Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:06

D

2D/3D/4D Data

► Query Evaluation Techniques for Multidimensional Data

DAC

► Discretionary Access Control

Daplex

Tore Risch
Uppsala University, Uppsala, Sweden

Definition

Daplex is a query language based on a functional data model [1] with the same name. The Daplex data model represents data in terms of entities and functions. The Daplex data model is close to the entity-relationship (ER) model with the difference that relationships between entities in Daplex have a logical direction, whereas ER relationships are directionless. Unlike ER entities, relationships, and properties are all represented as functions in Daplex. Also, entity types are defined as functions without arguments returning sets of a built-in type ENTITY. The entity types are organized in a type/subtype hierarchy. Functions represent properties of entities and relationships among entities. Functions also represent derived information. Functions may be set-valued and are invertible. The database is represented as tabulated function extents. Database updates change the function extents.

The Daplex query language has been very influential for many other query languages, both relational, functional, and object oriented. Queries are expressed declaratively in an iterative fashion over sets similar to the FLWR semantics of the XQuery language. Daplex queries cannot return entities but a value returned from a query must always be a literal.

The query language further includes schema (function) definition statements, update statements, constraints, etc.

Key Points

Daplex functions are defined using a DECLARE statement, for example:

DECLARE name(Student) = STRING

Where “Student” is a user defined entity type and “STRING” is a built-in type. Set valued functions are declared by a “ = > >” notation, e.g.

DECLARE course(Student) = > > Course

Entity types are functions returning the built-in type ENTITY, for example:

DECLARE Person() = > > ENTITY

Inheritance among entity types is defined by defining entities as functions returning supertypes, for example:

DECLARE Student() = > > Person

Functions may be overloaded on different entity types.

Queries in Daplex are expressed using a FOR EACH – SUCH THAT – PRINT fashion similar to the FLWR semantics of XQuery. For example:

```
FOR EACH X IN Employee
SUCH THAT Salary(X) > Salary(Manager (X))
PRINT Name(X)
```

The PRINT statement is here not regarded as a side effect but rather as defining the result set from the query. Derived functions are defined though the DEFINE statement, e.g.

```
DEFINE Course(Student) = > Course
SUCH THAT
    FOR SOME Enrollment
        Stud#(Student) = Stud#(Enrollment) AND
        Course#(Enrollment) = Course#(Course)
```

Daplex was first implemented in the Multibase system [2]. There, it was used as a multi-database query language to query data from several databases.

The P/FDM [1] data model and query language is close to Daplex. The query languages OSQL and AmosQL are also based on Daplex. These languages extend Daplex with object identifiers (OIDs) to represent actual entities and thus queries can there return entities as OIDs.

Cross-references

► AmosQL
► Functional Data Model
► OSQL
► P/FDM
► Query Language
► XQuery

Recommended Reading

1. Gray P.M.D., Kerschberg L., King P.J.H., and Poulovassilis A. (eds.). The Functional Approach to Data Management. Springer, Berlin, 2004.
2. Landers T. and Rosenberg R.L. An overview of Multibase. In Proceedings of the Second International Symposium on Distributed Databases, Berlin, Germany. North Holland, 1982, pp. 153–184.
3. Shipman D.W. The functional data model and the data language DAPLEX. ACM Trans. Database Syst., 6(1):140–173, 1981.

DAS

► Direct Attached Storage

Data

► Information Quality Assessment

Data Acquisition

► Data Acquisition and Dissemination in Sensor Networks

Data Acquisition and Dissemination in Sensor Networks

Turkmen Canli, Ashfaq Khokhar
University of Illinois at Chicago, Chicago, IL, USA

Synonyms

Data gathering; Data collection; Data acquisition

Definition

Wireless sensor networks (WSNs) are deployed to monitor and subsequently communicate various aspects of physical environment, e.g., acoustics, visual, motion, vibration, heat, light, moisture, pressure, radio, magnetic, biological, etc. Data acquisition and dissemination protocols for WSNs are aimed at collecting information from sensor nodes and forwarding it to the subscribing entities such that maximum data rate is achieved while maximizing the overall network life time. The information can be simple raw data or processed using basic signal processing techniques such as filtering, aggregation/compression, event detection, etc.

Historical Background

Wireless sensor networks consist of tiny dispensable smart sensor nodes, with limited battery power and processing/communication capabilities. In addition, these networks also employ more powerful "sink" node(s) that collect information from the sensor nodes and facilitate interfacing with the outside computing and communication infrastructure. WSNs are configured to execute two fundamental tasks: information acquisition/collection at the sink nodes, and dissemination of information to the nodes across the network. Existing data acquisition and dissemination techniques have been investigated for different levels of application abstractions including: structured data collection [1,5,6,11,12] in a query-database paradigm, and raw data acquisition in for field reconstruction and event recognition at the sink nodes [2,3,4,7,9,10,13].

Foundations

In WSNs, devices have limited battery life, which is generally considered non-replenishable, therefore the pertinent challenge is to design protocols and algorithms that maximize the network life time. In data acquisition and dissemination tasks, data volume is high therefore another optimization criteria is to increase throughput while reducing power consumption. Several data collection protocols aiming at data reduction using data transformation techniques have been suggested [2,10]. In [10], authors have proposed the use of wavelet compression to reduce data volume in structure monitoring WSN applications, thus resulting in low power consumption and reduced communication latency. Similar data transformation and compression techniques have been used to compute summary of the raw data [2].

In most of the data collection algorithms, the network nodes are organized into logical structures, and communication among the nodes and with the sink is realized using such logical structures. For example, in tree based data acquisition protocols, a collection tree is built that is rooted at the data collection center such as the sink node [8]. The dissemination of the data requests from the participating nodes and collection of data from the sensor nodes are accomplished using this tree. A cluster based data acquisition mechanism has been proposed in [3]. As shown in Fig. 1, nodes are organized into a fixed number of clusters, and nodes within each cluster dynamically elect a cluster head.

The data acquisition is carried out in two phases. In the first phase, cluster heads collect data from their cluster nodes. In the second phase, cluster heads send collected data to the nodes that have subscribed to the data. The cluster heads are re-elected to balance energy consumption among the nodes in the cluster. Zhang et al. [13] have proposed an adaptive cluster based data collection protocol that dynamically adjusts the number of cluster heads to the traffic load in the network. This dynamic traffic model is developed at the sink node.

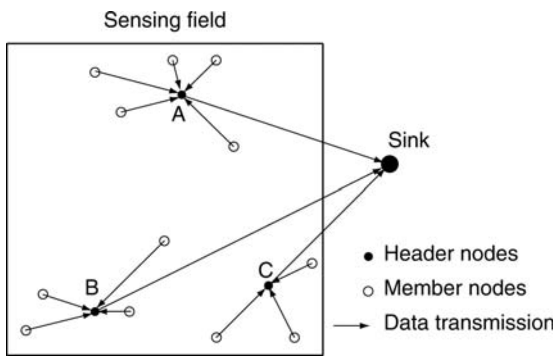
In [7], network is divided into virtual grids and sensor nodes in each grid are classified as either gateway nodes or internal nodes. For example, in Fig. 2 nodes B, G are selected as gateway nodes that are responsible for transmitting data to nodes outside the grid. By doing so, data contention and redundant data transmission of a packet are reduced, which saves energy.

The common characteristic of all the aforementioned protocols is the pro-actively built routing infrastructure. As an alternative, authors in [4] have proposed the directed diffusion approach. The routing infrastructure is constructed on the fly. The sink node disseminates its interest to the network and gradients are set-up from nodes that match the sink's interest. There may be more than one path from a sensor node to the sink node. Sink nodes regulate data rate across all the paths.

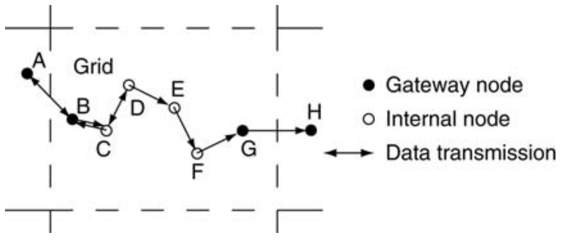
Data acquisition and dissemination techniques designed with a higher level of application abstraction model the sensor network as a distributed database system. In these techniques, data center disseminates its queries, and database operation such as "join" or "select" operations are computed distributively using sensor nodes that have the requested data. For example, in [12] a new layer, referred to as query layer, is proposed that is logically situated between the network and application layers of the network protocol stack. This layer processes descriptive queries and determines a power efficient execution plan that makes use of in-network processing and aggregation operations. An in-network aggregation is realized by packet merging or by incrementally applying aggregation operators such as min, max, count, or sum.

Cougar system [11] presents a framework for specifying a Query execution plan in WSNs. It allows specification of routing among sensor nodes and execution of aggregate operation over the collected data.

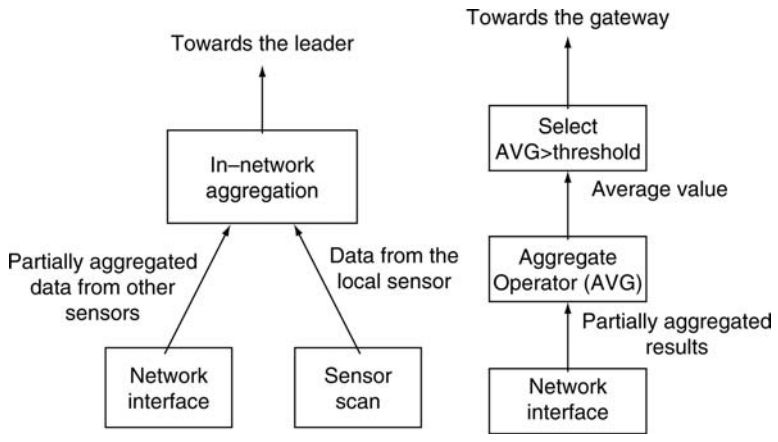
As depicted in Fig. 3, each sensor node samples the environment as specified by the query. According to the execution plan, sampled data is sent to a leader node, or together with the partially aggregated data received from other nodes, an aggregation operators is applied. The new partially aggregated value is then sent towards the leader node. Partial aggregation is possible only for the aggregation operators that can be computed incrementally. The volume of data is decreased by partial or incremental aggregation. The responsibility of the leader node is to combine all the partially



Data Acquisition and Dissemination in Sensor Networks. Figure 1. Clustering concept as proposed in LEACH [3].



Data Acquisition and Dissemination in Sensor Networks. Figure 2. Principle of LAF.



Data Acquisition and Dissemination in Sensor Networks. Figure 3. Query plan at a source and leader node [11].

aggregated results and report it to the gateway node if the value exceed the set threshold.

In TinyDB framework [5], WSN is viewed as one big relational table. The columns of the table correspond to the type of phenomenon observed, i.e, humidity, temperature, etc. The aim is to reduce the power consumption during data acquisition by in-network processing of raw data. In other words, it addresses questions such as when data should be sampled for a particular query, which sensors should respond to a query, in which order sensors should sample the data, and how to achieve balance between in-network processing of the raw samples and collection of the raw samples at sink nodes without any processing. Moreover, the structured query language (SQL) is extended specifically for sensor network applications. Keywords such as, `SAMPLE`, `ON EVENT` and `LIFETIME` have been added to SQL language to facilitate realization of basic sensor network applications.

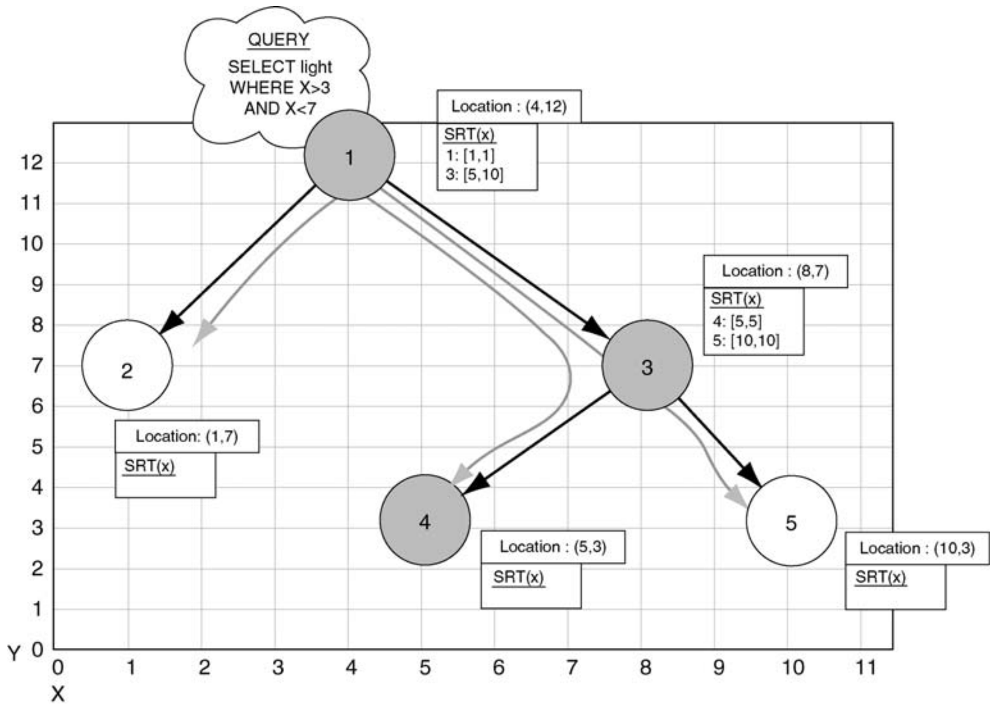
Through `SAMPLE` clause, the sampling rate of the sensors can be controlled. `LIFETIME` allows automatic sampling rate adjustment for a given lifetime. `ON EVENT` is used for trigger, i.e. query is executed only when the specified event occurs. A query sample shown below [5] illustrates the use of extended SQL. In this example query, the sampling period is set via introducing `SAMPLE` clause. The query planner needs meta data information regarding power consumption, sensing and communication costs to compute lifetime approximation and execute the `LIFETIME` clause. `ON EVENT` type queries may trigger multiple instances of same type of internal query.

```
SELECT COUNT(*)
FROM sensors AS s, recentLight AS r1
WHERE r1.nodeid=s.nodeid
AND s.light < r1.light
SAMPLE INTERVAL 10s
```

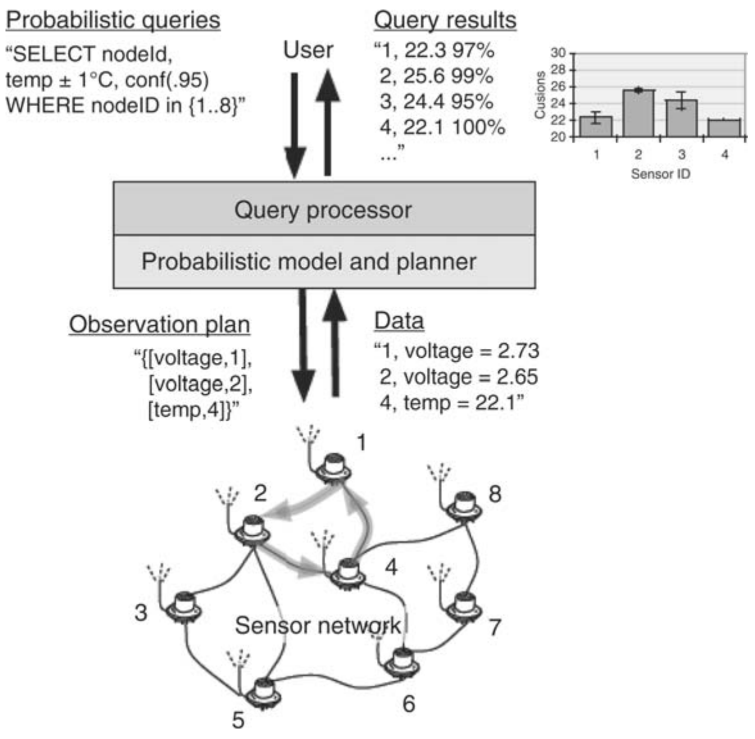
Optimization of the data sampling order can also reduce energy consumption significantly. Typically, a sensor node has more than one on-board sensor, i.e. nodes may have all temperature, humidity, pressure, etc. sensors on a single sensing platform. If in the query it is requested to report the temperature of the nodes where humidity value is greater than some threshold, it would be inefficient to simultaneously sample temperature and humidity values. The energy spent on sampling temperature values where humidity is less than the threshold could be saved by reordering the predicate evaluation [5].

The semantic tree (SRT) [5] is a mechanism that allows nodes to find out whether their children have the data for the incoming query. Every parent node stores the range of its children's values. Therefore when query arrives to a node it is not forwarded to those children that do not have the data. For instance in Fig. 4, node 1 will not send query request to node 2, similarly, node 3 will not send query to node 5.

Authors in [1] have used probabilistic models to facilitate efficient query processing and data acquisition on sensor networks. The idea is to build statistical model of the sensor readings from stored and current readings of the sensors. Whenever an SQL query is submitted to the network, constructed model is used to provide answers. Accuracy of the requested data can be specified by setting the



Data Acquisition and Dissemination in Sensor Networks. Figure 4. A semantic routing tree in use for a query. Gray arrows indicate flow of the query down the tree, gray nodes must produce or forward results in the query [5].



Data Acquisition and Dissemination in Sensor Networks. Figure 5. Model based querying in sensor networks [1].

confidence interval in the SQL statement. Figure 5 illustrates typical SQL query. Depending on error tolerance levels, response to the query may involve collecting information from every sensor node, for example, if 100% accuracy is required. On the other hand, if the error tolerance is high, query can be answered by only using the constructed model.

Key Applications

Building Health Monitoring, Micro-climate Monitoring, Habitat Monitoring, Hazardous Environment Monitoring.

Cross-references

► Ad-hoc Queries in Sensor Networks
► Continuous Queries in Sensor Networks
► Data Aggregation in Sensor Networks
► Data Compression in Sensor Networks
► Data Estimation in Sensor Networks
► Data Fusion in Sensor Networks
► Data Storage and Indexing in Sensor Networks
► Database Languages for Sensor Networks
► Model-Based Querying in Sensor Networks
► Query Optimization in Sensor Networks
► Sensor Networks

Recommended Reading

1. Deshpande A., Guestrin C., Madden S.R., Hellerstein J.M., and Hong W. Model-driven data acquisition in sensor networks. 2004.
2. Ganesan D., Greenstein B., Perelyubskiy D., Estrin D., and Heidemann J. An evaluation of multi-resolution storage for sensor networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003, pp. 89–102.
3. Heinzelman W.R., Chandrakasan A., and Balakrishnan H. Energy-efficient communication protocol for wireless microsensor networks. In Proc. 33rd Annual Hawaii Conf. on System Sciences, 2000, pp. 8020.
4. Intanagonwiwat C., Govindan R., Estrin D., Heidemann J., and Silva F. Directed diffusion for wireless sensor networking. IEEE/ACM Trans. Netw., 11(1):2–16, 2003.
5. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. The design of an acquisitional query processor for sensor networks. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 491–502.
6. Madden S.R., Franklin M.J., Hellerstein J.M., and Hong W. TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst., 30(1):122–173, 2005, doi:http://doi.acm.org/10.1145/1061318.1061322.
7. Sabbineni M.H. and Chakrabarty S.M.K. Location-aided flooding: an energy-efficient data dissemination protocol for wireless sensor networks. IEEE Trans. Comput., 54(1):36–46, 2005.
8. Szewczyk R., Osterweil E., Polastre J., Hamilton M., Mainwaring A., and Estrin D. Habitat monitoring with sensor networks. Commun. ACM, 47(6):34–40, 2004.
9. Xi Y., Yang W., Yamauchi N., Miyazaki Y., Baba N., and Ikeda H. Real-time data acquisition and processing in a miniature wireless monitoring system for strawberry during transportation. Proc. Int. Technical Conf. of IEEE Region 10 (Asia Pacific Region), 2006.
10. Xu N., Rangwala S., Chintalapudi K.K., Ganesan D., Broad A., Govindan R., and Estrin D. A wireless sensor network for structural monitoring. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 13–24.
11. Yao Y. and Gehrke J. The cougar approach to in-network query processing in sensor networks. SIGMOD Rec., 31(3):9–18, 2002.
12. Yao Y. and Gehrke J. Query processing in sensor networks. 2003.
13. Zhan X., Wang H., and Khokhar A. An energy-efficient data collection protocol for mobile sensor networks. Vehicular Technology Conference, 2006, pp. 1–15.

Data Aggregation in Sensor Networks

JUN YANG[1], KAMESH MUNAGALA[1], ADAM SILBERSTEIN[2]
[1]Duke University, Durham, NC, USA
[2]Yahoo! Research Silicon Valley, Santa Clara, CA, USA

Definition

Consider a network of N sensor nodes, each responsible for taking a reading v_i $(1 \leq i \leq N)$ in a given epoch. The problem is to compute the result of an aggregate function (cf. *Aggregation*) over the collection of all readings $v_1, v_2, \ldots, v_N$ taken in the current epoch. The final result needs to be available at the base station of the sensor network. The aggregate function ranges from simple, standard SQL aggregates such as SUM and MAX, to more complex aggregates such as top-k, median, or even a contour map of the sensor field (where each value to be aggregated is a triple $\langle x_i, y_i, z_i \rangle$, with x_i and y_i denoting the location coordinates of the reading z_i).

In battery-powered wireless sensor networks, energy is the most precious resource, and radio communication is often the dominant consumer of energy. Therefore, in this setting, the main optimization objective is to minimize the total amount of communication needed in answering an aggregation query. A secondary objective is to balance the energy consumption across all sensor nodes, because the first node to run out of battery may render a large portion of the network inaccessible.

There are many variants of the above problem definition. For example, the aggregation query may be *continuous* (cf. *Continuous Queries in Sensor Networks*) and produce a new result for each epoch; not all nodes may participate in aggregation; the result may be needed not at the base station but at other nodes in the network. Some of these variants are further discussed below.

Historical Background

Early deployments of wireless sensor networks collect and report all data to the base station without summarization or compression. This approach severely limits the scale and longevity of sensor networks, because nodes spend most their resources forwarding data on behalf of others. However, many applications do not need the most detailed data; instead, they may be interested in obtaining a summary view of the sensor field, monitoring outlier or extreme readings, or detecting events by combining evidence from readings taken at multiple nodes. Data aggregation is a natural and powerful construct for applications to specify such tasks.

Data aggregation is supported by *directed diffusion* [9], a data-centric communication paradigm for sensor networks. In directed diffusion, a node diffuses its interests for data in the network. Then, nodes with data matching the interests return the relevant data along the reverse paths of interest propagation. Intermediate nodes on these paths can be programmed to aggregate relevant data as it converges on these nodes. Systems such as *TinyDB* and *Cougar* take a database approach, providing a powerful interface for applications to pose declarative queries including aggregation over a sensor network (cf. *Database Languages for Sensor Network Tasking*), and hiding the implementation and optimization details from application programmers. The seminal work by Madden et al. [12] on *TAG* (*Tiny AGgregation*) is the first systematic study of database-style aggregation in sensor networks. One of the first efforts at supporting more sophisticated aggregation queries beyond SQL aggregates is the work by Hellerstein et al. [8], which shows how to extend TAG to compute contour maps, wavelet summaries, and perform vehicle tracking. Since these early efforts, the research community has made significant progress in sensor data aggregation; some of the developments are highlighted below.

Foundations

The key to efficient sensor data aggregation is in-network processing. On a typical sensor node today, the energy cost of transmitting a byte over wireless radio is orders-of-magnitude higher than executing a CPU instruction. When evaluating an aggregate query, as data converges on an intermediate node, this node can perform aggregate computation to reduce the amount of data to be forwarded, thereby achieving a favorable tradeoff between computation and communication.

To illustrate, consider processing a simple SUM aggregate with TAG [12]. TAG uses a routing tree rooted at the base station spanning all nodes in the network. During aggregation, each node listens to messages from its children in the routing tree, computes the sum of all values in these messages and its own reading, and then transmits this result – which equals the sum of all readings in the subtree rooted at this node – to its parent. To conserve energy, each node only stays awake for a short time interval to listen, compute, and transmit. To this end, TAG coordinates the nodes' communication schedules: the beginning of a parent node's interval must overlap with the end of its children's intervals, so that the parent is awake to receive children's transmissions. Overall, to compute SUM, each node needs to send only one constant-size message, so the total amount of communication is $\Theta(N)$. In comparison, for the naïve approach, which sends all readings to the root, the amount of data that needs to be forwarded by a node increases closer to the root, and the total amount of communication can be up to $\Theta(Nd)$, where d is the depth of the routing tree. Clearly, in-network aggregation not only decreases overall energy consumption, but also balances energy consumption across nodes.

The above algorithm can be generalized to many other aggregates. Formally, an aggregate function can be implemented using three functions: an *initializer* f_i converts an input value into a *partial aggregate record; a merging function* f_m combines two partial aggregate records into one; finally, an *evaluator* f_e computes the final result from a partial aggregate record. During aggregation, each node applies f_i to its own reading; each non-leaf node invokes f_m to merge partial aggregate records received from its children with that of its own; the root uses f_e to compute the aggregate result from the final partial aggregate record. As an example, standard deviation can be computed (in theory, without regard to numerical stability) using the following functions, where the partial aggregate record is a triple $\langle s, r, n \rangle$ consisting of a sum (s), a sum of squares (r), and a count (n):

$$f_i(v) = \langle v, v^2, 1 \rangle,$$

$$f_m(\langle s_1, r_1, n_1 \rangle, \langle s_2, r_2, n_2 \rangle) = \langle s_1 + s_2, r_1 + r_2, n_1 + n_2 \rangle,$$

$$f_e(\langle s, r, n \rangle) = \frac{1}{n}\sqrt{nr - s^2}.$$

In general, one cannot expect the partial aggregate record to be of constant size for arbitrary aggregate functions. Consider the following examples. For a top-k aggregate, which finds the k largest values, a partial aggregate record needs to be of size $\Theta(k)$. For exact median computation, the size requirement becomes $\Theta(N)$, which is no better than the naïve approach of sending all readings to the root. For contour map construction, the size of the partial aggregate records depends on the complexity of the field being sensed, and can be $\Theta(N)$ in the worst case.

Approximate Aggregation

Approximation is a popular and effective technique for bounding the communication cost in evaluating complex aggregation functions. The basic idea is to replace an exact partial aggregate record with an approximate partial aggregate record that consumes less space and is therefore cheaper to send. An early example of applying this idea is the work by Hellerstein et al. [8]. Since then, a diverse collection of approximation methods has been developed for many aggregate functions; a few illustrative examples are given below.

To compute order-statistics (e.g., quantile queries including top-k and median), Greenwald and Khanna [7] propose a technique based on *ε-approximate quantile summaries*. An ε-approximate quantile summary for a collection S of sensor readings is an ordered subset $\{q_i\}$ of S, where each q_i is associated with a lower bound $r\min_i$ and an upper bound $r\min_i$ on q_i's rank within S, and the difference between $r\max_{i+1}$ and $r\min_i$ is no greater than $2\varepsilon|S|$. Any quantile query over S can be answered instead on this summary within an additive rank error of $\varepsilon|S|$. Specifically, a query requesting the r-th ranked reading can be answered by returning q_j from the summary, where $r - \varepsilon|S| \leq r\min_j$ and $r\max \leq r + \varepsilon|S|$. Greenwald and Khanna represent a partial aggregate record sent up from a node u by a set of quantile summaries – at most one for each class numbered 1 through $\log N$ – which together disjointly cover all readings in the subtree rooted at u; the summary for class i covers between 2^i and $2^{i+1}-1$ readings using at most $(\log N/\varepsilon + 1)$ of these readings. Each sensor node starts with an $\varepsilon/2$-approximate summary of all its local readings. Each intermediate node merges summaries from its children together with its own summary into up to $\log N$ merged summaries, prunes each of them down to the maximum size allowed, and then sends them up to the parent. Finally, the root merges all summaries into a single one and prunes it down to $(\log N/\varepsilon + 1)$ readings. Although pruning introduces additional error, the use of per-class summaries bounds the error in a class-i summary to $\varepsilon/2 + (i/(2\log N/\varepsilon))$, which in turn allows the error in the final summary to be bounded by ε. Overall, the communication cost incurred by each node during aggregation is only $o(\log^2 N/\varepsilon)$.

Silberstein et al. [15] approach the problem of computing top-k aggregates using a very different style of approximation. Instead of having each node always sending the top k readings in its subtree, a node sends only the top k' readings among its local reading and those received from its children, where $k' \leq k$. The appropriate setting of k' for each node is based on the samples of past sensor readings, or, more generally, a model capturing the expected behavior of sensor readings (cf. *Model-Based Querying in Sensor Networks*). Intuitively, a subtree that tends to contribute few of the top values will be allotted a smaller k'. Unlike the ε-approximate quantile summaries, which provide hard accuracy guarantees, the accuracy of this approach depends on how well the past samples or the model reflect the current behavior of readings. Nevertheless, the approach can be augmented by transmitting additional information needed to establish the correctness of some top-k answers, thereby allowing the approximation quality to be assessed.

As a third example, the contour map of a sensor field is a complex spatial aggregate defined over not only values but also locations of the sensor readings. In this case, the partial aggregate record produced by a node is a compact, usually lossy, representation of the contour map encompassing all readings in this node's subtree. In Hellerstein et al. [8], each contour in the map is represented by an orthogonal polygon whose edges follow pre-imposed 2-d rectangular grid lines. This polygon is obtained by starting with the minimum bounding rectangle of the contour, and then repeatedly subtracting the largest-area rectangle that does not contain any point in the contour, until a prescribed limit on the number of vertices is reached. Gandhi et al. [5] use general polygons instead of orthogonal ones. During aggregation, each node constructs and sends to its parent an approximate description of its contour map consisting of up to *k* possibly disconnected line segments. The root then connects and untangles such line segments to obtain the final contour map. It is shown that the approximation error in the *k* -segment representation produced by distributed aggregation is within a constant factor of the smallest possible error attainable by any *k* segments, and it is conjectured that the resulting contour map has size *o(k)*.

Duplicate-Insensitive Aggregation

Approximate aggregation methods based on *duplicate-insensitive synopses* (cf. *Data Sketch/Synopsis*) are especially worth noting because of their resiliency against failures, which are common in sensor networks. Tree-based aggregation techniques are vulnerable to message failures. If a message carrying the partial aggregate record from a node fails, all information from that subtree will be lost, resulting in significant error. Sending the same message out on multiple paths towards the base station decreases the chance of losing all copies, and is a good solution for aggregation functions such as MAX. However, for other aggregation functions whose results are sensitive to duplicates in their inputs, e.g., SUM and COUNT, having multiple copies of the same partial aggregation record causes a reading to participate multiple times in aggregation, leading to incorrect results. In general, if an aggregation method is *order- and duplicate-insensitive (ODI)* [14], it can be implemented with more failure-resilient routing structures such as directed acyclic graphs, without worrying about the duplicates they introduce. The challenge, then, is in designing ODI aggregation methods for duplicate-sensitive aggregation functions.

Duplicate-insensitive synopses provide the basis for computing many duplicate-sensitive aggregation functions approximately in an ODI fashion. This approach is pioneered by Considine et al. [1] and Nath et al. [14]. To illustrate the idea, consider COUNT, which is duplicate-sensitive. Nodes in the sensor network are organized into rings centered at the base station, where the i -th ring includes all nodes at i hops away from the base station. A partial aggregate record is a Flajolet-Martin sketch (cf. *FM Sketch*), a fixed-size bit-vector for estimating the number of distinct elements in a multi-set. This sketch is duplicate-insensitive by design: conceptually, it is obtained by hashing each element to a bitmap index (using an exponential hash function) and setting that bit to one. During aggregation, each node first produces a sketch for its local sensors. A node in the i-th ring receives sketches from its neighbors (i.e., nodes within direct communication distance) in the $(i+1)$-th ring, takes the bitwise OR of all these sketches and its own, and broadcasts the result sketch to all its neighbors in the $(i-1)$-th ring. Taking advantage of broadcast communication, each node sends out only one message during aggregation, but the information therein can reach the base station via multiple paths, boosting reliability. The overall COUNT can be estimated accurately with high probability using sketches of size $\Theta(\log N)$.

The failure-resiliency feature comes with two costs in the above approach: the final answer is only approximate, and the size of each message is larger than the tree-based exact-aggregation approach. Manjhi et al. [13] have developed an adaptive, hybrid strategy that combines the advantages of the two approaches by applying them to different regions of the network, and dynamically exploring the tradeoffs between the two approaches.

Temporal Aspects of Aggregation

The preceding discussion has largely ignored the temporal aspects of aggregation. In practice, aggregation queries in sensor networks are often *continuous* (cf. *Continuous Queries in Sensor Networks*). In its simplest form, such a query executes continuously over time and produces, for each epoch, an aggregate result computed over all readings acquired in this epoch. A key optimization opportunity is that sensor readings often are temporally correlated and do not change haphazardly over time. Intuitively, rather than re-aggregating from scratch in every epoch, evaluation efforts should focus only on relevant changes since the last epoch.

An effective strategy for implementing the above intuition is to install on nodes local constraints that dictate when changes in subtrees need to be reported. These constraints carry memory about past readings and filter out reports that do not affect the current aggregate result, thereby reducing communication. For example, to compute MAX continuously, Silberstein et al. [16] set a threshold at each node, which is always no less than its local reading and its children's thresholds. A node sends up the current maximum value in its subtree only if that value exceeds the threshold; the threshold is then adjusted higher. If the current global maximum falls, nodes with thresholds higher than the new candidate maximum must be visited to find the new maximum; at the same time, their thresholds are adjusted lower. Thresholds control the tradeoff between reporting and querying costs, and can be set adaptively at runtime. Alternatively, optimum settings can be found by assuming adversarial data behavior or using predictive models of data behavior.

As another example, consider continuous SUM. Unlike MAX, even a small change in one individual reading affects the final SUM result. Approximation is thus needed to do better than one message per node per epoch. Deligiannakis et al. [4] employ an interval constraint at each node, which bounds the sum of all current readings within the subtree. In each epoch, the node sends its estimate of this partial sum to its parent only if this value falls outside its interval; the interval then recenters at this value. The length of the interval controls the error allowance for the subtree. Periodically, based on statistics collected, the interval lengths are adjusted by recursively redistributing the total error allowed in the final result to all nodes.

Continuous versions of more complex queries, for which approximation is needed to reduce the size of partial aggregate records, have also been studied. For example, Cormode et al. [2] show how to continuously compute ε-approximate quantile summaries using a hierarchy of constraints in the network to filter out insignificant changes in subtree summaries. As with other techniques described above, a major technical challenge lies in allocating error tolerance to each constraint; optimum allocation can be computed for predictive models of data behavior. Xue et al. [17] consider the continuous version of the contour map query. Instead of sending up the entire partial aggregate record (in this case, a contour map for the subtree), only its difference from the last transmitted version needs to be sent.

In the continuous setting, aggregation can apply not only spatially to the collection of readings acquired in the same epoch, but also temporally over historical data (e.g., recent readings in a sliding window). Cormode et al. [3] consider the problem of continuously computing time-decaying versions of aggregation functions such as SUM, quantiles, and heavy hitters. The contribution of a reading taken at time t' to the aggregate result at the current time $t' = t' + \Delta$ is weighted by a user-defined *decay function* $f(\Delta) \leq 0$, which is non-increasing with Δ. The solution is based on duplicate-insensitive sketching techniques, and it approximates a general decay function using a collection of sliding windows of different lengths.

Other Aspects of Aggregation in Sensor Networks

Besides the above discussion, there are many other aspects of sensor data aggregation that are not covered by this entry; some of them are outlined briefly below.

Most techniques presented earlier opportunistically exploit in-network processing, whenever two partial aggregate records meet at the same node following their standard routes to the base station. More generally, routing can be made aggregation-driven [11] by encouraging convergence of data that can be aggregated more effectively (e.g., the merged partial aggregate record uses less space to achieve the required accuracy).

Oftentimes, only a sparse subset of nodes contribute inputs to aggregation, and this subset is not known a priori, e.g., when aggregation is defined over the output of a filter operation evaluated locally at each node. The challenge in this scenario is to construct an ad hoc aggregation tree of high quality in a distributed fashion [6].

Finally, for some applications, the final aggregate result is needed at all nodes in the sensor network as opposed to just the base station. *Gossiping* is an effective technique for this purpose [10], which relies only on local communication and does not assume any particular routing strategy or topology.

Key Applications

Aggregation is a fundamental query primitive indispensible to many applications of sensor networks (cf. *Sensor Network, Applications of Senor Network Data Management*). It is widely used in expressing and implementing common sensor network tasks such as summarization, compression, monitoring, and event detection. Even for applications that are interested in collecting all detailed sensor readings, aggregation can be used in monitoring system and data characteristics, which support maintenance of the sensor network and optimization of its operations.

Cross-references

► Ad-Hoc Queries in Sensor Networks
► Aggregation

► Continuous Queries in Sensor Networks
► Data Compression in Sensor Networks
► Data Fusion in Sensor Networks

Recommended Reading

1. Considine J., Li F., Kollios G., and Byers J. Approximate aggregation techniques for sensor databases. Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 449–460.
2. Cormode G., Garofalakis M., Muthukrishnan S., and Rastogi R. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 25–36.
3. Cormode G., Tirthapura S., and Xu B. Time-decaying sketches for sensor data aggregation. ACM Symposium on Principles of Distributed Computing, 2007, pp. 215–224.
4. Deligiannakis A., Kotidis Y., and Roussopoulos N. Hierarchical in-network data aggregation with quality guarantees. Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 658–675.
5. Gandhi S., Hershberger J., and Suri S. Approximate isocontours and spatial summaries for sensor networks. Proc. 6th Int. Symp. Inf. Proc. in Sensor Networks, 2007, pp. 400–409.
6. Gao J., Guibas L.J., Milosavljevic N., and Hershberger J. Sparse data aggregation in sensor networks. Proc. 6th Int. Symp. Inf. Proc. in Sensor Networks, 2007, pp. 430–439.
7. Greenwald M. and Khanna S. Power-conserving computation of order-statistics over sensor networks. Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2004, pp. 275–285.
8. Hellerstein J.M., Hong W., Madden S., and Stanek K. Beyond average: toward sophisticated sensing with queries. Proc. 2nd Int. Workshop Inf. Proc. in Sensor Networks, 2003, pp. 63–79.
9. Intanagonwiwat C., Govindan R., and Estrin D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. Proc. 6th Annual Int. Conf. on Mobile Computing and Networking, 2000, pp. 56–67.
10. Kempe D., Dobra A., and Gehrke J. Gossip-based computation of aggregate information. IEEE Symposium on Foundations of Computer Science, 2003, pp. 482–491.
11. Luo H., Y. Liu, and S. Das. Routing correlated data with fusion cost in wireless sensor networks. IEEE Transactions on Mobile Computing, 11(5):1620–1632, 2006.
12. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. TAG: a tiny aggregation service for ad-hoc sensor networks. Proc. 5th USENIX Symp. on Operating System Design and Implementation. 2002.
13. Manjhi A., Nath S., and Gibbons P.B. Tributaries and deltas: efficient and robust aggregation in sensor network streams. Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 287–298.
14. Nath S., Gibbons P.B., Seshan S., and Anderson Z.R. Synopsis diffusion for robust aggregation in sensor networks. Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems. 2004, pp. 250–262.
15. Silberstein A., Braynard R., Ellis C., and Munagala K. A sampling-based approach to optimizing top-k queries in sensor networks. Proc. 22nd Int. Conf. on Data Engineering. 2006.
16. Silberstein A., Munagala K., and Yang J. Energy-efficient monitoring of extreme values in sensor networks. Proc. ACM SIGMOD Int. Conf. on Management of Data. 2006.
17. Xue W., Luo Q., Chen L., and Liu Y. Contour map matching for event detection in sensor networks. Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 145–156.

Data Analysis

► Data Mining

Data Anomalies

► Data Conflicts

Data Broadcasting, Caching and Replication in Mobile Computing

Panos K. Chrysanthis[1], Evaggelia Pitoura[2]
[1]University of Pittsburgh, Pittsburgh, PA, USA
[2]University of Ioannina, Ioannina, Greece

Synonyms

Data dissemination; Push/pull delivery; Data copy

Definition

Mobile computing devices (such as portable computers or cellular phones) have the ability to communicate while moving by being connected to the rest of the network through a wireless link. There are two general underlying infrastructures: single-hop and multi-hop ones. In *single-hop* infrastructures, each mobile device communicates with a stationary host, which corresponds to its point of attachment to the wired network. In *multi-hop* infrastructures, an ad-hoc wireless network is formed in which mobile hosts participate in routing messages among each other. In both infrastructures, the hosts between the source (or sources) and the requester of data (or data sink) form a dissemination tree. The hosts (mobile or stationary) that form the dissemination tree may store data and participate in computations towards achieving *in network* processing. Challenges include [14], (i) *intermittent connectivity*, which refers to both short and long periods of network unavailability, (ii) *scarcity of resources*, including storage and battery life, and (iii) *mobility* itself.

To handle these challenges, data items may be stored locally (cached or replicated) at the requester or at the intermediate nodes of the dissemination tree. Cache and replication aim at increasing availability in the case of network disconnections or host failures as well as at handling intermittent connectivity. Mobility introduces additional challenges in maintaining cache and replica consistency and in replica placement protocols. Wireless data delivery in both infrastructures physically supports broadcasting. This broadcast facility has been used for providing a push-mode of data dissemination where a server broadcasts data to a large client population often without an explicit request from the clients. Issues addressed by related research include broadcast scheduling and organization (i.e, which items to broadcast and in which order), indexing broadcast data, and update propagation.

Historical Background

Mobile computing can be traced back to file systems and the need for disconnected operations in the late 1980s. With the rapid growth in mobile technologies and the cost effectiveness in deploying wireless networks in the 1990s, the goal of mobile computing was the support of AAA (anytime, anywhere and any-form) access to data by users from their portable computers, mobile phones and other devices with small displays and limited resources. These advances motivated research in data management in the early 1990s.

Foundations

Data Broadcasting

Many forms of wireless network infrastructures rely on broadcast technology to deliver data to large client populations. As opposed to point-to-point data delivery, broadcast delivery is scalable, since a single broadcast response can potentially satisfy many clients simultaneously. There are two basic modes of broadcast data delivery: pull-based and push-based. With *push-based* data delivery, the server sends data to clients without an explicit request. With *pull-based* or *on-demand* broadcast delivery, data are delivered only after a specific client request. In general, access to broadcast data is sequential with clients monitoring the broadcast channel and retrieving any data items of interest as they arrive. The smallest access unit of broadcast data is commonly called a *bucket* or *page*.

Scheduling and Organization

A central issue is determining the content of the broadcast or *broadcast scheduling*. Scheduling depends on whether we have on demand, push or hybrid delivery. In on-demand broadcast, there is an up-link channel available to clients to submit requests. The item to be broadcast next is chosen among those for which there are pending requests. Common heuristics for on-demand scheduling include First Come First Served and Longest Wait First [7]. The $R \times W$ strategy selects the data item with the maximal $R \times W$ value, where R is the number of pending requests for an item and W the amount of time that the oldest pending request for that item has spent waiting to be served [3]. More recent schemes extended $R \times W$ to consider the semantics of the requested data and applications such as subsumption properties in data cubes [15]. Push-based broadcast scheduling assumes a-priori knowledge of client access distributions and prepares an off-line schedule. Push-based data delivery is often periodic. In hybrid broadcast, the set of items is partitioned, so that some items are pushed, i.e., broadcast continuously, and the rest are pulled, i.e., broadcast only after being requested [2]. Commonly, the partition between push and pull data is based on popularity with the most popular items being pushed periodically and the rest delivered on demand. One problem is that for push items, there is no way to detect any changes in their popularity. One solution is to occasionally stop broadcasting some pushed items. This forces clients to send explicit requests for them, which can be used to estimate their popularity [16]. An alternative that avoids flooding of requests requires a percentage of the clients to submit an explicit request irrespective of whether or not a data item appears on the broadcast [5].

The organization of the broadcast content is often called *broadcast program*. In general, broadcast organizations can be classified as either *flat* where each item is broadcast exactly once or *skewed* where an item may appear more than once. One can also distinguish between *clustered* organizations, where data items having the same or similar values at some attribute appear consecutively, and *non-clustered* ones, where there is no such correlation. In skewed organizations, the broadcast frequency of each item depends on its popularity. For achieving optimal access latency or response time, it was shown that (i) the relative number of appearances of items should be proportional to the square root of their access probabilities and (ii) successive broadcasts of the same item should be at equal distances [7]. It was also shown that the *Mean Aggregate Access* (MAD) policy that selects to broadcast next the item whose access probability $\times$ the interval since its last broadcast is the highest achieves close to optimal response time [17]. Along these lines, a practical skewed push broadcast organization is that of *broadcast disks* [1]. Items are assigned to virtual disks with different "speeds" based on their popularity with popular items being assigned to fast disks. The spin speed of each disk is simulated by the frequency with which the items assigned to it are broadcast. For example, the fact that a disk D_1 is three times faster than a disk D_2, means that items assigned to D_1 are broadcast three times as often as those assigned to D_2. To achieve this, each disk is split into smaller equal-sized units called chunks, where the number of chunks per disk is inversely proportional to the relative frequence of the disk. The broadcast program is generated by broadcasting one chunk from each disk and cycling through all the chunks sequentially over all the disks.

Indexing

To reduce energy consumption, a mobile device may switch to *doze* or *sleep* mode when inactive. Thus, research in wireless broadcast also considers reducing the *tuning time* defined as the amount of time a mobile client remains active listening to the broadcast. This is achieved by including index entries in the broadcast so that by reading them, the client can determine when to tune in next to access the actual data of interest. Adding index entries increases the size of the broadcast and thus may increase access time. The objective is to develop methods for allocating index entries together with data entries on the broadcast channel so that both access and tuning time are optimized. In *(1, m) indexing* [18], an index for all data items is broadcast following every fraction ($1/m$) of the broadcast data items. *Distributed indexing* [18] improves over this method by instead of replicating the whole index m times, each index segment describes only the data items that follow it. Following the same principles, different indexing schemes have been proposed that support different query types or offer different trade-offs between access and tuning time. Finally, instead of broadcasting an index, *hashing-based techniques* have also been applied.

Data Caching and Replication

A mobile computing device (such as a portable computer or cellular phone) is connected to the rest of the network through a wireless link. Wireless communication has a double impact on the mobile device since the limited bandwidth of wireless links increases the response times for accessing remote data from a mobile host and transmitting as well as receiving of data are high energy consumption operations. The principal goal of caching and replication is to store appropriate pieces of data locally at the mobile device so that it can operate on its own data, thus reducing the need for communication that consumes both energy and bandwidth. Several cost-based caching policies along the principles of *greedy-dual* ones have been proposed that consider energy cost.

In the case of broadcast push, the broadcast itself can be viewed as a "cache in the air." Hence, in contrast to traditional policies, performance can be improved by clients caching those items that are accessed frequently by them but are not popular enough among all clients to be broadcast frequently. For instance, a cost-based cache replacement policy selects as a victim the page with the lowest p/x value, where p is the local access probability of the page and x its broadcast frequency [1]. Prefetching can also be performed with low overhead, since data items are broadcast anyway. A simple prefetch heuristic evaluates the worth of each page on the broadcast to determine whether it is more valuable than some other page in cache and if so, it swaps the cache page with the broadcast one.

Replication is also deployed to support *disconnected operation* that refers to the autonomous operation of a mobile client, when network connectivity becomes either unavailable (for instance, due to physical constraints), or undesirable (for example, for reducing power consumption). Preloading or prefetching data to sustain a forthcoming disconnection is often termed *hoarding*. Optimistic approaches to consistency control are typically deployed that allow data to be accessed concurrently at multiple sites without a priori synchronization between the sites, potentially resulting in short term inconsistencies. At some point, operations performed at the mobile device must be synchronized with operations performed at other sites. Synchronization depends on the level at which correctness is sought. This can be roughly categorized as replica-level correctness and transaction-level correctness. At the *replica level*, correctness or coherency requirements are expressed per item in terms of the allowable divergence among the values of the copies of each item. At the *transaction level*, the strictest form of correctness is achieved through global serializability that requires the execution of all transactions running at mobile and stationary hosts to be equivalent to some serial execution of the same transactions. With regards to update propagation with *eager replication*, all copies of an item are synchronized within a single transaction, whereas with *lazy replication*, transactions for keeping replica coherent execute as separate, independent database transactions after the original transaction commits.

Common characteristics of protocols for consistency in mobile computing include:

- The propagation of updates performed at the mobile site follows in general lazy protocols.
- Reads are allowed at the local data, while updates of local data are tentative in the sense that they need to be further validated before commitment.
- For integrating operations at the mobile hosts with transactions at other sites, in the case of replica-level consistency, copies of each item are reconciled following some conflict resolution protocol. At the transaction-level, local transactions are validated against some application or system level criterion. If the criterion is met, the transaction is committed. Otherwise, the execution of the transaction is either aborted, reconciled or compensated.

Representative approaches along these lines include *isolation-only transactions* in Coda, *mobile open-nested transactions* [6], *two-tier replications* [8], *two-layer transactions* [10] and *Bayou* [9].

When local copies are read-only, a central issue is the design of efficient protocols for disseminating server updates to mobile clients. A server is called *stateful*, if it maintains information about its clients and the content of their caches and *stateless* otherwise. A server may use broadcasting to efficiently propagate update reports to all of its clients. Such update reports vary on the type of information they convey to the clients, for instance, they may include just the identifiers of the updated items or the updated values themselves. They may also provide information for individual items or aggregate information for sets of items. Update propagation may be either synchronous or asynchronous. In *asynchronous* methods, update reports are broadcast as the updates are performed. In *synchronous* methods, the server broadcasts an update report periodically. A client must listen for the report first to decide whether its cache is valid or not. This adds some latency to query processing, however, each client needs only tune in periodically to read the report. The efficiency of update dissemination protocols for clients with different connectivity behavior, such as for workaholics (i.e., often connected clients) and sleepers (i.e., often disconnected clients), is evaluated in [4].

Finally, in the case of broadcast push-data delivery, clients may read items from different broadcast programs. The *currency* of the set of data items read by each client can be characterized based on the current values of the corresponding items at the server and on the temporal discrepancy among the values of the items in the set [13]. A more strict notion of correctness may be achieved through transaction-level correctness by requiring the client read-only transactions to be serializable with the server transactions. Methods for doing so include: (i) an *invalidation* method [12], where the server broadcasts an invalidation report that includes the data items that have been updated since the broadcast of the previous report, and transactions that have read obsolete items are aborted, (ii) *serialization graph testing* (SGT) [12], where the server broadcasts control information related to conflicting operations, and (iii) *multiversion broadcast* [11], where multiple versions of each item are broadcast, so that client transactions always read a consistent database snapshot.

Key Applications

Data broadcasting, caching and replication techniques are part of the core of any application that requires data sharing and synchronization among mobile devices and data servers. Such applications include vehicle dispatching, object tracking, points of sale (e.g., ambulance and taxi services, Fedex/UPS), and collaborative applications (e.g., homecare, video gaming). They are also part of embedded or light versions of database management systems that extend enterprise applications to mobile devices. These include among others Sybase Inc.'s SQL Anywhere, IBM's DB2 Everyplace, Microsoft SQL Server Compact, Oracle9i Lite and SQL Anywhere Technologies' Ultralite.

Cross-references

► Concurrency Control
► Hash-Based Indexing

► MANET Databases
► Mobile Databases
► Replicated Databases
► Transaction Management

Recommended Reading

1. Acharya S., Alonso R., Franklin M.J., and Zdonik S.B. Broadcast disks: data management for asymmetric communications environments. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 199–210.
2. Acharya S., Franklin M.J., and Zdonik S.B. Balancing push and pull for data broadcast. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 183–194.
3. Aksoy D. and Franklin M.J. RxW: a scheduling approach for large scale on-demand broadcast. IEEE/ACM Trans. Netw., 7(6):846–860, 1999.
4. Barbará D. and Imielinski T. Sleepers and workaholics: caching strategies in mobile environments. VLDB J., 4(4):567–602, 1995.
5. Beaver J., Chrysanthis P.K., and Pruhs K. To broadcast push or not and what? In Proc. 7th Int. Conf. on Mobile Data Management, 2006, pp. 40–45.
6. Chrysanthis P.K. Transaction processing in a mobile computing environment. In Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems, 1993, pp. 77–82.
7. Dykeman H.D., Ammar M.H., and Wong J.W. Scheduling algorithms for videotex systems under broadcast delivery. In Proc. IEEE Int. Conf. on Communications, 1986, pp. 1847–1851.
8. Gray J., Helland P., Neil P.O., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
9. Petersen K., Spreitzer M., Terry D.B. Theimer M., and Demers A.J. Flexible update propagation for weakly consistent replication. In Proc. 16th ACM Symp. on Operating System Principles, 1997, pp. 288–301.
10. Pitoura E. and Bhargava B. Data consistency in intermittently connected distributed systems. IEEE Trans. Knowl. Data Eng., 11(6):896–915, 1999.
11. Pitoura E. and Chrysanthis P.K. Exploiting versions for handling updates in broadcast disks. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 114–125.
12. Pitoura E. and Chrysanthis P.K. Scalable processing of read-only transactions in broadcast push. In Proc. 19th Int. Conf. on Distributed Computing Systems, 1999, pp. 432–439.
13. Pitoura E., Chrysanthis P.K., and Ramamritham K. Characterizing the temporal and semantic coherency of broadcast-based data dissemination. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 410–424.
14. Pitoura E. and Samaras G. Data Management for Mobile Computing. Kluwer, Boston, USA, 1998.
15. Sharaf MA. and Chrysanthis P.K. On-demand data broadcasting for mobile decision making. MONET, 9(6):703–714, 2004.
16. Stathatos K., Roussopoulos N., and Baras J.S. Adaptive data broadcast in hybrid networks. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 326–335.
17. Su C.J, Tassiulas L., and Tsotras V.J. Broadcast scheduling for information distribution. Wireless Netw., 5(2):137–147, 1999.
18. T I., Viswanathan S., and Badrinath B.R. Data on air: organization and access. IEEE Trans. Knowl. Data Eng., 9(3):353–372, 1997.

Data Cache

► Processor Cache

Data Cleaning

Venkatesh Ganti
Microsoft Research, Redmond, WA, USA

Definition

Owing to differences in conventions between the external sources and the target data warehouse as well as due to a variety of errors, data from external sources may not conform to the standards and requirements at the data warehouse. Therefore, data has to be transformed and cleaned before it is loaded into a data warehouse so that downstream data analysis is reliable and accurate. *Data Cleaning* is the process of standardizing data representation and eliminating errors in data. The data cleaning process often involves one or more tasks each of which is important on its own. Each of these tasks addresses a part of the overall data cleaning problem. In addition to tasks which focus on transforming and modifying data, the problem of diagnosing quality of data in a database is important. This diagnosis process, often called data profiling, can usually identify data quality issues and whether or not the data cleaning process is meeting its goals.

Historical Background

Many business intelligence applications are enabled by data warehouses. If the quality of data in a data warehouse is poor, then conclusions drawn from business data analysis could also be incorrect. Therefore, much emphasis is placed on cleaning and maintaining high quality of data in data warehouses. Consequently, the area of data cleaning received considerable attention in the database community. An early survey of automatic data cleaning techniques can be found in [14]. Several companies also started developing domain-specific data cleaning solutions (especially for the customer address domain). Over time, several generic data cleaning techniques have been also been

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:08

developed (e.g., [10,5,15,8,9,1]) and, domain neutral commercial data cleaning software also started making its appearance (e.g., [13,11]).

Foundations

1 Main Data Cleaning Tasks

In this section, the goals of several data cleaning tasks are introduced informally. The set of tasks mentioned below consists of those addressing commonly encountered problems in data cleaning and may not be a comprehensive list. However, note that most of the tasks mentioned below are important whether one wants to clean data at the time of loading a data warehouse or at the time of querying a database [6].

1.1 Column Segmentation Consider a scenario where a customer relation is being imported to add new records to a target customer relation. Suppose the address information in the target relation is split into its constituent attributes [street address, city, state, and zip code] while in the source relation they are all concatenated into one attribute. Before the records from the source relation could be inserted in the target relation, it is essential to segment each address value in the source relation to identity the attribute values at the target. The goal of a *column segmentation* task is to split an incoming string into segments, each of which may be inserted as attribute values at the target. A significant challenge to be addressed by this task is to efficiently match sub-strings of an input string with patterns such as regular expressions and with members of potentially large reference tables in order to identify values for target attributes. Note that, in general data integration may involve more complex schema transformations than achieved by the column segmentation task.

1.2 Record Matching Consider a scenario where a new batch of customer records is being imported into a sales database. In this scenario, it is important to verify whether or not the same customer is represented in both the existing as well as the incoming sets and only retain one record in the final result. Due to representational differences and errors, records in both batches could be different and may not match exactly on their key attributes (e.g., name and address or the CustomerId). The goal of a *record matching* task is to identify record pairs, one in each of two input relations, which correspond to the same real world entity. Challenges to be addressed in this task include (i) identification of criteria under which two records represent the same real world entity, and (ii) efficient computation strategies to determine such pairs over large input relations.

1.3 Deduplication Consider a scenario where one obtains a set of customer records or product records from an external (perhaps low quality) data source. This set may contain multiple records representing the same real world (customer or product) entity. It is important to "merge" records representing the same entity into one record in the final result. The goal of a *deduplication* task is to partition a relation into disjoint sets of records such that each group consists of records which represent the same real world entity. Deduplication may (internally) rely on a record matching task but the additional responsibility of further grouping records based on pairwise matches introduces new challenges. The output of record matching may not be transitively closed. For instance, a record matching task comparing record pairs in a relation may output pairs (r_1, r_2) and (r_2, r_3) as matches, but not (r_1, r_3). Then, the problem of deriving a partitioning that respects the pairwise information returned by record matching is solved by deduplication.

1.4 Data Standardization Consider a scenario where a relation contains several customer records with missing zip code or state values, or improperly formatted street address strings. In such cases, it is important to fill in missing values and adjust, where possible, the format of the address strings so as to return correct results for analysis queries. For instance, if a business analyst wants to understand the number of customers for a specific product by zip code, it is important for all customer records to have correct zip code values. The task of improving the quality of information within a database is often called *data standardization.* Similar tasks also occur in various other domains such as product catalog databases. The data standardization task may also improve the effectiveness of record matching and deduplication tasks.

1.5 Data Profiling The process of cleansing data is often an iterative and continuous process. It is important to "evaluate" quality of data in a database before one initiates data cleansing process, and subsequently assesses its success. The process of evaluating data quality is called *data profiling*, and typically involves gathering several aggregate data statistics which constitute the *data profile*, and ensuring that the values match up with expectations. For example, one may expect the customer name and address columns together to uniquely determine each customer record in a Customer relation. In such a case, the number of unique [name, address] values must be close to that of the total number of records in the Customer relation. Note that a large subset of elements of a data profile may each be obtained using one or more SQL queries. However, because all the elements of a data profile are computed together, there is an opportunity for a more efficient computation strategy. Further, the data profile of a database may also consist of elements which may not easily be computed using SQL queries.

Besides the set of data cleaning tasks mentioned above, other data cleaning tasks such as filling in missing values, identifying incorrect attribute values and then automatically correcting them based on known attribute value distributions are also important for applications such as cleaning census data.

2 Data Cleaning Platforms

The above requirements for a variety of data cleaning tasks have led to the development of utilities that support data transformation and cleaning. Such software falls into two broad categories:

Vertical Solutions: The first category consists of verticals such as Trillium [15] that provide data cleaning functionality for specific domains, e.g., addresses. Since they understand the domain where the vertical is being applied, they can fine tune their software for the given domain. However, by design, these are not generic and hence cannot be applied to other domains.

Horizontal Platforms: The other approach of building data cleaning software is to define and implement basic data cleaning operators. The broad goal here is to define a set of domain neutral operators, which can significantly reduce the load of developing common data cleaning tasks such as those outlined above. An example of such a basic operator is the set similarity join which may be used for identifying pairs of highly similar records across two relations (e.g., [16,4]). The advantage is that custom solutions for a variety of data cleaning tasks may now be developed for specialized domains by composing one or more of these basic operators along with other (standard or custom) operators. These basic operators do the heavy lifting and thus make the job of developing data cleaning programs easier. Examples of such platforms include AJAX [7,8] and Data Debugger [3].

The above mentioned data cleaning operators may then be included in database platforms so as to enable programmers to easily develop custom data cleaning solutions. For instance, *ETL (extract-transform-load)* tools such as Microsoft SQL Server Integration Services (SSIS) [13] and IBM Websphere Information Integration [11] that can be characterized as "horizontal" *platforms*. These platforms are applicable across a variety of domains, and provide a set of composable operators (e.g., relational operators) enabling users to build programs involving these operators. Further, these platforms also allow users to build their own custom operators which may then be used in these programs. Hence, such ETL platforms provide a great vehicle to include core data cleaning operators.

Key Applications

Data cleaning technology is critical for several information technology initiatives (such as data warehousing and business intelligence) which consolidate, organize, and analyze structured data. Accurate data cleaning processes are typically employed during data warehouse construction and maintenance to ensure that subsequent business intelligence applications yield accurate results.

A significant amount of recent work has been focusing on extracting structured information from documents to enable structured querying and analysis over document collections [2,12]. Invariably, the extracted data is unclean and many data cleaning tasks discussed above are applicable in this context as well.

Cross-references

► Column segmentation
► Constraint-driven database repair
► Data deduplication
► Data profiling
► Record matching
► Similarity functions for data cleaning

Recommended Reading

1. Borkar V. Deshmukh V. and Sarawagi S. Automatic segmentation of text into structured records. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.
2. Cafarella M.J. Re C. Suciu D. Etzioni O. and Banko M. Structured querying of the web 2007.
3. Chaudhuri S. Ganti V. and Kaushik. R. Data debugger: an operator-centric approach for data quality solutions. IEEE Data Eng. Bull., 2006.

4. Chaudhuri S. Ganti V. and Kaushik. R. A primitive operator for similarity joins in data cleaning. In Proc. 22nd Int. Conf. on Data Engineering 2006.
5. Cohen. W. Integration of heterogeneous databases without common domains using queries based on textual similarity. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998.
6. Fuxman A. Fazli E. and Miller. R.J. Conquer: efficient management of inconsistent databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
7. Galhardas H. Florescu D. Shasha D. and Simon. E. An extensible framework for data cleaning. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
8. Galhardas H. Florescu D. Shasha D. Simon E. and Saita. C. Declarative data cleaning: language, model, and algorithms. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
9. Gravano L. Ipeirotis P.G. Jagadish H.V. Koudas N. Muthukrishnan S. and Srivastava. D. Approximate string joins in a database (almost) for free. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
10. Hernandez. M. and Stolfo. S. The merge/purge problem for large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995.
11. IBM Websphere information integration. http://ibm.ascential.com.
12. Ipeirotis P.G. Agichtein E. Jain P. and Gravano. L. To search or to crawl? towards a query optimizer for text-centric tasks. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.
13. Microsoft SQL Server 2005 integration services.
14. Rahm E. and Do. H.H. Data cleaning: problems and current approaches. IEEE Data Engineering Bulletin, 2000.
15. Raman V. and Hellerstein. J. An interactive framework for data cleaning. Technical report, Univeristy of California, Berkeley, 2000.
16. Sarawagi S. and Kirpal. A. Efficient set joins on similarity predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
17. Trillium Software. www.trilliumsoft.com/trilliumsoft.nsf.

Data Collection

► Data Acquisition and Dissemination in Sensor Networks

Data Compression in Sensor Networks

Amol Deshpande
University of Maryland, College Park, MD, USA

Synonyms

Distributed source coding; Correlated data collection; Data suppression

Definition

Data compression issues arise in a sensor network when designing protocols for efficiently collecting all data observed by the sensor nodes at an Internet-connected base station. More formally, let X_i denote an attribute being observed by a node in the sensor network – X_i may be an environmental property being sensed by the node (e.g., *temperature*), or it may be the result of an operation on the sensed values (e.g., in an anomaly-detection application, the sensor node may continuously evaluate a filter such as "*temperature* > 100" on the observed values). The goal is to design an energy-efficient protocol to periodically collect the observed values of all such attributes (denoted $X_1, \ldots, X_n$) at the base station, at a frequency specified by the user. In many cases, a bounded-error approximation might be acceptable, ie., the reported values may only be required to be within $\pm \in$ of the observed values, for a given $\in$. The typical optimization metric is the total energy expended during the data collection process, commonly approximated by the total communication cost. However, metrics such as minimizing the maximum energy consumption across all nodes or maximizing the lifetime of the sensor network may also be appropriate in some settings.

Key Points

The key issue in designing data collection protocols is modeling and exploiting the strong spatio-temporal correlations present in most sensor networks. Let X_i^t be a random variable that denotes the value of X_i at time t (assuming time is discrete), and let $H(X_i^t)$ denote the *information entropy* of X_i^t. In most sensor network deployments, especially in environmental monitoring applications, the data generated by the sensor nodes is typically highly correlated both in time and in space — in other words, $H(X_i^{t+1}|X_i^t) \ll H(X_i^{t+1})$, and $H(X_1^t, \ldots, X_n^t) \ll H(X_1^t) + \ldots H(X_n^t)$. These correlations can usually be captured quite easily by constructing predictive models using either prior domain knowledge or historical data traces. However, because of the distributed nature of data generation in sensor networks, and the resource-constrained nature of sensor nodes, traditional data compression techniques cannot be easily adapted to exploit such correlations.

The distributed nature of data generation has been well-studied in the literature under the name of *Distributed Source Coding*, whose foundations were

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:08

laid almost 35 years ago by Slepian and Wolf [6]. Their seminal work proves that it is theoretically possible to encode the correlated information generated by distributed data sources at the rate of their joint entropy even if *the data sources do not communicate with each other*. However this result is non-constructive, and constructive techniques are known only for a few specific distributions [4]. More importantly, these techniques require precise and perfect knowledge of the correlations. This may not be acceptable in practical sensor networks, where deviations from the modeled correlations must be captured accurately. Pattem et al. [3] and Chu et al. [2], among others, propose practical data collection protocols that exploit the spatio-temporal correlations while guaranteeing correctness. However, these protocols may exploit only some of the correlations, and further require the sensor nodes to communicate with each other (thus increasing the overall cost).

In many cases, it may not be feasible to construct a predictive model over the sensor network attributes, as required by the above approach, because of mobility, high failure rates or inherently unpredictable nature of the monitored phenomena. Suppression-based protocols, that monitor local constraints and report to the base station only when the constraints are violated, may be used instead in such scenarios [5].

Sensor networks, especially wireless sensor networks, exhibit other significant peculiarities that make the data collection problem challenging. First, sensor nodes are typically computationally constrained and have limited memories. As a result, it may not be feasible to run sophisticated data compression algorithms on them.

Second, the communication in wireless sensor networks is typically done in a broadcast manner – when a node transmits a message, all nodes within the radio range can receive the message. This enables many optimizations that would not be possible in a one-to-one communication model.

Third, sensor networks typically exhibit an extreme asymmetry in the computation and communication capabilities of the sensor nodes compared to the base station. This motivates the design of pull-based data collection techniques where the base station takes an active role in the process. Adler [1] proposes such a technique for a one-hop sensor network. The proposed algorithm achieves the information-theoretical lower bound on the number of bits *sent* by the sensor nodes, while at the same time offloading most of the compute-intensive work to the base station. However, the number of bits *received* by the sensor nodes may be very high.

Finally, sensor networks typically exhibit high message loss and sensor failure rates. Designing robust and fault-tolerant protocols with provable guarantees is a challenge in such an environment.

Cross-references

▶ Continuous Queries in Sensor Networks
▶ Data Aggregation in Sensor Networks
▶ Data Fusion in Sensor Networks
▶ In-Network Query Processing
▶ Model-based Querying in Sensor Networks

Recommended Reading

1. Adler M. Collecting correlated information from a sensor network. In: Proc. 16th Annual ACM -SIAM Symp. on Discrete Algorithms. 2005.
2. Chu D., Deshpande A., Hellerstein J., and Hong W. Approximate data collection in sensor networks using probabilistic models. In Proc. 22nd Int. Conf. on Data Engineering. 2006.
3. Pattem S., Krishnamachari B., and Govindan R. The impact of spatial correlation on routing with compression in wireless sensor networks. In Proc. 3rd Int. Symp. Inf. Proc. in Sensor Networks. 2004.
4. Pradhan S. and Ramchandran K. Distributed source coding using syndromes (DISCUS): Design and construction. IEEE Trans. Inform. Theory, 49(3), 2003.
5. Silberstein A., Puggioni G., Gelfand A., Munagala K., and Yang J. Making sense of suppressions and failures in sensor data: a Bayesian approach. In: Proc. 33rd Int. Conf. on Very Large Data Bases. 2007.
6. Slepian D. and Wolf J. Noiseless coding of correlated information sources. IEEE Trans. Inform. Theory, 19(4), 1973.

Data Confidentiality

▶ Security Services

Data Conflicts

Hong-Hai Do
SAP AG, Dresden, Germany

Synonyms

Data problems; Data quality problems; Data anomalies; Data inconsistencies; Data errors

Definition

Data conflicts are deviations between data intended to capture the same state of a real-world entity. Data with conflicts are often called "dirty" data and can mislead analysis performed on it. In case of data conflicts, data cleaning is needed in order to improve the data quality and to avoid wrong analysis results. With an understanding of different kinds of data conflicts and their characteristics, corresponding techniques for data cleaning can be developed.

Historical Background

Statisticians were probably the first who had to face data conflicts on a large scale. Early applications, which needed intensive resolution of data conflicts, were statistical surveys in the areas of governmental administration, public health, and scientific experiments. In 1946, Halbert L. Dunn already observed the problem of duplicates in data records of a person's life captured at different places [3]. He introduced the term Record Linkage to denote the process to resolve the problem, i.e., to obtain and link all unique data records to a consistent view on the person. In 1969, Fellegi and Sunter provided a formal mathematical model for the problem and thereby laid down the theoretical foundation for numerous record linkage applications developed later on [5].

Soon it became clear that record linkage is only the tip of the iceberg of the various problems, such as wrong, missing, inaccurate, and contradicting data, which makes it difficult for humans and applications to obtain a consistent view on real-world entities. In the late 1980s, computer scientists began to systematically investigate all problems related to data quality, increasingly from a practical perspective in the context of business applications. This was essentially pushed by the need to integrate data from heterogeneous sources for business decision making and by the emergence of enterprise data warehouses at the beginning of the 1990s. To date, various research approaches and commercial tools have been developed to deal with the different kinds of data conflicts and to improve data quality [1,2,4,7].

Foundations

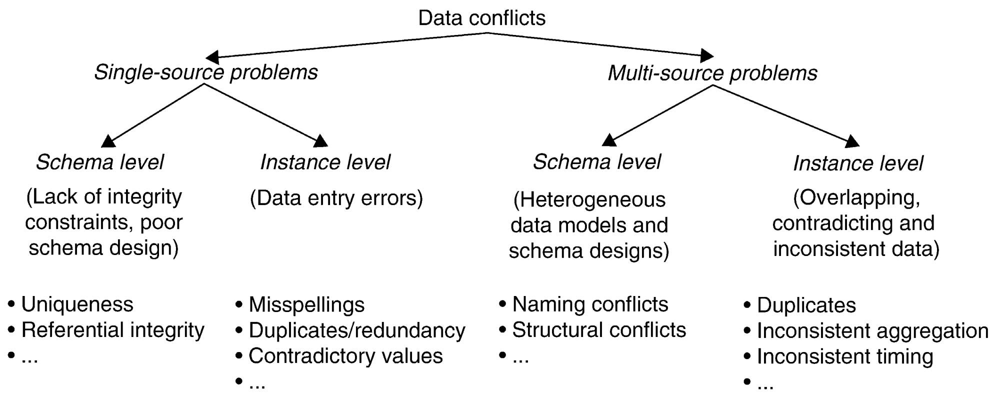
Classification of Data Conflicts

As shown in Fig. 1, data conflicts can be classified according to the following criteria:

- *Single-source* versus *multi-source*: Data conflicts can occur among data within a single source or between different sources.
- *Schema-level* versus *instance-level*: Schema-level conflicts are caused by the design of the data schemas. Instance-level conflicts, on the other hand, refer to problems and inconsistencies in the actual data contents, which are not visible at the schema level.

Figure 1 also shows typical data conflicts for the various cases. While not shown, the single-source conflicts occur (with increased likelihood) in the multi-source case, too, besides specific multi-source conflicts.

Single-Source Data Conflicts The data quality of a source largely depends on the degree to which it is governed by schema and integrity constraints controlling permissible data values. For sources without a schema,



Data Conflicts. Figure 1. Examples of multi-source problems at schema and instance level.

such as files, there are few restrictions on what data can be entered and stored, giving rise to a high probability of errors and inconsistencies. Database systems, on the other hand, enforce restrictions of a specific data model (e.g., the relational approach requires simple attribute values, referential integrity, etc.) as well as application-specific integrity constraints. Schema-related data quality problems thus occur because of the lack of appropriate model-specific or application-specific integrity constraints, e.g., due to data model limitations or poor schema design, or because only a few integrity constraints were defined to limit the overhead for integrity control. Instance-specific problems are related to errors and inconsistencies that cannot be prevented at the schema level (e.g., misspellings).

Both schema- and instance-level conflicts can be further differentiated according to the different problem scopes *attribute*, *record*, *record type*, and *source*. In particular, a data conflict can occur within an individual attribute value (*attribute*), between attributes of a record (*record*), between records of a record type (*record type*), and between records of different record types (*source*). Examples of data conflicts in each problem scope are shown and explained in Tables 1 and 2 for the schema and instance level, respectively. Note that uniqueness constraints specified at the schema level do not prevent duplicated instances, e.g., if information on the same real world entity is entered twice with different attribute values (see examples in Table 2).

Multi-Source Data Conflicts The problems present in single sources are aggravated when multiple sources need to be integrated. Each source may contain dirty data and the data in the sources may be represented differently, may overlap, or contradict. This is because the sources are typically developed, deployed and maintained independently to serve specific needs. This results in a large degree of heterogeneity with respect to database management systems, data models, schema designs, and the actual data.

At the schema level, data model and schema design differences are to be addressed by the steps of schema translation and schema integration, respectively. The main problems with respect to schema design are naming and structural conflicts. Naming conflicts arise when the same name is used for different objects (homonyms) or different names are used for the same object (synonyms). Structural conflicts occur in many variations and refer to different representations of the same object in different sources, e.g., attribute versus table representation, different component structure, different data types, different integrity constraints, etc.

In addition to schema-level conflicts, many conflicts appear only at the instance level. All problems from the single-source case can occur with different representations in different sources (e.g., duplicated records, contradicting records). Furthermore, even when there are the same attribute names and data types, there may be different value representations (e.g., M/F vs. Male/Female for marital status) or different interpretation of the values (e.g., measurement units Dollar vs. Euro) across sources. Moreover, information in the sources may be provided at different aggregation levels (e.g., sales per product vs. sales per product group) or refer to different points in time (e.g., current sales as of yesterday for Source 1 vs. as of last week for Source 2).

A main problem for cleaning data from multiple sources is to identify overlapping data, in particular matching records referring to the same real-world entity (e.g., a particular customer). This problem is

Data Conflicts. Table 1. Examples for single-source problems at schema level (violated integrity constraints)

Scope	Type of conflict	Dirty data	Reasons/Remarks
Attribute	Illegal values	birthdate = 13/30/1970	Values outside of domain range
Record	Violated attribute dependencies	city = "Redmond," zip = 77777	City and zip code should correspond
Record type	Uniqueness violation	emp_1 = (name = "John Smith," SSN = "123456"), emp_2 = (name = "Peter Miller," SSN = "123456")	Uniqueness for SSN (social security number) violated
Source	Referential integrity violation	emp = (name = "John Smith," deptno = 127)	Referenced department (127) not defined

Data Conflicts. Table 2. Examples for single-source problems at instance level

Scope	Type of conflict	Dirty data	Reasons/Remarks
Attribute	Missing values	phone = 9999–999999	Unavailable values during data entry (dummy values or null)
	Misspellings	city = "London"	Usually typos, phonetic errors
	Cryptic values, Abbreviations	experience = "B"; occupation = "DB Prog."	Use of code lists
	Embedded values	name = "J. Smith 02/12/70 New York"	Multiple values entered in one attribute (e.g. in a free-form field)
	Misfielded values	city = "Germany"	City field contains value of country field
Record	Violated attribute dependencies	city = "Redmond," zip = 77777	City and zip code should correspond
Record type	Word transpositions	$name_1$ = "J. Smith," $name_2$ = "Miller P."	Usually in a free-form field
	Duplicated records	emp_1 = (name = "John Smith," . . .); emp_2 = (name = "J. Smith," . . .)	Same employee represented twice due to some data entry errors
	Contradicting records	emp_1 = (name = "John Smith," bdate = 02/12/70); emp_2 = (name = "John Smith," bdate = 12/12/70)	The same real world entity is described by different values
Source	Wrong references	emp = (name = "John Smith," deptno = 17)	Referenced department (17) is defined but wrong

also referred to as the record linkage problem, the object identity problem, or the deduplication problem. Frequently, the information is only partially redundant and the sources may complement each other by providing additional information about an entity. Thus duplicate information should be purged and complementing information should be consolidated and merged in order to achieve a consistent view of real-world entities.

The two relational sources, Source 1 and Source 2, in the example of Fig. 2 exhibit several kinds of conflicts. At the schema level, there are name conflicts (synonyms *Customer* vs. *Client*, *CID* vs. *Cno*, *Sex* vs. *Gender*) and structural conflicts (different structures for names *Name* vs. {*LastName*, *FirstName*}, and for addresses {*Street*, *City*, *Zip*} vs. *Address*). At the instance level, one can see that there are different gender representations ("0"/"1" vs. "*F*"/"*M*") and presumably a duplicate record (*Kristen Smith*). The latter observation also reveals that while *CID* and *Cno* are both source-specific identifiers, their contents are not comparable between the sources; different numbers ("11," "49") refer to the same person while different persons have the same number ("24").

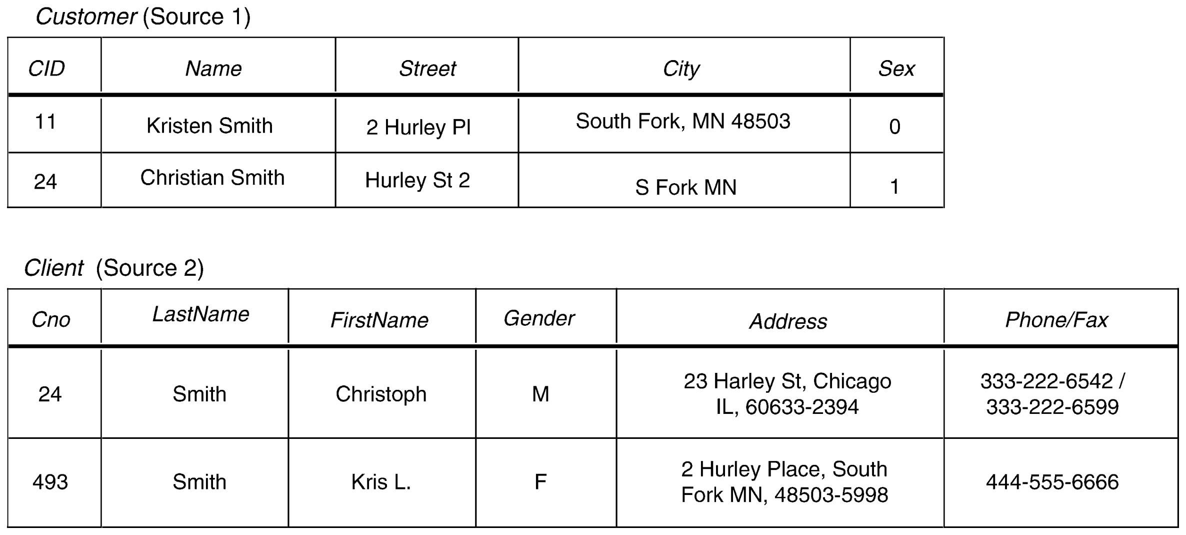
Dealing with Data Conflicts

Data conflicts can be dealt with in a preventive and/or corrective way. As resolving existing data conflicts is generally an expensive task, preventing dirty data to be entered is promising to ensure a high data quality. This requires appropriate design of the database schema with corresponding integrity constraints and strict enforcement of the constraints in the databases and data entry applications. For most applications, however, a corrective strategy, i.e. data cleaning (a.k.a. cleansing or scrubbing), is needed in order to remove conflicts from given data and make it suitable for analysis. Typically, this process involves thorough analysis of the data to detect conflicts and transformation of the data to resolve the identified conflicts.

Key Applications

Data Warehousing

Data warehousing aims at a consolidated and consistent view of enterprise data for business decision making. Transactional and non-transactional data from a variety of sources is aggregated and structured typically in a multidimensional schema to effectively support dynamic



Customer (Source 1)

CID	*Name*	*Street*	*City*	*Sex*
11	Kristen Smith	2 Hurley Pl	South Fork, MN 48503	0
24	Christian Smith	Hurley St 2	S Fork MN	1

Client (Source 2)

Cno	*LastName*	*FirstName*	*Gender*	*Address*	*Phone/Fax*
24	Smith	Christoph	M	23 Harley St, Chicago IL, 60633-2394	333-222-6542 / 333-222-6599
493	Smith	Kris L.	F	2 Hurley Place, South Fork MN, 48503-5998	444-555-6666

Data Corruption. Figure 2. Classification of data conflicts in data sources.

querying and reporting, such as Online Analytical Processing (OLAP). As multiple data sources are considered, the probability that some of the sources contain conflicting data is high. Furthermore, the correctness of the integrated data is vital to avoid wrong conclusions. Due to the wide range of possible data inconsistencies and the sheer data volume, data cleaning is one of the biggest problems for data warehousing. Data conflicts need to be detected and resolved during the so-called ETL process (Extraction, Transformation, and Load), when source data is integrated from corresponding sources and stored into the data warehouse.

Data Mining

Data mining, or knowledge discovery, is the analysis of large data sets to extract new and useful information. Developed algorithms utilize a number of techniques, such as data visualization (charts, graphs), statistics (summarization, regression, clustering), and artificial intelligence techniques (classification, machine learning, and neural networks). As relevant data typically needs to be integrated from different sources, data warehousing represents a promising way to build a suitable data basis for data mining. However, due to performance reasons on large data sets, specialized data mining algorithms often operate directly on structured files. In either case, resolving data conflicts to obtain correct data is crucial for the success of data mining. On the other hand, the powerful data mining algorithms can also be utilized to analyze dirty data and discover data conflicts.

Cross-references

► Data Cleaning
► Data Quality
► Duplicate Detection

Recommended Reading

1. Barateiro J. and Galhardas H. A survey of data quality tools. Datenbank-Spektrum, 14:15–21, 2005.
2. Batini C. and Scannapieco M. Data Quality – Concepts, Methodologies and Techniques. Springer, Berlin, 2006.
3. Dunn H.L. Record linkage. Am. J. Public Health, 36(12):1412–1416, 1946.
4. Elmagarmid A.K., Ipeirotis P.G., and Verykios V.S. Duplicate record detection – a survey. IEEE Trans. Knowl. Data Eng., 19(1):1–16, 2007.
5. Fellegi I.P. and Sunter A.B. A theory for record linkage. J. Am. Stat. Assoc., 64(328):1183–1210, 1969.
6. Kim W., Choi B.-J., Kim S.-K., and Lee D. A taxonomy of dirty data. Data Mining Knowl. Discov., 7(1):81–99, 2003.
7. Rahm E. and Do H.-H. Data cleaning – problems and current approaches. IEEE Techn. Bull. Data Eng., 23(4):3–13, 2000.

Data Copy

► Data broadcasting, caching and replication

Data Corruption

► Storage Security

Data Deduplication

► Record Matching

Data Dependency

► Database Dependencies

Data Dictionary

James Caverlee
Texas A&M University, College Station, TX, USA

Synonyms

System catalog; Metadata repository

Definition

A data dictionary catalogs the definitions of data elements, data types, data flows and other conventions that are used in an information system. Data dictionaries have been widely adopted by both (i) the database community, where a dictionary typically describes database entities, schemas, permissions, etc.; and (ii) the software development community, where a dictionary typically describes flows of information through the system. In essence, a data dictionary is a virtual database of metadata about an information system itself. A data dictionary may also be referred to as a "system catalog."

Key Points

Understanding and managing an information system – both from a design and from an implementation point-of-view – requires some documentation of the schema, capabilities, constraints, and other descriptive features of the system. This documentation is typically embodied by a data dictionary – that is, a repository of information for an information system that describes the entities represented as data, including their attributes and the relationships between them [3].

The importance of a systematic way to store and manage the metadata associated with an information system has been well known since the earliest days of database and large-scale systems development. By the time the relational model was garnering attention in the 1970s, metadata management via a system catalog was a standard feature in database management systems (DBMSs) like System R [1] and INGRES [5]. Around the same time, the structured analysis approach for large-scale systems development also advocated for the use of a data dictionary [2].

The phrase *data dictionary* has two closely related meanings: (i) as documentation primarily for consumption by human users, administrators, and designers; and (ii) as a mini-database managed by a DBMS and tightly coupled with the software components of the DBMS.

In the first meaning, a data dictionary is a document (or collection of documents) that provides a conceptual view of the structure of an information system for those developing, using, and maintaining the system. In this first meaning, a data dictionary serves to document the system design process, to identify the important characteristics of the system (e.g., schemas, constraints, data flows), and to provide the designers, users, and administrators of the system a central metadata repository [6]. A data dictionary can provide the names of tables and fields, types for data attributes, encoding information, and further details of an overall structure and usage. The owner of a database or database administrator (DBA) might provide it as a book or a document with additional descriptions and diagrams, or as generated documentation derived from a database. Database users and application developers then benefit from the data dictionary as an accepted reference, though this hardcopy version is not always provided nor required.

In the second meaning, a data dictionary is a mini-database tightly coupled and managed by an information system (typically a DBMS) for supporting query optimization, transaction processing, and other typical features of a DBMS. When used in this sense, a data dictionary is often referred to as a *catalog* or as a *system catalog*. As a software component of a database or a DBMS, a data dictionary makes up all the metadata and additional functions needed for a database manipulation language (DML) to select, insert, and generally operate on data. A database user will do this in conjunction with a high-level programming language or from a textual or graphical user interface (GUI). The data dictionary for a database or DBMS typically has these elements:

- Descriptions of tables and fields
- Permissions information, such as usernames and privileges
- How data is indexed

- Referential integrity constraints
- Definitions for database schemas
- Storage allocation parameters
- Usage statistics
- Stored procedures and database triggers

For an example, a developer unfamiliar with what tables are available within a database could query the virtual INFORMATION_SCHEMA database, which serves as the data dictionary for MySQL databases [4].

Besides this low-level version of a data dictionary, some software frameworks add another layer of abstraction to create a high-level data dictionary as well. This layer can reduce development time by providing features not supported at the lower level, such as alternative database scheme models. One example is Object-Relational Mapping (ORM), which seeks to map the data types created in the Object-Oriented Programming (OOP) paradigm to a relational database.

Cross-references

► Metadata
► Metadata Repository

Recommended Reading

1. Astrahan M. et al. (1979) System R: a relational data base management system. IEEE Comput. 12(5):42–48, 1979.
2. Demarco T. Structured Analysis and System Specification. Yourdon, 1978.
3. Elmasri R. and Navathe S. Fundamentals of database systems. Addison-Wesley, Reading, MA, 2000.
4. MySQL MySQL 5.0 Reference Manual, 2008.
5. Stonebraker M., Wong E., Kreps P., and Held G. The design and implementation of INGRES. ACM Trans. Database Syst., 1(3):189–222, 1976.
6. Yourdon E. Modern Structured Analysis. Yourdon 1989.

Data Dissemination

► Data broadcasting, caching and replication

Data Encryption

Ninghui Li
Purdue University, West Lafayette, IN, USA

Synonyms

Encryption; Cipher

Definition

Data encryption is the process of transforming data (referred to as plaintext) to make it unreadable except to those possessing some secret knowledge, usually referred to as a key. The result of the process is encrypted data (referred to as ciphertext). Data encryption aims at preserving confidentiality of messages. The reverse process of deriving the plaintext from the ciphertext (using the key) is known as decryption. A cipher is a pair of algorithms which perform encryption and decryption. The study of data encryption is part of cryptography. The study of how to break ciphers, i.e., to obtaining the meaning of encrypted information without access to the key, is called cryptanalysis.

Historical Background

Encryption has been used to protect communications since ancient times by militaries and governments to facilitate secret communication. The earliest known usages of cryptography include a tool called Scytale, which was used by the Greeks as early as the seventh century BC, and the Caesar cipher, which was used by Julius Caesar in the first century B.C.

The main classical cipher types are transposition ciphers, which rearrange the order of letters in a message, and substitution ciphers, which systematically replace letters or groups of letters with other letters or groups of letters. Ciphertexts produced by classical ciphers always reveal statistical information about the plaintext. Frequent analysis can be used to break classical ciphers.

Early in the twentieth century, several mechanical encryption/decryption devices were invented, including rotor machines – most famously the Enigma machine used by Germany in World War II. Mechanical encryption devices, and successful attacks on them, played a vital role in World War II.

Cryptography entered modern age in the 1970s, marked by two important events: the introduction of the U.S. Data Encryption Standard and the invention of public key cryptography. The development of digital computers made possible much more complex ciphers. At the same time, computers have also assisted cryptanalysis. Nonetheless, good modern ciphers have stayed ahead of cryptanalysis; it is usually the case that use of a quality cipher is very efficient (i.e., fast and requiring few resources), while breaking it requires an effort many orders of magnitude larger, making cryptanalysis so inefficient and impractical as to be effectively impossible.

Today, strong encryption is no longer limited to secretive government agencies. Encryption is now widely used by the financial industry to protect money transfers, by merchants to protect credit-card information in electronic commerce, by corporations to secure sensitive communications of proprietary information, and by citizens to protect their private data and communications.

Foundations

Data encryption can be either secret-key based or public-key based. In secret-key encryption (also known as symmetric encryption), a single key is used for both encryption and decryption. In public-key encryption (also known as asymmetric encryption), the encryption key (also called the public key) and the corresponding decryption key (also called the private key) are different. Modern symmetric encryption algorithms are often classified into stream ciphers and block ciphers.

Stream Ciphers

In a stream cipher, the key is used to generate a pseudo-random key stream, and the ciphertext is computed by using a simple operation (e.g., bit-by-bit XOR or byte-by-byte modular addition) to combine the plaintext bits and the key stream bits. Mathematically, a stream cipher is a function $f:\{0,1\}^\ell \rightarrow \{0,1\}^m$, where ℓ is the key size, and m determines the length of the longest message that can be encrypted under one key; m is typically much larger than ℓ. To encrypt a message x using a key k, one computes $c = f(k) \oplus x$, where $\oplus$ denote bit-by-bit XOR. To decrypt a ciphertext c using key k, one computes $f(k) \oplus c$.

Many stream ciphers implemented in hardware are constructed using linear feedback shift registers (LFSRs). The use of LFSRs on their own, however, is insufficient to provide good security. Additional variations and enhancements are needed to increase the security of LFSRs.

The most widely-used software stream cipher is RC4. It was designed by Ron Rivest of RSA Security in 1987. It is used in popular protocols such as Secure Sockets Layer (SSL) (to protect Internet traffic) and WEP (to secure wireless networks).

Stream ciphers typically execute at a higher speed than block ciphers and have lower hardware complexity. However, stream ciphers can be susceptible to serious security problems if used incorrectly; in particular, the same starting state (i.e., the same generated key stream) must never be used twice.

Block Ciphers

A block cipher operates on large blocks of bits, often 64 or 128 bits. Mathematically, a block cipher is a pair of functions $\mathcal{E}:\{0,1\}^\ell \times \{0,1\}^n \rightarrow \{0,1\}^n$ and $\mathcal{D}:\{0,1\}^\ell \times \{0,1\}^n \rightarrow \{0,1\}^n$, where ℓ is the key size and n is the block size. To encrypt a message x using key k, one calculates $\mathcal{E}(k,x)$, which is often written as $\mathcal{E}_k[x]$. To decrypt a ciphertext c using key k, one calculates $\mathcal{D}(k,c)$, often written as $\mathcal{D}_k[c]$. The pair $\mathcal{E}$ and $\mathcal{D}$ must satisfy

$$\forall k \in \{0,1\}^\ell \;\; \forall x \in \{0,1\}^n \;\; \mathcal{D}_k[\mathcal{E}_k[x]] = x.$$

The two most widely used block ciphers are the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES).

DES is a block cipher selected as Federal Information Processing Standard for the United States in 1976. It has subsequently enjoyed widespread use internationally. The block size of DES is 64 bits, and the key size 56 bits. The main weakness of DES is its short key size, which makes it vulnerable to bruteforce attacks that try all possible keys.

One way to overcome the short key size of DES is to use Triple DES (3DES), which encrypts a 64-bit block by running DES three times using three DES keys. More specifically, let $(\mathcal{E},\mathcal{D})$ be the pair of encryption and decryption functions for DES, then the encryption function for 3DES is

$$3\mathrm{DES}_{k_1,k_2,k_3}(x) = \mathcal{E}_{k_1}[\mathcal{D}_{k_2}[\mathcal{E}_{k_3}(x)]].$$

AES was announced as an U.S. Federal Information Processing Standard on November 26, 2001 after a 5-year selection process that is opened to the public. It became effective as a standard May 26, 2002. The algorithm is invented by Joan Daemen and Vincent Rijmen and is formerly known as Rijndael. AES uses a block size of 128 bits, and supports key sizes of 128 bits, 192 bits, and 256 bits.

Because messages to be encrypted may be of arbitrary length, and because encrypting the same plaintext under the same key always produces the same output, several modes of operation have been invented which allow block ciphers to provide confidentiality for messages of arbitrary length. For example, in the electronic codebook (ECB) mode, the message is divided into blocks and each block is encrypted separately. The disadvantage of this method is that identical plaintext blocks are encrypted into identical ciphertext blocks.

It is not recommended for use in cryptographic protocols. In the cipher-block chaining (CBC) mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block is dependent on all plaintext blocks processed up to that point. Also, to make each message unique, an initialization vector must be used in the first block and should be chosen randomly. More specifically, to encrypt a message x under key k, let x_1, $x_2,\ldots,x_m$ denote the message blocks, then the ciphertext is $c_0||c_1||\ldots||c_m$ where $||$ denote concatenation, c_0 = IV, the randomly chosen initial value, and $c_i = \mathcal{E}_k[x_i] \oplus c_{i-1}$ for $1 \leq i \leq m$. Other well-known modes include Cipher feedback (CFB), Output feedback (OFB), and Counter (CTR).

Public Key Encryption Algorithms

When using symmetric encryption for secure communication, the sender and the receiver must agree upon a key and the key must kept secret so that no other party knows the key. This means that the key must be distributed using a secure, but non-cryptographic, method; for example, a face-to-face meeting or a trusted courier. This is expensive and even impossible in some situations. Public key encryption was invented to solve the key distribution problem. When public key encryption is used, users can distribute public keys over insecure channels.

One of the most widely used public-key encryption algorithm is RSA. RSA was publicly described in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman at MIT; the letters RSA are the initials of their surnames. To generate a pair of RSA public/private keys, one does the following: choose two distinct large prime numbers p, q, calculate $N = pq$ and $\phi(N) = (p - 1)(q - 1)$, choose an integer e such that $1 < e < \phi(N)$, and e and $\phi(N)$ share no factors other than 1. The public key is (N, e), and the private key is (N, d), where $ed \equiv 1(\mathrm{mod}\ \phi(N))$. A message to be encrypted is encoded using a positive integer x where $x < N$. To encrypt x, compute $c = x^e \bmod N$. To decrypt a ciphertext c, compute $c^e \bmod N$. Practical RSA implementations typically embed some form of structured, randomized padding into the value x before encrypting it. Without such padding, the ciphertext leaks some information about the plaintext and is generally considered insecure for data encryption. It is generally presumed that RSA is secure if N is sufficiently large. The lengths of N are typically 1,024–4,096 bits long.

A central problem for public-key cryptography is proving that a public key is authentic and has not been tampered with or replaced by a malicious third party. The usual approach to this problem is to use a public-key infrastructure (PKI), in which one or more third parties, known as certificate authorities, certify ownership of key pairs.

Asymmetric encryption algorithms are much more computationally intensive than symmetric algorithms. In practice, public key cryptography is used in combination with secret-key methods for efficiency reasons. For encryption, the sender encrypts the message with a secret-key algorithm using a randomly generated key, and that random key is then encrypted with the recipient's public key.

Attack Models

Attack models or attack types for ciphers specify how much information a cryptanalyst has access to when cracking an encrypted message. Some common attack models are:

- *Ciphertext-only attack*: the attacker has access only to a set of ciphertexts.
- *Known-plaintext attack*: the attacker has samples of both the plaintext and its encrypted version (ciphertext).
- *Chosen-plaintext attack*: the attacker has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts.
- *Chosen-ciphertext attack*: the attacker has the capability to choose a number of ciphertexts and obtain the plaintexts.

Key Applications

Data encryption is provided by most database management systems. It is also used in many settings in which database is used, e.g., electronic commerce systems.

Cross-references

► Asymmetric Encryption
► Symmetric Encryption

Recommended Reading

1. Federal information processing standards publication 46-3: data encryption standard (DES), 1999.
2. Federal information processing standards publication 197: advanced encryption standard, Nov. 2001.
3. Diffie W. and Hellman M.E. New directions in cryptography. IEEE Trans. Inform. Theory, 22:644–654, 1976.

4. Kahn D. The codebreakers: the comprehensive history of secret communication from ancient times to the internet. 1996.
5. Menezes A.J., Oorschot P.C.V., and Vanstone S.A. Handbook of applied cryptography (revised reprint with updates). CRC, West Palm Beach, FL, USA, 1997.
6. Rivest R.L., Shamir A., and Adleman L.M. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21:120–126, 1978.
7. Singh S. The code book: the science of secrecy from ancient Egypt to quantum cryptography. Anchor, Garden City, NY, USA, 2000.

Data Errors

► Data Conflicts

Data Estimation in Sensor Networks

Le Gruenwald
University of Oklahoma, Norman, OK, USA

Synonyms

Data imputation

Definition

In wireless sensor networks, sensors typically transmit their data to servers at predefined time intervals. In this environment, data packets are very susceptible to losses, delays or corruption due to various reasons, such as power outage at the sensor's node, a higher bit error rate of the wireless radio transmissions compared to the wire communication alternative, an inefficient routing algorithm implemented in the network, or random occurrences of local interferences (e.g., mobile radio devices, microwaves or broken line-of-sight path). To process queries that need to access the missing data, if repeated requests are sent to sensors asking them to resend the missing information, this would incur power-costly communications as those sensors must be constantly in the listening mode. In addition, it is not guaranteed that those sensors would resend their missing data or would resend them in a timely manner. Alternatively, one might choose to estimate the missing data based on the underlying structure or patterns of the past reported data. Due to the low power-cost of computation, this approach represents an efficient way of answering queries that need to access the missing information. This entry discusses a number of existing data estimation approaches that one can use to estimate the value of a missing sensor reading.

Key Points

To estimate the value of a missing sensor reading, the quality of service in terms of high estimate accuracy and low estimation time needs to be observed. Data estimation algorithms can be divided into three major groups: (i) traditional statistical approaches; (ii) statistical-based sensor/stream data approaches, and (iii) association rule data mining based approaches. Many traditional statistical data approaches are not appropriate for wireless sensor networks as they require either the entire data set to be available or data to be missed at random, or do not consider relationships among sensors. Some statistical based sensor/stream data estimation algorithms include SPIRIT [3] and TinyDB [2]. SPIRIT is a pattern discovery system that uncovers key trends within data of multiple time series. These trends or correlations are summarized by a number of hidden variables. To estimate current missing values, SPIRIT applies an auto-regression forecasting model on the hidden variables. TinyDB is a sensor query processing system where missing values are estimated by taking the average of all the values reported by the other sensors in the current round. Two association rule based data estimation algorithms are WARM [1] and FARM [1]. WARM identifies sensors that are related to each other in a sliding window containing the latest w rounds using association rule mining. When the reading of one of those sensors is missing, it uses the readings of the other related sensors to estimate the missing reading. FARM is similar to WARM except that it does not use the concept of sliding window and it considers the freshness of data.

Cross-references

► Association Rule Mining
► Data Quality
► Sensor Networks
► Stream Data Management

Recommended Reading

1. Gruenwald L., Chok H., and Aboukhamis M. Using data mining to estimate missing sensor data. In Proceedings of the Seventh IEEE ICDM Workshop on Optimization-Based Data Mining Techniques with Applications. Omaha, NE, 2007, pp. 207–212.

2. Madden S., Franklin M., Hellerstein J., and Hong W. TinyDB: an acquisitional query processing system for sensor networks. Trans. Database Syst., 30(1):122–173, 2005.
3. Papadimitriou S., Sun J., and Faloutsos C. Pattern discovery in multiple time-series. In Proceedings of the 31st International Conference on Very Large Data Bases. Trondheim, Norway, 2005, pp. 697–708.

Data Exchange

Lucian Popa
IBM Almaden Research Center, San Jose, CA, USA

Synonyms

Data translation; Data migration; Data transformation

Definition

Data exchange is the problem of materializing an instance of a target schema, given an instance of a source schema and a specification of the relationship between the source schema and the target schema. More precisely, a *data exchange setting* is a quadruple of the form ($\mathbf{S}$, $\mathbf{T}$, Σ_{st}, Σ_t), where $\mathbf{S}$ is the source schema, $\mathbf{T}$ is the target schema, Σ_{st} is a schema mapping that expresses constraints between $\mathbf{S}$ and $\mathbf{T}$, and Σ_t is a set of constraints on $\mathbf{T}$. Such a setting gives rise to the following *data exchange problem*: given an instance I over the source schema $\mathbf{S}$, find an instance J over the target schema $\mathbf{T}$ such that I and J together satisfy the schema mapping Σ_{st}, and J satisfies the target constraints Σ_t. Such an instance J is called a *solution* for I in the data exchange setting. In general, many different solutions for an instance I may exist. The main focus of the data exchange research is to study the space of all possible solutions, to identify the "best" solutions to materialize in a practical application, and to develop algorithms for computing such a best solution.

Historical Background

The first systems supporting the restructuring and translation of data were built several decades ago. An early such system was EXPRESS [21], which performed data exchange between hierarchical schemas. The need for systems supporting data exchange has persisted over the years and has become more pronounced with the proliferation of data in various formats ranging from traditional relational database schemas to semi-structured/XML schemas and scientific formats. An example of a modern data exchange system is Clio [18,20], a schema mapping prototype developed at IBM Almaden Research Center and in collaboration with University of Toronto that influenced both theoretical and practical aspects of data exchange.

The data exchange problem is related to the data integration problem [16] in the sense that both problems are concerned with management of data stored in heterogeneous formats. The two problems, however, are different for the following reasons. In data exchange, the main focus is on actually *materializing* a target instance (i.e., a solution) that reflects the source data as accurately as possible. This presents a challenge, due to the inherent under-specification of the relationship between the source and the target, which means that in general there are many different ways to materialize such a target instance. In contrast, a target instance need not be materialized in data integration.There, the main focus is on answering queries posed over the target schema using views that express the relationship between the target and source schemas.

Fagin et al. [8] were the first to formalize the data exchange problem and to embark on an in-depth investigation of the foundational and algorithmic issues that surround it. Their framework focused on data exchange settings in which $\mathbf{S}$ and $\mathbf{T}$ are relational schemas, Σ_{st} is a set of tuple-generating dependencies (tgds) between $\mathbf{S}$ and $\mathbf{T}$, also called *source-to-target tgds*, and Σ_t is a set of tgds and equality-generating dependencies (egds) on $\mathbf{T}$. Fagin et al. isolated a class of solutions for the data exchange problem, called *universal solutions*, and showed that they have good properties that justify selecting them as the preferred solutions in data exchange. Universal solutions are solutions that can be homomorphically mapped into every other solution; thus, intuitively, universal solutions are the most general solutions. Moreover, in a precise sense, universal solutions represent the entire space of solutions. One of the main results in [8] is that, under fairly general conditions (*weak acyclicity* of the set of target tgds), a *canonical* universal solution can be computed (if solutions exist) in polynomial time, by using the classical chase procedure [2].

In general, universal solutions need not be unique. Thus, in a data exchange setting, there may be many universal solutions for a given source instance. Fagin, Kolaitis and Popa [9] addressed the issue of further isolating a "best" universal solution, by using the concept of the *core* of a graph or a structure [14]. By definition, the core of a structure is the smallest substructure that is also a homomorphic image of that structure. Since all universal solutions for a source instance *I* are homomorphically equivalent, it follows that they all have the same core (up to isomorphism). It is then shown in [9] that this core is also a universal solution, and hence the smallest universal solution. The uniqueness of the core of a universal solution together with its minimality make the core an ideal solution for data exchange. In a series of papers that started with [9] and continued with [12,13], it was shown that the core of the universal solutions can be computed in polynomial time, for data exchange settings where Σ_{st} is a set of source-to-target tgds and Σ_t is the union of a weakly-acyclic set of tgds with a set of edgs. This is in contrast with the general case of computing the core of an arbitrary structure, for which it is known that, unless P=NP, there is no polynomial-time algorithm.

There are quite a few papers on data exchange and theory of schema mappings that extended or made use of the concepts and results introduced in [8,9]. Some of the more representative ones addressed: extensions to XML data exchange [1], extensions to peer data exchange [11], the study of solutions under the closed-world assumption (CWA) [17], combined complexity of data exchange [15], schema mapping composition [10,19] and schema mapping inversion [7]. The Clio system, which served as both motivation and implementation playground for data exchange, was the first to use source-to-target dependencies as a language for expressing schema mappings [20]. Mapping constraints, expressed as either embedded dependencies (which comprise tgds and egds) or as equalities between relational or SQLS expressions, also play a central role in the model management framework of Bernstein and Melnik [3].

Foundations

Given a source schema **S** and a target schema **T** that are assumed to be disjoint, a *source-to-target dependency* is, in general, a formula of the form $\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \chi_{\mathbf{T}}(\mathbf{x}))$, where $\phi_{\mathbf{S}}(\mathbf{x})$ is a formula, with free variables **x**, over the source schema **S**, and $\chi_{\mathbf{T}}(\mathbf{x})$ is a formula, with free variables **x**, over the target schema **T**. The notation **x** signifies a vector of variables $x_1,\ldots,x_k$. A *target* dependency is, in general, a formula over the target schema **T** (the formalism used to express a target dependency may be different in general from those used for the source-to-target dependencies). The source schema may also have dependencies that are assumed to be satisfied by every source instance. Source dependencies do not play a direct role in data exchange, because the source instance is *given*.

The focus in [8] and in most of the subsequent papers on data exchange theory is on the case when **S** and **T** are relational schemas and when the dependencies are given as *tuple-generating dependencies (tgds) and equality-generating dependencies (egds)* [2]. More precisely, each source-to-target dependency in Σ_{st} is assumed to be a tad of the form

$$\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x},\mathbf{y})),$$

where $\phi_{\mathbf{S}}(\mathbf{x})$ is a conjunction of atomic formulas over **S** and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over **T**. All the variables in **x** are assumed to appear in $\phi_{\mathbf{S}}(\mathbf{x})$. Moreover, each target dependency in Σ_t is either a tad (of the form shown below left) or an edg (of the form shown below right):

$$\forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x},\mathbf{y})) \quad \forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_1 = x_2))$$

In the above, $\phi_{\mathbf{T}}(\mathbf{x})$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over **T**, where all the variables in **x** appear in $\phi_{\mathbf{T}}(\mathbf{x})$, and x_1, x_2 are among the variables in **x**. An often used convention is to drop the universal quantifiers in front of a dependency, and implicitly assume such quantification. However, the existential quantifiers are explicitly written down.

Source-to-target tgds are a natural and expressive language for expressing the relationship between a source and a target schema. Such dependencies are semi-automatically derived in the Clio system [20] based on correspondences between the source schema and the target schema. In turn, such correspondences can either be supplied by a human expert or discovered via schema matching techniques. Source-to-target tgds are also equivalent to the language of "sound" *global-and-local-as-view (GLAV)* assertions often used in data integration systems [16].

It is natural to take the target dependencies to be tgds and egds: these two classes together comprise the (embedded) implicational dependencies [6]. However, it is somewhat surprising that tgds, which were originally "designed" for other purposes (as constraints), turn out to be ideally suited for specifying desired data transfer.

Example 1. Figure 1b shows a source schema (on the left) and a target schema (on the right) with

I

src_1.students:

001 Mary CS120 A
005 John CS500 B

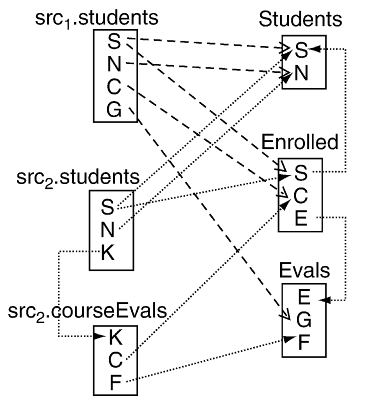
src_2.students:

001 Mary K7
001 Mary K4

src_2.courseEvals:

K7 CS120 file01
K4 CS200 file07

a A source instance



b Source and target schemas with their correspondences

Σ_{st}:

(t_1) $src_1.students(s,n,c,g) \rightarrow$
$\exists E\, \exists F\, [students(s,n) \wedge enrolled(s,c,E) \wedge evals(E,g,F)]$

(t_2) $src_2.students(s,n,k) \wedge src_2.courseEvals(k,c,f) \rightarrow$
$\exists E\, \exists G\, [students(s,n) \wedge enrolled(s,c,E) \wedge evals(E,G,F)]$

Σ_t:

(i_1) $enrolled(s,c,e) \rightarrow \exists N\ students(s,N)$
(i_2) $enrolled(s,c,e) \rightarrow \exists G\, \exists F\ evals(e,G,F)$
(e_1) $enrolled(s,c,e) \wedge enrolled(s,c,e') \rightarrow (e = e')$
(e_2) $evals(e,g,f) \wedge evals(e,g',f') \rightarrow (g = g')$
(e_3) $evals(e,g,f) \wedge evals(e,g',f') \rightarrow (f = f')$

c Dependencies: Σ_{st} and Σ_t

Data Exchange. Figure 1. A data exchange example.

correspondences between their attributes. The source schema models two different data sources or databases, src_1 and src_2, each representing data about students. The first source consists of one relation, src_1.*students*, while the second source consists of two relations, src_2.*students* and src_2.*courseEvals*. The attributes S, N, C, G, F represent, respectively,"student id," "student name," "course," "grade" (only in src_1), and "file evaluation" (a written evaluation that a student receives for a course; only in src_2). The attribute K in src_2 is used to link students with the courses they take; more concretely, K plays the role of a foreign key in src_2.*students* and the role of a key in src_2.*courseEvals*. As seen in the instance in Fig. 1a, information in the two sources may overlap: the same student can appear in both sources, with each source providing some information that the other does not have (e.g., either grade or file evaluation).

The two data sources are mapped into a target schema with three relations: *students* (with general student information), *enrolled* (listing course entries for each student), and *evals* (with evaluation entries per student and per course). The attribute E (evaluation id) is used to link enrollment entries with the associated evaluation records (E is a foreign key in *enrolled* and a key in *evals*). Similarly, the attribute S (student id) links *enrolled* with *students*.

The relationship between the individual attributes in the schemas is described by the arrows or correspondences that "go" between the attributes. However, the more precise mapping between the schemas is given by the set $\Sigma_{st} = \{t_1, t_2\}$ of source-to-target tgds that is shown in Fig. 1c.

The first source-to-target tad, t_1, specifies that for each tuple in src_1.*students* there must exist three corresponding tuples in the target: one in *students*, one in *enrolled*, and one in *evals*. Moreover, t_1 specifies how the four components of the source tuple (i.e., *s*, *n*, *c*, *g*) must appear in the target tuples. The tad also specifies the existence of "unknown" values (via the existential variables E and F) for the target attributes that do not have any corresponding attribute in the source. Note that existential variables can occur multiple times; in the example, it is essential that the same variable E is used in both *enrolled* and *evals* so that the association between students, courses and their grades is not lost in the target.

The second source-to-target tad, t_2, illustrates a case where the source pattern (the premise of the tad) is not limited to one tuple of a relation but encodes a join between multiple relations. In general, not all variables in the source pattern must occur in the target (e.g., k does not occur in the target). In this example, t_2 plays a "complementary" role to t_1, since it maps a different source that contains file evaluations rather than grades.

The target dependencies in Σ_t are formulas expressed solely in terms of the target schema that further constrain the space of possible target instances. In this example, the tgds i_1 and i_2 are inclusion dependencies that encode referential integrity constraints from *enrolled* to *students* and *evals*, respectively. The egds e_1, e_2 and e_3 encode functional dependencies. that must be satisfied. In particular, e_1 requires that a student and a course must have a unique evaluation id, while e_2 and e_3 together specify that the evaluation id must be a key for *evals*.

Solutions In general, in a data exchange setting (**S**, **T**, Σ_{st},Σ_t), there can be zero or more *solutions J* for a given source instance *I*. In other words, there can be zero or more target instances *J* such that: (1) *J* satisfies the target dependencies in Σ_t, and (2) *I* together with *J* satisfy the source-to-target dependencies in Σ_{st}. The latter condition simply means that the instance $\langle I, J\rangle$ that is obtained by considering together all the relations in *I* and *J* satisfies Σ_{st}. Note that $\langle \mathrm{I}, J\rangle$ is an instance over the union of the schemas **S** and **T**.

Example 2. Figure 2 illustrates three target instances that are plausible for the source instance *I* shown in Fig. 1a, and given the dependencies Σ_{st} and Σ_t in Fig.1b. Consider the first instance J_0, shown in Fig. 2a. It can be seen that $\langle I, J_0\rangle$ satisfies all the source-to-target dependencies in Σ_{st}; in particular, for any combination of the source data in *I* that satisfies the premises of some source-to-target tad in Σ_{st}, the "required" target tuples exist in J_0. Note that in J_0, the special values $E_1, \ldots, E_4$, F_1, F_2, G_3 and G_4 are used to represent "unknown" values, that is, values that do not occur in the source instance. Such values are called *labeled nulls* or *nulls* and are to be distinguished from the values occurring in the source instance, which are called *constants*. (See the later definitions.) It can then be seen that J_0 fails to satisfy the set Σ_t of target dependencies; in particular, the egd e_1 is not satisfied (there are two *enrolled* tuples for student 001 and course CS120 having different evaluation ids, E_1 and E_3). Thus, J_0 is not a solution for *I*.

On the other hand, the two instances J_1 and J_2 shown in Fig. 2b and c, respectively, are both solutions for *I*. The main difference between J_1 and J_2 is that J_2 is more "compact": the same null E_2 is used as an evaluation id for two different pairs of student and course (in contrast, J_1 has different nulls, E_2 and E_4).

In this example, J_1 and J_2 illustrate two possible ways of filling in the target that both satisfy the given specification. In fact, there are infinitely many possible solutions: one could choose other nulls or even constants instead of $E_1, E_2, \ldots$, or one could add "extra" target tuples, and still have all the dependencies satisfied. This raises the question of which solutions to choose in data exchange and whether some solutions are better than others.

Universal solutions. A key concept introduced in [8] is that of *universal solutions*, which are the most general among all the possible solutions.

Let $\underline{\mathrm{Const}}$ be the set, possibly infinite, of all the values (also called *constants*) that can occur in source instances. Moreover, assume an infinite set $\underline{\mathrm{Var}}$ of values, called *labeled nulls*, such that $\underline{\mathrm{Var}} \cap \underline{\mathrm{Const}} = \emptyset$. The symbols I, I', I_1, $I_2, \ldots$ are reserved for instances over the source schema **S** and with values in $\underline{\mathrm{Const}}$. The symbols J, J', J_1, $J_2, \ldots$ are reserved for instances over the target schema **T** and with values in $\underline{\mathrm{Const}} \cup \underline{\mathrm{Var}}$. All the target instances considered, and in particular, the solutions of a data exchange problem, are assumed to have values in $\underline{\mathrm{Const}} \cup \underline{\mathrm{Var}}$. If *J* is a target instance, then $\underline{\mathrm{Const}}(J)$ denotes the set of all constants occurring in *J*, and $\underline{\mathrm{Var}}(J)$ denotes the set of labeled nulls occurring in *J*.

Let J_1 and J_2 be two instances over the target schema. A *homomorphismh* : $J_1 \rightarrow J_2$ is a mapping from $\underline{\mathrm{Const}}(J_1) \cup \underline{\mathrm{Var}}(J_1)$ to $\underline{\mathrm{Const}}(J_2) \cup \underline{\mathrm{Var}}(J_2)$ such that: (1) $h(c) = c$, for every $c \in \underline{\mathrm{Const}}(J_1)$; (2) for every tuple

$\underline{J_0}$

students:
001 Mary
005 John
enrolled:
001 CS120 E_1
005 CS500 E_2
001 CS120 E_3
001 CS200 E_4
evals:
E_1 *A* F_1
E_2 *B* F_2
E_3 G_3 *File01*
E_4 G_4 *File07*

a A target instance that is not a solution

$\underline{J_1}$

students:
001 Mary
005 John

enrolled:
001 CS120 E_1
005 CS500 E_2
001 CS200 E_4

evals:
E_1 *A* *File01*
E_2 *B* F_2
E_4 G_4 *File07*

b A universal solution

$\underline{J_2}$

students:
001 Mary
005 John

enrolled:
001 CS120 E_1
005 CS500 E_2
001 CS200 E_2

evals:
E_1 *A* *File01*
E_2 *B* *File07*

c A solution that is not universal

Data Exchange. Figure 2. Examples of target instances.

t in a relation R of J_1, the tuple $h(t)$ is in the relation R of J_2 (where, if $t=(a_1,\ldots,a_s)$, then $h(t)=(h(a_1),\ldots,h(a_s))$). The instance J_1 is *homomorphically equivalent* to the instance J_2 if there are homomorphisms $h: J_1 \rightarrow J_2$ and $h': J_2 \rightarrow J_1$.

Consider a data exchange setting $(\mathbf{S},\mathbf{T},\Sigma_{st},\Sigma_t)$. If I is a source instance, then a *universal solution for I* is a solution J for I such that for every solution J' for I, there exists a homomorphism $h: J \rightarrow J'$.

Example 3. The solution J_2 in Fig. 2c is not universal. In particular, there is no homomorphism from J_2 to the solution J_1 in Fig. 2b. Specifically, the two *enrolled* tuples $(005,\ CS500,\ E_2)$ and $(001, CS200, E_2)$ of J_2 cannot be mapped into tuples of J_1 (since E_2 cannot be mapped into both E_2 and E_4). Thus, J_2 has some "extra" information that does not appear in all solutions (namely, the two *enrolled* tuples for $(005, CS500)$ and $(001, CS200)$ sharing the same evaluation id). In contrast, a universal solution has only information that can be homomorphically mapped into every possible solution. It can be shown that J_1 is such a universal solution, since it has a homomorphism to every solution (including J_2).

As the above example suggests, universal solutions are the preferred solutions in data exchange, because they are at least as general as any other solution (i.e., they do not introduce any "extra" information).

Computing universal solutions with the chase. Fagin et al. [8] addressed the question of how to check the existence of a universal solution and how to compute one, if one exists. They showed that, under a *weak acyclity* condition on the set Σ_t of target dependencies, universal solutions exist whenever solutions exist. Moreover, they showed that, under the same condition, there is a polynomial-time algorithm for computing a *canonical* universal solution, if a solution exists; this algorithm is based on the classical chase procedure.

Intuitively, the following procedure is applied to produce a universal solution from a source instance I: start with the instance $\langle I, \emptyset \rangle$ that consists of I for the source, and the empty instance for the target; then chase $\langle I, \emptyset \rangle$ with the dependencies in Σ_{st} and Σ_t in some arbitrary order and for as long as they are applicable. Each chase step either adds new tuples in the target or attempts to equate two target values (possibly failing, as explained shortly). More concretely, let $\langle I, J \rangle$ denote an intermediate instance in the chase process (initially $J = \emptyset$). Chasing with a source-to-target tad $\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ amounts to the following: check whether there is a vector $\mathbf{a}$ of values that interprets $\mathbf{x}$ such that $I \vDash \phi_{\mathbf{S}}(\mathbf{a})$, but there is no vector $\mathbf{b}$ of values that interprets $\mathbf{y}$ such that $J \vDash \psi_{\mathbf{T}}(\mathbf{a}, \mathbf{b})$; if such $\mathbf{a}$ exists, then add new tuples to J, where fresh new nulls $\mathbf{Y}$ interpret the existential variables $\mathbf{y}$, such that the resulting target instance satisfies $\psi_{\mathbf{T}}(\mathbf{a}, \mathbf{Y})$. Chasing with a target tad is defined similarly, except that only the target instance is involved. Chasing with a target egd $\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_1 = x_2)$ amounts to the following: check whether there is a vector $\mathbf{a}$ of values that interprets $\mathbf{x}$ such that $J \vDash \phi_{\mathbf{T}}(\mathbf{a})$ and such that $a_1 \neq a_2$; if this is the case, then the chase step attempts to identify a_1 and a_2, as follows. If both a_1 and a_2 are constants then the chase fails; it can be shown that there is no solution (and no universal solution) in this case. If one of a_1 and a_2 is a null, then it is replaced with the other one (either a null or a constant); this replacement is global, throughout the instance J. If no more chase steps are applicable, then the resulting target instance J is a universal solution for I.

Example 4. Recall the earlier data exchange scenario in Fig. 1. Starting from the source instance I, the source-to-target tgds in Σ_{st} can be applied first. This process adds all the target tuples that are "required" by the tuples in I and the dependencies in Σ_{st}. The resulting target instance after this step is an instance that is identical, modulo renaming of nulls, to the instance J_0 in Fig. 2a. Assume, for simplicity, that the result *is* J_0.

The chase continues by applying the dependencies in Σ_t to J_0. The tgds (i_1) and (i_2) are already satisfied. However, the egd (e_1) is not satisfied and becomes applicable. In particular, there are two *enrolled* tuples with the same student id and course but different evaluation ids (E_1 and E_3). Since E_1 and E_3 are nulls, the chase with e_1 forces the replacement of one with the other. Assume that E_3 is replaced by E_1. After the replacement, the two *enrolled* tuples become identical (hence, one is a duplicate and is dropped). Moreover, there are now two *evals* tuples with the same evaluation id (E_1). Hence, the egds e_2 and e_3 become applicable. As a result, the null G_3 is replaced by the constant A and the null F_1 is replaced by the constant *file*01. The resulting target instance is the instance J_1 in Fig. 2b, which is a canonical universal solution for I.

As a remark on the expressive power of dependencies (and of their associated chase), note how the above process has merged, into the same tuple, information from two different sources (i.e., the grade, and respectively, the file, for student 001 and course *CS*120). Also note that the chase is, inherently, a recursive procedure.

In general, the chase with an arbitrary set of target tgds and egds may not terminate. Hence, it is natural to ask for sufficient conditions for the termination of the chase. An extensively studied condition that guarantees termination is that the target tgds in Σ_t form a *weakly acyclic set of tgds* (the latter is also known as a set of constraints with *stratified witnesses*) [8,5]. Two important classes of dependencies that are widely used in database dependency theory, namely sets of full target tgds and acyclic sets of inclusion dependencies, are special cases of weakly acyclic sets of tgds.

The following theorem summarizes the use of chase in data exchange and represents one of the main results in [8].

Theorem 1 *Assume a data exchange setting* (**S**, **T**, Σ_{st},Σ_t) where Σ_{st} *is a set of source-to-target tgds, and* Σ_t *is the union of a weakly acyclic set of tgds with a set of egds. Then: (1) The existence of a solution can be checked in polynomial time. (2) A universal solution exists if and only if a solution exists. (3) If a solution exists, then a universal solution can be produced in polynomial time using the chase.*

The weak acyclicity restriction is essential for the above theorem to hold. In fact, it was shown in [15] that if the weak-acyclicity restriction is removed, the problem of checking the existence of solutions becomes undecidable.

Multiple universal solutions and core. In general, in a data exchange setting, there can be many universal solutions for a given source instance. Nevertheless, it has been observed in [9] that all these universal solutions share one common part, which is the *core* of the universal solutions. The core of the universal solutions is arguably the "best" universal solution to materialize, since it is the unique most compact universal solution. It is worth noting that the chase procedure computes a universal solution that may not necessarily be the core. So, additional computation is needed to produce the core.

A universal solution that is not a core necessarily contains some "redundant" information which does not appear in the core. Computing the core of the universal solutions could be performed, conceptually, in two steps: first, materialize a canonical universal solution by using the chase, then remove the redundancies by taking to the core. The second step is tightly related to *conjunctive query minimization* [4], a procedure that is in general intractable. However, by exploiting the fact that in data exchange, the goal is on computing the core of a universal solution rather than that of an arbitrary instance, polynomial-time algorithms were shown to exist for certain large classes of data exchange settings. Specifically, for data exchange settings where Σ_{st} is a set of arbitrary source-to-target tgds and Σ_t is a set of egds, two polynomial-time algorithms, the *blocks* algorithm and the *greedy* algorithm, for computing the core of the universal solutions were given in [9]. By generalizing the blocks algorithm, this tractability case was further extended in [12] to the case where the target tgds are full and then in [13] to the more general case where the target tgds form a weakly acyclic set of tgds.

Further results on query answering. The semantics of data exchange problem (i.e., which solution to materialize) is one of the main issues in data exchange. Another main issue is that of answering queries formulated over the target schema. Fagin et al. [8] adopted the notion of the "certain answers" in incomplete databases for the semantics of query answering in data exchange. Furthermore, they studied the issue of when can the certain answers be computed based on the materialized solution alone; in this respect, they showed that in the important case of unions of conjunctive queries, the certain answers can be obtained simply by running the query on an arbitrary universal solution and by eliminating the tuples that contain nulls. This in itself provided another justification for the "goodness" of universal solutions. The followup paper [9] further investigated the use of a materialized solution for query answering; it showed that for the larger class of existential queries, evaluating the query on the core of the universal solutions gives the best approximation of the certain answers. In fact, if one redefines the set of certain answers to be those that occur in every universal solution (rather than in every solution), then the core gives the exact answer for existential queries.

Key Applications

Schema mappings are the fundamental building blocks in information integration. Data exchange gives theoretical foundations for schema mappings, by studying the transformation semantics associated to a schema mapping. In particular, universal solutions are the main concept behind the "correctness" of any program, query or ETL flow that implements a schema mapping specification. Data exchange concepts are also essential in the study of the operators on schema mappings such as composition of sequential schema mappings and inversion of schema mappings.

Cross-references

► Data Integration
► Schema Mapping
► Schema Mapping Composition

Recommended Reading

1 Arenas M. and Libkin L. XML data exchange: consistency and query answering. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 13–24.
2. Beeri C. and Vardi M.Y. A proof procedure for data dependencies. J. ACM (JACM), 31(4):718–741, 1984.
3. Bernstein P.A. and Melnik S. Model management 2.0: manipulating richer mappings. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1–12.
4. Chandra A.K. and Merlin P.M. Optimal implementation of conjunctive queries in relational data bases. In Proc. 9th Annual ACM Symp. on Theory of Computing, 1977, pp. 77–90.
5. Deutsch A. and Tannen V. XML queries and constraints, containment and reformulation. Theor. Comput. Sci. (TCS), 336 (1):57–87, 2005.
6. Fagin R. Horn clauses and database dependencies. J. ACM (JACM), 29(4):952–985, 1982.
7. Fagin R. Inverting schema mappings. ACM Trans. Database Syst. (TODS), 32(4), 2007.
8. Fagin R., Kolaitis P.G., Miller R.J., and Popa L. Data exchange: semantics and query answering. Theor. Comput. Sci. (TCS), 336 (1):89–124, 2005.
9. Fagin R., Kolaitis P.G., and Popa L. Data exchange: getting to the core. ACM Trans. Database Syst. (TODS), 30(1):174–210, 2005.
10. Fagin R., Kolaitis P.G., Popa L., and Tan W.-C. Composing schema mappings: second-order dependencies to the rescue. ACM Trans. Database Syst. (TODS), 30(4):994–1055, 2005.
11. Fuxman A., Kolaitis P.G., Miller R.J., and Tan W.-C. Peer data exchange. ACM Trans. Database Syst. (TODS), 31(4): 1454–1498, 2006.
12. Gottlob G. Computing cores for data exchange: new algorithms and practical solutions. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005.
13. Gottlob G. and Nash A. Data exchange: computing cores in polynomial time. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 40–49.
14. Hell P. and Nešetřil J. The core of a graph. Discrete Math, 109:117–126, 1992.
15. Kolaitis P.G., Panttaja J., and Tan W.C. The complexity of data exchange. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 30–39.
16. Lenzerini M. Data Integration: A Theoretical Perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
17. Libkin L. Data exchange and incomplete information. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 60–69.
18. Miller R.J., Haas L.M., and Hernández M.A. Schema mapping as query discovery. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 77–88.
19. Nash A., Bernstein P.A., and Melnik S. Composition of mappings given by embedded dependencies. ACM Trans. Database Syst. (TODS), 32(1):4, 2007.
20. Popa L., Velegrakis Y., Miller R.J., Hernández M.A., and Fagin R. Translating Web data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 598–609.
21. Shu N.C., Housel B.C., Taylor R.W., Ghosh S.P., and Lum V.Y. EXPRESS: A Data EXtraction, Processing, and REStructuring System. ACM Trans. Database Syst. (TODS), 2(2):134–174, 1977.

Data Expiration

► Temporal Vacuuming

Data Extraction

► Screen Scraper

Data Flow Diagrams

► Activity Diagrams

Data Fusion

► Semantic Data Integration for Life Science Entities

Data Fusion in Sensor Networks

Aman Kansal, Feng Zhao
Microsoft Research, Redmond, WA, USA

Synonyms

Distributed sensor fusion

Definition

Data fusion in sensor networks is defined as the set of algorithms, processes, and protocols that combine data from multiple sensors. The goal may be to extract information not readily apparent in an individual sensor's data, improve the quality of information compared to that provided by any individual data, or improve the operation of the network by optimizing usage of its resources.

For instance, the output of a magnetic sensor and an audio sensor may be combined to detect a vehicle (new information), outputs of multiple vibration sensors may be combined to increase the signal to noise ratio (improving quality), or a passive infrared sensor may be combined with a camera in a people detection network to reduce the frame-rate of the camera for conserving energy (improving operation).

The sensors fused may be of the same or different types. Key features of data fusion in sensor networks, that distinguish it from other methods to combine multiple sources of data, are that the former methods are designed to (i) reduce the amount of communication among and from the sensors, and (ii) increase the life-time of the sensor network when all or some of the sensors are battery operated. The methods used to combine the data may leverage past observations from the same sensors, previously known models of the sensed phenomenon, and other information in addition to the sensor data.

Historical Background

Data fusion in sensor networks is founded on the methods developed for fusion of data in several older systems that used multiple sensors but not under the same system constraints as are typical of sensor networks. The Radar systems used in World War II present one of the first notable examples of such systems. The advantages that these systems demonstrated in robustness to failure of a fraction of sensors increased quality of data due to increased dimensionality of the measurement space. Since then, better discrimination between available hypotheses have led to the use of fusion methods in many sensor systems.

Most of the algorithms for sensor data processing can be viewed as derivatives of the Kalman filter and related Bayesian methods [7,9]. Decentralized forms of these methods have been developed to take data from multiple sensors as input and produced a single higher quality fused output [15,16]. Latest advances in these areas are discussed at the IEEE Sensor Array and Multichannel Signal Processing Workshop and the International Conference on Information Fusion among other venues. Other related works on fusion of sensor data are found in computer vision, tracking, and defense applications [2,3,5,6,8].

The achievable advantage in reducing signal distortion through fusion of multiple sensor inputs has been derived using information theoretic methods for general and Gaussian models for the phenomenon and sensor noise [1,12,13].

Foundations

The basic data fusion problem may be expressed as follows. The sensor network is deployed to measure a phenomenon represented by a state vector $\mathbf{x}^{(\mathbf{t})}$. For example, if the phenomenon is an object being tracked, $\mathbf{x}^{(\mathbf{t})}$ may represent a vector of position and velocity of the object at time t. The observation model at sensor i, that relates the observation $\mathbf{z}$ to state $\mathbf{x}$, is assumed to be Gaussian in many of the works, for computational tractability:

$$\mathbf{z}_i(t) = \mathbf{H}_i(\mathbf{x}(t)) + \mathbf{w}_i(t) \tag{1}$$

Linear models where $\mathbf{H}$ is a matrix are often used.

The *belief* about the phenomenon at time t is defined to be the *a posteriori* distribution of $\mathbf{x}$:

$$p(\mathbf{x}|\mathbf{z}_1, \ldots, \mathbf{z}_n) \tag{2}$$

where n is the number of sensors. The belief is sufficient to characterize the phenomenon and compute typical statistics such as the expected value of $\mathbf{x}$ and its residual uncertainty after the estimation.

The centralized methods to determine this belief from the observations require a knowledge of measurements $\mathbf{z}_i$ from all the sensors. The decentralized Kalman filters, such as [15,16], typically assume that each of the n sensors in the sensor network is connected to every other sensor and an $O(n^2)$ communication overhead is acceptable. This design may be used for fusion in systems with a small number of sensors or when high data rate communication links are present among sensors, such as networks of defense vehicles. However, such a high overhead is not acceptable in sensor networks based on embedded and wireless platforms. In these systems, both due to the large number of sensors and the low data rates supported by their radios for battery efficiency, the fusion methods must minimize

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:11

the communication requirements. Such fusion methods are said to be distributed.

One approach [17,18] to realize a distributed fusion method is to selectively use only a subset of the large number of sensors that have the most relevant information about the phenomenon being sensed. The fusion method then is also required to provide for appropriate selection of these most informative sensors and to dynamically adapt the set of selected sensor as the phenomenon evolves over time. To this end, a quantitative measure of the usefulness of a sensor for fusion is introduced, referred to as the *information content* of that sensor. An information utility function is defined:

$$\psi : \mathcal{P}(\mathcal{R}^d) \rightarrow \mathcal{R} \tag{3}$$

that acts on the class $\mathcal{P}(\mathcal{R}^d)$ of all probability distributions on $\mathcal{R}^d$ and returns a real number, with d being the dimension of $\mathbf{x}$. Specific choices of ψ are derived from information measures known from information theory, such as entropy, the Fischer information matrix, the size of the covariance ellipsoid for Gaussian phenomena models, sensor geometry based measures and others. An example form of ψ if information theoretic entropy is used, is:

$$\psi(p_{\mathbf{x}}) = \int_S p_{\mathbf{x}}(x) log(p_{\mathbf{x}}(x))dx \tag{4}$$

where S represents the domain of $\mathbf{x}$ and $p_{\mathbf{x}}$ is its probability distribution. Let $U \subset \{1,\ldots,n\}$ be the set of sensors whose measurements have been incorporated into the belief, i.e., the current belief is:

$$p(\mathbf{x}|\{\mathbf{z}_i\}_{i\in U}) \tag{5}$$

If the measurement from sensor j is also selected for inclusion in the computation of belief, the belief becomes:

$$p(\mathbf{x}|\{\mathbf{z}_i\}_{i\in U} \cup \{\mathbf{z}_j\}) \tag{6}$$

To select the sensor that has the maximum information content, the sensor j should be selected to maximize the information utility of the belief after including $\mathbf{z}_j$. Noting that j is to be selected from set $A = \{1, \ldots, n\} - U$, the best sensor $\hat{j}$ is:

$$\hat{j} = arg_{j\in A} max\ \psi\big(p(\mathbf{x}|\{\mathbf{z}_i\}_{i\in U} \cup \{\mathbf{z}_j\})\big) \tag{7}$$

However, in practice, the knowledge about $\mathbf{z}_j$ is not available before having selected j. The most likely best sensor j can then be selected by computing the expectation of the information utility with respect to $\mathbf{z}_j$:

$$\hat{j} = arg_{j\in A} max\ E_{\mathbf{z}_j}[\psi\big(p(\mathbf{x}|\{\mathbf{z}_i\}_{i\in U} \cup \{\mathbf{z}_j\})\big)|\{\mathbf{z}_i\}_{i\in U}] \tag{8}$$

among other options.

Also, the cost of communication from a sensor is explicitly modeled. Suppose the current belief is held at sensor l, referred to as the leader node. Suppose $M_c(l, j)$ denotes the cost of communication to sensor j. Then the sensor selection method choses the best sensor as follows:

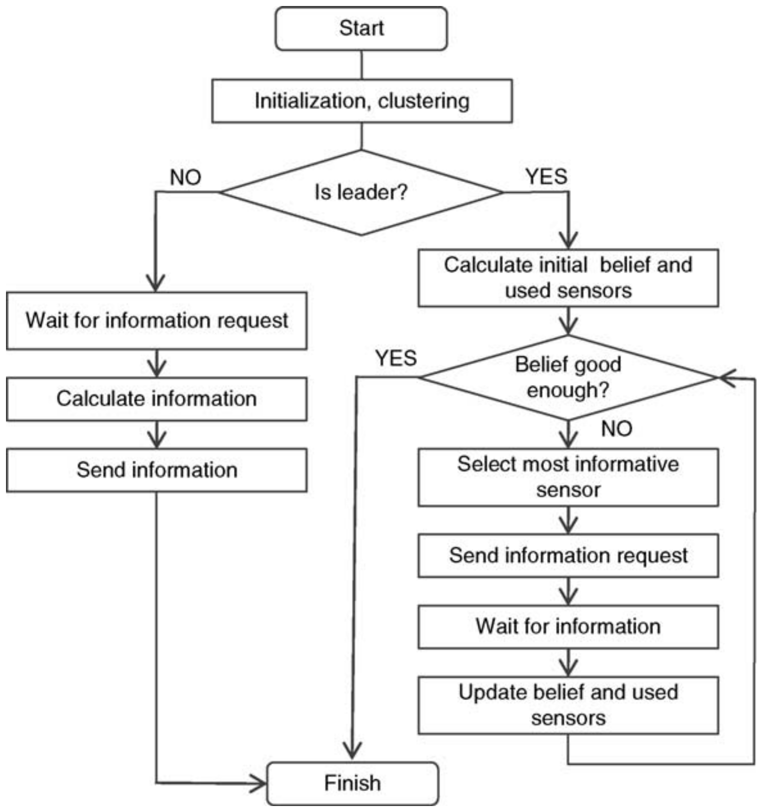
$$\hat{j} = arg_{j\in A}\max[\alpha M_u(j) - (1-\alpha)M_c(l,j)] \tag{9}$$

where $M_u(j)$ denotes the expectation of the information utility as expressed in (8), and $\alpha \in [0,1]$ balances the contribution from M_u and M_c.

These fundamentals can be used to develop a distributed data fusion algorithm for a sensor network. Suppose the sensor nodes are synchronized in time, and the phenomenon is initially detected at time $t = 0$. At this time, a known distributed leader election algorithm may be executed to select a node l as the leader, which computes the initial belief using only its own observation. It now selects the next sensor to be included in the belief calculation using (9). The process continues until the belief is known to a satisfactory quality as characterized by known statistical measures. The flowchart of the fusion algorithm followed at all the nodes is shown in Fig. 1.

The sensor selection process used in the algorithm above is a greedy one – at each selection step, it only considers a single sensor that optimizes the selection metric. It is possible that selecting multiple sensors at the same time yields a better choice. This can be achieved at the cost of higher computation complexity by selecting a set of sensors instead a single sensor j in (9), using appropriate modifications to the cost and utility metrics.

The distributed fusion method described above limits the number of sensors used and hence significantly reduces the communication overhead as compared to the $O(n^2)$ overhead of decentralized methods. It is well-suited for problems where the sensed phenomenon is localized, such as an object being tracked, since the number of most informative nodes selected can then yield fused results close to that provided by the entire network.


Start
Initialization, clustering
Is leader?
NO
YES
Wait for information request
Calculate information
Send information
Calculate initial belief and used sensors
Belief good enough?
YES
NO
Select most informative sensor
Send information request
Wait for information
Update belief and used sensors
Finish

Data Fusion in Sensor Networks. Figure 1. Distributed sensor fusion based on selecting only the most informative and cost effective sensors.

In another approach, a distributed Kalman filter is derived. The sensing model is as expressed before in (1). However, instead of using a centralized Kalman filter to estimate the state **x**, *n* micro-Kalman filters, each executing locally at the *n* sensor nodes, using the measurements from only the local node, are used. In addition, two consensus problems are solved using a method that requires communication only with one hop wireless neighbors [10,11]. This approach is more relevant when the observations from all the nodes in the network are important, such as when measuring a distributed phenomenon.

Distributed fusion methods that are tolerant to communication losses and network partitioning have also been developed using message passing on junction trees [14] and techniques from assumed density filtering [4].

Additionally, distributed methods are also available for cases where the sensed phenomenon is not itself required to be reproduced but only some of its properties are to be obtained. These properties may be global and depend on the measurements of all the sensors in the network, such as the number of targets in the region, the contours of a given phenomenon value in a heat map, or tracking relations among a set of objects [17].

The field of data fusion in sensor networks is rapidly evolving with new advances being presented frequently at forums including the ACM/IEEE IPSN and ACM SenSys.

Key Applications

Infrastructure monitoring, pervasive health-care, defense, scientific experimentation, environmental sensing, urban monitoring, home automation, supply-chain control, industrial control, business process monitoring, security.

Data Sets

Several data sets collected from experimental sensor network deployments are available for researchers to test their data fusion methods:

Center for Embedded Networked Sensing, http://www.sensorbase.org/

Intel Lab Data, http://db.csail.mit.edu/labdata/lab data.html

Cross-references

► Data Aggregation in Sensor Networks
► Data Estimation in Sensor Networks
► In-Network Query Processing

Recommended Reading

1. Berger T., Zhang Z., and Vishwanathan H. The CEO problem. IEEE Trans. Inform. Theory, 42(3):887–902, 1996.
2. Brooks R.R. and Iyengar S.S. Multi-sensor fusion: Fundamentals and applications with software. Prentice-Hall, Englewood, Cliffs, NJ, 1997.
3. Crowley J.L. and Demazeau Y. Principles and techniques for sensor data fusion. Signal Process., 32(1–2):5–27, 1993.
4. Funiak S., Guestrin C., Paskin M., and Sukthankar R. Distributed inference in dynamical systems. In Advances in Neural Information Processing Systems 19, B. Scholkopf, J. Platt, and T. Hoffman (eds.). MIT, Cambridge, MA, 2006, pp. 433–440.
5. Hall D.L. and McMullen S.A.H. Mathematical Techniques in Multisensor Data Fusion. Artech House, 2004.
6. Isard M. and Blake A. Condensation – conditional density propagation for visual tracking. Int. J. Comput. Vision, 29(1):5–28, 1998.
7. Jazwinsky A. Stochastic processes and filtering theory. Academic, New York, 1970.
8. Lodaya M.D. and Bottone R. Moving target tracking using multiple sensors. In Proc. SPIE, Vol. 4048, pp. 333–344, Signal and Data Processing of Small Targets 2000, O.E. Drummond (ed.)., vol. 4048 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference,* 2000, pp. 333–344.
9. Maybeck P. Cox I. and Wilfong G. (eds.). Autonomous robot vehicles, chap. The Kalman filter: An introduction to concepts. Springer, Berlin, 1990.
10. Olfati-Saber R. Distributed kalman filtering for sensor networks. In Proc. 46th IEEE Conf. on Decision and Control. 2007.
11. Olfati-Saber R. and Shamma J.S. Consensus filters for sensor networks and distributed sensor fusion. In Proc. 44th IEEE Conf. on Decision and Control. 2005.
12. Oohama Y. The rate distortion function for the quadratic Gaussian CEO problem. IEEE Trans. Inform. Theory, 44(3), 1998.
13. Pandya A., Kansal A., Pottie G.J., and Srivastava M.B. Fidelity and resource sensitive data gathering. In 42nd Allerton Conference. 2004.
14. Paskin M., Guestrin C., and McFadden J. A robust architecture for distributed inference in sensor networks. In Proc. 4th Int. Symp. Inf. Proc. in Sensor Networks. 2005.
15. Rao B., Durrant-Whyte H., and Sheen J. A fully decentralized multi-sensor system for tracking and surveillance. Int. J. Robot. Res., 12.
16. Speyer J.L. Computation and Transmission requirements for a decentralized linear-quadratic-Gaussian control problem. IEEE Trans. Automat. Control, 24(2):266–269, 1979.
17. Zhao F., Liu J., Liu J., Guibas L., and Reich J. Collaborative Signal and Information Processing: An Information Directed Approach. Proc. IEEE, 91(8):1199–1209, 2003.
18. Zhao F. and Guibas L. Wireless Sensor Networks: An Information Processing Approach. Morgan Kaufmann, 2004.

Data Gathering

► Data Acquisition and Dissemination in Sensor Networks

Data Grids

► Storage Grid

Data Imputation

► Data Estimation in Sensor Networks

Data Inconsistencies

► Data Conflicts

Data Integration

► Information Integration

Data Integration Architectures and Methodology for the Life Sciences

Alexandra Poulovassilis
University of London, London, UK

Definition

Given a set of biological data sources, *data integration* is the process of creating an integrated resource combining data from the data sources, in order to allow queries and analyses that could not be supported by the individual data sources alone. Biological data

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:11

sources are characterized by their high degree of heterogeneity, in terms of their data model, query interfaces and query processing capabilities, data types used, and nomenclature adopted for actual data values. Coupled with the variety, complexity and volumes of biological data that are becoming increasingly available, integrating biological data sources poses many challenges, and a number of methodologies, architectures and systems have been developed to support it.

Historical Background

If an application requires data from different data sources to be integrated in order to support users' queries and analyses, one possible solution is for the required data transformation and aggregation functionality to be encoded into the application's programs. However, this may be a complex and lengthy process, and may also affect the robustness and maintainability of the application. These problems have motivated the development of architectures and methodologies which abstract out data transformation and aggregation functionality into generic data integration software.

Much work has been done since the early 1990s in developing architectures and methodologies for integrating biological data sources in particular. Many systems have been developed which create and maintain integrated data resources: examples of significant systems are DiscoveryLink [7], K2/Kleisli [3], Tambis [6], SRS [16], Entrez [5], BioMart [4]. The main aim of such systems is to provide users with the ability to formulate queries and undertake analyses on the integrated resource which would be very complex or costly if performed directly on the individual data sources, sometimes prohibitively so.

Providing access to a set of biological data sources via one integrated resource poses several challenges, mainly arising from the large volumes, variety and complexity of othe data, and the autonomy and heterogeneity of the data sources [2,8,9]. Data sources are developed by different people in differing research environments for differing purposes. Integrating them to meet the needs of new users and applications requires the reconciliation of their different data models, data representation and exchange formats, content, query interfaces, and query processing capabilities. Data sources are in general free to change their data formats and content without considering the impact this may have on integrated resources derived from them. Integrated resources may themselves serve as data sources for higher-level integrations, resulting in a network of dependencies between biological data resources.

Foundations

Three main methodologies and architectures have been adopted for biological data integration, *materialized, virtual* and *link-based*:

- With *materialized integration*, data from the data sources is imported into a data warehouse and it is transformed and aggregated as necessary in order to conform to the warehouse schema. The warehouse is the integrated resource, typically a relational database. Queries can be formulated with respect to the warehouse schema and their evaluation is undertaken by the database management system (DBMS), without needing to access the original data sources.
- With *virtual integration,* a schema is again created for the integrated resource. However, the integrated resource is represented by this schema, and the schema is not populated with actual data. Additional mediator software is used to construct mappings between the data sources and the integrated schema. The mediator software coordinates the evaluation of queries that are formulated with respect to the integrated schema, utilizing the mappings and the query processing capabilities of the database or file management software at the data sources. Data sources are accessed via additional "wrapper" software for each one, which presents a uniform interface to the mediator software.
- With *link-based integration* no integrated schema is created. Users submit queries to the integration software, for example via a web-based user interface. Queries are formulated with respect to data sources, as selected by the user, and the integration software provides additional capabilities for facilitating query formulation and speeding up query evaluation. For example, SRS [16] maintains indexes supporting efficient keyword-based search over data sources, and also maintains cross-references between different data sources which are used to augment query results with links to other related data.

A link-based integration approach can be adopted if users will wish to query the data sources directly, will only need to pose keyword and navigation-style queries, and the scientific hypotheses that they will be investigating will not require any significant transformation or aggregation of the data. Otherwise, the adoption of a materialized or virtual integration approach is indicated. The link-based integration approach is discussed further in [1] and in the entry on Pathway Databases.

A key characteristic of materialized or virtual integration is that the integrated resource can be queried as though it were itself a single data source rather than an integration of other data sources: users and applications do not need to be aware of the schemas or formats of the original data sources, only the schema/format of the integrated resource. Materialized integration is usually chosen for query performance reasons: distributed access to remote data sources is avoided and sophisticated query optimization techniques can be applied to queries submitted to the data warehouse. Other advantages are that it is easier to clean and annotate the source data than by using mappings within a virtual integration approach. However, maintaining a data warehouse can be complex and costly, and virtual integration may be the preferred option if these maintenance costs are too high, or if it is not possible to extract data from the data sources, or if access to the latest versions of the data sources is required.

Examples of systems that adopt the materialized integration approach are GUS [3], BioMart [4], Atlas [14], BioMap [12]. With this approach, the standard methodology and architecture for data warehouse creation and maintenance can be applied. This consists of first extracting data from the data sources and transporting it into a "staging" area. Data from the data sources will need to be re-extracted periodically in order to identify changes in the data sources and to keep the warehouse up-to-date. Data extraction from each data source may be either full extraction or incremental extraction. With the former, the entire source data is re-extracted every time while with the latter it is only relevant data that has changed since the previous extraction. Incremental extraction is likely to be more efficient but for some data sources it may not be possible to identify the data that has changed since the last extraction, for example due to the limited functionality provided by the data sources, and full extraction may be the only option. After the source data has been brought into the staging area, the changes from the previous versions of the source data are determined using "difference" algorithms (in the case of full re-extraction) and the changed data is transformed into the format and data types specified by the warehouse schema. The data is then "cleaned" i.e., errors and inconsistencies are removed, and it is loaded into the warehouse. The warehouse is likely to contain materialized views which transform and aggregate in various ways the detailed data from the data sources. View maintenance capabilities provided by the DBMS can be used to update such materialized views following insertions, updates and deletions of the detailed data. It is also possible to create and maintain additional "data marts" each supporting a set of specialist users via a set of additional views specific to their requirements. The warehouse serves as the single data source for each data mart, and a similar process of extraction, transformation, loading and aggregating occurs to create and maintain the data mart.

One particular characteristic of biological data integration, as compared with business data integration for example, is the prevalence of both automated and manual annotation of data, either prior to its integration, or during the integration process, or both. For example, the Distributed Annotation System (DAS) (http://www.biodas.org) allows annotations to be generated and maintained by the owners of data resources, while the GUS data warehouse supports annotations that track the origins of data, information about algorithms or annotation software used to derive new inferred data, and who performed the annotation and when. Being able to find out the provenance of any data item in an integrated resource is likely to be important for users, and this is even more significant in biological data integration where multiple annotation processes may be involved.

Another characteristic of biological data integration is the wide variety of nomenclatures adopted by different data sources. This greatly increases the difficulty of aggregating their data and has led to the proposal of many standardized ontologies, taxonomies and controlled vocabularies to help alleviate this problem e.g., from the Gene Ontology (GO) Consortium, Open Biomedical Ontologies (OBO) Consortium, Microarray Gene Expression Data (MGED) Society and Proteomics Standards Initiative (PSI). The role of ontologies in scientific data integration is discussed in the entry on Ontologies in Scientific Data Integration. Another key issue is the need to resolve possible inconsistencies in the ways that biological entities are identified within the data sources. The same biological entity may be identified differently in different data sources or, conversely, the same identifier may be used for different biological entities in different data sources. There have been a number of initiatives to address the problem of inconsistent identifiers e.g., the Life Sciences Identifiers (LSID) initiative and the International Protein Index (IPI). Despite such initiatives, there is still a legacy of large numbers of non-standardized identifiers in biological datasets and therefore techniques are needed for associating biological entities independently of their identifiers. One technique is described in [12] where a clustering approach is used to identify sets of data source entities that are likely to refer to the same real-world entity.

Another area of complexity is that data sources may evolve their schemas over time to meet the needs of new applications or new experimental techniques (henceforth, the term "data source schema" is used to encompass also the data representation and exchange formats of data sources that are not databases). Changes in the data source schemas may require modification of the extraction-transformation-loading (ETL), view materialization and view maintenance procedures. Changes in the warehouse schema may impact on data marts derived from it and on the procedures for maintaining these.

Turning now to virtual data integration, architectures that support virtual data integration typically include the following components:

- A Repository for storing information about data sources, integrated schemas, and the mappings between them.
- A suite of tools for constructing integrated schemas and mappings, using a variety of automatic and interactive methods.
- A Query Processor for coordinating the evaluation of queries formulated with respect to an integrated schema; the Query Processor first reformulates such a query, using the mappings in the Repository, into an equivalent query expressed over the data source schemas; it then optimizes the query and evaluates it, submitting as necessary sub-queries to the appropriate data source Wrappers and merging the results returned by them.
- An extensible set of Wrappers, one for each type of data source being integrated; each Wrapper extracts metadata from its data source for storage in the Repository, translates sub-queries submitted to it by the Query Processor into the data source's query formalism, issues translated sub-queries to the data source, and translates sub-query results returned by the data source into the Query Processor's data model for further post-processing by the Query Processor.

An integrated schema may be defined in terms of a standard data modeling language, or it may be a source-independent ontology defined in an ontology language and serving as a "global" schema for multiple potential data sources beyond the specific ones that are being integrated (as in TAMBIS for example). The two main integration methodologies are *top-down* and *bottom-up*. With top-down integration, the integrated schema, *IS*, is first constructed, or may already exist from previous schema design, integration or standardization efforts. The set of mappings, *M*, between *IS* and the data source schemas are then defined. With bottom-up integration, an initial version of *IS* and *M* are first constructed – for example, these may be based on just one of the data source schemas. The integrated schema *IS* and the set of mappings *M* are then incrementally extended by considering in turn each of the other data source schemas: for each object *O* in each source schema, *M* is modified so as to encompass the mapping between *O* and *IS*, if it is possible to do so using the current *IS*; otherwise, *IS* extended as necessary in order to encompass the data represented by *O*, and *M* is then modified accordingly.

A mixed top-down/bottom-up approach is also possible: an initial *IS* may exist from a previous design or standardization activity, but it may need to be extended in order to encompass additional data arising from the set of data sources being integrated within it. With either top-down, bottom-up or mixed integration, it is possible that *IS* will not need to encompass all of the data of the data sources, but only a subset of the data which is sufficient for answering key queries and analyses – this avoids the possibly complex process of constructing a complete integrated schema and set of mappings.

There are a number of alternatives to defining the set of mappings *M* above, and different data integration systems typically adopt different approaches: with the *global-as-view* (GAV) approach, each mapping relates one schema object in *IS* with a view that is defined over the source schemas; with the *local-as-view* (LAV) approach, each mapping relates one schema object in one of the source schemas with a view defined over *IS*; and with the *global-local-as-view* (GLAV) approach, each mapping relates a view over a source schema with a view over *IS* [10,11]. Another approach is the *both-as-view* [13] approach supported by the AutoMed system. This provides a set of primitive transformations on schemas, each of which adds, deletes or renames a schema object. The semantic relationships between objects in the source schemas and the integrated schema are represented by reversible sequences of such transformations. The ISPIDER project [15] uses AutoMed for virtual integration of several Grid-enabled proteomics data sources.

In addition to the approach adopted for specifying mappings between source and integrated schemas, different systems may also make different assumptions about the degree of semantic overlap between the data sources: some systems assume that each data source contributes to a different part of the integrated resource (e.g., K2/Kleisli); some relax this assumption but do not undertake any aggregation of duplicate or overlapping data that may be present in the data sources (e.g., TAMBIS); and some can support aggregation at both the schema and the data levels (e.g., AutoMed). The degree of data source overlap impacts on the degree of schema and data aggregation that will need to be undertaken by the mappings, and hence on their complexity and the design effort involved in specifying them. The complexity of the mappings in turn impacts on the sophistication of the query processing mechanisms that will be needed in order to optimize and evaluate queries posed on the integrated schema.

Key Applications

- Integrating, analyzing and annotating genomic data.
- Predicting the functional role of genes and integrating function-specific information.
- Integrating organism-specific information.
- Integrating and analyzing chemical compound data and metabolic pathway data to support drug discovery.
- Integrating protein family, structure and pathway data with gene expression data, to support functional genomics data analysis.
- Integrating, analyzing and annotating proteomics data sources recording data from experiments on protein separation and identification.
- Supporting systems biology research.
- Integrating phylogenetic data sources for genealogical reconstruction.
- Integrating data about genomic variations in order to analyze the impact of genomic variations on health.
- Integrating genomic and proteomic data with clinical data to support personalized medicine.

Future Directions

Identifying semantic correspondences between different data sources is a necessary prerequisite to integrating them. This is still largely a manual and time-consuming process undertaken with significant input from domain experts. Semi-automatic techniques are being developed to alleviate this problem, for example name-based or structural comparisons of source schemas, instance-based matching at the data level to determine overlapping schema concepts, and annotation of data sources with terms from ontologies to facilitate automated reasoning over the data sources.

The transformation of source data into an integrated resource may result in loss of information, for example due to imprecise knowledge about the semantic correspondences between data sources. This is leading to research into capturing within the integrated resource incomplete and uncertain information, for example using probabilistic or logic-based representations and reasoning.

Large amounts of information are potentially available in textual form within published scientific articles. Automated techniques are being developed for extracting information from such sources using grammar and rule-based approaches, and then integrating this information with other structured or semi-structured biological data.

Traditional approaches to data integration may not be sufficiently flexible to meet the needs of distributed communities of scientists. Peer-to-peer data integration techniques are being developed in which there is no single administrative authority for the integrated resource and it is maintained instead by a community of peers who exchange, transform and integrate data in a pair-wise fashion and who cooperate in query processing over their data.

Finally, increasing numbers of web services are being made available to access biological data and computing resources - see, for example, the entry on Web Services and the Semantic Web for Life Science Data. Similar problems arise in combining such web services into larger-scale workflows as in integrating biological data sources: the necessary services are often created independently by different parties, using different technologies, formats and data types, and therefore additional code needs to be developed to transform the output of one service into a format that can be consumed by another.

Cross-references

► Data Provenance
► Ontologies and Life Science Data Management
► Ontologies in Scientific Data Integration
► Pathway Databases
► Provenance of Scientific Databases
► Semantic Data Integration for Life Science Entities
► Web Services and the Semantic Web for Life Science Data

Recommended Reading

1. Cohen-Boulakia S., Davidson S., Froidevaux C., Lacroix Z., and Vidal M.E. Path-based systems to guide scientists in the maze of biological data sources. J. Bioinformatics Comput. Biol., 4(5):1069–1095, 2006.
2. Davidson S., Overton C., and Buneman P. Challenges in integrating biological data sources. J. Comput. Biol., 2(4):557–572, 1995.
3. Davidson S.B., et al. K2/Kleisli and GUS: experiments in integrated access to genomic data sources. IBM Syst. J., 40 (2):512–531, 2001.
4. Durnick S., et al. Biomart and Bioconductor: a powerful link between biological databases and microarray data analysis. Bioinformatics, 21(16):3439–3440, 2005.
5. Entrez – the life sciences search engine. Available at: http://www.ncbi.nlm.nih.gov/Entrez
6. Goble C.A., et al. Transparent access to multiple bioinformatics information sources. IBM Syst. J., 40(2):532–551, 2001.
7. Haas L.M., et al. Discovery Link: a system for integrated access to life sciences data sources. IBM Syst. J., 40(2):489–511, 2001.
8. Hernandez T. and Kambhampati S. Integration of biological sources: current systems and challenges ahead. ACM SIGMOD Rec., 33(3):51–60, 2004.
9. Lacroix Z. and Critchlow T. Bioinformatics: Managing Scientific Data. The Morgan Kaufmann Series in Multimedia Information and Systems. Morgan Kaufmann, San Francisco, CA, 2004.
10. Lenzerini M. Data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
11. Madhavan J. and Halevy A.Y. Composing mappings among data sources. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 572–583.
12. Maibaum M., et al. Cluster based integration of heterogeneous biological databases using the AutoMed toolkit. In Proceedings of the Second International Workshop on Data Integration in the Life Sciences (DILS'05). Lecture Notes in Computer Science, 3615. 2005, pp. 191–207.
13. McBrien P. and Poulovassilis A. Data integration by bi-directional schema transformation rules. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 227–238.
14. Shah S.P., et al. Atlas – a data warehouse for integrative bioinformatics. BMC Bioinformatics, 6:34, 2005.
15. Zamboulis L., et al. Data access and integration in the ISPIDER Proteomics Grid. In Proceedings of the Third International Workshop on Data Integration in the Life Sciences (DILS '06). Springer, Lecture Notes in Computer Science, 5075. 2006, pp. 3–18.
16. Zdobnov E.M., Lopez R., Apweiler R., and Etzold T. The EBI SRS Server – recent developments. Bioinformatics, 18(2):368–373, 2002.

Data Integration in Web Data Extraction System

Marcus Herzog[1,2]
[1]Vienna University of Technology, Vienna, Austria
[2]Lixto Software GmbH, Vienna, Austria

Synonyms

Web information integration and schema matching; Web content mining; Personalized Web

Definition

Data integration in Web data extraction systems refers to the task of providing a uniform access to multiple Web data sources. The ultimate goal of Web data integration is similar to the objective of *data integration* in database systems. However, the main difference is that Web data sources (i.e., Websites) do not feature a structured data format which can be accessed and queried by means of a query language. In contrast, *Web data extraction systems* need to provide an additional layer to transform Web pages into (semi)-structured data sources. Typically, this layer provides an extraction mechanism that exploits the inherent document structure of HTML pages (i.e., the document object model), the content of the document (i.e., text), visual cues (i.e., formatting and layout), and the inter document structure (i.e., hyperlinks) to extract data instances from the given Web pages. Due to the nature of the Web, the data instances will most often follow a semi-structured schema. Successful data integration then requires to solve the task of reconciling the syntactic and semantic heterogeneity, which evolves naturally from accessing multiple independent Web sources. Semantic heterogeneity can be typically observed both on the schema level and the data instance level. The output of the Web data integration task is a unified data schema along with consolidated data instances that can be queried in a structured way. From an operational point of view, one can distinguish between on-demand integration of Web data (also referred to as metasearch) and off-line integration of Web data similar to the ETL process in data warehouses.

Historical Background

The concept of data integration was originally conceived by the database community. Whenever data are not stored in a single database with a single data schema, data integration needs to resolve the structural and semantic heterogeneity found in databases built by different parties. This is a problem that researches have been addressing for years [8]. In the context of web data extraction systems, this issue is even more pressing due to the fact that web data extraction systems usually deal with schemas of semi-structured data, which are more flexible both from a structural and semantic perspective. The Information Manifold [12] was one of the systems that not only integrated relational databases but also took Web sources into account. However, these Web sources were structured in nature and were queried by means of a Web form. Answering a query involved a join across the relevant web sites. The main focus of the work was on providing a mechanism to describe declaratively the contents and query capabilities of the available information sources.

Some of the first research systems which covered the aspects of data integration in the context of Web data extraction systems were ANDES, InfoPipes, and a framework based on the Florid system. These systems combine languages for web data extraction with mechanisms to integrate the extracted data in a homogeneous data schema. ANDES [15] is based on the Extensible Stylesheet Language Transformations (XSLT) for both data extraction and data integration tasks. The ANDES framework merges crawler technology with XML-based extraction techniques und utilized templates, (recursive) path expressions, and regular expressions for data extraction, mapping, and aggregation. ANDES is primarily a software framework, requiring application developers to manually build a complete process from components such as Data Retriever, Data Extractor, Data Checker, and Data Exporter.

The InfoPipes system [10] features a workbench for visual composition of processing pipelines utilizing XML-based processing components. The components are defined as follows: Source, Integration, Transformation, and Deliverey. Each of those components features a configuration dialog to interactively define the configuration of the component. The components can be arranged on the canvas of the workbench and can be connected to form information processing pipelines, thus the name InfoPipes. The Source component utilized ELOG programs [13] to extract semi-structured data from Websites. All integration tasks are subsequently performed on XML data. The Integration component also features a visual dialog to specify the reconciliation of the syntactic and semantic heterogeneity in the XML documents. These specifications are then translated into appropriate XSLT programs to perform the reconciliation during runtime.

In [14] an integrated framework for Web exploration, wrapping, data integration, and querying is described. This framework is based on the Florid [13] system and utilizes a rule-based object-oriented language which is extended by Web accessing capabilities and structured document analysis. The main objective of this framework is to provide a unified framework – i.e., data model and language – in which all tasks (from Web data extraction to data integration and querying) are performed. Thus, these tasks are not necessarily separated, but can be closely intertwined. The framework allows for modeling the Web both on the page level as well as on the parse-tree level. Combined rules for wrapping, mediating, and Web exploration can be expressed in the same language and with the same data model.

More recent work can be found in the context of Web content mining. Web content mining focuses on extracting useful knowledge from the Web. In Web content mining, Web data integration is a fundamental aspect, covering both schema matching and data instance matching.

Foundations

Semi-structured Data

Web data extraction applications often utilize XML as data representation formalism. This is due to the fact that the semi-structured data format naturally matches with the HTML document structure. In fact, XHTML is an application of XML. XML provides a common syntactic format. However, it does not offer any means for addressing the semantic integration challenge. Query languages such as XQuery [5], XPath [1] or XSLT [11] provide the mechanism to manipulate the structure and the content of XML documents. These languages can be used as basis for implementing integration systems. The semantic integration aspect has to be dealt with on top of the query language.

Schema and Instance Matching

The main issue in data integration is the finding the semantic mapping between a number of data sources. In the context of Web extraction systems, these sources are web pages or more generally websites. There are three distinct approaches to the matching problem: manual, semiautomatic, or automatic matching. In the manual approach, an expert needs to define the mapping by using a toolset. This is of course time consuming. Automatic schema matching in contrast is AI-complete [3] and well researched in the database community [16], but typically still lacks reliability. In the semiautomatic approach, automatic matching algorithms suggest certain mappings which are validated by an expert. This approach saves time due to filtering out the most relevant matching candidates.

An example for manual data integration framework is given in [6]. The Harmonize framework [9] deals with business-to-business (B2B) integration on the "information" layer by means of an ontology-based mediation. It allows organizations with different data standards to exchange information seamlessly without having to change their proprietary data schemas. Part of the Harmonize framework is a mapping tool that allows for manually generating mapping rules between two XML schema documents.

In contrast to the manual mapping approach, automated schema mapping has to rely on clues that can be derived from the schema descriptions: utilizing the similarities between the names of the schema elements or taking the amount of overlap of data values or data types into account.

While matching schemas is already a time-consuming task, reconciling the data instances is even more cumbersome. Due to the fact that data instances are extracted from autonomous and heterogeneous websites, no global identifiers can be assumed. The same real world entity may have different textual representations, e.g., "CANOSCAN 3000ex 48 Bit, 1200×2400 dpi" and "Canon CanoScan 3000ex, 1200 × 2400dpi, 48Bit." Moreover, data extracted from the Web is often incomplete and noisy. In such a case, a perfect match will not be possible. Therefore, a similarity metric for text joins has to be defined. Most often the widely used and established cosine similarity metric [17] from the information retrieval field is used to identify string matches. A sample implementation of text joins for Web data integration based on an unmodified RDBMS is given in [17]. Due to the fact that the number of data instances is much higher than the number of schema elements, data instance reconciliation has to rely on automatic procedures.

Web Content Mining

Web content mining uses the techniques and principles of data mining to extract specific knowledge from Web pages. An important step in Web mining is the integration of extracted data. Due to the fact that Web mining has to work on Web-scale, a fully automated process is required. In the Web mining process, Web data records are extracted from Web pages which serve as input for the subsequent processing steps. Due to the large scale approach of Web mining it calls for novel methods that draw from a wide range of fields spanning data mining, machine learning, natural language processing, statistics, databases, and information retrieval [4].

Key Applications

Web data integration is required for all applications that draw data from multiple Web sources and need to interpret the data in a new context. The following main application areas can be identified:

Vertical Search

In contrast to web search as provided by major search engines, vertical search targets a specific domain such as e.g., travel offers, job offers, or real estate offers. Vertical search applications typically deliver more structured results than conventional web search engines. While the focus of the web search is to cover the breath of all available websites and deliver the most relevant websites for a given query, vertical search typically searches less websites, but with the objective to retrieve relevant data objects. The output of a vertical search query is a result set that contains e.g., the best air fares for a specific route. Vertical search also needs to address the challenge of searching the deep Web, i.e., extracting data by means of automatically utilizing web forms. Data integration in the context of vertical search is both important for interface matching, i.e., merge the source query interfaces and map onto a single query interface, and result data object matching, where data extracted from the individual websites is matched against a single result data model.

Web Intelligence

In Web Intelligence applications, the main objective is to gain new insights from the data extracted on the

Web. Typical application fields are market intelligence, competitive intelligence, and price comparison. Price comparison applications are probably the most well known application type in this field. In a nutshell these applications aggregate data from the Web and integrate different Web data sources according to a single data schema to allow for easy analysis and comparison of the data. Schema matching and data reconciliation are important aspects with this type of applications.

Situational Applications

Situational applications are a new type of application where people with domain knowledge can build an application in a short amount of time without the need to setup an IT project. In the context of the Web, Mashups are addressing these needs. With Mashups, readymade widgets are used to bring together content extracted from multiple websites. Additional value is derived by exploiting the relationship between the different sources, e.g., visualizing the location of offices in a mapping application. In this context, Web data integration is required to reconcile the data extracted from different Web sources and to resolve the references to real world objects.

Cross-references

► Data Integration
► Enterprise Application Integration
► Enterprise Information Integration
► Schema Matching
► Web Data Extraction

Recommended Reading

1. Baumgartner R., Flesca S., and Gottlob G. Visual web information extraction with Lixto. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 119–128.
2. Berglund A., Boag S., Chamberlin D., Rernandez M.F., Kay M., Robie J., and Simeon J. (eds.). XML XPath Language 2.0. W3C Recommendation, 2007.
3. Bernstein P.A., Melnik S., Petropoulos M., and Quix C. Industrial-strength schema matching. SIGMOD Record, 33 (4):38–43, 2004.
4. Bing L. and Chen-Chuan-Chang K. Editorial: special issue on web content mining. ACM SIGKDD Explorations Newsletter, 6(2):1–4, 2004.
5. Boag S., Chamberlin D., Fernandez M.F., Florescu D., Robie J., and Simeon J. (eds.). XQuery 1.0. An XML Query Language. W3C Recommendation, 2007.
6. Fodor O. and Werthner E. Harmonise: a step toward an interoperable e-tourism marketplace. Intl. J. Electron. Commerce, 9(2):11–39, 2005.
7. Gravano L., Panagiotis G.I., Koudas N., and Srivastava D. Text joins in an RDBMS for web data integration. In Proc. WWW Conf. Budapest, Hungary, 2003, pp. 90–101.
8. Halevy A., Rajaraman A., and Ordille J. Data integration: the teenage years. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 9–18.
9. Harmonise Framework. Available at: http://sourceforge.net/projects/hmafra/.
10. Herzog M. and Gottlob G. InfoPipes: a flexible framework for m-commerce applications. In Proc. Second Int. Workshop on Technologies for E-Services. Springer, 2001, pp. 175–186.
11. Kay M. (ed.). XSL Transformations. Version 2.0. W3C Recommendation, 2007.
12. Kirk T., Levy A.Y., Sagiv Y., and Srivastava D. The information manifold. In Proc. Working Notes of the AAAI Spring Symp. on Information Gathering from Heterogeneous, Distributed Environments. Stanford University. AAAI Press, 1995, pp. 85–91.
13. Ludäscher B., Himmeröder R., Lausen G., May W., and Schlepphorst C. Managing semistructured data with florid: a deductive object-oriented perspective. Inf. Syst., 23(9):589–613, 1998.
14. May W. and Lausen G. A uniform framework for integration of information from the web. Inf. Syst., 29:59–91, 2004.
15. Myllymaki J. Effective web data extraction with standard XML technologies. Comput. Networks, 39(5):653–644, 2002.
16. Rahm E. and Bernstein P.A. A survey of approaches to automatics schema matching. VLDB Journal, 10(4):334–350, 2001.
17. Salton G. and McGill M.J. Introduction to Modern Information Retrieval. McGraw-Hill, New York, NY, 1983.

Data Integrity Services

► Security Services

Data Lineage

► Data Provenance

Data Manipulation Language

► Query Language

Data Map

► Thematic Map

Data Mart

Il-Yeol Song
Drexel University, Philadelphia, PA, USA

Definition

A data mart is a small-sized data warehouse focused on a specific subject. While a data warehouse is meant for an entire enterprise, a data mart is built to address the specific analysis needs of a business unit. Hence, a data mart can be defined as "a small-sized data warehouse that contains a subset of the enterprise data warehouse or a limited volume of aggregated data for specific analysis needs of a business unit, rather than the needs of the whole enterprise." Thus, an enterprise usually ends up having many data marts.

Key Points

While a data warehouse is for a whole enterprise, a data mart focuses on a specific subject of a specific business unit. Thus, the design and management of a data warehouse must consider the needs of the whole enterprise, while those of a data mart are focused on the analysis needs of a specific business unit such as the sales department or the finance department. Thus, a data mart shares the characteristics of a data warehouse, such as being subject-oriented, integrated, nonvolatile, and a time-variant collection of data [1]. Since the scope and goal of a data mart is different from a data warehouse, however, there are some important differences between them:

- While the goal of a data warehouse is to address the needs of the whole enterprise, the goal of a data mart is to address the needs of a business unit such as a department.
- While the data of a data warehouse is fed from OLTP (Online Transaction Processing) systems, those of a data mart are fed from the enterprise data warehouse.
- While the granularity of a data warehouse is raw at its OLTP level, that of a data mart is usually lightly aggregated for optimal analysis by the users of the business unit.
- While the coverage of a data warehouse is fully historical to address the needs of the whole enterprise, that of a data mart is limited for the specific needs of a business unit.

An enterprise usually ends up having multiple data marts. Since the data to all the data marts are fed from the enterprise data warehouse, it is very important to maintain the consistency between a data mart and the data warehouse as well as among data marts themselves. A way to maintain the consistency is to use the notion of conformed dimension. A conformed dimension is a standardized dimension or a master reference dimension that are shared across multiple data marts [2].

The technology for advanced data analysis for the business intelligence in the context of data mart environment is called OLAP (Online Analytic Processing). Two OLAP technologies are ROLAP (Relational OLAP) and MOLAP (Multidimensional OLAP). In ROLAP, data are structured in the form a star schema or a dimensional model. In MOLAP, data are structured in the form of multidimensional data cubes. For MOLAP, a specialized OLAP software is used to support the creation of data cubes and OLAP operations such as drill-down and roll-up.

Cross-references

► Active and Real-Time Data Warehousing
► Business Intelligence
► Data Mining
► Data Warehouse
► Data Warehouse Life-Cycle and Design
► Data Warehouse Maintenance
► Data Warehouse Metadata
► Data Warehouse Security
► Data Warehousing and Quality Data Management for Clinical Practice
► Data Warehousing for Clinical Research
► Data Warehousing Systems: Foundations and Architectures
► Dimension
► Evolution and Versioning
► Extraction
► Multidimensional Modeling
► On-Line Analytical Processing
► Transformation and Loading

Recommended Reading

1. Inmon W.H. Building the Data Warehouse, 3rd edn. Wiley, New York, 2002.
2. Kimball R. and Ross M. The Data Warehouse Toolkit, 2nd edn. Wiley, New York, 2002.

Data Migration

► Data Exchange

Data Mining

Jiawei Han
University of Illinois at Urbana-Champaign, Urbana, IL, USA

Synonyms

Knowledge discovery from data; Data analysis; Pattern discovery

Definition

Data mining is the process of discovering knowledge or patterns from massive amounts of data. As a young research field, data mining represents the confluence of a number of research fields, including database systems, machine learning, statistics, pattern recognition, high-performance computing, and specific application fields, such as WWW, multimedia, and bioinformatics, with broad applications. As an interdisciplinary field, data mining has several major research themes based on its mining tasks, including pattern-mining and analysis, classification and predictive modeling, cluster and outlier analysis, and multidimensional (OLAP) analysis. Data mining can also be categorized based on the kinds of data to be analyzed, such as multi-relational data mining, text mining, stream mining, web mining, multimedia (or image, video) mining, spatio-temporal data mining, information network analysis, biological data mining, financial data mining, and so on. It can also be classified based on the mining methodology or the issues to be studied, such as privacy-preserving data mining, parallel and distributed data mining, and visual data mining.

Historical Background

Data mining activities can be traced back to the dawn of early human history when data analysis methods (e.g., statistics and mathematical computation) were needed and developed for finding knowledge from data. As a distinct but interdisciplinary field, knowledge discovery and data mining can be viewed as starting at The First International Workshop on Knowledge Discovery from Data (KDD) in 1989. The first International Conference on Knowledge Discovery and Data Mining (KDD) was held in 1995. Since then, there have been a number of international conferences and several scientific journals dedicated to the field of knowledge discovery and data mining. Many conferences on database systems, machine learning, pattern recognition, statistics, and World-Wide Web have also published influential research results on data mining. There are also many textbooks published on data mining, such as [5,7,8,11,12], or on specific aspects of data mining, such as data cleaning [4] and web mining [2,9]. Recently, there is also a trend to organize dedicated conferences and workshops on mining specific kinds of data or specific issues on data mining, such as International Conferences on Web Search and Data Mining (WSDM).

Foundations

The overall knowledge discovery process usually consists of a few steps, including (i) data preprocessing, e.g., data cleaning and data integration (and possibly building up data warehouse), (ii) data selection, data transformation (and possibly creating data cubes by multidimensional aggregation), and feature extraction, (iii) data mining, (iv) pattern or model evaluation and justification, and (v) knowledge update and application. Data mining is an essential step in the knowledge discovery process.

As a dynamic research field, many scalable and effective methods have been developed for mining patterns and knowledge from an enormous amount of data, which contributes to theories, methods, implementations, and applications of knowledge discovery and data mining. Several major themes are briefly outlined below.

Mining Interesting Patterns from Massive Amount of Data

Frequent patterns are the patterns (e.g., itemsets, subsequences, or substructures) that occur frequently in data sets. This line of research started with association rule mining [1] and has proceeded to mining sequential patterns, substructure (or subgraph) patterns, and their variants. Many scalable mining algorithms have been developed and most of them explore the A priori (or *downward closure*) property of frequent patterns, i.e., *any subpattern of a frequent pattern is frequent.* However, to make discovered patterns truly useful in many applications, it is important to study how to mine interesting frequent patterns [6], e.g., the patterns that satisfy certain constraints, patterns that reflect strong correlation relationships, compressed patterns, and the patterns with certain distinct features. The discovered patterns can also be used for classification, clustering, outlier analysis, feature selection (e.g., for index construction), and semantic annotation.

Scalable Classification and Predictive Modeling

There are many classification methods developed in machine learning [10] and statistics [8], including decision tree induction, rule induction, naive-Bayesian, Bayesian networks, neural networks, support vector machines, regression, and many statistical and pattern analysis methods [5,7,8,11,12]. Recent data mining research has been exploring scalable algorithms for such methods as well as developing new classification methods for handling different kinds of data, such as data streams, text data, web data, multimedia data, and high-dimensional biological data. For example, a pattern-based classification method, called DDPMine [3], that first extracts multidimensional features by discriminative frequent pattern analysis and then performs classification using these features has demonstrated high classification accuracy and efficiency.

Cluster and Outlier Analysis

Data mining research has contributed a great deal to the recent development of scalable and effective cluster analysis methods. New methods have been proposed to make partitioning and hierarchical clustering methods more scalable and effective. For example, the micro-clustering idea in BIRCH [13] has been proposed that first groups objects into tight, micro-clusters based on their inherent similarity, and then performs flexible and efficient clustering on top of a relatively small number of micro-clusters. Moreover, new clustering methodologies, such as density-based clustering, link-based clustering, projection-based clustering of high-dimensional space, user-guided clustering, pattern-based clustering, and (spatial) trajectory clustering methods have been developed and various applications have been explored, such as clustering high-dimensional microarray data sets, image data sets, and interrelated multi-relational data sets. Furthermore, outlier analysis methods have been investigated, which goes beyond typical statistical distribution-based or regression deviation-based outlier analysis, and moves towards distance-based or density-based outlier analysis, local outlier analysis, and trajectory outlier analysis.

Multidimensional (OLAP) Analysis

Each object/event in a dataset usually carries multidimensional information. Mining data in multidimensional space will substantially increase the power and flexibility of data analysis. By integration of data cube and OLAP (online analytical processing) technologies with data mining, the power and flexibility of data analysis can be substantially increased. Data mining research has been moving towards this direction with the proposal of OLAP mining, regression cubes, prediction cubes, and other scalable high-dimensional data analysis methods. Such multidimensional, especially high-dimensional, analysis tools will ensure that data can be analyzed in hierarchical, multidimensional structures efficiently and flexibly at user's finger tips. OLAP mining will substantially enhance the power and flexibility of data analysis and lead to the construction of easy-to-use tools for the analysis of massive data with hierarchical structures in multidimensional space.

Mining Different Kinds of Data

Different data mining methods are often needed for different kinds of data and for various application domains. For example, mining DNA sequences, moving object trajectories, time-series sequences on stock prices, and customer shopping transaction sequences require rather different sequence mining methodology. Therefore, another active research frontier is the development of data- or domain-specific mining methods. This leads to diverse but flourishing research on mining different kinds of data, including multi-relational data, text data, web data, multimedia data, geo-spatial data, temporal data, data streams, information networks, biological data, financial data, and science and engineering data.

Key Applications

Data mining claims a very broad spectrum of applications since in almost every domain, there is a need for scalable and effective methods and tools to analyze massive amounts of data. Two applications are illustrated here as examples:

Biological Data Mining

The fast progress of biomedical and bioinformatics research has led to the accumulation of an enormous amount of biological and bioinformatics data. However, the analysis of such data poses much greater challenges than traditional data analysis methods. For example, genes and proteins are gigantic in size (e.g., a

DNA sequence could be in billions of base pairs), very sophisticated in function, and the patterns of their interactions are largely unknown. Thus it is a fertile field to develop sophisticated data mining methods for in-depth bioinformatics research. Substantial research is badly needed to produce powerful mining tools for biology and bioinformatics studies, including comparative genomics, evolution and phylogeny, biological data cleaning and integration, biological sequence analysis, biological network analysis, biological image analysis, biological literature analysis (e.g., PubMed), and systems biology. From this point view, data mining is still very young with respect to biology and bioinformatics applications.

Data Mining for Software Engineering

Software program executions potentially (e.g., when program execution traces are turned on) generate huge amounts of data. However, such data sets are rather different from the datasets generated from the nature or collected from video cameras since they represent the executions of program logics coded by human programmers. It is important to mine such data to monitor program execution status, improve system performance, isolate software bugs, detect software plagiarism, analyze programming system faults, and recognize system malfunctions.

Data mining for software engineering can be partitioned into static analysis and dynamic/stream analysis, based on whether the system can collect traces beforehand for post-analysis or it must react at real time to handle online data. Different methods have been developed in this domain by integration and extension of the methods developed in machine learning, data mining, pattern recognition, and statistics. For example, statistical analysis such as hypothesis testing approach can be performed on program execution traces to isolate the locations of bugs which distinguish program success runs from failing runs. Despite of its limited success, it is still a rich domain for data miners to research and further develop sophisticated, scalable, and real-time data mining methods.

Future Directions

There are many challenging issues to be researched further, and therefore, there are great many research frontiers in data mining. Besides the mining of biological data and software engineering data, as well as the above introduced advanced mining methodologies, a few more research directions are listed here.

Mining Information Networks

Information network analysis has become an important research frontier, with broad applications, such as social network analysis, web community discovery, computer network analysis, and network intrusion detection. However, information network research should go beyond explicitly formed, homogeneous networks (e.g., web page links, computer networks, and terrorist e-connection networks) and delve deeply into implicitly formed, heterogeneous, dynamic, interdependent, and multidimensional information networks, such as gene and protein networks in biology, highway transportation networks in civil engineering, theme-author-publication-citation networks in digital libraries, wireless telecommunication networks among commanders, soldiers and supply lines in a battle field.

Invisible Data Mining

It is important to build data mining functions as an invisible process in many systems (e.g., rank search results based on the relevance and some sophisticated, preprocessed evaluation functions) so that users may not even sense that data mining has been performed beforehand or is being performed and their browsing and mouse clicking are simply using the results of or further exploration of data mining. Google has done excellent invisible data mining work for web search and certain web analysis. It is highly desirable to introduce such functionality to many other systems.

Privacy-Preserving Data Mining

Due to the security and privacy concerns, it is appealing to perform effective data mining without disclosure of private or sensitive information to outsiders. Much research has contributed to this theme and it is expected that more work in this direction will lead to powerful as well as secure data mining methods.

Experimental Results

There are many experimental results reported in numerous conference proceedings and journals.

Data Sets

There are many, many data sets (mostly accessible on the web) that can be or are being used for data mining.

University of California at Irvine has an online repository of large data sets which encompasses a wide variety of data types, analysis tasks, and application areas. The website of UCI Knowledge Discovery in Databases Archive is http://kdd.ics.uci.edu.

Researchers and practitioners should work on real data sets as much as possible to generate data mining tools for real applications.

URL to Code

Weka (http://www.cs.waikato.ac.nz/ml/weka) presents a collection of machine learning and data mining algorithms for solving real-world data mining problems.

RapidMiner (http://rapid-i.com), which was previously called YALE (Yet Another Learning Environment), is a free open-source software for knowledge discovery, data mining, and machine learning.

IlliMine (IlliMine.cs.uiuc.edu) is a collection of data mining software derived from the research of the Computer Science department at the University of Illinois at Urbana-Champaign.

For frequent pattern mining, the organizers of the FIMI (Frequent Itemset Mining Implementations) workshops provides a repository for frequent itemset mining implementations at http://fimi.cs.helsinki.fi.

There are many other websites providing source or object codes on data mining.

Cross-references

► Association Rules
► Bayesian Classification
► Classification
► Classification by Association Rule Analysis
► Clustering Overview and Application/Partitional Clustering Algorithms
► Data, Text, and Web Mining in Healthcare
► Decision Rule Mining in Rough Set Theory
► Decision Tree Classification
► Decision Trees
► Dimentionality reduction
► Event pattern detection
► Event Prediction
► Exploratory data analysis
► Frequent graph patterns
► Frequent itemset mining with constraints
► Frequent itemsets and association rules
► Machine learning in Computational Biology
► Mining of Chemical Data
► Opinion mining
► Pattern-growth methods
► Privacy-preserving data mining
► Process mining
► Semi-supervised Learning
► Sequential patterns
► Spatial Data Mining
► Spatio-temporal Data Mining
► Stream Mining
► Temporal Data Mining
► Text Mining
► Text mining of biological resources
► Visual Association Rules
► Visual Classification
► Visual Clustering
► Visual Data Mining

Recommended Reading

1. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
2. Chakrabarti S. 1Mining the Web: Statistical Analysis of Hypertex and Semi-Structured Data. Morgan Kaufmann, Los Altos, CA, 2002.
3. Cheng H., Yan X., Han J., and Yu P.S. Direct discriminative pattern mining for effective classification. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 169–178.
4. Dasu T. and Johnson T. Exploratory Data Mining and Data Cleaning. Wiley, New York, 2003.
5. Duda R.O., Hart P.E., and Stork D.G. Pattern Classification, 2nd edn. Wiley, New York, 2001.
6. Han J., Cheng H., Xin D., and Yan X. Frequent pattern mining: Current status and future directions. Data Min. Knowl. Disc., 15:55–86, 2007.
7. Han J. and Kamber M. Data Mining: Concepts and Techniques, 2nd edn. Morgan Kaufmann, Los Altos, CA, 2006.
8. Hastie T., Tibshirani R., and Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2001.
9. Liu B. Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Springer, 2006.
10. Mitchell T.M. Machine Learning. McGraw-Hill, New York, 1997.
11. Tan P., Steinbach M., and Kumar V. Introduction to Data Mining. Addison Wesley, 2005.
12. Witten I.H. and Frank E. Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, 2005.
13. Zhang T., Ramakrishnan R., and Livny M. BIRCH: an efficient data clustering method for very large databases. In Proc. ACM-SIGMOD Int. Conf. on Management of Data, 1996, pp. 103–114.

Data Mining in Bioinformatics

► Machine Learning in Computational Biology

Data Mining in Computational Biology

► Machine Learning in Computational Biology

Data Mining in Moving Objects Databases

► Spatio-Temporal Data Mining

Data Mining in Systems Biology

► Machine Learning in Computational Biology

Data Mining Pipeline

► KDD Pipeline

Data Mining Process

► KDD Pipeline

Data Model Mapping

► Logical Database Design: From Conceptual to Logical Schema

Data Organization

► Indexing and Similarity Search

Data Partitioning

Daniel Abadi
Yale University, New Haven, CT, USA

Definition

Data Partitioning is the technique of distributing data across multiple tables, disks, or sites in order to improve query processing performance or increase database manageability. Query processing performance can be improved in one of two ways. First, depending on how the data is partitioned, in some cases it can be determined *a priori* that a partition does not have to be accessed to process the query. Second, when data is partitioned across multiple disks or sites, *I/O parallelism* and in some cases *query parallelism* can be attained as different partitions can be accessed in parallel. Data partitioning improves database manageability by optionally allowing backup or recovery operations to be done on partition subsets rather than on the complete database, and can facilitate loading operations into rolling windows of historical data by allowing individual partitions to be added or dropped in a single operation, leaving other data untouched.

Key Points

There are two dominant approaches to data partitioning.

Horizontal partitioning divides a database table tuple-by-tuple, allocating different tuples to different partitions. This is typically done using one of five techniques:

1. *Hash partitioning* allocates tuples to partitions by applying a hash function to an attribute value (or multiple attribute values) within the tuple. Tuples with equivalent hash function values get allocated to the same partition.
2. *Range partitioning* allocates tuples to partitions by using ranges of attribute values as the partitioning criteria. For example, tuples from a customer table with last name attribute beginning with "A"–"C" are mapped to partition 1, "D"–"F" mapped to partition 2, etc.
3. *List partitioning* allocates tuples to partitions by associating a list of attribute values with each partition. Using range or list partitioning, it can be difficult to ensure that each partition contains approximately the same number of tuples.
4. *Round-robin partitioning* allocates the ith tuple from a table to the $(i \ mod \ n)^{th}$ partition where n is the total number of partitions.
5. *Composite partitioning* combines several of the above techniques, typically range partitioning followed by hash partitioning.

Vertical partitioning divides a table column-by-column, allocating different columns (or sets of columns) to different partitions. This approach is less frequently used relative to horizontal partitioning since it is harder to parallelize query processing over multiple vertical partitions, and merging or joining partitions is often necessary at query time. *Column-stores* are databases that specialize in vertical partitioning, usually taking the approach to the extreme, storing each column separately.

Cross-references

► Horizontally Partitioned Data
► Parallel Query Processing

Data Pedigree

► Data Provenance

Data Perturbation

► Matrix Masking

Data Privacy and Patient Consent

David Hansen[1], Christine M. O'Keefe[2]
[1]The Australian e-Health Research Centre, Brisbane, QLD, Australia
[2]CSIRO Preventative Health National Research Flagship, Acton, ACT, Australia

Synonyms

Data protection

Definition

Data privacy refers to the interest individuals and organisations have in the collection, sharing, use and disclosure of information about those individuals or organizations. Common information types raising data privacy issues include health (especially genetic), criminal justice, financial, and location. The recent rapid growth of electronic data archives and associated data technologies has increased the importance of data privacy issues, and has led to a growing body of legislation and codes of practice.

Patient consent, in relation to data, refers to a patient's act of approving the collection, sharing, use or disclosure of information about them. It is important because data with appropriate patient consent often falls into exception clauses in privacy legislation.

The challenge in data privacy is to balance the need to share and use data with the need to protect personally identifiable information and respect patient consent. For example, a person may have an expectation that their health data is being held securely but is available to clinicians involved in their care. In addition, researchers seek access to health data for medical research and to answer questions of clinical and policy relevance. Part of the challenge is that there are a range of views about, for example, what constitutes legitimate uses of data, what level of consent is required and how fine-grained that consent should be, what constitutes a clinical "care team," and what privacy legislation should cover.

The development of technologies to support data privacy and patient consent is currently attracting much attention internationally.

Historical Background

Privacy issues began to attract attention in Europe and North America in the 1960s, and shortly thereafter in Australia. Probably a large factor in the relatively late recognition of privacy as a fundamental right is that most modern invasions of privacy involve new technology. For example, before the invention of computer databases, data were stored on paper in filing cabinets which made it difficult to find and use the information. The traditional ways of addressing the harm caused by invasions of privacy through invoking trespass, assault or eavesdropping were no longer sufficient in dealing with invasions of privacy enacted with modern information technologies.

In the health care area, the notion of consent arose in the context of both treatment and clinical trials in research into new treatments. Patients undergoing a procedure or treatment would either be assumed to have given implicit consent by their cooperation, or would be required to sign an explicit statement of consent. Participants in clinical trials would be asked to give a formal consent to a trial, considered necessary because of the risk of harm to the individual due to the unknown effects of the intervention under research. The notion of consent has transferred to the context of the (primary) use of data for treatment and the (secondary) use of data for research. For example, implied or expressed consent would be required for the transfer of medical records to a specialist or may be required for the inclusion of data in a research database or register. Increasingly, and somewhat controversially, health data are being made available for research without patient consent, but under strict legal and ethical provisions. Where this is the case, ethics committees are given the responsibility to decide if the research benefits outweigh to a substantial degree the public interest in protecting privacy. Ethics committees will generally look for de-identified data to be used wherever practical.

A recent development related to the privacy of health data is the concept of participation or moral rights (http://www.privireal.org/). This can be viewed as an objection to the use of personal information on moral grounds; that individuals should have the right to know, and veto, how their data is used, even when there is no risk to the release of that personal information.

The major data privacy-related legislative provisions internationally are Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data, the Health Insurance Portability and Accountability Act (HIPAA) enacted by the US Congress in 1996, the Data Protection Act 1998 of the UK Parliament and the Australian Privacy Act 1988. There are apparent differences between these provisions in terms of both their scope and philosophy.

Recent technological approaches to data privacy include: tight governance and research approvals processes, restricted access through physical and IT security, access control mechanisms, de-identification, statistical disclosure limitation and remote server technology for remote access and remote execution. Patient consent is often built into the approvals process, but can also be a component of the access control mechanism. In all cases the purpose of access to the data is of prime importance.

Foundations

The use of sensitive personal data falls broadly into two areas: primary and secondary use.

Primary use of data refers to the primary reason the data was captured. Increasing amounts of personal health data are being stored in databases for the purpose of maintaining a life long health record of an individual. While in most cases this will provide for more appropriate care, there will be times when the individual will not want sensitive data revealed, even to a treating clinician. This may particularly be the case when the data is judged not relevant to the current medical condition.

In some electronic health record programs, patients will have the option to opt-in or opt-out of the program. This can be a controversial aspect of health record systems and often legislation is needed to support this aspect of any collection of health related data.

Once the patient data has been entered into an electronic health record system, there will generally be several layers of security protecting the data. Authentication is often performed using a two pass system – a password and a certificate are required to enter the system, to ensure that the person accessing the system is correctly authenticated. Once the person is authenticated access to appropriate data must be managed. Often a Role Based Access System (RBAC) [5] is implemented to ensure that access to the data is only granted to people who should have it. There are also now XML based mark up languages, such as XACML, which enable Role Based Access Control rules to be encoded using the language and hence shared by multiple systems. Audit of access to electronically stored health data is also important, to enable review access granted to data.

Storage and transmission of data is also an issue for privacy and confidentiality of patient data.

Encryption of the data when being transmitted is one way of ensuring the security of the data. There are also new computer systems which support Mandatory Access Control (MAC) which embed the security algorithms in the computer hardware, rather than at the software level, which are now available for the storage and access of electronic data. Security Assertion Markup Language (SAML) token are one technology which is being used to store privacy and security information with health data as they are transmitted between computers.

The USA HIPAA legislation covers the requirements of how to capture, store and transmit demographic and clinical data.

The secondary use of data is more problematic when it comes to data privacy and patient consent, since it refers to usage of the data for purposes which are not directly related to the purpose for which it was collected. This includes purposes like medical research and policy analysis, which are unlikely to have a direct affect on the treatment of the patient whose data is used.

There are a range of technological approaches to the problem of enabling the use of data for research and policy analysis while protecting privacy and confidentiality. None of these technologies provides a complete answer, for each must be implemented within an appropriate legislative and policy environment and governance structure, with appropriate management of the community of authorised users and with an appropriate level of IT security including user authentication, access control, system audit and follow-up. In addition, none of the technologies discussed here is the only solution to the problem, since there are many different scenarios for the use of data, each with a different set of requirements. It is clear that different technologies and approaches have different strengths and weaknesses, and so are suitable for different scenarios.

A high level discussion of the problem of enabling the use of data while protecting privacy and confidentiality typically discusses two broad approaches. The first is restricted access, where access is only provided to approved individuals and for approved purposes. Further restrictions can be imposed, such as access to data only given at a restricted data centre, restrictions on the types of analyses which can be conducted and restrictions on the types of outputs which can be taken out of a data centre. There can be a cost associated with access to the data. The second is restricted or altered data, where something less than the full data set is published or the data are altered in some way before publication. Restricted data might involve removing attributes, aggregating geographic classifications or aggregating small groups of data. For altered data, some technique is applied to the data so that the released dataset does not reveal private or confidential information. Common examples here include the addition of noise, data swapping or the release of synthetic data. Often these broad approaches are used in combination.

Below, three current technological approaches to the problem are reviewed. These fall into the category restricted or altered data described above, and all are used in combination with restricted access.

The first approach is to release de-identified data to researchers under strict controls. De-identification is a very complex issue surrounded by some lack of clarity and standard terminology. It is also very important as it underpins many health information privacy guidelines and legislation.

First, it is often not at all clear what is meant when the term "de-identified" is used to refer to data. Sometimes it appears to mean simply that nominated identifiers such as name, address, date of birth and unique identifying numbers have been removed from the data. At other times its use appears to imply that individuals represented in a data set cannot be identified from the data – though in turn it can be unclear what this means. Of course simply removing nominated identifiers is often insufficient to ensure that individuals represented in a data set cannot be identified – it can be a straightforward matter to match some of the available data fields with the corresponding fields from external data sets, and thereby obtain enough information to determine individuals' names either uniquely or with a low uncertainty. In addition, sufficiently unusual records in a database without nominated identifiers can sometimes be recognized. This is particularly true of health information or of information which contains times and/or dates of events.

The second approach is statistical disclosure control where the techniques aim to provide researchers with useful statistical data at the same time as preserving privacy and confidentiality.

It is widely recognized that any release of data or statistical summaries increases the risk of identification of some individual in the relevant population, with the consequent risk of harm to that individual through inference of private information about them. On the other hand, attempts to limit such disclosures can adversely affect the outcomes or usefulness of statistical analyses conducted on the data. Statistical disclosure control theory attempts to find a balance between these opposing objectives.

Statistical disclosure control techniques can be organized into categories in several different ways. First, there are different techniques for tabular data (where data are aggregated into cells) versus microdata (individual level data). Second, techniques can be perturbative or non-perturbative. Perturbative methods operate by modifying the data, whereas non-perturbative methods do not modify the data. Perhaps the most well-known perturbative method is the addition of random "noise" to a dataset, and perhaps the most well-known non-perturbative method is cell suppression. In fact, current non-perturbative methods operate by suppressing or reducing the amount of information released, and there is much ongoing debate on whether a good perturbative method gives more useful information than a non-perturbative method. On the other hand, it has been noted that perturbative techniques which involve adding noise provide weak protection and are vulnerable to repeated queries, essentially because the noise becomes error in models of the data. There is much activity directed at developing perturbative techniques that do not suffer from this problem.

Virtually every statistical disclosure control technique can be implemented with differing degrees of intensity, and hence depends on a parameter which is usually pre-specified.

Remote analysis servers are designed to deliver useful results of user-specified statistical analyses with acceptably low risk of a breach of privacy and confidentiality.

The third approach is the technology of remote analysis servers. Such servers do not provide data to users, but rather allow statistical analysis to be carried out via a remote server. A user submits statistical queries by some means, analyses are carried out on the original data in a secure environment, and the user then receives the results of the analyses. In some cases the output is designed so that it does not reveal private information about the individuals in the database.

The approach has several advantages. First, no information is lost through confidentialization, and there is no need for special analysis techniques to deal with perturbed data. In many cases it is found to be easier to confidentialize the output of an analysis, in comparison to trying to confidentialize a dataset when it is not known which analyses will be performed.

However, analysis servers are not free from the risk of disclosure, especially in the face of multiple, interacting queries. They describe the risks and propose quantifiable measures of risk and data utility that can be used to specify which queries can be answered and with what output. The risk-utility framework is illustrated for regression models.

Each of the broad technologies is implemented within the context that the analyst is trusted to comply with legal and ethical undertakings made. However, the different approaches have been designed with different risks of disclosure of private information, and so rely more or less heavily on trust. De-identification requires the greatest trust in the researcher, while remote servers require the least. Statistical disclosure control, whether used alone or in combination with a remote analysis server, is somewhere inbetween these two extremes. De-identification provides the most detailed information to the researcher, while remote servers provide the least. Again, Statistical Disclosure Control is inbetween.

These mechanisms of maintaining data privacy are made more difficult when it is necessary to link two or more databases together for the purpose of building a linked data set for analysis. This sort of research is increasingly being used as a way of reducing the cost of collecting new data and to make better use of existing data. The difficulties are two fold. First, there is the linkage step, i.e. recognizing that patients are the same across the data sets. Recent work has included blindfolded linkage methodologies [1,2] and encryption techniques [4] as ways of linking datasets while not revealing any patient information. The second difficulty lies in the greater chance of revealing information from a linked data set than a single data set, especially when combining data from multiple modalities, e.g. health data and geographic data.

Key Applications

As discussed above, data privacy and patient consent has an impact on large sections of the health care industry and the biomedical research community. There are many applications which will need to consider data privacy and patient consent issues. Below is a discussion of Electronic Health Records, possibly the fastest growing example of data collected for primary use, and medical research, often the largest source of requests for data for secondary use.

Electronic Health Records

Electronic Health Records (EHR) are the fastest growing example of an application which concern data privacy and patient consent. Increasing amounts of personal health data are being stored in databases for the purpose of maintaining a life long health record of an individual.

The data is being stored according to a number of different international and local standards and development for the format of the data, for example openEHR (http://www.openehr.org/), and the transmission of the data, such as HL7 (http://www.hl7.org/). Some of this data is stored using codes from clinical terminologies, while some of it will be free text reports. This data are being stored so that the data are available to clinicians for the purpose of treating the individual.

While in most cases this will provide more appropriate care, there will be times when the individual will not want sensitive data revealed, even to a treating clinician. This may particularly be the case when the data is judged not relevant to the current medical condition. With a number of countries introducing Electronic Health Records, there are concerns over who will have access to the data. Generally these are governed by strict privacy policies, as well as allowing patients the opportunity to have some level of control over whether data is added to the EHR or not.

Medical Research

Secondary use of data is primarily concerned with providing health data for clinical or medical research. For most secondary data use, it is possible to use de-identified data, as described above. Increasingly, secondary use of data involves the linkage of data sets to bring different modalities of data together, which raises more concerns over the privacy of the data as described above. The publication of the Human Genome gave rise to new ways of finding relationships between clinical disease and human genetics. The increasing use and storage of genetic information also impacts the use of familial records, since the information about the patient also provides information on the patient's relatives. The issues of data privacy and patient confidentiality and the use of the data for medical research are made more difficult in this post-genomic age.

Cross-references

► Access Control
► Anonymity
► Electronic Health Records
► Exploratory Data Analysis
► Health Informatics Databases
► Privacy Policies and Preferences
► Privacy-Enhancing Technologies
► Privacy-Preserving data mining
► Record Linkage

Recommended Reading

1. Agrawal R., Evfimievski A., and Srikant R. Information sharing across private databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 86–97.
2. Churches T. and Christen P. Some methods for blindfolded record linkage. BMC Med. Inform. Decis. Making, 4:9, 2004.
3. Domingo-Ferrer J. and Torra V. (eds.). Privacy in Statistical Databases. Lect. Notes Comput. Sci., Vol 3050. Springer Berlin Heidelberg, 2004.
4. OKeefe C.M., Yung M., and Baxter R. Privacy-preserving linkage and data extraction protocols. In Workshop on Privacy in the Electronic Society in conjunction with the 11th ACM CCS Conference, 2004.
5. Sandhu R.S., Coyne E.J., Feinstein H.L., and Youman C.E. Role-based access control models. IEEE Comput., 29(2):38–47, 1996.

Data Problems

► Data Conflicts

Data Profiling

THEODORE JOHNSON
AT&T Labs – Research, Florham, Park, NJ, USA

Synonyms

Database profiling

Definition

Data profiling refers to the activity of creating small but informative summaries of a database [5]. These summaries range from simple statistics such as the number of records in a table and the number of distinct values of a field, to more complex statistics such as the distribution of n-grams in the field text, to structural properties such as keys and functional dependencies. Database profiles are useful for database exploration, detection of data quality problems [4], and for schema matching in data integration [5]. Database exploration helps a user identify important database properties, whether it is data of interest or data quality problems. Schema matching addresses the critical question, "do two fields or sets of fields or tables represent the same information?" Answers to these questions are very useful for designing data integration scripts.

Historical Background

Databases which support a complex organization tend to be quite complex also. Quite often, documentation and metadata are incomplete and outdated, no DBA understands the entire system, and the actual data fails to match documented or expected properties [2]. These problems greatly complicate already difficult tasks such as database migration and integration, and in fact database profiling was originally developed for their support. Newer developments in database profiling support database exploration, and finding and diagnosing data quality problems.

Foundations

A *database profile* is a collection of summaries about the contents of a database. These summaries are usually collected by making a scan of the database (some profiles use sampled data, and some require multiple complex queries). Many of the profile statistics are collected by the DBMS for query optimization. If the optimizer statistics are available, they can be retrieved instead of calculated – though one must ensure that the optimizer's statistics are current before using them. Profile summaries are typically stored in a database for fast interactive retrieval.

Basic Statistics

These statistics include schema information (table and field names, field types, etc.) and various types of counts, such as the number of records in a table, and the number of special values of a field (typically, the number of null values).

Distributional Statistics

These statistics summarize the *frequency distribution* of field values: for each distinct value of a field, how often does it occur. Examples include the number of distinct values of a field, the entropy of the frequency distribution [10], and the most common values and their frequency of occurrence. Another summary is the *inverse frequency distribution*, which is the distribution of the frequency distribution (e.g., the number of distinct values which occur once, occur twice, and so on). While the inverse frequency distribution is often small, it can become large and in general needs to be summarized also.

Textual Summaries

A textual summary represents the nature of the data in a field. These summaries are very useful for exploring the value and pattern distributions and for field content matching in a schema matching task, i.e., to determine whether or not two fields across tables or databases represent similar content.

Textual summaries apply to fields with numeric as well as string data types. The issue is that many identifiers are numeric, such as telephone numbers, Social Security numbers, IP addresses, and so on. These numeric identifiers might be stored as numeric or string fields in a given table – or even combined with other string fields. To ensure that field matches can be made in these cases, numeric fields should be converted to their string representation for textual matching. Patterns (say, regular expressions) which most field values conform to are very useful in identifying anomalies and data quality issues.

Minhash Signatures: One type of summary is very useful in this regard, the *minhash signature* [1]. To compute a minhash signature of a field, one starts with N hash functions from the field domain to the integers. For each hash function, compute the hash value of each field value, and collect the minimum. The collection of minimum hash values for each hash function constitutes the minhash signature.

For example, suppose our set consists of $X = \{3,7,13,15\}$ and our two hash functions are $h_1(x) = x \bmod 10$, and $h_2(x) = x \bmod 5$ (these are simple but very poor hash functions). Then, $\min\{h_1(x) \mid x \text{ in } X\} = 3$, and $\min\{h_2(x) \mid x \text{ in } X\} = 2$. Therefore the minhash signature of X is $\{3,2\}$.

A surprising property of the minhash signature is its ability to determine the intersection of two sets. Given two minhash signatures for sets A and B, the number of hash functions with the same minimum value divided by the number of hash functions is an estimator for the *resemblance* of two sets, which is the size of the intersection of A and B divided by the union of A and B ($\rho = |A \cap B|/|A \cup B|$). Given knowledge of $|A|$ and $|B|$ (from the distributional statistics), an estimate of the size of the intersection is $|A \cap B| = \rho(|A| + |B|)/(1 + \rho)$.

If one extends the minhash signature to include the number of times that the minimum value of a hash function occurred, one can summarize the tail of the inverse frequency distribution [3]. Augmenting the minhash signature summary with the counts of the most frequent values (from the distributional statistics), which constitute the head, completes the summary.

Substring summaries: Another type of textual summary determines if two fields have textually similar information, i.e., many common substrings. As with approximate string matching [6], these summaries rely on *q-grams* – all consecutive *q*-letter sequences in the field values. One type of approximate textual summary collects the distribution of all q-grams within a field's values, and summarizes this distribution using a *sketch* such as the minhash signature or the min-count sketch [3]. Two fields are estimated to be textually similar if their q-gram distributions, represented by the sketches, are similar.

Structural Summaries

Some summaries represent patterns among fields in a table. The two most common examples are *keys* and *functional dependencies* (FDs). Since a table might be corrupted by data quality problems, another type of structural summary are *approximate* keys and *approximate FDs* [7], which hold for most (e.g., 98%) of the records in a table.

Key and FD finding is a very expensive procedure. Verifying whether or not a set X of fields is a key can be performed using a count distinct query on X, which returns the number of distinct values of X. Now, X is an (approximate) key if the number of distinct values of X is (approximately) equal to the size of the table. And, an FD X → Y (approximately) holds if the number of distinct values of X is (approximately) equal to the number of distinct values of (X U Y). An exhaustive search of a d field table requires 2^d expensive count-distinct operations on the table. There are several ways to reduce this cost. For one, keys and FDs with a small number of fields are more interesting than ones with a large number of fields (large keys and FDs are likely spurious – because of the limited size of the table – and likely do not indicate structural properties of the data). If the maximum set of fields is k (e.g., $k = 3$), then the search space is limited to $O(d^k)$. The search space can be trimmed further by searching for *minimal* keys and functional dependencies [7]. Finally, one can hash string fields to reduce the cost of computing count distinct queries over them (if exact keys and FD are required, a candidate key or FD found using hashing must be verified by a query over the actual table).

Samples

A random sample of table records also serves as summary of the contents of a table. A few sampled rows of a table are surprisingly informative, and can be used to estimate a variety of distributions, e.g., identifying the most frequent values or patterns and their frequencies in a field.

Sampling can be used to accelerate the expensive computation of profile data. For example, when computing keys and FDs on a large table (e.g., one with many records), one can sample the table and compute keys and FDs over the sample. If a key or FD is (approximately) valid over the base table then it is also valid on a sample. But, it is possible that a key or FD which is valid on the sample may not be a valid on the base table. Therefore, a random sample can be used to identify candidate keys and FDs. If exact keys and FDs are needed, candidates can be verified by queries over the actual table.

A minhash signature can be computed over sampled data. Suppose that F and G are again keys with identical sets of strings and of size S, and that one computes minhash signatures over F' and G', which are sampled from F and G (respectively) at rate p. Then, the resemblance of F' and G' is

$$\begin{aligned}\rho' &= |F' \cap G'|/(|F'| + |G'| - |F' \cap G'|)\\ &= p^2 S/(2pS - p^2 S) = p/(2-p) \approx p/2\end{aligned}$$

While the resemblance decreases linearly with p (and experiences a larger decrease than the sample intersection), minhash signatures have the advantage of being small. A profiling system which collects minhash signatures might use a signature size of, e.g., 250 hashes – small enough that an exhaustive search for matching fields can be performed in real time. Very large tables are sampled to accelerate the computation of the minhash signatures, say $p = .1$. When comparing two fields which both represent identical key values, there will be about 13 matches on average – enough to provide a reliable signal. In contrast, small uniform random samples of similar sizes drawn from the two fields may not provide accurate estimates of the resemblance. For instance, collecting 250 samples from a table with 1,000,000 rows requires a sampling rate of $p = 0.00025$, meaning that the intersection of the samples of F and G is very likely to be empty.

While random sampling is a common data synopsis used to estimate a wide variety of data properties, its use as a database profile is limited. For one, a random sample cannot always provide an accurate estimation of the number of distinct values in a field, or of the frequency distribution of a field. Table samples are also ineffective for computing the size of the intersection of fields. Suppose that fields F and G are keys and contain the same set of strings, and suppose that they are sampled at rate p. Then, the size of the intersection of the sample is $p^2|F| = p^2|G|$. One can detect that F and G are identical if the size of the intersected sample is p times the size of the samples. However, if p is small enough for exhaustive matching (e.g., $p = 0.001$), then $p^2|F|$ is likely to be very small – and therefore an unreliable indicator of a field match.

Implementation Considerations

Profiling a very large database can be a time-consuming and computationally intensive procedure. A given DBMS might have features, such as sampling, grouping sets, stored procedures, user-defined aggregate functions, etc., which can accelerate the computation of various summaries. Many profile statistics are computed by the DBMS for the query optimizer, and might be made available to users.

However, database profiles are often used to compare data from different databases. Each of these databases likely belongs to its own administrative domain, which will enable or disable features depending on the DBA's needs and preferences. Different databases often reside in different DBMSs. Therefore, a profiling tool which enables cross-database comparisons must in general make use of generic DBMS facilities, making use of DBMS-specific features as an optimization only.

Modes of Use

The types of activities supported by database profiles can be roughly categorized into *database exploration* and *schema matching*. Database exploration means to help a user identify important database properties, whether it is data of interest, data quality problems, or properties that can be exploited to optimize database performance. For example, the user might want to know which are the important tables in a database, and how do they relate (how can they be joined). The number of records in a table is a good first indicator of the importance of a table, and a sample of records is a good first indicator of the kind of data in the table. The most frequent values of a field will often indicate the field's default values (often there is more than one). Other types of information, e.g., keys and field resemblance, help to identify join paths and intra-table relationships.

By collecting a sequence of historical snapshots of database profiles, one can extract information about how the database is updated. A comparison of profiles can indicate which tables are updated, which fields tend to change values, and even reveal changes in database maintenance procedures [3]. For example, in the two large production databases studied in [3], only 20–40% of the tables in the database changed at all from week to week. Furthermore, most of the tables which ever changed experienced only a small change. Only 13 of the 800 + tables were found to be dynamic.

A schema matching activity asks the question, "do these two instances represent the same thing?" – fields, sets of fields, tables, etc. For example, textual summaries are designed to help determine if two fields have the same (or nearly the same) contents. However, any single type of information (schema, textual, distributional) can fail or give misleading results in a large number of cases. The best approach is to use all available information [11].

Key Applications

Data profiling techniques and tools have been developed for database exploration, data quality exploration, database migration, and schema matching. Systems and products include Bellman [4], Ascential [8] and Informatica [9].

Cross-references

► Count-Min Sketch
► Data Sketch/Synopsis
► Hash Functions

Recommended Reading

1. Broder A. On the resemblance and containment of documents. In Proc. IEEE Conf. on Compression and Comparison of Sequences, IEEE Computer Society, 1997, pp. 21–29.
2. Dasu T. and Johnson T. Exploratory Data Mining and Data Cleaning. Wiley Interscience, New York, 2003.
3. Dasu T., Johnson T., and Marathe A. Database exploration using database dynamics. IEEE Data Eng. Bull. 29(2):43–59, 2006.
4. Dasu T., Johnson T., Muthukrishnan S., and Shkapenyuk V. Mining database structure; or, how to build a data quality browser. In Proc. ACM SIGMOD Int. Conf. on Management of data, 2002, pp. 240–251.
5. Evoke Software. Data Profiling and Mapping, The Essential First Step in Data Migration and Integration Projects. Available at: http://www.evokesoftware.com/pdf/wtpprDPM.pdf 2000.
6. Gravano L., Ipeirotis P.G., Jagadish H.V., Koudas N., Muthukrishnan S., and Srivastava D. Approximate String Joins in a Database (Almost) for Free. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 491–500.
7. Huhtala Y., Karkkainen J., Porkka P., and Toivonen H. TANE: an efficient algorithm for discovering functional and approximate dependencies. Comp. J., 42(2):100–111, 1999.
8. IBM Websphere Information Integration. Available at: http://ibm.ascential.com
9. Informatica Data Explorer. Available at: http://www.informatica.com/products_services/data_explorer
10. Kang J. and Naughton J.F. On schema matching with opaque column names and data values. In Proc. ACM SIGMOD Int.

Conf. on Management of Data, San Diego, CA, 2003, pp. 205–216.
11. Shen W., DeRose P., Vu L., Doan A.H., and Ramakrishnan R. Source-aware entity matching: a compositional approach. In Proc. 23rd Int. Conf. on Data Engineering, pp. 196–205.

Data Protection

► Data Privacy and Patient Consent
► Storage Protection

Data Provenance

Amarnath Gupta
University of California San Diego, La Jolla, CA, USA

Synonyms

Provenance metadata; Data lineage; Data tracking; Data pedigree

Definition

The term “data provenance” refers to a record trail that accounts for the origin of a piece of data (in a database, document or repository) together with an explanation of how and why it got to the present place.

Example: In an application like Molecular Biology, a lot of data is derived from public databases, which in turn might be derived from papers but after some transformations (only the most significant data were put in the public database), which are derived from experimental observations. A provenance record will keep this history for each piece of data.

Key Points

Databases today do not have a good way of managing provenance data and the subject is an active research area. One category of provenance research focuses on the case where one database derives some of its data by querying another database, and one may try to “invert” the query to determine which input data elements contribute to *this data element*. A different approach is to explicitly add annotations to data elements to capture the provenance. A related issue is to keep process provenance, especially in business applications, where an instrumented business process capturing software is used to track the data generation and transformation life cycle. While keeping a trail of provenance data is beneficial for many applications, storing, managing and searching provenance data introduces an overhead.

Cross-references

► Annotation
► Provenance

Recommended Reading

1. Bose R. and Frew J. Lineage retrieval for scientific data processing: a survey. ACM Comput. Surv., 37(1):1–28, 2005.
2. Buneman P., Khanna S., Tajima K., and Tan W.-C. Archiving scientific data. In Proc. ACM SIGMOD Conf. on Management of Data, 2002, pp. 1–12
3. Buneman P., Khanna S., and Tan W.C. On propagation of deletions and annotations through views. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2002, pp. 150–158.
4. Simmhan Y.L., Plale B., and Gannon D. A Survey of Data Provenance Techniques. Technical Report TR618, Department of Computer Science, Indiana University, 2005.
5. Widom J. Trio: A System for Integrated Management of Data, Accuracy, and Lineage. In Proc. Second Biennial Conference on Innovative Data Systems Research, 2005, pp. 262–276.

Data Quality

► Information Quality and Decision Making
► Information Quality Policy and Strategy

Data Quality Assessment

Carlo Batini
University of Milano – Bicocca, Milan, Italy

Synonyms

Data quality measurement; Data quality benchmarking

Definition

The goal of the assessment activity in the area of data quality methodologies is to provide a precise evaluation and diagnosis of the state of databases and data flows of an information system with regard to data quality issues. In the assessment the evaluation is performed measuring the quality of data collections along relevant quality dimensions. The term (data quality) *measurement* is used to address the issue of measuring the value of a set of data quality dimensions. The term (data quality) *assessment* is used when such measurements are analyzed in order to enable a diagnosis of the quality of the data collection. The term (data quality) *benchmarking* is used when the output of the assessment is compared against reference indices, representing average values or best practices values in similar organizations. The term (data quality) *readiness* aims at assessing the overall predisposition of the organization in accepting and taking advantages of data quality improvement programs.

The assessment activity may concern: (i) the schema of the data base (the intension), (ii) the values of data (the extension), and (iii) the costs of poor data quality to the organization. Therefore, the principal outputs of assessment methodologies are: (i) measurements of the quality of databases and data flows, both schemas and values, (ii) costs to the organization due to the present low data quality, and (iii) a comparison with data quality levels considered acceptable from experience, or else a benchmarking with best practices, together with suggestions for improvements.

Historical Background

Ever since computer applications have been used to automate more and more business and administrative activities, it has become clear that available data often result from inaccurate observations, imputation, and elaborations, resulting in data quality problems. More importantly, in the last decades, information systems have been migrating from a hierarchical/monolithic to a network-based structure; therefore, the potential sources that organizations can use for the purpose of their businesses are dramatically increased in size and scope. Data quality problems have been further worsened by this evolution, since the external sources are created and updated at different times and by different organizations or persons and are characterized by various degrees of trustworthiness and accuracy, frequently unknown a priori. As a consequence, the overall quality of the data that flow between information systems may rapidly degrade over time if both processes and their inputs are not themselves subject to quality assessment.

Foundations

The typical steps to assess data quality are:

1. *Data analysis*, which examines data schemas and performs interviews to reach a complete understanding of data and related architecture and management rules.
2. *Requirements analysis*, which surveys the opinion of data users and administrators to identify quality issues and set new quality targets.
3. *Identification of critical areas*, which selects the most relevant databases and data flows to be assessed quantitatively.
4. *Process modeling*, which provides a model of the processes producing or updating data.
5. *Measurement of quality*, which selects relevant quality dimensions, defines corresponding metrics and performs the actual measurement.

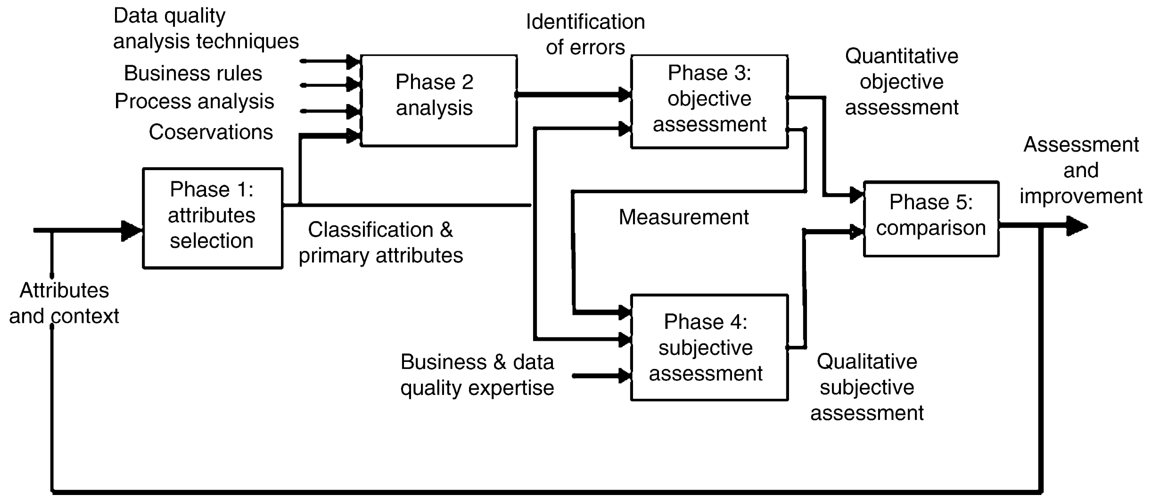
The usual process followed in the *measurement of quality* step has two main activities: qualitative assessment, based on subjective judgments of experts, and objective assessment, based on measures of data quality dimensions.

Qualitative assessment is performed through questionnaires and interviews with stakeholders and with internal and external users, with the goal of understanding the consequences and impact of poor data quality on the work of internal users and on products and services provided to information consumers, and the extent the needs of external users and customers are currently satisfied.

Quantitative assessment is based on the selection of quality dimensions and their measurement through metrics. Over 50 quality dimensions have been proposed in the literature (see the respective entry and [2] for a thorough description of dimensions and proposed classifications). The most frequently mentioned concern the values of data, and are accuracy, completeness, currency/timeliness, and inconsistency.

Examples of methodologies for the choice of dimensions and measures and for the objective vs. subjective evaluation are given in [3,7–9]. With regard to dimension classification, dimensions are classified in [7] into *sound*, *useful*, *dependable*, and *usable*, according to their positioning in quadrants related to “product quality/service quality” and “conforms to specifications/meets or exceeds consumer expectations” coordinates. The goal of the classification is to provide a context for each individual quality dimension and metric, and for consequent evaluation. In the following the five phases of the methodology proposed in [3] are described in more detail (see Fig. 1).

Phase 1, *attribute selection*, concerns the identification, description, and classification of the main data attributes to be assessed. Then, they are characterized


Data quality analysis techniques
Business rules
Process analysis
Coservations
Phase 2 analysis
Identification of errors
Phase 3: objective assessment
Quantitative objective assessment
Assessment and improvement
Phase 1: attributes selection
Classification & primary attributes
Measurement
Phase 5: comparison
Attributes and context
Phase 4: subjective assessment
Business & data quality expertise
Qualitative subjective assessment

Data Quality Assessment. Figure 1. The main phases of the assessment methodology described in [2].

according to their meaning and role. The possible characterizations are *qualitative/categorical*, *quantitative/numerical*, and *date/time*.

In Phase 2, *analysis*, data quality dimensions and integrity constraints to be measured are identified. Statistical techniques are used for the inspection of data. Selection and inspection of dimensions is related to process analysis, and has the final goal of discovering the main causes of erroneous data, such as unstructured and uncontrolled data loading and data updating processes. The result of the analysis on selected dimensions leads to a report with the identification of the errors.

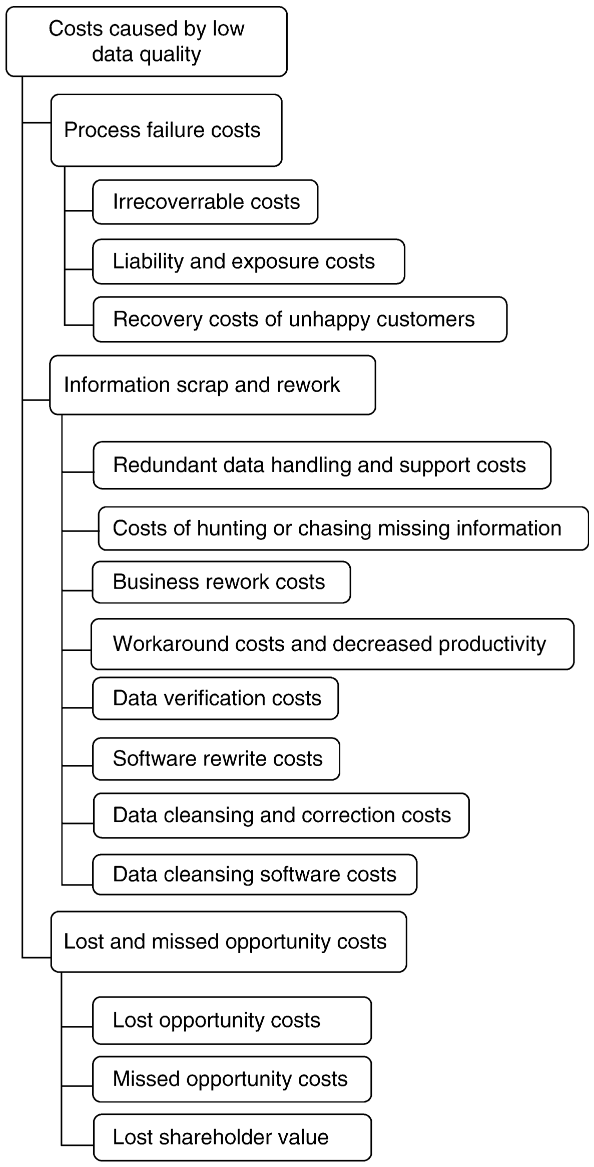
In Phase 3, *objective/quantitative assessment*, appropriate indices are defined for the evaluation and quantification of the global data quality level. The number of low quality data items for the different dimensions and the different data attributes is first evaluated with statistical and/or empirical methods, and, subsequently, normalized and summarized.

Phase 4 deals with *subjective/qualitative assessment*. The qualitative assessment is obtained by merging independent evaluations from (i) business experts, who analyze data from a business process point of view, (ii) final users (e.g., for financial data, a trader), and (iii) data quality experts, who have the role of analyzing data and examining its quality.

Finally, in the *comparison* phase objective and subjective assessments are compared. For each attribute and quality dimension, the distance between the percentages of erroneous observations obtained from quantitative analysis, mapped in a discrete domain, and the quality level defined by the judgment of the evaluations is calculated. Discrepancies are analyzed by the data quality experts, to further detect causes of errors and to find alternative solutions to correct them.

The above mentioned methodologies, although not explicitly, refer to the assessment of structured data, namely, data represented in terms of typed files or relational tables and databases. Recently, the attention in data quality assessment has moved towards semi-structured and un-structured data. Assessment methodologies for evaluating specific qualities of web sites are proposed in [1,6]. Atzeni et al. [1] is specifically focused on *accessibility*, evaluated on the basis of a mixed quantitative/qualitative assessment. The quantitative assessment activity checks the guidelines provided by the World Wide Web Consortium in (W3C. http://www.w3.org/WAI/). The qualitative assessment is based on experiments performed with disabled users. Fraternali et al. [6] focuses on the *usability* of the site and proposes an approach based on the adoption of *conceptual logs*, which are web usage logs enriched with meta-data derived from the application of conceptual specifications expressed by the conceptual schema of the web site.

Since data and information are often the most relevant resource consumed in administrative and business processes, several authors consider the evaluation of costs of poor data quality as part of the data quality assessment problem. Figure 2 shows the classification proposed in [4], for which comments follow:


Costs caused by low data quality
Process failure costs
Irrecoverrable costs
Liability and exposure costs
Recovery costs of unhappy customers
Information scrap and rework
Redundant data handling and support costs
Costs of hunting or chasing missing information
Business rework costs
Workaround costs and decreased productivity
Data verification costs
Software rewrite costs
Data cleansing and correction costs
Data cleansing software costs
Lost and missed opportunity costs
Lost opportunity costs
Missed opportunity costs
Lost shareholder value

Data Quality Assessment. Figure 2. A comprehensive classification of costs of poor data quality [4].

- *Process failure costs* result when poor quality information causes a process to not perform properly. As an example, inaccurate mailing addresses cause correspondence to be misdelivered.
- *Information scrap and rework* costs occur every time data of poor quality requires several types of defect management activities, such as reworking, cleaning, or rejecting. Examples of this category are (i) redundant data handling, if the poor quality of a source makes it useless, time and money has to be spent to collect and maintain data in another database, (ii) business rework costs, due to re-performing failed processes, such as resending correspondence, (iii) data verification costs, e.g., when data users do not trust the data, they perform their own quality inspection.
- *Loss and missed opportunity costs* correspond to the revenues and products not realized because of poor information quality. For example, due to low accuracy of customer e-mail addresses, a percentage of customers already acquired cannot be reached in periodic advertising campaigns, resulting in lower revenues, roughly proportional to the decrease of accuracy in addresses.

Data quality assessment has been investigated also under a managerial perspective. Following the results of the assessment, a managerial activity might be the analysis of the main barriers in the organization to the quality management perspective in terms of resistance to change processes, control establishment, information sharing, and quality certification.

Key Applications

Quality assessment is used in a large set of business and administrative activities, such as organization assessment, strategic planning, supply chain, marketing, selling, demographic studies, health experiments, management of health files, census applications, epidemiological analyses. The perception of the importance of quality assessment is increasing in the area of risk management, such as operational risk management related to the Basel II norms.

Future Directions

Open areas of research in data quality assessment concern quality dimensions and the relationship between data quality assessment and process quality assessment.

The first area concerns assessment of a wider set of dimensions, such as performance, availability, security, with concern also to risk management, and investigation on dependencies among dimensions. For example, a dependency among currency and accuracy is the rule "70% of all outdated data is also inaccurate." Knowledge about dependencies can greatly aid in finding causes of low data quality, and in conceiving improvement activities.

The relationship between data quality and process quality is a wide area of investigation, due to the relevance and diversity of characteristics of business processes in organizations. The different impacts of data quality at the three typical organizational levels, namely operations, the tactical level, and the strategic level, are analyzed in [10] reporting interviews and the outcomes of several proprietary studies. Data quality and its relationship with the quality of services, products, business operations, and consumer behavior is investigated in very general terms in [9,11]. The symmetric problem of investigating how to improve information production processes positively influences data quality is analyzed in [10].

Cross-references

► Data Quality Dimensions
► Design for Data Quality
► Information Quality Assessment
► Information Quality Policy and Strategy
► Quality in Data Warehouses

Recommended Reading

1. Atzeni P., Merialdo P., and Sindoni G. Web site evaluation: methodology and case study. In Proc. Int. Workshop on Data Semantics in Web Information Systems, 2001.
2. Batini C. and Scannapieco M. Data Quality: Concepts, Methodologies and Techniques. Springer, 2006.
3. De Amicis F. and Batini C. A methodology for data quality assessment on financial data. Stud. Commn. Sci., 4(2):115–136, 2004.
4. English L.P. Improving Data Warehouse and Business Information Quality. Wiley, 1999.
5. English L.P. Process management and information quality: how improving information production processes improves information (product) quality. In Proc. Seventh Int. Conf. on Information Quality (IQ 2002). MIT Sloan School of Management, Cambridge, MA, 2002, pp. 206–209.
6. Fraternali P., Lanzi P.L., Matera M., and Maurino A. Model-driven web usage analysis for the evaluation of web application quality. J. Web Eng., 3(2):124–152, 2004.
7. Kahn B., Strong D.M., and Wang R.Y. Information quality benchmarks: product and service performance. Commn. ACM, 45(4):184–192, 2002.
8. Lee Y.W., Strong D.M., Kahn B.K., and Wang R.Y. AIMQ: a methodology for information quality assessment. Inf. Manag., 40(2):133–146, 2001.
9. Pipino L., Lee Y.W., and Wang R.Y. Data quality assessment. Commn. ACM, 45(4):211–218, 2002.
10. Redman T.C. The impact of poor data quality on the typical enterprise. Commn. ACM, 41(2):70–82, 1998.
11. Sheng Y.H. Exploring the mediating and moderating effects of information quality on firms? Endeavor on information systems. In Proc. Eighth Int. Conf. on Information Quality (IQ 2003). MIT Sloan School of Management, Cambridge, MA, 2003, pp. 344–353.

Data Quality Attributes

► Data Quality Dimensions

Data Quality Benchmarking

► Data Quality Assessment

Data Quality Criteria

► Data Quality Dimensions

Data Quality Dimensions

Kai-Uwe Sattler
Technical University of Ilmenau, llmenau, Germany

Synonyms

Data quality criteria; Data quality attributes; Data quality measurement

Definition

Data quality (DQ) is usually understood as a multi-dimensional concept. The dimensions represent the views, criteria, or measurement attributes for data quality problems that can be assessed, interpreted, and possibly improved individually. By assigning scores to these dimensions, the overall data quality can be determined as an aggregated value of individual dimensions relevant in the given application context.

Historical Background

Since the mid-1990s data quality issues have been addressed by systematic research studies. In this context, relevant dimensions of data quality have also been investigated. One of the first empirical studies by Wang and Strong [6] has identified 15 relevant dimensions out of 179 gathered criteria. This list was later supplemented by other researchers. Initially, there were proposed divergent definitions of the same dimensions, mostly due to different views, e.g., management perspectives versus data-oriented perspectives as well as application-specific views. In addition, several classifications for data quality problems and criteria were proposed.

To date there is still a different understanding of several dimensions, depending on the application scenario and its requirements. However, there exists a set of agreed upon dimensions that are relevant in most domains.

Foundations

The selection of dimensions relevant in a given scenario is mostly application-dependent. In addition, many dimensions are not independent and, therefore, should not be used together. However, because quality dimensions characterize potential data quality problems they can be classified according some important characteristics. In the following, some representative classifications are introduced followed by a discussion of the most important dimensions:

Classifications

A first approach for classifying DQ dimensions proposed by Redman [5] is based on DQ problems or conflicts by considering the different levels where they can occur:

- The *intensional level* comprises criteria concerning the content of the conceptual schema relevance, clarity of definition, the scope, the level of detail (e.g., granularity of attributes, the precision of the attribute domains) as well as consistency and flexibility.
- The *extensional level* considers the data values comprising criteria such as accuracy and correctness of values, timeliness, and completeness of data.
- The *level of data representation* addresses problems related to the data format, e.g., interpretability, portability, adequateness.

In contrast to this data-oriented approach, the classification introduced by Naumann [4] is more comprehensive. Dimensions are classified into four sets:

1. *Content-related* dimensions consider the actual data and therefore data-intrinsic properties such as accuracy, completeness, and relevance.
2. *Technical* dimensions address aspects of the hard- and software used for maintaining the data. Examples are availability, latency, response time, but also price.
3. *Intellectual* dimensions represent subjective aspects, such as trustworthiness or reputation.
4. *Instantiation-related* dimensions concern the presentation of data, e.g., the amount of data, understandability, and verifiability.

An alternative way of classifying DQ dimensions is to look at the process of data evolution by analogy of data with products. In [3] an approach is presented promoting hierarchical views on data quality following the steps of the data life cycle: collection, organization, presentation, and application. Based on an analysis of possible root causes for poor quality relevant dimensions can be identified and assigned to the different DQ views:

- *Collection quality* refers to problems during data capturing, such as observation biases or measurement errors. The relevant dimensions are, among others, accuracy, completeness, and trustworthiness of the collector.
- *Organization quality* deals with problems of data preparation and manipulation for storing it in a database. It comprises dimensions such as consistency, storage, and retrieval efficiency. Furthermore, collection quality is also a component of organization quality.
- *Presentation quality* addresses problems during processing, re-interpretation, and presentation of data. Dimensions are for example interpretability, formality as well as the organization quality component.
- *Application quality* concerns technical and social constraints preventing an efficient utilization of data and comprises dimensions like timeliness, privacy, and relevance in addition to the presentation quality component.

Among all these dimensions the most important ones in many application scenarios are completeness, accuracy, consistency, and timeliness that are now described in detail.

Completeness

Missing or incomplete data is one of the most important data quality problem in many applications. However, there are different meanings of completeness.

An obvious and often used definition is the absence of null values or more exactly the ratio of non-null values and the total number of values. This measure can be easily assessed. Given a relation $R(A_1,\ldots,A_n)$ then N_{A_i} denotes the set of all non-null values in A_i:

$$N_{A_i} = \{t \in R | NotNull(t.A_i)\}$$

Completeness $Q_C(A_i)$ can be now defined as:

$$Q_C(A_i) = \frac{|N_{A_i}|}{|R|}$$

This can be also extended to take tuples into account instead of single values by determining the number of tuples containing no null values:

$$Q_C(R) = \frac{|N_{A_1,\ldots,A_n}|}{|R|}$$

Note that null can have different meanings which have to be treated in a special way: it could represent a missing value or simply a not-applicable case, e.g. a customer without a special delivery address.

Sometimes, not all attributes are of equal importance, e.g. whereas a customer identifier is always required, the customer's email address is optional. In this case, weights can be assigned to the individual attributes or rules of the form "if A_1 is not available (`null`) then A_2 is important, otherwise not" are used.

This notion of completeness concerns only the data inside the database. An alternative definition for completeness is the portion of real-world objects stored in the database. It addresses the case that for instance not all customers are represented in the database and therefore the data is incomplete. This is also known as *coverage*. However, assessing this completeness is often much more difficult because it requires either additional metadata (e.g., it is known that the DBLP digital library contains only computer science literature) or a (manual) checking with the real world, possibly supported by sampling.

Besides these extensional views, completeness can be also interpreted from an intensional point of view. Here, completeness (or *density*) is defined as the number of attributes represented in the database compared to the required real-world properties. Again, assessing this kind of completeness requires manual inspection.

Improvement of completeness is generally achieved by choosing better or additional data sources. In some cases, null values can be replaced with the help of dictionaries or reference sources (e.g., an address database). Depending of the usage of data missing numeric values can be sometimes also imputed based on knowledge about data characteristics, such as value distribution and variance.

Accuracy

A second data quality problem is often caused by measurement errors, observation biases or simply improper representation. Accuracy can be defined as the extent to which data are correct, reliable, and certified free of error. Note that the meaning of correctness is application-dependent: it can specify the distance to the actual real-world value or just the optimal degree of detail of an attribute value. Assuming a table representing sales volumes for products a value \$10,000 could be interpreted as inaccurate if the actual value, e.g., obtained in a different way, is \$10,500. However, if the user is interested only in some sales categories (low: $\leq$ 20K, high: $>$ 20K) the value is accurate.

In order to assess accuracy for a given value v the real world value $\overline{v}$ or at least a reference value is needed. Then, the distance can be easily computed for numeric values as $|v - \overline{v}|$ or – for textual values – as the syntactic distance using the edit distance measure. However, particularly for textual attributes sometimes the semantic distance has to be considered, e.g., the strings "Munich" and "München" are syntactically different but represent the same city. Solving this problem requires typically a dictionary or ontology.

Based on the distance of single attribute values, the accuracy of tuples or the whole relation can be computed as shown above for completeness, for example by determining the fraction of tuples with only correct values.

An improvement of accuracy is often possible only by removing inexact values or preferably by applying data cleaning techniques.

Consistency

Though modern database systems provide advanced support for ensuring integrity and consistency of data, there are many reasons why inconsistency is a further important data quality problem. Thus, consistency as a DQ dimension is defined as the degree to which data managed in a system satisfy specified constraints or business rules. Such rules can be classic database integrity constraints, such as uniqueness of customer identifiers or referential integrity (e.g., "for each order, a customer record must exist,") or more advanced business rules describing relationships between attributes (for instance "age = current-date − data-of-birth," "driver license number can only exist, if age $\geq$ 16.") These rules have to be specified by the user or can be derived automatically from training data by applying rule induction. Using a set **B** of such rules, the set of tuples from a relation R satisfying these rules can be determined:

$$W_{\mathbf{B}} = \{t \in R | \mathit{Satisfies}(t, \mathbf{B})\}$$

Then, the consistency measure for relation R can be computed as the fraction of tuples in $W_{\mathbf{B}}$ as shown above.

As for accuracy, an improvement of consistency can be achieved by removing or replacing inconsistent data.

Timeliness

Another reason for poor quality of data is outdated data. This problem is captured by the dimension timeliness describing the degree to which the provided data is up-to-date. Depending on the application this is not always the same as the ordinary age (time between creation of data and now). For instance, in a stock information system stock quotes data from 2000 are outdated if the user is interested in the current quotes. But if he asks for stock quotes from the time of the dot-com bubble, it would be still sufficient.

Therefore, both the age $age(v)$ of a value v (as the time between observation and now) and the update frequency $f_u(v)$ (updates per time unit) have to be considered, where $f_u(v) = 0$ means the value is never updated. Using this information, timeliness $Q_T(v)$ of v can be computed as

$$Q_T(v) = \frac{1}{f_u(v) \cdot age(v) + 1}$$

This takes into account that an object with a higher update frequency ages faster and that objects that are never updated have the same timeliness.

Further Dimensions

Finally, the following further dimensions are also important for many applications.

Relevance, also known from information retrieval, is the degree to which the provided information satisfies the users need. The problem of relevance occurs mainly if keyword-based search is used for querying data or documents. In database systems using exact queries, relevance is inherently high.

Response time measures the delay between the submission of a request (e.g. a query) and the arrival of the complete response. Though a technical criterion, response time is particularly important for users, because they usually do not want to wait more than a couple of seconds for an answer. Related to response time is *latency* defining the delay to the arrival of the first result data. Often, a small latency compensates a larger response time in user satisfaction.

Believability, trustworthiness, and reputation are dimensions which often depend on each other. Believability and trustworthiness can be understood as the degree to which data is accepted by the user as correct, accurate or complete. In contrast, reputation describes the degree to which a data (source) has a good standing by users. Reputation is based on the memory and summary of behavior from past transactions, whereas believability is more an subjective expectation.

Key Applications

DQ dimensions are primarily used for quality assessment. They define the criteria under which data quality is measured and for which quality scores can be derived. A further application are data quality models for explicitly representing data quality scores that can be used for annotating the data.

Cross-references

► Data Conflicts
► Data Quality Assessment
► Data Quality Models

Recommended Reading

1. Batini C. and Scannapieco M. Data Quality – Concepts, Methodologies and Techniques. Springer, 2006.
2. Gertz M., Özsu M.T., Saake G., and Sattler K. Report on the Dagstuhl Seminar: data quality on the Web. SIGMOD Rec., 33(1):127–132, 2004.
3. Liu L. and Chi L. Evolutional data quality: a theory-specific view. In Proc. Int. Conf. on Information Quality (IQ 2002). MIT, 2002, pp. 292–304.
4. Naumann F. Quality-Driven Query Answering for Integrated Information Systems. LNCS 2261, Springer, Berlin, 2002.
5. Redman T. Data Quality for the Information Age. Artech House, Norwood, MA, USA, 1996.
6. Wang R. and Strong D. Beyond Accuracy: What Data Quality Means to Data Consumers. J. Inf. Syst., 12(4):5–34, 1996.
7. Wang R., Ziad M., and Lee Y. Data Quality. Kluwer, Boston, MA, USA, 2001.

Data Quality Measurement

► Data Quality Dimensions
► Data Quality Assessment

Data Quality Models

Monica Scannapieco
University of Rome, Rome, Italy

Synonyms

Data quality representations

Definition

Data quality models extend traditional models for databases for the purpose of representing data quality dimensions and the association of such dimensions to data. Therefore, data quality models allow analysis of a set of data quality requirements and their representation in terms of a conceptual schema, as well as accessing and querying data quality dimensions by means of a logical schema. Data quality models also include process models tailored to analysis and design of quality improvement actions. These models permit tracking data from their source, through various manipulations that data can undergo, to their final usage. In this way, they support the detection of causes of poor data quality and the design of improvement actions.

Historical Background

Among the first data quality models, in 1990 the *polygen model* [5] was proposed for explicitly tracing the origins of data and the intermediate sources used to arrive at that data. The model is targeted to heterogeneous distributed systems and is a first attempt to represent and analyze the provenance of data, which has been recently investigated in a more general context.

In the mid-1990's, there was a first proposal of extending the relational model with quality values associated to each attribute value, resulting in the *quality attribute model* [6]. An extension of the Entity Relationship model was also proposed in the same years ([4], and [7], Chapter 3), similarly focused on associating quality dimensions, such as accuracy or completeness, to attributes. More recently, models for associating quality values to data-oriented XML documents have been investigated (e.g., [2]). Such models are intended to be used in the context of distributed and cooperative systems, in which the cooperating organizations need to exchange data each other, and it is therefore critical for them to be aware of the quality of such data. These models are semi-structured, thus allowing each organization to export the quality of its data with a certain degree of flexibility; quality dimensions can be associated to various elements of the data model, ranging from the single data value to the whole data source, in this way being different from the previous attribute-based models.

The principal data quality models that are oriented towards process representation are based on the principle that data can be seen as a particular product of a manufacturing activity, and so descriptive models (and methodologies) for data quality can be based on models conceived in the last two centuries for manufacturing traditional products. The *Information Product Map (IP-MAP)* [3] is a significant example of such models and follows this view, being centered on the concept of information product. The IP-MAP model has been extended in several directions (see [1], Chap. 3). Indeed, more powerful mechanisms have been included, such as *event process chain diagrams* representing the business process overview, the interaction model (how company units interact), the organization model (who does what), the component model (what happens), and the data model (what data is needed). A further extension called *IP-UML* consists of a UML profile for data quality based on IP-MAP.

Foundations

Data quality models can be distinguished in data-oriented models, focused on the representation of data quality dimensions, and process-oriented models focused on the representation of the processes that manipulate data and on their impact on the data quality.

Data-oriented models include extensions of the Entity Relationship model, of the relational model, and of the XML data model.

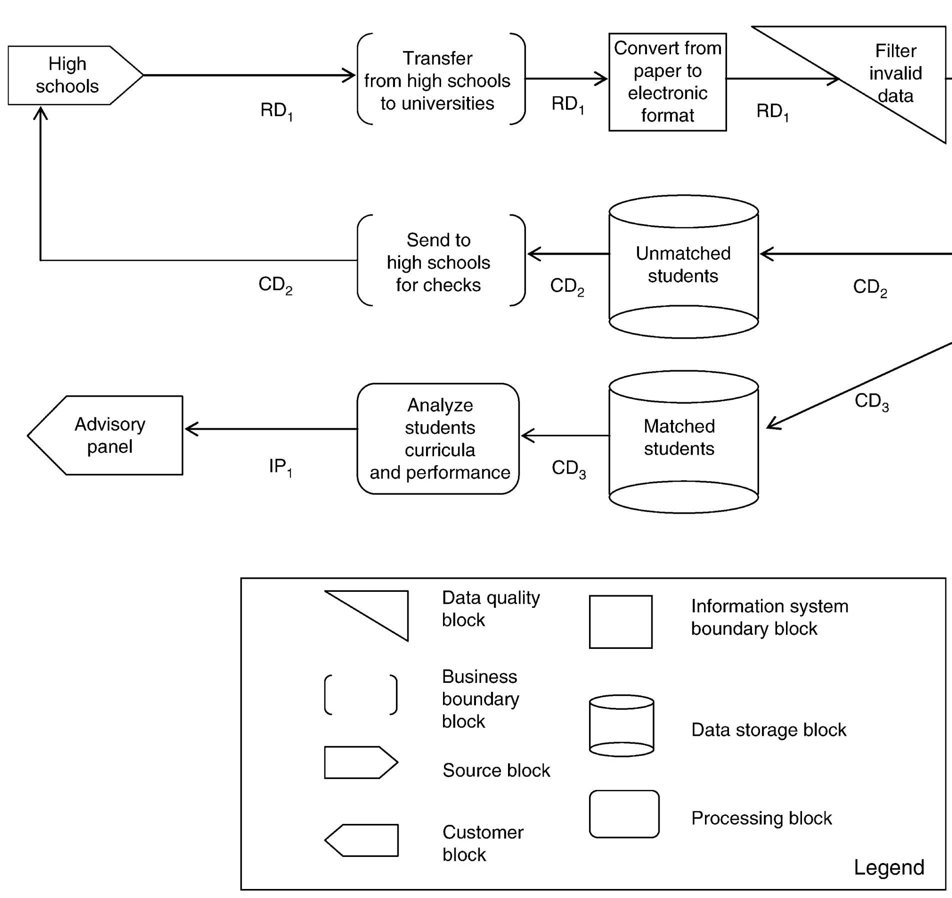
When extending the Entity Relationship model for representing data quality, one possibility is to introduce two types of entities, explicitly defined to express quality dimensions and their values: a data quality dimension entity and a data quality measure entity. The goal of the data quality dimension entity is to represent possible pairs `<DimensionName, Rating>` of dimensions and corresponding ratings resulting from measurements. The data quality dimension entity characterizes the quality of an attribute and the scale may obviously depend on the attribute. In these cases, it is necessary to extend the properties of the data quality dimension entity to include the attribute, that is `<DimensionName, Attribute, Rating>`.

In order to represent metrics for dimensions, and the relationship with entities, attributes, and dimensions, the model introduces the data quality measure entity; its attributes are `Rating`, the values of which depend on the specific dimension modeled, and `DescriptionOfRating`. The complete data quality schema, shown by means of the example in Fig. 1, is made up of:

- The original data schema, in the example represented by the entity `Class` with the attribute `Attendance`.
- The `DQ Dimension` entity with a pair of attributes <`DimensionName`, `Rating` >.
- The relationship between the entity `Class`, the related attribute `Attendance`, and the `DQ Dimension` entity with a many-to-many relationship `ClassAttendanceHas`; a distinct relationship has to be introduced for each attribute of the entity `Class`.
- The relationship between the previous structure and the `DQ Measure` entity with a new representation structure that extends the Entity Relationship model, and relates entities and relationships.

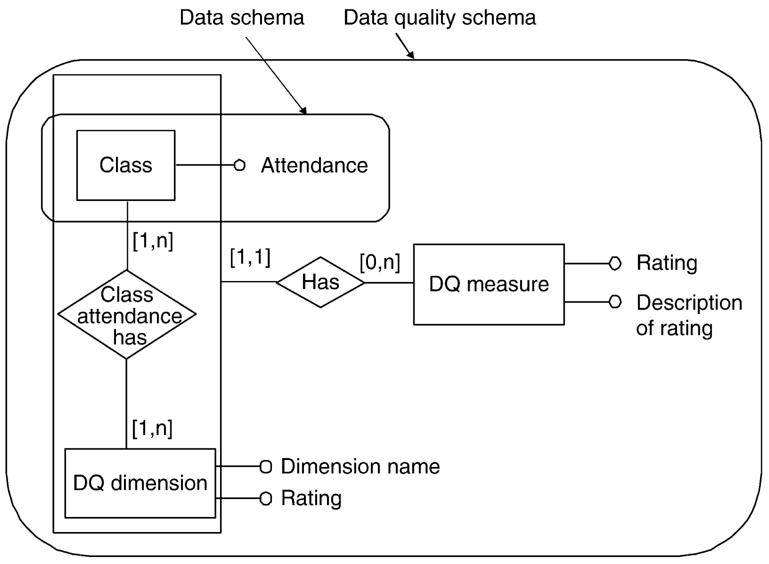
An extension of the relational data model is provided by the quality attribute model, explained in the following by means of the example shown in Fig. 2.

The figure shows a relational schema `Employee`, defined on attributes `EmployeeId`, `Address`, `DateofBirth`, and others, and one of its tuples. Relational schemas are extended adding an arbitrary number of underlying levels of quality indicators (only one level in the figure) to the attributes of the schema, to which they are linked through a quality key. In the example, the attribute `EmployeeId` is extended with one quality attribute, namely accuracy, the attribute `Address` with two quality attributes, namely accuracy and currency, while the attribute `DateofBirth` is extended with accuracy and completeness. The

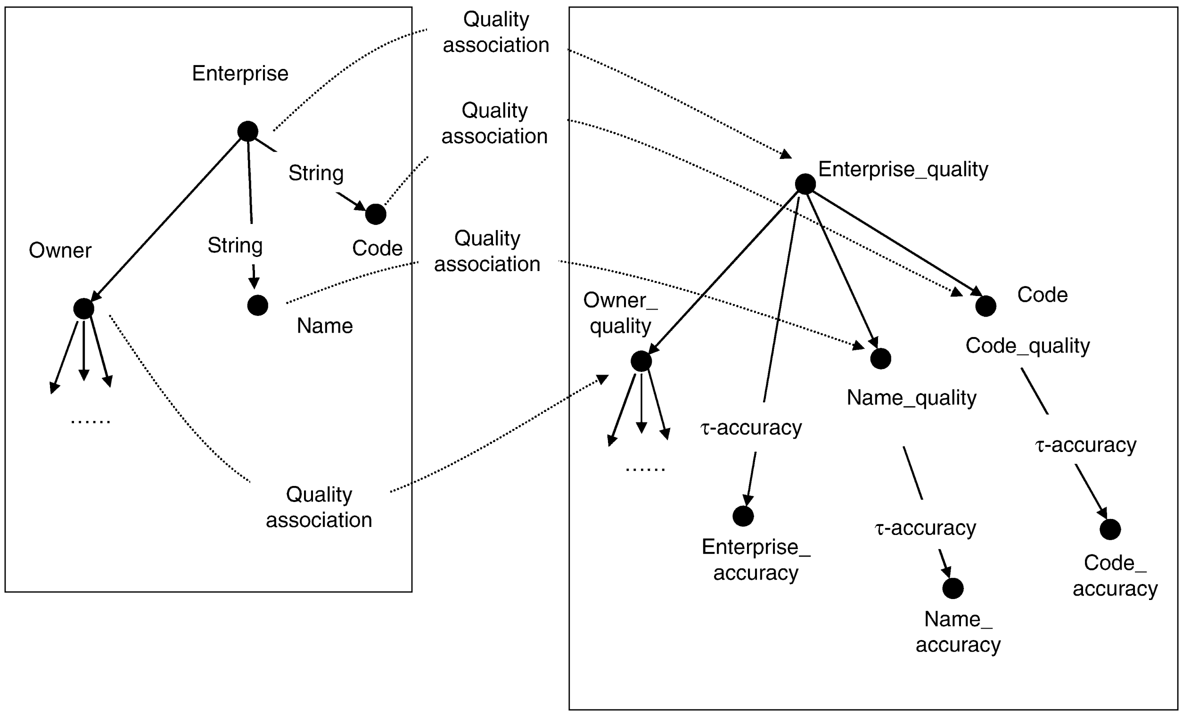

High schools
RD1
Transfer from high schools to universities
RD1
Convert from paper to electronic format
RD1
Filter invalid data
CD1
High schools students database
CD1
Perform record linkage
CD2
Unmatched students
CD2
Send to high schools for checks
CD2
RD2
University students database
CD3
Matched students
CD3
Analyze students curricula and performance
IP1
Advisory panel
Data quality block
Information system boundary block
Business boundary block
Data storage block
Source block
Processing block
Customer block
Legend

Data Quality Models. Figure 1. An example of IP-MAP.

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:14



Data Quality Models. Figure 2. An extension of the entity relationship model.



Data Quality Models. Figure 3. An example of a D^2Q schema.

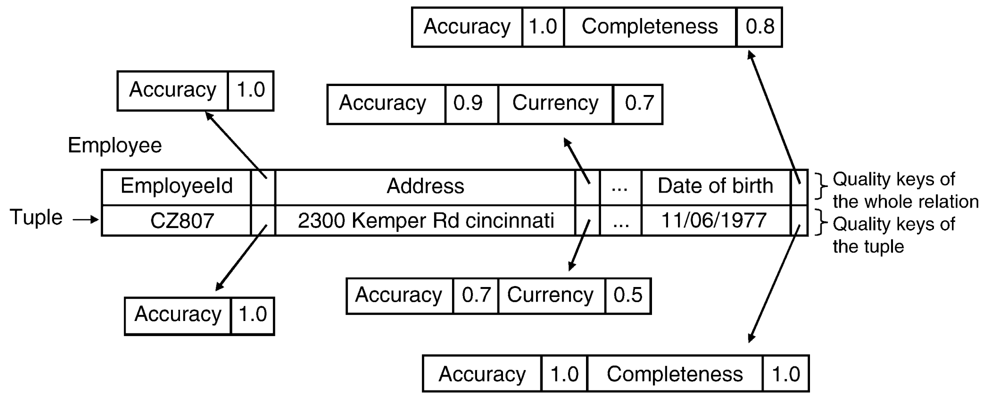
values of such quality attributes measure the quality dimensions' values associated with the whole relation instance (top part of the figure). Therefore, completeness equal to 0.8 for the attribute DateofBirth means that the 80% of the tuples have a non-null value for such an attribute. Similar structures are used for the instances of quality indicator relations (bottom part of the figure); if there are n attributes of the relational schema, n quality tuples will be associated to each tuple in the instance.

The model called Data and Data Quality (D^2Q) is among the first models for associating quality values to data-oriented XML documents. D^2Q can be used in order to certify dimensions like accuracy, consistency, completeness, and currency of data. The model is semi-structured, thus allowing each organization to export the quality of its data with a certain degree of flexibility. More specifically, quality dimension values can be associated with various elements of the data model, ranging from the single data value to the whole data source. The main features of the D^2Q model are summarized as follows:

- A data class and a data schema are introduced to represent the business data portion of the D^2Q model.
- A quality class and a quality schema correspond to the quality portion of the D^2Q model.
- A quality association function that relates nodes of the graph corresponding to the data schema to nodes of the graph corresponding to the quality schema. Quality associations represent biunivocal functions among all nodes of a data schema and all non-leaf nodes of a quality schema.

In Fig. 3, an example of a D^2Q schema is depicted. On the left-hand side of the figure, a data schema is shown representing enterprises and their owners. On the right-hand side, the associated quality schema is represented. Specifically, two quality classes, `Enterprise_Quality` and `Owner_Quality` are associated with the `Enterprise` and `Owner` data classes. Accuracy nodes are shown for both data classes and related properties. For instance, `Code_accuracy` is an accuracy node (of type *τ-accuracy*) associated with the `Code` property, while `Enterprise_accuracy` is an accuracy node associated with the data class `Enterprise`. The arcs connecting the data schema and the quality schema with the quality labels represent the quality association functions. The D^2Q model is intended to be easily translated into the XML data model. This is important for meeting the interoperability requirements that are particularly stringent in cooperative systems.

Process-oriented models have their principal representative in the Information Product Map (IP-MAP) model. An information product map is a graphical model designed to help people comprehend, evaluate, and describe how an information product, such as an invoice, a customer order, or a prescription, is assembled in a business process. IP-MAPs are designed to help analysts visualize the information production process, identify ownership of process phases, understand information and organizational boundaries, and estimate time and quality metrics associated with the current production process. There are eight types of construct blocks that can be used to form the IP-MAP: source, customer, data quality, processing, data storage, decision, business boundary, and information systems boundary. An example of information product map is shown in Fig. 4. Information products (IP in the figure) are produced by means of processing activities and data quality checks on raw data (RD), and semi-processed information called component data (CD). In the example, it is assumed that high schools and universities of a district have decided to cooperate in order to improve their course offering to students, avoiding overlap and being more effective in the education value chain. To this end, they have to share


Accuracy 1.0 Completeness 0.8
Accuracy 1.0
Accuracy 0.9 Currency 0.7
Employee
EmployeeId
Address
...
Date of birth
Quality keys of the whole relation
Tuple
CZ807
2300 Kemper Rd cincinnati
11/06/1977
Quality keys of the tuple
Accuracy 1.0
Accuracy 0.7 Currency 0.5
Accuracy 1.0 Completeness 1.0

Data Quality Models. Figure 4. An extension of the relational model.

historical data on students and their curricula. Therefore, they perform a record linkage activity that matches students in their education life cycle ("Perform Record Linkage" block). To reach this objective, high schools periodically supply relevant information on students; in case it is in paper format, the information has to be converted in electronic format. At this point, invalid data are filtered and matched with the database of university students. Unmatched students are sent back to high schools for clerical checks, and matched students are analyzed. The result of the analysis of curricula and course topics are sent to the advisory panel of the universities.

Key Applications

Data-oriented and process-oriented data quality models can be used to represent quality dimensions and quality related activities thus supporting techniques for data quality improvement. However, such techniques seldom rely on the described model extensions, with the distinctive exception of the IP-MAP model. Indeed, only a few prototypical DBMSs have experienced the adoption of some of the approaches mentioned. This is mainly due to the complexity of the representational structures proposed in the different approaches. Indeed, measuring data quality is not an easy task, hence models that impose to associate data quality dimensions values at attribute level have proven not very useful in practice. A greater flexibility is more useful in real applications, like for instance, in scenarios of e-Government or e-Commerce. In these scenarios, which involve cooperation between different organizations, a more successful case is provided by XML data exchanged with associated quality profiles, which are based on semi-structured data quality models.

Future Directions

The future of research on models appears to be in provenance and semi-structured data quality models. In open information systems and in peer-to-peer ones, knowing the provenance and having a flexible tool to associate quality to data is crucial. Indeed, such systems have to be able to trace the history of data and to certify the level of quality of the retrieved data.

Cross-references

► Data Quality Dimension
► Entity Relationship Model
► Information Product Management
► Provenance
► Relational Data Model
► Semi-Structured Data
► XML

Recommended Reading

1. Batini C. and Scannapieco M. Data Quality: Concepts, Methodologies, and Techniques. Springer, Berlin, 2006.
2. Scannapieco M., Virgillito A., Marchetti C., Mecella M., and Baldoni R. The DaQuinCIS architecture: a platform for exchanging and improving data quality in cooperative information systems. Inf. Syst., 29(7):551–582, 2004.
3. Shankaranarayan G., Wang R.Y., and Ziad M. Modeling the manufacture of an information product with IP-MAP. In Proc. Fifth Int. Conf. on Information Quality (ICIQ 2000), MIT, 2000, pp. 1–16.
4. Storey V.C. and Wang R.Y. An analysis of quality requirements in database design. In Proc. Fourth Int. Conf. on Information Quality (IQ 1998), MIT, 1998, pp. 64–87.
5. Wang R.Y. and Madnick S.E. A polygen model for heterogeneous database systems: the source tagging perspective. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 519–538.
6. Wang R.Y., Reddy M.P., and Kon H. Toward data quality: an attribute-based approach. DSS, 13(3–4):349–372, 1995.
7. Wang R.Y., Ziad M., and Lee Y.W. Data Quality. Kluwer, Boston, MA, USA, 2001.

Data Quality Problems

► Data Conflicts

Data Quality Representations

► Data Quality Models

Data Rank/Swapping

Josep Domingo-Ferrer
The Public University of Tarragona, Tarragona, Spain

Synonyms

Data swapping; Rank swapping

Definition

Data swapping was originally designed by Dalenius and Reiss [1] as a masking method for statistical disclosure control of databases containing only categorical attributes. The basic idea behind the method is to transform a database by exchanging values of confidential attributes among individual records. Records are exchanged in such a way that low-order frequency counts or marginals are maintained.

Rank swapping is a variant of data swapping [2,3]. First, values of an attribute X_i are ranked in ascending order, then each ranked value of X_i is swapped with another ranked value randomly chosen within a restricted range (e.g. the rank of two swapped values cannot differ by more than $p\%$ of the total number of records, where p is an input parameter). This algorithm is independently used on each original attribute in the original data set.

Key Points

It is reasonable to expect that multivariate statistics computed from data swapped with this algorithm will be less distorted than those computed after an unconstrained swap. In empirical work on SDC scores, rank swapping with small swapping range has been identified as a particularly well-performing method in terms of the trade-off between disclosure risk and information loss. Consequently, it is one of the techniques that have been implemented in the $\mu - Argus$ package [3].

Cross-references

► Data Rank/Swapping
► Disclosure Risk
► Inference Control in Statistical Databases
► Information Loss Measures
► K-anonymity
► Microaggregation
► Microdata
► Microdata rounding
► Noise Addition
► Non-perturbative masking methods
► Pram
► Record Linkage
► Synthetic Microdata
► SDC Score
► Tabular Data

Recommended Reading

1. Dalenius T. and Reiss S.P. Data-swapping: a technique for disclosure control (extended abstract). In Proc. ASA Section on Survey Research Methods. American Statistical Association, Washington DC, 1978, pp. 191–194.
2. Domingo-Ferrer J. and Torra V. A quantitative comparison of disclosure control methods for microdata. In Confidentiality, Disclosure and Data Access: Theory and Practical Applications for Statistical Agencies. P. Doyle, J.I. Lane, J.J.M. Theeuwes, and L. Zayatz (eds.). Amsterdam, North-Holland, 2001, pp. 111–134.
3. Hundepool A., Van de Wetering A., Ramaswamy R., Franconi F., Polettini S., Capobianchi A., De Wolf P.-P., Domingo-Ferrer J., Torra V., Brand R. and Giessing S. μ-Argus User's Manual version 4.1, February 2007. http://neon.vb.cbs.nl/CASC

Data Reconciliation

► Constraint-Driven Database Repair

Data Reduction

Rui Zhang
University of Melbourne, Melbourne, VIC, Australia

Definition

Data reduction means the reduction on certain aspects of data, typically the volume of data. The reduction can also be on other aspects such as the dimensionality of data when the data is multidimensional. Reduction on any aspect of data usually implies reduction on the volume of data.

Data reduction does not make sense by itself unless it is associated with a certain purpose. The purpose in turn dictates the requirements for the corresponding data reduction techniques. A naive purpose for data reduction is to reduce the storage space. This requires a technique to compress the data into a more compact format and also to restore the original data when the data needs to be examined. Nowadays, storage space may not be the primary concern and the needs for data reduction come frequently from database applications. In this case, the purpose for data reduction is to save computational cost or disk access cost in query processing.

Historical Background

The need for data reduction arises naturally. In early years (pre-1990's), storage was quite limited and expensive. It fostered the development of a class of techniques called *compression techniques* to reduce the data volume for lower consumption of resources such as storage space or bandwidth in telecommunication settings. Another requirement for a compression technique is to reproduce the original data (from the compressed data) for reading. Here "reading" has different meanings depending on the data contents. It means "listening" for audio data, "viewing" for video data, "file reading" for general contents, etc. Therefore the reproduction of the data should be either exactly the same as the original data or very close to the original data by human perception. For example, MP3 is an audio compression technique which makes a compressed audio sound almost the same to the original one. Until today, compression techniques is still a very active research topic. But, instead of concerning data size of kilobytes or megabytes as in the early years, today's compression techniques concern data size of gigabytes or even terabytes.

As the rapid advances of hardware technologies, storage limit is no longer the most critical issue in many cases. Another huge force driving the need for data reduction appears in database applications. Storing the data may not be a problem, but retrieving data from the storage (typically hard disk) is still a quite expensive operation due to the slow improvement in disk seek time. Database queries commonly need to retrieve large amount of data from the disk. Therefore data reduction is compelling for providing high performance in query processing. Different from data compression, data reduction in database applications usually do not need to generate a reproduction that is exactly the same as the original data or sounds/looks very close to the original data. Instead, an approximation of the intended answer suffices, which gives more flexibility for data reduction.

Traditionally, data reduction techniques have been used in database systems to obtain summary statistics, mainly for estimating costs of query plans in a query optimizer. Here, an approximation of the expected cost suffices as an estimate. At the same time, highly reduced data (summary statistics) is essential to make evaluation of the query plans much cheaper than evaluation of the query.

In the last two decades, there has been enormous interest in *online analytic processing (OLAP)*, which is characterized by complex queries involving group-by and aggregation operators on extremely large volume of data. OLAP is mainly performed in *decision support* applications, which analyze data and generate summaries from data. Organizations need these results to support high-level decision making. The data typically comprises of data consolidated from many sources of an organization, forming a repository called a *data warehouse*. In face of the high data volume, efficient OLAP calls for data reduction techniques. Due to the analytical and exploratory nature of the queries, approximate answers are usually acceptable and the error tolerance can sometimes be quite high.

Foundations

Compression techniques and data reduction techniques in databases are discussed separately below due to the differences in their purposes and general characteristics. Compression techniques are more often studied in the information retrieval research community while data reduction techniques in databases are studied in the database research community. Compression techniques is a subcategory of data reduction techniques, although sometimes the term *compression technique* is used in a less strict way to refer to data reduction in general.

Compression techniques involve the processes of *encoding* and *decoding*. Encoding converts the original data to a more compact format based on a mapping from *source messages* into *codewords*. Decoding performs the inverse operation to reproduce the original data. If the reproduction is exactly the same as the original data, the compression technique is *lossless*; otherwise, it is *lossy*. Lossless compression techniques are used for generally any data format without needing to know the contents or semantics of the data. Popular techniques include *ZIP* invented by Phil Katz in late 1980s and *RAR* invented by Eugene Roshal in early 1990s. If some domain knowledge on the data is available, usually *lossy* compression techniques yield better compression rates. For example, JEPG, MP3 and MPEG are popular compression techniques for audio, image and video data, respectively. Lossy compression techniques leave out the less important information and noise to achieve higher compression. More concretely, the MP3 audio encoding format removes the audio details most human beings cannot hear to make the compressed audio sound like a faithful reproduction of the original uncompressed one. Different compression techniques mainly differ in the mapping from source messages into codewords. A survey of compression techniques is given in [6]. Readers interested in recent research results in compression techniques are referred to the proceedings of the Data Compression Conference [1].

Data reduction in databases can make use of various techniques. Popular ones include histograms, clustering,

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:15

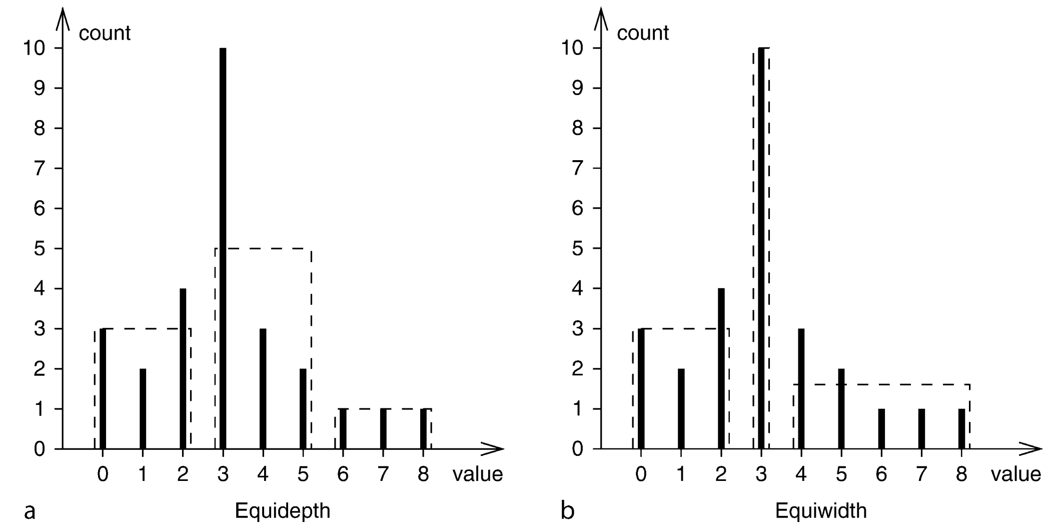
singular value decomposition (SVD), discrete wavelet transform (DWT), etc. The techniques can be divided into two categories, *parametric* and *nonparametric* techniques, depending on whether the technique assumes a certain model. Histograms and clustering are nonparametric techniques while SVD and DWT are parametric techniques. A summary of data reduction techniques for databases can be found in [3].

Histograms

A histogram is a data structure used to approximate the distribution of values. The value domain is divided into subranges called *buckets*. For each bucket, a count of the data items whose values fall in the bucket is maintained. Therefore a histogram basically contains the information of the boundaries of the buckets and the counts. The data distribution can be approximated by the average values of the buckets and the counts. Commonly, two types of histograms are used, *equiwidth* and *equidepth* histograms, distinguished by how the buckets are determined. In an equiwidth histogram, the length of every bucket is the same while in an equidepth histogram, the count for every bucket is the same (Sometimes, an exact same count cannot be achieved and then the counts for the buckets are approximately the same.). Figure 1 shows an example data distribution in the value domain [0,8] and the equiwidth and equidepth histograms for the data assuming three buckets. A thick vertical line represents the count for a value; a dashed line represent a bucket range and the estimated count for the values in the bucket. The estimated count of a certain value is simply the average count in the bucket. In the equiwidth histogram (Fig. 1a), each bucket has the range of length 3. The estimated counts for most values are quite close to the actual values. Equiwidth histograms are simple and easy to maintain, but the estimate is less accurate for skewed distribution such as the count of value 3. This problem is alleviated in the equidepth histogram (Fig. 1b). Each bucket has the count of about 9. The estimate count for value 3 is very accurate. The disadvantage of equidepth histograms is that determining the boundaries of buckets is more difficult. There are other types of histograms such as *compressed* histograms and *v-optimal* histograms. A thorough classification on various histograms can be found in [7].

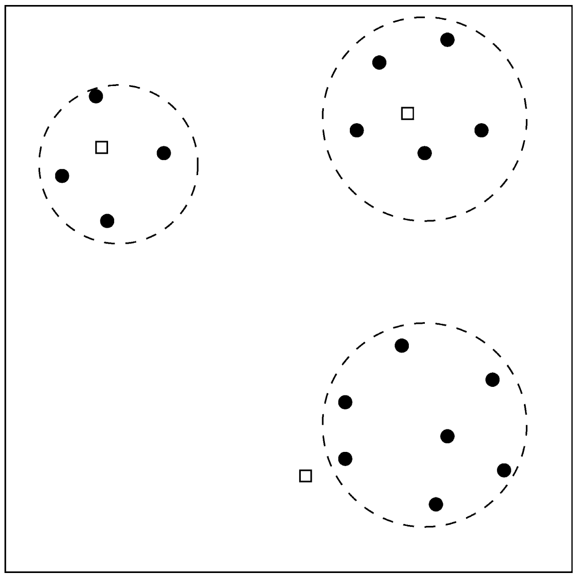
Clustering

Clustering is a technique to partition objects into groups called *clusters* such that the objects within a group are similar to each other. After clustering, operations can be performed on objects collectively as groups. The information of data can be represented at the cluster level and hence greatly reduced. The data to perform clustering on usually contain multiple attributes. Therefore each data object can be represented by a multidimensional point in space. The similarity is measured by a distance function. Typically, a metric function, such as Euclidean



Data Reduction. Figure 1. Histograms.

distance, is used as the distance function. Given a data set, there is no single universal answer for the problem of clustering. The result of clustering depends on the requirements or the algorithm used to perform clustering. A classic algorithm is the *k-means* algorithm, which partitions the data into *k* clusters. Given a data set and a number *k*, the algorithm first picks *k* points randomly or based on some heuristics to serve as cluster centroids. Second, every point (object) is assigned to its closest centroid. Then the centroid for each cluster is recomputed based on the current assignment of points. If the newly computed centroids are different from the previous ones, all the points are assigned to their closest centroids again and then the centroids are computed again. This process is repeated until the centroids do not change. Figure 2 shows an example where $k = 3$. The black dots are data points, squares are initial centroids and the dashed circles show the resultant clusters. In the beginning, the value of *k* is given by the user in a very subjective manner, usually depending on the application needs. Another algorithm called *k-medoid* works in a very similar manner but with a different way of choosing their cluster representatives, called *medoids*, and with a different stop condition. Recently, algorithms designed for large data sets were proposed in the database research community such as BIRCH [9] and CURE [4].



Data Reduction. Figure 2. *k*-means clustering.

Singular Value Decomposition (SVD)

Any $m \times n$ real matrix A can be decomposed as follows:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^t \qquad (1)$$

where $\mathbf{U}$ is a column-orthonormal $m \times r$ matrix, r is the rank of the matrix $\mathbf{A}$, $\mathbf{S}$ is a diagonal $r \times r$ matrix and $\mathbf{V}$ is a column-orthonormal $n \times r$ matrix (bold symbols are used to represent matrices and vectors). It can be further expressed in the *spectral decomposition* [5] form:

$$\mathbf{A} = \lambda_1 \mathbf{u}_1 \mathbf{v}_1^t + \lambda_2 \mathbf{u}_2 \mathbf{v}_2^t + \ldots + \lambda_r \mathbf{u}_r \mathbf{v}_r^t \qquad (2)$$

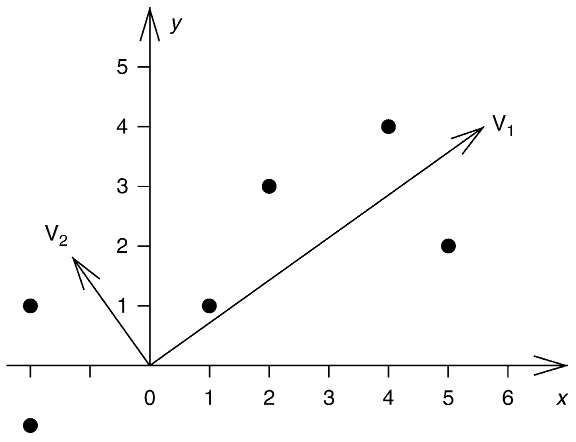
where $\mathbf{u}_i$ and $\mathbf{v}_i$ are column vectors of $\mathbf{U}$ and $\mathbf{V}$, respectively, and λ_i are the diagonal elements of $\mathbf{S}$. $\mathbf{A}$ can be viewed as m n-dimensional points (each row being a point). Because $\mathbf{v}_i$ are orthogonal vectors, they form a new basis of the space. λ_i represents the importance of the basis vector $\mathbf{v}_i$ (dimension i) and $\mathbf{u}_i$ represents the coordinates of the m points in dimension i in this new coordinate system. Assume that λ_i are sorted in descending order. Then, $\mathbf{v}_1$ is the direction (dimension) with the largest dispersion (variance) of the points; $\mathbf{v}_2$ is the direction with the second largest dispersion of the points, and so on. If the last few λ_i values are small and one omits them when calculating $\mathbf{A}$, the resulted error will be very small. Therefore SVD is widely used in dimensionality reduction and matrix approximation. The following is an example, with $\mathbf{A}$ given as

$$\mathbf{A} = \begin{bmatrix} -2 & 1 \\ -2 & -1 \\ 1 & 1 \\ 2 & 3 \\ 4 & 4 \\ 5 & 2 \end{bmatrix}$$

The SVD of $\mathbf{A}$ is

$$\mathbf{A} = \begin{bmatrix} -0.118 & 0.691 \\ -0.250 & 0.125 \\ 0.158 & 0.079 \\ 0.383 & 0.441 \\ 0.633 & 0.316 \\ 0.593 & -0.454 \end{bmatrix} \begin{bmatrix} 8.82 & 0 \\ 0 & 2.87 \end{bmatrix} \begin{bmatrix} 0.811 & 0.585 \\ -0.585 & 0.811 \end{bmatrix}$$

Here, $\lambda_1 = 8.82, \lambda_2 = 2.87, \mathbf{v}_1 = \begin{bmatrix} 0.811 \\ 0.585 \end{bmatrix}$ and $\mathbf{v}_2 = \begin{bmatrix} -0.585 \\ 0.811 \end{bmatrix}$. Figure 3 shows the data points, and the directions of $\mathbf{v}_1$ and $\mathbf{v}_2$.



Data Reduction. Figure 3. SVD.

Discrete Wavelet Transform (DWT)

Wavelet Transform is a commonly used signal processing technique like other transforms such as *Fourier Transform* or *Cosine Transform*. In databases, commonly used is the discrete version called Discrete Wavelet Transform (DWT). After applying DWT, a multi-resolution decomposition of the original signal is obtained in the form of wavelet coefficients. The wavelet coefficients are projections of the signal onto a set of orthogonal basis vectors. The choice of the basis vectors determines the type of DWT. The most popular one is the Haar transform, which is easy to implement and fast to compute. Some of the wavelet coefficients obtained may be small, therefore they can be replaced by zeros and hence the data is reduced. Inverse DWT can be applied on the reduced wavelet coefficients to get an approximation of the original signal. This is basically how DWT is used for compression. DWT based compression provides better lossy compression than Discrete Fourier Transform and Discrete Cosine Transform.

In the Haar transform, elements of a signal are processed pairwise. Specifically, the average and difference of every two neighboring elements are computed. The averages serve as a lower-resolution approximation of the signal and the differences (divided by 2) are the coefficients. For example, signal $S = \{2, 4, 5, 5, 3, 1, 2, 2\}$. Computing the average of every two neighboring elements results in a lower-resolution signal $S_1 = \{3, 5, 2, 2\}$. The coefficients are obtained by computing the difference of every two neighboring elements divided by 2, which is $D_1 = \{-1, 0, 1, 0\}$. S can be restored exactly by adding (or subtracting) the coefficient to the corresponding element in S_1. For example, $S(1) = S_1(1) + D_1(1) = 3 + (-1) = 2$; $S(2) = S_1(1) - D_1(1) = 3 - (-1) = 4$. Similarly, a even lower-resolution signal S_2 can be obtained by applying the same process on S_1. This can be done recursively until the length of the signal becomes 1. The full decomposition on S is shown in Fig. 4. The Haar transform of S is given as the average over all elements (3), and all the coefficients, $S' = \{3, 1, -1, 0, -1, 0, 1, 0\}$.

Resolution	Averages	Coeffecients
8	2, 4, 5, 5, 3, 1, 2, 2	
4	3, 5, 2, 2	−1, 0, 1, 0
2	4, 2	−1, 0
1	3	1

Data Reduction. Figure 4. Haar transform.

Key Applications

Data Storage and Transfer

Compression techniques are essential for data storage and transfer in many applications.

Database Management Systems

Histograms is a popular technique for maintaining summary information in database management systems. It is especially useful for a cost-based query optimizer.

OLAP

Due to the huge volume of data in OLAP applications, data reduction techniques such as sampling are commonly used to obtain quick approximate answers.

Multimedia Data

Multimedia data is characterized by large size. Therefore data reduction techniques are usually applied on multimedia data from storage to data processing. For example, the MP3, JPEG, MPEG formats for audio, image and video data, respectively, all use compression techniques. The new JPEG digital image standard, JPEG-2000, uses DWT for all its codecs [8]. Similarity search on multimedia data usually needs to deal with very high-dimensional point representations. SVD can be used to reduce the dimensionality to achieve better search performance. In a recent paper [2], DWT is used to represent 3D objects to obtain better data retrieval performance.

Taxonomy

Clustering is widely used in almost all taxonomy applications such as taxonomies of animals, plants, diseases and celestial bodies. It can also help visualization through a hierarchically clustered structure.

Cross-references

► Clustering
► Discrete Wavelet Transform
► Histograms
► Nonparametric Data Deduction
► Parametric Data Deduction
► Singular Value Decomposition

Recommended Reading

1. http://www.cs.brandeis.edu/~dcc/index.html.
2. Ali M.E., Zhang R., Tanin E., and Kulik L. A motion-aware approach to continuous retrieval of 3D objects. In ICDE. 2008.
3. Barbará D., DuMouchel W., Faloutsos C., Haas P.J., Hellerstein J.M., Ioannidis Y.E., Jagadish H.V., Johnson T., Ng R.T., Poosala V., Ross K.A., and Sevcik K.C. The new jersey data reduction report. IEEE Data Eng. Bull., 20(4):3–45, 1997.
4. Guha S., Rastogi R., and Shim K. CURE: an efficient clustering algorithm for large databases. In SIGMOD Conference. 1998, pp. 73–84.
5. Jolliffe I.T. Principal component analysis. Springer, Berlin, 1986.
6. Lelewer D.A. and Hirschberg D.S. Data compression. ACM Comput. Surv., 19(3):261–296, 1987, http://doi.acm.org/10.1145/45072.45074.
7. Poosala V., Ioannidis Y.E., Haas P.J., and Shekita E.J. Improved histograms for selectivity estimation of range predicates. In SIGMOD Conference. 1996, pp. 294–305.
8. The JPEG 2000 standard. http://www.jpeg.org/jpeg2000/index.html.
9. Zhang T., Ramakrishnan R., and Livny M. BIRCH: an efficient data clustering method for very large databases. In SIGMOD Conference. 1996, pp. 103–114.

Data Replication

Bettina Kemme
McGill University, Montreal, QC, Canada

Synonyms

Database replication; Replication

Definition

Using data replication, each logical data item of a database has several physical copies, each of them located on a different machine, also referred to as site or node. Depending on the context and the type of replication architecture, the term *replica* can refer to one of the physical copies of a particular data item, or to an entire site with all its data copies. Data replication can serve different purposes. First, it can be used to increase *availability* and provide *fault-tolerance* since the data can, in principle, be accessed as long as one replica is available. Second, it can provide good *performance*. By storing replicas close to users that want to access the data, replication allows *fast local access*. Third, access requests can be distributed across the replicas. By adding more replicas to the system a higher incoming workload can be handled, and hence, a higher throughput can be achieved. Thus, replication is a means to achieve *scalability*. Finally, for some applications, replication is a natural choice, e.g., if mobile users have to access data while disconnected from their corporate data server.

The main challenge of replication is that the replicas have to be kept consistent when updates occur. This is the task of *replica control*. It has to translate the read and write operations that users submit on the logical data items into operations on the physical copies. In the most common approach, read operations are performed on one replica while write operations have to be performed on all replicas (ROWA, or read-one-write-all approach). Ideally, all copies of a data item have the same value at all times. In reality, many different *consistency models* have been developed that reflect the needs for different application domains. Additionally, *replica allocation algorithms* have the task to decide where and when to install or remove replicas. They have to find a trade-off between the performance benefits for read operations and the overhead of keeping the copies consistent.

Historical Background

Data replication has gained attention in two different communities. The database community typically considers the replication of data items of a database, e.g., records or tables of a relational database. It has to consider that transactions and queries not only access individual data items, but read and write a whole set of related data items. In contrast, the distributed systems community mainly focuses on replication techniques for objects that are typically accessed individually, such as distributed file systems and web-servers. Nevertheless, in all the application types, similar issues arise regarding replica control and allocation, and the associated coordination and communication overhead. Thus, there has always been an active exchange of research ideas, such as [1,5], and there exist several publication venues where work from both communities appears.

Work on database replication had an early peak in the 1980s, where it was first introduced for availability purposes, and most approaches provided strong consistency properties. A good overview is given in [2]. A seminal paper by Gray et al. in 1996 [6] revived research in this area. It provided a first characterization of replica control algorithms and presented an analytical model showing that existing strong consistency solutions come with a large performance penalty. Since then, replication has remained an active research area. Emphasis has been put on reducing the high communication and coordination overhead of the early solutions. One research direction aims at reducing the costs by delaying the sending of updates to remote replicas. However, in this case, replicas might have stale or even inconsistent data. Solutions have been proposed to avoid inconsistencies (e.g., [3]), to provide limits on the staleness of the data (e.g., [8]), and to detect and then reconcile inconsistencies [9]. Another research direction has developed techniques to provide strong consistency guarantees at acceptable costs, for example, by taking advantage of multicast and group maintenance primitives provided by group communication systems [14].

In the distributed systems community, early work focused on replicated file systems (e.g., [10,13]). Later, web-server replication [12] and file replication in peer-2-peer systems, (e.g., [7]) have attained considerable attention. A wide range of consistency models has been defined to accommodate application needs. Also, a large body of literature exists regarding object replication for fault-tolerance purposes [4,11].

Foundations

Replica Control

Replica control, which has the task of keeping the copies consistent despite updates, is the main issue to be tackled by any replication solution. Replica control has to decide which data copies read operations should access, and when and how to update individual data copies in case of write operations. Thus, most of the work done in the area of database replication is to some degree associated with replica control. The entry *Replica Control* provides a detailed overview of the main challenges.

Replica Control and Concurrency Control

In order to work properly with a database system providing transactional semantics, replica control has to be integrated with concurrency control. Even in a nonreplicated and nondistributed system, as soon as transactions are allowed to execute concurrently, concurrency control mechanisms restrict how the read and write operations of different transactions may interleave in order to provide each transaction a certain level of isolation. If data items are now replicated, the issue becomes more challenging. In particular, different transactions might start their execution on different replicas making it difficult to detect conflicts.

For a nonreplicated database system, the most studied isolation level is *serializability*, which indicates that the concurrent execution of transactions should be equivalent to a serial execution of the same transactions. This is typically achieved via locking, optimistic, or multi-version concurrency control. Thus, one of the first steps in the research of replicated databases was to define a corresponding correctness criterion 1-copy-serializability, which requires that the concurrent execution of transactions over all replicas is equivalent to a serial execution over a single logical copy of the database. Many replica control algorithms have been developed to provide this correctness criterion, often extending the concurrency control algorithms of nonreplicated systems to work in a replicated environment.

In early solutions, replicas run some form of coordination during transaction execution to guarantee an appropriate serialization. This type of protocols is often called *eager* or *synchronous* since all replicas coordinate their operations before transaction commit. Textbooks on distributed systems typically contain a chapter on these replica control algorithms since they serve as a nice example of how to design coordination protocols in a distributed environment. A problem with most of these traditional replication solutions is that they induce a large increase in transaction response times which is often not acceptable from an application point of view.

More recent approaches addressed this issue and designed replica control algorithms providing 1-copy-serializability or other strong consistency levels that are tuned for performance. Many envision a cluster architecture, where a set of database replicas is connected via a fast local area network. In such a network, eager replication can be feasible since communication delays are short. Replication is used to provide both scalability and fault-tolerance. The entries *Replication based on Group Communication* and *Replication for Scalability* describe recent, efficient replica control algorithms providing strong consistency levels for cluster architectures.

Consistency Models and Conflict Resolution

By definition, eager replication incorporates coordination among replicas before transaction commit. Alternatively, *lazy* replication (also called asynchronous or optimistic) allows a transaction to commit data changes at one replica without coordination with other replicas. For instance, all update transactions could be executed and committed at a specific primary replica which then propagates changes to other replicas sometime after commit. Then, secondary replicas lag behind the current state at the primary. Alternatively, all replicas might accept and execute updates, and eventually propagate the changes to the rest of the replicas. In this case, replicas can become inconsistent. Conflicting updates are only detected after update propagation and inconsistent data has to be reconciled. In this context, a considerable body of research exists defining correctness criteria weaker than 1-copy-serializability. In particular, many formalisms exist that allow the defining of limits to the allowed divergence between the copies of a data item.

Availability

Replication can increase the availability of the system since, in principle, a data item is available as long as one replica is accessible. However, in practice, it is not trivial to design a high-available replication solution. As discussed before, most replication algorithms require all updates to be performed at all replicas (ROWA, or read-one-write-all). If this is taken in the strict sense, then, if one replica is update transactions cannot execute, and the availability observed by update transactions is actually lower than in a nonreplicated system. To avoid this, most replica control algorithms actually implement a read-one-write-all-available (ROWAA) strategy that only updates replicas that are actually accessible. When a replica recovers after a crash, it first has to get the current state from the available replicas. This can be a complex process. The possibility of network partitions imposes an additional challenge. Although a particular replica might be up and running, it might not be accessible because of an interruption in the network connectivity. Many commercial database systems offer specialized high-availability replication solutions. Typically, a primary database is replicated at a backup database system. All transactions are executed at the primary that sends updates to the backend. Only when the primary crashes, the backup takes over.

Quorum systems are an alternative high-availability replication approach. In quorum approaches both read and write operations have to access a quorum of data replicas. For example, a quorum could be a majority of replicas. This guarantees that any two operations overlap in at least one replica. Thus, each read operation reads at least one replica that has the most recent update, and any two concurrent write operations are serialized at least at one replica. There exist many ways to define quorums differing in the structure and the sizes of the quorums. Quorums are an elegant solution to network partitions and are attractive for write intensive applications since writes do not need to access all replicas. However, they have worse performance than ROWA for the more common read-intensive applications.

Replica Allocation

Using *full replication*, every site has copies of all existing data items. This simplifies the execution of read operations but has high update costs since all sites have to perform the updates for all data items. In contrast, in *partial replication* each site has only copies of some of the data items. The advantage is that an update on a data item does not lead to costs at all sites but only at those that contain a replica of the data item. However, read operations become more challenging. First, in a wide-area setting, if no local replica is available, read operations observe higher delays since they have to access a remote replica. Furthermore, if a request has to access several objects within a single query, the query might have to access data items on different sites, leading to distributed queries. Thus, replica allocation algorithms have to decide on the placement of replicas considering issues such as communication and update costs.

Related to replica allocation is the task to adjust the replication configuration automatically and dynamically to the needs of the application. This is particularily interesting in a cluster-based configuration where sites are located in a local area network and replication is used for load distribution and fault-tolerance. In here, configuration does not only relate to the number of replicas needed, but also to the question of how to distribute load across replicas, how to accomodate different applications on the replicated data, and how to optimally use all available resources. An important issue is that advanced information systems do not only have a database system but consist of a multi-tier architecture with web-servers, application servers, and database systems that interact with each other.

Materialized views are a special form of data replication, where the data retrieved by the most typical database queries is stored in a pre-formatted form. Typically, queries can be run over materialized views as if they were base tables. Since the data is already in the format needed by the query, query processing time can be considerably reduced. However, updates on the views are usually disallowed but have to be performed directly on the base tables. Special refresh algorithms then guarantee that materialized views are updated when the base data changes.

Replication in Various Computing Environments

Replication is a fundamental technique for data management that can be applied in various computing environments. The different purposes replication can generally have in LANs (clusters) and in wide-area environments have already been discussed above.

Peer-to-peer (P2P) networks are a special form of wide-area environment. In here, each site is both client and server. For instance, each peer can provide storage space to store documents or data items that can be queried by all peers in the system. Or it provides processing capacity that can be used by other peers to perform complex calculations. In turn, it can request data items or processing capacity from other peers. A large body of research has developed algorithms that decide where to store data items and how to find them in the P2P network. Replication plays an important task for fault-tolerance, fast access, and load distribution.

Replication also plays an important role in mobile environments that differ in some fundamental ways from wired networks. Firstly, communication between mobile units and the servers on the standard network is typically slow and unreliable. Secondly, mobile devices are usually less powerful and have considerably less storage space than standard machines leading to an asymmetric architecture. Furthermore, mobile devices, such as laptops, are often disconnected from the network and only reconnect periodically. Thus, having replicas locally on the mobile device provides increased availability during disconnection periods.

Key Applications

Data replication is widely used in practice. Basically, all database vendors offer a suite of replication solutions. Additionally, replication is often implemented ad-hoc at the application layer or as a middleware layer as the need arises. Replication is used in many application domains. Below some examples are listed.

- Companies use high-availability replication solutions for their mission critical data that has to be available 24/7. Examples are banking or trading applications.
- Companies use cluster replication, ideally with autonomic behavior, in order to provide a scalable and fault-tolerant database backend for their business applications. In particular companies that do e-business with a large number of users resort to database replication to be able to obtain the required throughput. This also includes techniques such as materialized views.
- Globally operating companies often have databases located at various sites. Parts of these databases are replicated at the other locations for fast local access. Examples are companies maintaining warehouses at many locations.
- Replication of web-sites is a common technique to achieve load-balancing and fast local access. As the information shown on the web becomes more and more dynamic (i.e., it is retrieved from the database in real-time), database replication techniques need to be applied.
- Companies that have employees working off-site, such as consulting or insurance companies, use mobile replication solutions. Data replicas are downloaded to mobile units such as laptops in order to work while disconnected. Upon reconnection to the network, the replicas are reconciled with the master replica on the database server. Typically, database vendors provide specialized software in order to allow for such types of replication.
- There exist many P2P-based document sharing systems, e.g., to share music, video files. Entire file systems can be built on P2P networks.

- Data Warehouses can be seen as a special form of replication where the transactional data is copied and reformatted to be easily processed by data analysis tools.

Future Directions

Being a fundamental data management technology, data replication solutions will need to be adjusted and revisited whenever new application domains and computing environments are developed. Thus, data replication will likely be a topic that will be further explored as our IT infrastructure and our demands change.

Cross-references

► Autonomous Replication
► Consistency Models for Replicated Data
► 1-Copy-Serializability Consistency
► Data Broadcasting, Caching and Replication
► Distributed Database Design
► Middleware Support for Database Replication and Caching
► Optimistic Replication and Resolution
► Partial Replication
► P2P Storage
► Quorum Systems
► Replication
► Replica Control
► Replica Freshness
► Replication based on Group Communication
► Replication for High Availability
► Replication for Scalability
► Replication in Multi-Tier Architectures
► Strong Consistency Models for Replicated Data
► Traditional Concurrency Control for Replicated Database
► WAN Data Replication
► Weak Consistency Models for Replicated Data

Recommended Reading

1. Alonso G., Charron-Bost B., Pedone F., and Schiper A. (eds.), A 30-year Perspective on Replication, Monte Verita, Switzerland, 2007.
2. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, Boston, MA, 1987.
3. Breitbart Y., Komondoor R., Rastogi R., Seshadri S., and Silberschatz A. Update Propagation Protocols For Replicated Databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 97–108.
4. Budhiraja N., Marzullo K., Schneider F.B., and Toueg S. The primary-backup approach. In Distributed Systems (2nd edn.). S. Mullender (ed.). Addison Wesley, New York, NY, pp. 199–216.
5. Cabrera L.F. and Pâris J.F. (eds.), Proceedings of the First Workshop on the Management of Replicated Data, IEEE Computer Society Press, 1990.
6. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc, ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
7. Lv Q., Cao P., Cohen E., Li K., and Shenker S. Search and replication in unstructured peer-to-peer networks. In Proceedings of the 16th Annual International Conference on Supercomputing. New York, NY, 2002, pp. 84–95.
8. Röhm U., Böhm K., Schek H.J., and Schuldt H. FAS - a freshness-sensitive coordination middleware for a cluster of OLAP components. In Proc. Int. Conf. on Very Large Data Bases (VLDB). Hong Kong, China, 2002, pp. 754–765.
9. Saito Y. and Shapiro M. Optimistic replication. ACM Comput. Surv., 37(1):42–81, 2005.
10. Satyanarayanan M., Kistler J.J., Kumar P., Okasaki M.E., Siegel E.H., and Steere D.C. Coda: a highly available file system for a distributed workstation environment. IEEE Trans. Comput., 39 (4):447–459, 1990.
11. Schneider F.B. Replication management using the state-machine approach. In Distributed Systems (2nd edn.), S. Mullender (ed.). Addison Wesley, New York, NY, 1993, pp. 169–198.
12. Sivasubramanian S., Szymaniak M., Pierre G., and van Steen M. Replication for web hosting systems. ACM Comput. Surv., 36 (3):291–334, 2004.
13. Terry D.B., Theimer M., Petersen K., Demers A.J., Spreitzer M., and Hauser C. Managing update conflicts in Bayou, a weakly connected replicated storage system. In Proc. 15th ACM Symp. on Operating System Principles (SOSP). Copper Mountain Resort, CO, 1995, pp. 172–183.
14. Wiesmann M. and Schiper A. Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng., 17(4):551–566, 2005.

Data Replication Protocols

► Replica Control

Data Sampling

QING ZHANG
The Australian e-health Research Center, Brisbane, QLD, Australia

Definition

Repeatedly choosing random numbers according to a given distribution is generally referred to as *sampling*. It is a popular technique for data reduction and

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:16

approximate query processing. It allows a large set of data to be summarized as a much smaller data set, the *sampling synopsis*, which usually provides an estimate of the original data with provable error guarantees. One advantage of the sampling synopsis is easy and efficient. The cost of constructing such a synopsis is only *proportional to the synopsis size*, which makes the sampling complexity potentially sublinear to the size of the original data. The other advantage is that the sampling synopsis represents parts of the original data. Thus many query processing and data manipulation techniques that are applicable to the original data, can be directly applied on the synopsis.

Historical Background

The notion of representing large data sets through small samples dates back to the end of nineteenth century and has led to the development of many techniques [5]. Over the past two decades, sampling techniques have been greatly developed in various database areas, especially on query optimization and approximate query processing.

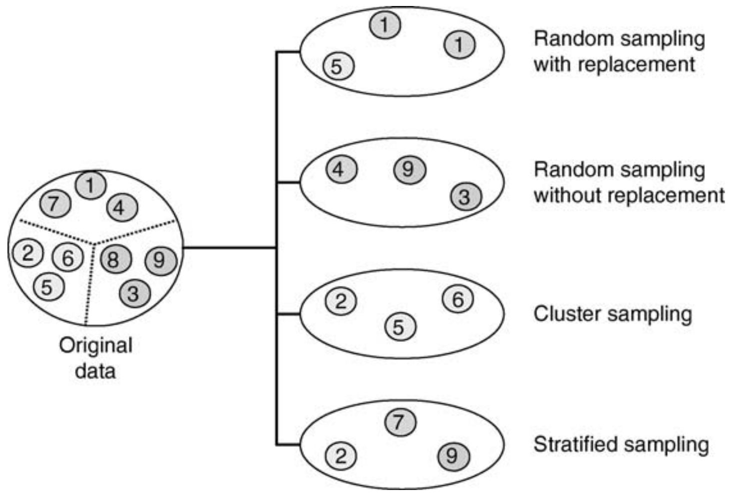
- *Query Optimization*: Query optimizer identifies an efficient execution plan for evaluating the query. The optimizer generates alternative plans and chooses the cheapest one. It uses the statistical information stored in the system catalog to estimate the cost of a plan. Sampling synopsis plays a critical role in the query optimization of a RDBMS. Some commercial products, such as DB2 and Oracle, have already adopted sampling techniques to estimate several catalog statistics. In the Heterogeneous DBMS, the global query optimizer also employs sampling techniques to estimate query plans when some local statistical information is unavailable [6].
- *Approximate Query Processing*: Sampling is mainly used to generate approximate numeric answers for aggregate queries over a set of records, such as COUNT, SUM, MAX, etc. Compared with other approximate query processing techniques, such as histogram and wavelet, sampling is easy to be implemented and efficient to generate approximate answers with error guarantees. Many prototypes on approximate query processing have adopted sampling approaches [2,3,4].

Foundations

A sampling estimation can be roughly divided into two stages. The first stage is to find a suitable sampling method to construct the sampling synopsis from the original data set. The second stage is to analyze the sampling estimator to find the characteristics (bounds and parameters) of its distribution.

Sampling Method: Existing sampling methods can be classified into two groups, the *uniform random sampling* and *biased sampling*. The uniform random sampling is a straightforward solution. Every tuple of the original data has the same probability to be sampled. Thus for aggregate queries, the estimation from samples is the expected value of the answer. Due to the usefulness of uniform random sampling, commercial DBMSs have already supports operators to collect uniform samples. However, there are some queries for which the uniform random sampling are less effective on estimation. Given a simple group-by query which intends to find the average value of different groups, a smaller group is often as important to the user as those larger groups. It is obvious that the uniform random sampling will not have enough information for the smaller group. That is why the biased sampling methods are developed in these cases. Stratified sampling, for example, is a typical biased sampling, which will be explained in detail later. The four basic sampling methods, two uniform sampling and two biased sampling, are listed below. Figure 1 shows corresponding sampling synopses generated by those methods on the sample data.

1. *Random sampling with replacement*: This method creates a synopsis by randomly drawing n of the N original data records, where the probability of drawing any record is $\frac{1}{N}$. In other words, the records that have already been drawn are not to be remembered. So the chance exists that a certain record will be repeatedly drawn in several runs.
2. *Random sampling without replacement*: This is similar to the random sampling with replacement method except that in each run the drawn record will be remembered. That is, the same record will not be chosen on subsequent runs. Although sampling without replacement appears to lead to better approximation results, sampling with replacement is significantly easier to be implemented and analyzed. Thus in practice the negligible difference between these two methods' effects makes the sampling with replacement a desirable alternative.
3. *Cluster sampling*: The N original data records are grouped into M mutually disjoint clusters. Then a



Data Sampling. Figure 1. Sampling methods (sampling synopsis size = 3).

random sampling on M clusters is obtained to form the cluster sampling synopsis. That is, the clusters are treated as sampling units so statistical analysis is done on a population of clusters.

4. *Stratified sampling*: Like the cluster sampling, the N records are grouped into M mutually disjoin clusters, called *strata*. A stratified sampling synopsis is generated through running a random sampling on each cluster. This method is especially useful when the original data has skew distribution. In this way, the cluster with smallest number of records will be sure to be represented in the synopsis.

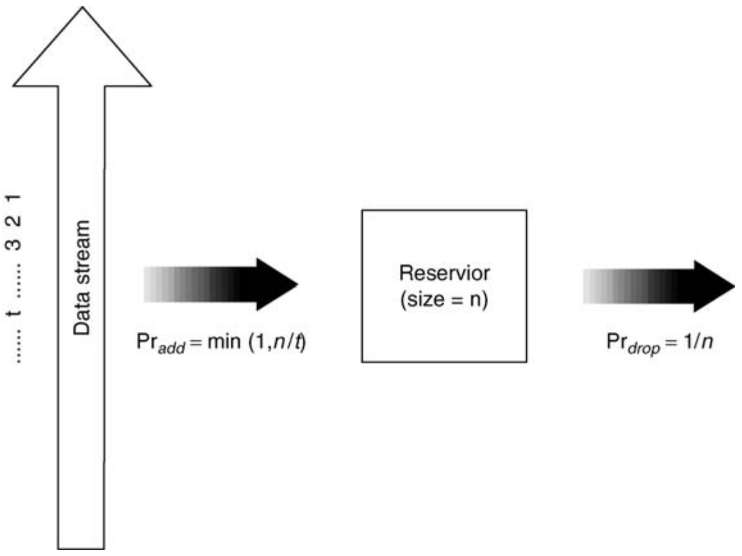
These basic sampling methods are straightforward solutions although they usually can not satisfy the error or space requirements. However, they are building blocks of more advanced methods that can either be used in certain situations or guarantee the estimation error with a confidence level.

Here is an example of a specially designed sampling method, which extends the basic random sampling method to be usable in the data stream environment. Note that the basic unbiased random sampling method requires a fixed data set with pre-defined data size. However, in a data stream environment, the size of the original data set is unknown. Thus the dynamic sampling method is required to get an unbiased sampling synopsis over the whole data stream. For this purpose, reservoir based sampling methods were originally proposed in [7].

Suppose constructing an unbiased sampling synopsis against a data stream T. A sample reservoir of n records is maintained from the stream. That is the first n records of T will be added to the reservoir for initialization. Any $t - th$ new coming record will be added to the reservoir with probability $\frac{n}{t}$. If a new record is added to the reservoir, any existing records of the reservoir will be discarded with probability $\frac{1}{n}$. Figure 2 demonstrates the construction steps of the reservoir. Finally, when all data of T has been processed, the n records of the reservoir form an unbiased random sampling synopsis of all the records of T. Similar reservoir based sampling method can also be developed for biased sampling [1].

Sampling Analysis : This stage analyzes the random variable generated by sampling methods. More specifically, it analyzes the distribution of the random variable through discovering its bound and distribution parameters. Given N records, assume that the function $f(N)$ represents an operation on the N records. Let S represent a sampling synopsis of N, and $f(S)$ is often closely related to $f(N)$ for most common operations, such as AVERAGE or MAX. Let $X = f(S)$. X is the random variable that are going to be analyzed. If $f(N)$ represents some linear aggregation functions, such as AVERAGE, X can be approximated as a normal distribution, according to the *Central Limit Theorem*. If however, $f(N)$ represents other functions, such as MAX, probabilistic bounds based on key distribution parameters, such as expectation $E(X)$ and variance $Var[X]$, need to be found. This is often quite acceptable as an alternative to characterize the entire distribution of X.

There exist a number of inequalities to estimate the probabilistic bound. These inequalities are collectively



Data Sampling. Figure 2. Random sampling with a reservoir.

known as tail bounds. Given a random variable X, if $E[X]$ is known, *Markov's Inequality* gives:

$$\forall\ a > 0, \quad \Pr(X \geq a) \leq \frac{E[X]}{a}$$

The variance of X, $Var[X]$ is defined as:

$$Var[X] = E[(X - E[X])^2]$$

A significantly stronger tail bound can be obtained by *Chebyshev's Inequality* if $Var[X]$ is known:

$$\forall\ a > 0, \quad \Pr(|X - E[X]| \geq a) \leq \frac{Var[X]}{a^2}$$

The third inequality is an extremely powerful tool called *Chernoff bounds*, which gives exponentially decreasing bounds on the tail distribution. These are derived by applying Markov's Inequality to $e^{tX}$for some well-chosen value t. Bounds derived from this approach are generally referred to collectively as Chernoff bounds. The most commonly used version of the Chernoff bound is for the tail distribution of a sum of independent 0–1 random variable. Let $X_1, \ldots, X_n$ be independent Poisson trials such that $\Pr(X_i) = p_i$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. For $0 < \delta < 1$,

$$\Pr(|X - \mu| \geq \mu\delta) \leq 2e^{-\mu\delta^2/3}$$

Finally, an example to illustrate the different tail bounding abilities of the three inequalities is given below. Suppose estimating the synopsis size generated by a random sampling with replacement of N data records. Each record has a same probability $\frac{1}{2}$ to be sampled. Let X denote the size of the sampling synopsis. Then the size expectation is $E[X] = \frac{N}{2}$. The probabilities of the synopsis size greater than $\frac{3}{4}N$, under estimations from different inequalities, are:

$$\text{Markov's Inequality}: \Pr\left(X \geq \tfrac{3}{4}N\right) \leq \tfrac{2}{3}$$

$$\text{Chebyshev's Inequality}: \Pr\left(X \geq \tfrac{3}{4}N\right) \leq \tfrac{4}{N}$$

$$\text{Chernoff Bounds}: \Pr\left(X \geq \tfrac{3}{4}N\right) \leq 2e^{-N/24}$$

Key Applications

Query Optimization

Data sampling is one of the primary techniques used by query optimizers. In some multi-dimensional cases, it becomes the only easy and viable solution.

Approximate Query Processing

Data sampling is one of the three major data deduction techniques (the other two are histogram and wavelet) employed by approximate query processors.

Data Streaming

Sampling is a simple yet powerful method for synopsis construction in data stream.

Cross-references

► Approximate Query Processing
► Data Reduction
► Data Sketch/Synopsis
► Histogram
► Query Optimization

Recommended Reading

1. Aggarwal C.C. On biased reservoir sampling in the presence of stream evolution. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.
2. Chaudhuri S. et al. Overcoming limitations of sampling for aggregation queries. In Proc. 17th Int. Conf. on Data Engineering, 2001.
3. Ganti V., Lee M.-L., and Ramakrishnan R. ICICLES: Self-tuning samples for approximate query answering. In Proc. 28th Int. Conf. on Very Large Data Bases. 2000.
4. Gibbons P.B. and Matias Y. 1New sampling-based summary statistics for improving approximate query answers. In Proc. ACM SIGMOD int. conf. on Management of data, 1998.
5. Kish L. Survey Sampling. Wiley, New York, xvi, 643, 1965.
6. Speegle G.D. and Donahoo M.J. Using statistical sampling for query optimization in heterogeneous library information systems. In Proc. 20th ACM Annual Conference on Computer science, 1993.
7. Vitter J.S. Random sampling with a reservoir. ACM Trans. Math. Softw., 11(1):37–57, 1985.

Data Sketch/Synopsis

XUEMIN LIN
University of New South Wales, Sydney, NSW, Australia

Synonyms

Summary

Definition

A *synopsis* of dataset D is an abstract of D. A *sketch* is also referred to an abstract of dataset D but is usually referred to an abstract in a sampling method.

Key Points

Sketch/synopsis techniques have many applications. They are mainly used for statistics estimation in query processing optimization and for supporting on-line data analysis via approximate query processing. The goal is to develop effective and efficient techniques to build a small space synopsis while achieving high precision. For instance, a key component in query processing optimization is to estimate the result sizes of queries. Many techniques [1,2] have been developed for this purpose, including histograms, wavelets, and join synopses.

In data stream applications, the space requirements of synopses/sketches are critical to keep them in memory for on-line query processing. Streams are usually massive in size and fast at arrival rates; consequently it may be infeasible to keep a whole data stream in memory. Many techniques [3] have been proposed with the aim to minimize the space requirement for a given precision guarantee. These [3] include heavy hitter, quantiles, duplicate-insensitive aggregates, joins, data distribution estimation, etc.

Cross-references

► Approximate Query Processing
► Histograms on Streams
► Wavelets on Streams

Recommended Reading

1. Alon N., Gibbons P.B., Matias Y., and Szegedy M. Tracking join and self-join sizes in limited storage. In Proc. 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS, 1999.
2. Gibbons P.B. and Matias Y. Synopsis data structures for massive data sets. In Proc. ACM-SIAM Symposium on Discrete Algorithms, SODA, 1999.
3. Zhang Y., Lin X., Xu J., Korn F., and Wang W. Space-efficient relative error order sketch over data streams. In Proc. 22nd Int. Conf. on Data Engineering, ICDE, 2006.

Data Skew

LUC BOUGANIM
INRIA, Rocquencourt, Le Chesnay, France

Synonyms

Biased distribution; Non-uniform distribution

Definition

Data skew primarily refers to a non uniform distribution in a dataset. Skewed distribution can follow common distributions (e.g., Zipfian, Gaussian, Poisson), but many studies consider Zipfian [3] distribution to model skewed datasets. Using a real bibliographic database, [1] provides real-world parameters for the Zipf distribution model. The direct impact of data skew on parallel execution of complex database queries is a poor load balancing leading to high response time.

Key Points

Walton et al. [2] classify the effects of skewed data distribution on a parallel execution, distinguishing *intrinsic skew* from *partition skew*. Intrinsic skew is skew inherent in the dataset (e.g., there are more citizens in Paris than in Waterloo) and is thus called *Attribute value skew (AVS)*. Partition skew occurs on parallel implementations when the workload is not evenly distributed between nodes, even when input data is uniformly distributed. Partition skew can further be classified in four types of skew. *Tuple placement skew (TPS)* is the skew introduced when the data is initially partitioned (e.g., with range partitioning). *Selectivity skew (SS)* is introduced when there is variation in the selectivity of select predicates on each node. *Redistribution skew (RS)* occurs in the redistribution step between two operators. It is similar to TPS. Finally *join product skew (JPS)* occurs because the join selectivity may vary between nodes.

Cross-references

► Query Load Balancing in Parallel Database Systems

Recommended Reading

1. Lynch C. Selectivity estimation and query optimization in large databases with highly skewed distributions of column values. Proc. 14th Int. Conf. on Very Large Data Bases, 1988, pp. 240–251.
2. Walton C.B., Dale A.G., and Jenevin R.M. A taxonomy and performance model of data skew effects in parallel joins. Proc. 17th Int. Conf. on Very Large Data Bases, 1991, pp. 537–548.
3. Zipf G.K. Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology. Addison-Wesley, Reading, MA, 1949.

Data Sorts

► Data Types in Scientific Data Management

Data Standardization

► Constraint-Driven Database Repair

Data Storage and Indexing in Sensor Networks

Philip B. Gibbons
Intel Research Pittsburgh, Pittsburgh, PA, USA

Definition

Sensor data can either be stored local to the sensor node that collected the data (*local storage*), transmitted to one or more collection points outside of the sensor network (*external storage*), or transmitted and stored at other nodes in the sensor network (*in-network storage*). There are trade-offs with each of these approaches, as discussed below, depending on the volume of data collected at each sensor node, the query workload, and the resource limitations of each node. Moreover, the local and in-network storage scenarios often require *in-network indexes* in order to reduce the overheads of answering queries on data stored within the sensor network. Such indexes can be classified as either *exact-match indexes* or *range indexes*.

Historical Background

External storage is in some sense the default approach for sensor networks, reflecting the common scenario in which the application is interested in all the collected sensor readings. Early work in *local storage* includes Cougar [11] and Tined [8]; both push SQLS-style queries out to data stored at the sensor nodes. In Tined and in Directed Diffusion [6] (another early work) the query workload dictates which sensors are turned on. This functionality can be used for external, local, or in-network storage, depending on where these collected data get stored. Seminal work on *in-network storage* includes the work on geographic hash tables (GHATS) [10], which support exact-match indexing. The authors advocate *data-centric storage*, a class of in-network storage in which data are stored according to named attribute values. Early work on supporting range indexes for in-network storage includes DIFFS [5] and DIM [7].

Foundations

External Storage, in which all the sensor readings are transmitted to collection points outside of the sensor network, has several important advantages. First, storage is plentiful outside the sensor network, so that all the data can be archived, as well as disseminated to any interested party (e.g., posted on the web). Archiving all the data is quite useful for testing out new theories and models on these historical data, and for forensic activities. Second, processing power, memory, and energy are plentiful outside the sensor network, so that queries and complex data analysis can be executed quickly and without exhausting the sensors' limited energy reserves. Finally, such data processing can be done using standard programming languages and tools (such as Matlow) that are not available on sensor nodes. On the other hand, external storage suffers the disadvantage that it incurs the costs (primarily energy, but also bandwidth and latency) of transmitting all the data to outside the network.

Local storage, in which sensor readings are stored local to the node that collected the data, avoids the costs of transmitting all the data. Instead, it incurs the costs of pushing queries into the network and returning the query answers. Queries are often flooded through the sensor network. A *collection tree* is constructed hop-by-hop from the query source (called the *root*), as follows [8]. The root broadcasts the query, and each sensor node that hears the broadcast makes the root its parent in the tree. These nodes in turn broadcast the query, and any node that hears one or more broadcasts (and is not yet in the tree) selects one of these nodes as its parent, and so on. This tree is used to collect query answers: Each leaf node sends its piece of the answer to its parent, and each internal node collects these partial answers from its children, combines them with its own piece of the answer, and sends the result to its parent. The process proceeds level-by-level up the tree to the root. Thus, the cost of pushing the query and gathering the answer can be high.

Nevertheless, the amount of data transmitted when using local storage can often be far less than when using external storage. First, queries are often long running (*continuous queries*); for such queries, the costs of query flooding and tree construction are incurred only once at the start of the query (although maintaining the tree under failures can incur some additional costs). Second, indexes can be used (as discussed below) to narrow the scope of the query to a subset of the nodes. Third, queries can be highly selective (e.g., looking for rare events such as an intruder sighting), so that most sensors transmit little or no real data. In camera sensor networks (e.g., Irises [4]), local filtering of images can result in query answers that are orders of magnitude smaller than the raw images. Fourth, many queries are amenable to efficient *in-network aggregation*, in which partial answers received from children can be combined into a single fixed-sized packet. For example, in a Sum query each internal node can send to its parent a single value equal to the sum of its value and the values received from its children. Finally, in sensor networks supporting queries of live data only (i.e., only the latest data are of interest), the amount of data sensed can far exceed the amount of data queried.

A key consideration when using local storage is that the amount of such storage is limited. Thus, at some point old data need to be discarded or summarized to make room for new data [2]. Moreover, the local storage is often flash memory, and hence flash-friendly techniques are needed to minimize the costs for accessing and updating locally stored data [9].

In-network storage, in which sensor readings are transmitted and stored at other nodes in the sensor network, falls in between the extremes of external storage and local storage. Caching data that passes through a sensor node during query processing is a simple form of in-network storage. As cached data become stale over time, care must be taken to ensure that the data meets the query's freshness requirements [4]. In Tined [8], "storage point" queries can be used to collect data satisfying a query (e.g., all temperature readings in the past 8 seconds, updated every second) at nodes within the network. In *data-centric storage* [10], data are stored according to named attribute values; all data with the same general name (e.g., intruder sightings) are stored at the same sensor node. Because data items are stored according to their names, queries can retrieve all data items associated with a target name from just a single "home" node for that name (as opposed to potentially all the nodes when using local storage). The approach relies on building and maintaining indexes on the names, so that both sensor readings and queries can be routed efficiently to the corresponding home node.

In-network indexes, in which the index is maintained inside the sensor network, are useful for both local storage and in-network storage. The goal is to map named data to the sensor node(s) that hold such data, in order to minimize the cost of answering queries. Queries using the index are guided to the sensor nodes holding the desired data. In Tined each internal node of a collection tree maintains a lower and upper bound on the attribute values in the subtree rooted at the node; this index is used to restrict the query processing to only subtrees containing the value(s) of interest. In Directed Diffusion, queries expressing interests in some target data are initially flooded out from the query node. Paths leading to nodes that are sources of the target data are reinforced, resulting in an ad hoc routing tree from the sources back to the query node. This can be viewed as an ad hoc query-specific index.

A *geographic hash table* (GHAT) [10] is an exact-match in-network indexing scheme for data-centric storage. A GHAT hashes a data name (called a *key*) to a random geographic location within the sensor field. The home node for a key is the sensor node that is geographically closest to the location returned by the hash of the key. (For fault tolerance and to mitigate hot spots, the data are also replicated on nearby nodes.) *Geographic routing* is used to route sensor readings and queries to the home node. In geographic routing each node knows its geographic coordinates and the coordinates of its neighbors. Upon receiving a packet, a node forwards the packet to the neighbor closest to the home node. In this way, the packet attempts to take a shortest path to the home node. The packet can get stuck, however, in a local minimum node v such that no neighbor of v is closer to the home node than v itself. Different geographic routing schemes provide different approaches for recovering from local minima, so that the home node can always be reached. Follow-on work on GHATS (e.g., [1]) has focused on improving the practicality (efficiency, robustness, etc.) of geographic routing and hence GHATS.

In-network storage based on a GHAT is less costly than local storage whenever the savings in data transmitted in querying exceed the additional costs of transmitting sensor data to home nodes. In a square sensor field of n sensors, routing to a home node takes $O(\sqrt{n})$ hops. Consider a workload where the transmitted sensor data is limited to E interesting events and there are Q queries each requesting all the events for a distinct named event type. With in-network storage based on a GHAT, the total hops is $O(\sqrt{n}(Q+E))$. With local storage, the total hops is $O(Qn)$, as it is dominated by the cost to flood the query. Thus, roughly, the in-network scheme is less costly when the number of events is at most a factor of $\sqrt{n}$ larger than the number of queries. However, this is in many respects a best case scenario for in-network storage, and in general, local or external storage can often be less costly than in-network storage. For example, with a single continuous query that aggregates data from all the sensor nodes for $t \gg 1$ time periods (i.e., $E = tn$), in-network storage based on a GHAT incurs $O(tn^{1.5})$ total hops while local storage with in-network aggregation incurs only $O(tn)$ total hops, as the cost is dominated by the t rounds of hops up the collection tree.

Moreover, a GHAT is not well-suited to answering range queries. To remedy this, a variety of data-centric storage schemes have been proposed that provide effective range indexes [5,7,3]. DIM [7], for example, presents a technique (inspired by k-d trees) for constructing a locality-preserving geographic hash function. Combined with geographic routing, this extends the favorable scenarios for in-network storage to include also multi-dimensional range queries.

In summary, which of the external, local, or in-network storage schemes is preferred depends on the volume of data collected at each sensor node, the query workload, and the resource limitations of each node.

Key Applications

Sensor networks, Applications of sensor network data management.

Cross-references

► Ad-Hoc Queries in Sensor Networks
► Applications of Sensor Network Data Management
► Continuous Queries in Sensor Networks
► Data Acquisition and Dissemination in Sensor Networks
► Data Aggregation in Sensor Networks
► Data Compression in Sensor Networks
► Data Fusion in Sensor Networks
► In-Network Query Processing
► Sensor Networks

Recommended Reading

1. Ee C.T., Ratnasamy S., and Shenker S. Practical data-centric storage. In Proc. 3rd USENIX Symp. on Networked Systems Design & Implementation, 2006, pp. 325–338.
2. Ganesan D., Greenstein B., Estrin D., Heidemann J., and Govindan R. Multiresolution storage and search in sensor networks. ACM Trans. Storage, 1(3):277–315, 2005.
3. Gao J., Guibas L.J., Hershberger J., and Zhang L. Fractionally cascaded information in a sensor network. In Proc. 3rd Int. Symp. Inf. Proc. in Sensor Networks, 2004, pp. 311–319.
4. Gibbons P.B., Karp B., Ke Y., Nath S., and Seshan S. IrisNet: An architecture for a worldwide sensor web. IEEE Pervasive Comput, 2(4):22–33, 2003.
5. Greenstein B., Estrin D., Govindan R., Ratnasamy S., and Shenker S. DIFS: A distributed index for features in sensor

networks. In Proc. First IEEE Int. Workshop on Sensor Network Protocols and Applications, 2003, pp. 163–173.

6. Intanagonwiwat C., Govindan R., Estrin D., Heidemann J., and Silva F. Directed diffusion for wireless sensor networking. IEEE/ACM Trans. Network., 11(1):2–16, 2003.
7. Li X., Kim Y.J., Govindan R., and Hong W. Multi-dimensional range queries in sensor networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003, pp. 63–75.
8. Madden S.R., Franklin M.J., Hellerstein J.M., and Hong W. TinyDB: An acquisitional query processing system for sensor networks. ACM Trans Database Syst, 30(1):122–173, 2005.
9. Mathur G., Desnoyers P., Ganesan D., and Shenoy P. Capsule: An energy-optimized object storage system for memory-constrained sensor devices. In Proc. 4th Int. Conf. on Embedded Networked Sensor Systems, 2006, pp. 195–208.
10. Ratnasamy S., Karp B., Shenker S., Estrin D., Govindan R., Yin L., and Yu F. Data-centric storage in sensornets with GHT, a geographic hash table. Mobile Networks Appl., 8(4):427–442, 2003. Springer.
11. Yao Y. and Gehrke J. Query processing for sensor networks. In Proc. First Biennial Conf. on Innovative Data Systems Research, 2003.

Data Stream

Lukasz Golab
AT&T Labs-Research, Florham Park, NJ, USA

Synonyms

Continuous data feed

Definition

A data stream *S* is an ordered collection of data items, $s_1, s_2, \ldots$, having the following properties:

- Data items are continuously generated by one or more sources and sent to one or more processing entities.
- The arrival order of data items cannot be controlled by the processing entities.

For instance, an Internet Service Provider (ISP) may be interested in monitoring the traffic on one or more of its links. In this case, the data stream consists of data packets flowing through the network. The processing entities, e.g., monitoring software, may be located directly on routers inside the ISP's network or on remote nodes.

Data streams may be classified into two types: based and derived. A base stream arrives directly from the source. A derived stream is a pre-processed base stream, e.g., an intermediate result of a query or a sub-query over one or more base streams. In the network monitoring scenario, the base stream corresponds to the actual IP packets, whereas a derived stream could contain aggregate measurements of traffic between each source and destination in a five-minute window.

Key Points

Depending upon the application, a data stream may be composed of raw data packets, relational tuples, events, pieces of text, or pieces of an XML document.

Furthermore, each data stream item may be associated with two timestamps: generation time (assigned by the source) and arrival time (assigned by the processing entity). The order in which items arrive may be different from their generation order, therefore these two timestamps may produce different orderings of the data stream.

A data stream may arrive at a very high speed (e.g., a router may process hundreds of thousands of packets per second) and its arrival rate may vary over time. Hence, a data stream may be unbounded in size. In particular, the processing entity may not know if and when the stream "ends."

Cross-references

► Stream-oriented query languages & operators
► Stream processing
► One-pass algorithm
► Stream mining
► Synopsis structure

Recommended Reading

1. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data streams. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. Madison, WI, 2002, pp. 1–16.
2. Golab L. and Özsu M.T. Issues in data stream management. ACM SIGMOD Rec., 32(2):5–14, 2003.
3. Muthukrishnan S. Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci., 1(2):1–67, 2005.

Data Stream Algorithm

► One-Pass Algorithm

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:17

Data Stream Management Architectures and Prototypes

Yanif Ahmad, Uğur Çetintemel
Brown University, Providence, RI, USA

Definition

Data stream processing architectures perform database-style query processing on a set of continuously arriving input streams. The core query executor in this type of architecture is designed to process continuous queries, rather than ad-hoc queries, by pushing inputs through a series of operators functioning in a pipelined and potentially non-blocking manner. Stream processing applications perform explicit read and write operations to access storage via asynchronous disk I/O operations. Other architectural components that differ significantly from standard database designs include the stream processor's scheduler, storage manager and queue manager.

Historical Background

Support database-style query processing for long-running applications that operate in high (data) volume environments, and impose high throughput and low latency requirements on the system. There have been several efforts from both the academic and industrial communities at developing functional prototypes of stream processing engines, to demonstrate their usefulness and to better understand the challenges posed by data stream applications. The first general purpose relational stream processing architectures appeared from the research community in 2001–2002, while the initial industrial offerings began to appear in 2003–2004. As a historical note, non-relational approaches to stream or event processing have existed in many forms prior to the early 2000s, especially in the embedded systems and signal processing communities.

Foundations

An SPE has a radically different architecture than that of a traditional database engine. Conceptually, the architectural differences can be captured by the following key characteristics:

1. Continuous query model
2. Inbound processing model
3. Single-process model

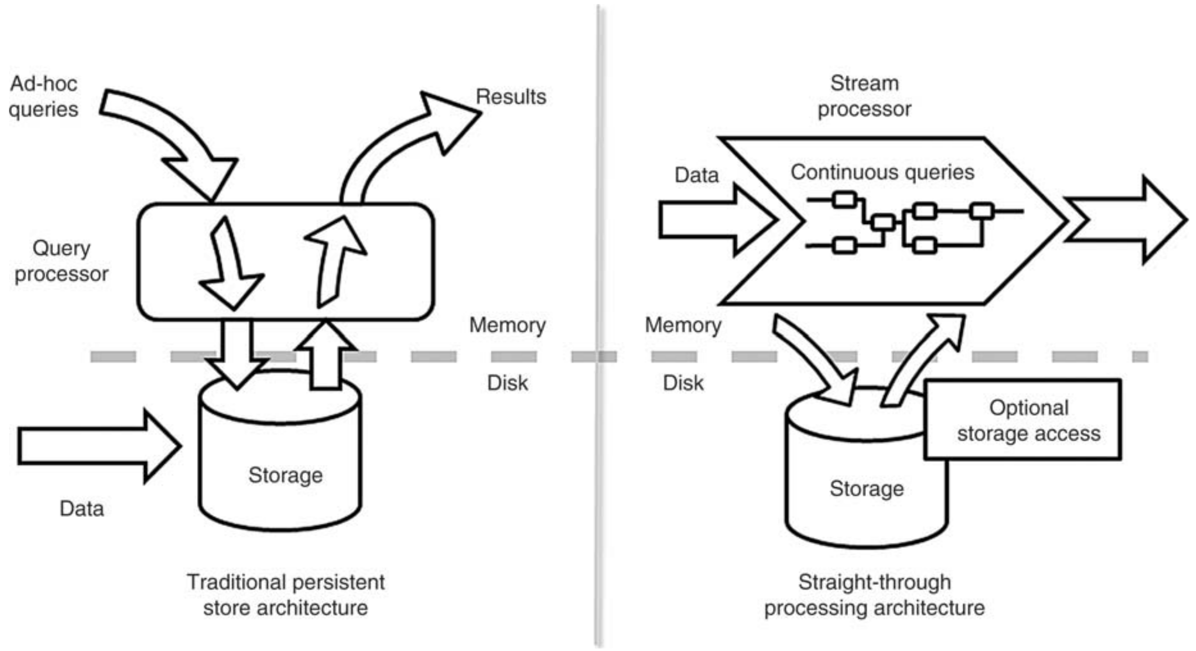
In the continuous query model, the queries execute continuously as new input data becomes available. This contrasts with the prevailing one-time query model where users (or clients) issue queries that process the available input data and produce one-time results. In other words, in the continuous model, the queries are persistent and input data is transient, whereas in the traditional model, queries are transient and input data is persistent, as illustrated in Fig. 1.

An SPE supports inbound processing, where incoming event streams immediately start to flow through the query processing operators as they enter the system. The operators process the events as they move, continuously producing results, all in main memory when possible. Read or write operations to storage are optional and can be executed asynchronously in many cases, when they are present. Inbound processing overcomes a fundamental limitation of the traditional outbound processing model, employed by all conventional database management systems, where data are always inserted into the database (usually as part of a transaction) and indexed as a first step before any processing can take place to produce results. By removing the storage from the critical path of processing, an SPE achieves significant performance gains compared to the traditional outbound processing approach.

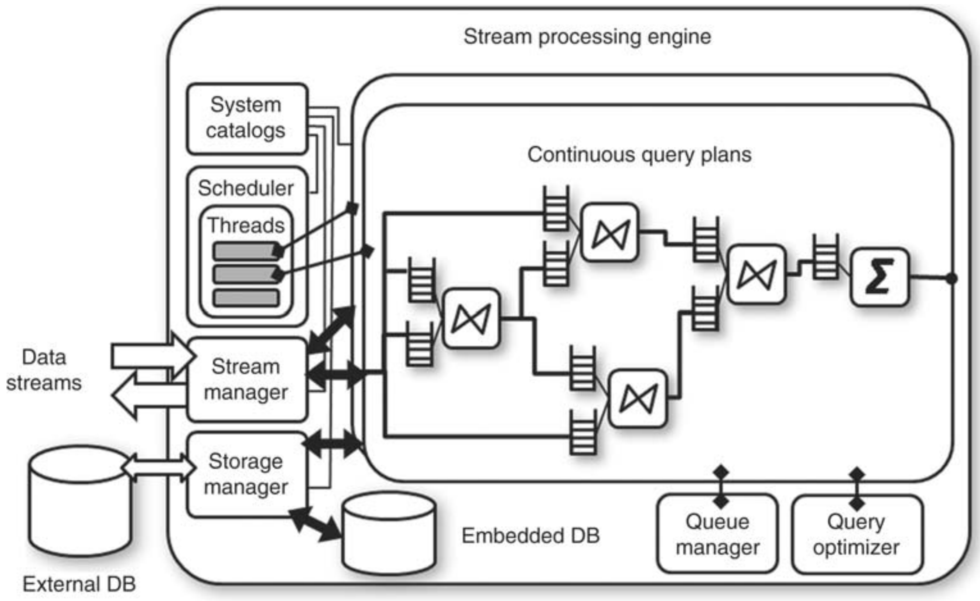
An SPE often adopts a single-process model, where all time-critical operations (including data processing, storage, and execution of custom application logic) are run in a single process space. This integrated approach eliminates high-overhead process context switches that are present in solutions that use multiple software systems to collectively provide the same set of capabilities, yielding high throughput with low latency.

The SPE prototypes developed independently in the academic community share core design principles and architectural components to implement push-based dataflows. At a high level, an SPE's core includes a query executor maintaining plans for users' queries, a queue manager and storage manager to govern memory resources and perform optional disk access and persistence, a stream manager to handle stream I/O with data sources and sinks, and a scheduler to determine an execution strategy. Figure 2 presents a diagrammatic overview of these core components.

SPEs implement continuous queries and inbound processing inside the query executor, by instantiating query plans with non-blocking operators that are capable of producing result tuples from each individual



Data Stream Management Architectures and Prototypes. Figure 1. Illustration of storage-oriented and streaming-oriented architectures. The former requires outbound processing of data, whereas the latter enables inbound (or straight-through) processing.



Data Stream Management Architectures and Prototypes. Figure 2. Architectural components of an SPE.

input tuple, or a window of tuples, depending on the operator type. This is in contrast to traditional operators that wait for relations to be scanned from disk. The query executor represents query plans as operators connected together with queues that buffer continuously-arriving inputs (as found in the Aurora and Stream prototypes [2,5]), and any pending outputs (for example Fjords in TelegraphCQ [8]), while each

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:18

operator performs its processing. A queue manager is responsible for ensuring the availability of memory resources to support buffering inputs and outputs, and interacts with other system components to engage alternative processing techniques when memory availability guarantees cannot be upheld [3].

Operators may choose to access inputs, or share state with other operators, through persistent storage. Disk access is provided through a storage manager that is responsible for maintaining cursors on external tables, and for performing asynchronous read and write operations while continuing to process data streams. Advanced storage manager features include the ability to spill operator queues and state to disk under dwindling memory availability, as well as the ability to maintain approximations of streams, queues and states. SPEs typically include a stream manager component to handle interaction with the network layer as data sources transmit stream inputs typically over TCP or UDP sockets. The stream manager is additionally responsible for any data format conversions through built-in adaptors, and to indicate the arrival of new inputs to the scheduler as the new inputs are placed on the operators' queues.

An operator scheduler [5,7] is responsible for devising an execution order based on various policies to ensure efficient utilization of system resources. These policies typically gather operator cost and selectivity statistics in addition to resource utilization statistics to determine a schedule that improves throughput and latencies. While SPEs execute under a single-process model, various scheduler threading designs have been proposed to provide query processing parallelism. Finally, SPEs also include query optimizers such as load shedders [15] and adaptive plan optimizers [6], that also monitor the state of the running query in terms of statistics and other optimizer-specific monitors to dynamically and adaptively determine advantageous modifications to query plans and operator internals.

Prototypes

The key features in the architectural design of stream processors primarily arose from academic prototypes, before being extended by industrial-grade tools based on the commercialization of the academic efforts. These features are described for a subset of the prototypes below.

Aurora/Borealis: The Aurora and Borealis [2,1] projects are first- and second-generation stream processing engines built in a collaboration by Brandeis, Brown and MIT. The Aurora engine was implemented from scratch in C++, and included the basic architectural components described above to produce a single-site design. Aurora was a general-purpose engine that provided a relational operator set to be used to construct queries visually in an editor, as a workflow. This workflow-style programming paradigm (sometimes referred to as "boxes-and-arrows") differed significantly from similar research projects which focused more on providing stream-oriented language extensions to SQL.

The Aurora architecture included a multi-threaded scheduler capable of supporting tuple-trains and superbox scheduling. Aurora also supported load shedding, the concept of selectively processing inputs in the presence of excessive load due to high-rate data streams. The Aurora engine also supported embedded tables to enable operators to share state. The embedded tables were implemented as BerkeleyDB stores and the core operator set included operators capable of performing a subset of SQL queries on these tables.

The Borealis project extended the Aurora architecture for a multi-site deployment, and implemented components to address the novel challenges exposed by distributed execution. These included a decentralized catalog structure maintaining metadata for the set of deployed queries, and a distributed statistics collector and optimizer. The optimizer targeted distributed query optimizations such as spreading load across multiple machines to achieve both a balanced allocation, and resilience to changes in load, in addition to a distributed load shedding mechanism that factored in the allocation of operators to sites.

Stream: The Stream project at Stanford [5] developed a C++ implementation of a stream processing engine with a similar high-level architecture to the design described above. The novel features of the Stream architecture included its use of the Continuous Query Language (CQL) which extended SQL with DDL statements to define data streams and subsequently provided DML clauses for several types of windowing operations on these streams.

The core engine included a single-threaded scheduler that continuously executes operators based on a scheduling policy, while the operators implement non-blocking query execution through the use of queues. In addition to this basic query executor, the Stream project studied various resource management, query approximation and adaptive query processing techniques. These included memory management techniques implemented by both a scheduling policy that executes groups of operators to minimize queue sizes, and by exploiting shared synopses (window implementations) and constraints in the arrival patterns of data streams (such as bounding the arrival delay between interacting inputs). In terms of query approximation, Stream provided both a load-shedding algorithm that approximates query results to reduce CPU load, in addition to synopsis compaction techniques that reduced memory requirements at operators. Finally, Stream is capable of adapting the running query through the aid of two components, a profiler that continuously collects statistics during query execution, and a reoptimizer component that maintains both filter and join orderings based on changing selectivities.

TelegraphCQ: In contrast to the ground up design of Aurora and Stream, the TelegraphCQ project [8] at UC Berkeley developed a stream processing engine on top of the PostgreSQL open-source database. This approach allowed the reuse of several PostgreSQL components, such as the system catalogs, parser and optimizer.

TelegraphCQ is divided into two high-level components, a frontend and a backend. The frontend is responsible for client interaction such as parsing and planning queries, in addition to returning query results. The TelegraphCQ backend is a continually executing process that performs the actual query processing, and adds query plans submitted by the frontend to the set of executable objects. The backend is implemented in a multi-threaded fashion enabling processing parallelism. The query executor in the TelegraphCQ backend is principally designed to support adaptive query processing through the use of Eddies to dynamically route tuples between a set of commutative operators (thus performing run-time reordering). The executor also leans heavily on exploiting opportunities for shared processing, both in terms of the state maintained internally within operators (such as aggregates), and in terms of common expressions used by selections through grouped filters. Finally, as a result of its PostgreSQL base, TelegraphCQ investigated query processing strategies combining the use of streamed data and historical data from a persistent source.

Gigascope: The Gigascope data stream engine [10] was developed at AT&T Labs-Research to primarily study network monitoring applications, for example involving complex network and protocol analyses of BGP updates and IP packets. Gigascope supports textual queries through GSQL, a pure stream query language that is a simplified form of standard SQL. GSQL queries are internally viewed as having a two-level structure, where queries consist of high-level and low-level operators comprising a graph-structured program, depending on the optimization opportunities determined by a query optimizer. Low-level operators are extremely lightweight computations to perform preliminary filtering and aggregation prior to processing high-level operators, and in some cases these low-level operators may be performed on the network cards of the machines present in the network monitoring application. Gigascope queries are thus compiled into C and C++ modules and linked into a run-time system for highly-efficient execution. Gigascope also investigated the blocking properties of both high- and low-level operators and developed a heartbeat mechanism to effectively alleviate operators' memory requirements.

Nile: The Nile stream processing engine was developed at Purdue on top of the Predator [14] object-relational DBMS. Nile implements data streams as an enhanced datatype in Predator and performs stream processing with the aid of a stream manager component. This stream manager is responsible for buffering input streams and handing data to the execution engine for query processing. Nile uses a separate thread for its stream manager, and performs round-robin scheduling for processing new inputs on streams.

In addition to the basic stream processing engine design, the Nile project investigated various query correctness issues and optimization opportunities arising in the stream processing context. This included studying scheduling strategies to exploit resource sharing amongst queries, for example sharing windowed join operators between multiple queries, and pipelining mechanisms based on strategies to expire tuples in multiple windows.

System S: The System S [12] project is a recent endeavor at IBM Research investigating large-scale distributed stream processing systems focusing primarily on analytical streaming applications through the use of data mining techniques. System S processes data streams with a dataflow-oriented operator network consisting of processing elements (PEs) that are distributed across a set of processing nodes (PNs) and communicate through a transport component known as the data fabric. Some of the prominent architectural features of System S include the design and implementation of streaming analytic operators, including clustering and decision-tree based algorithms, and appropriate resource management algorithms to support these types of operators, such as a variety of load shedding and diffusion algorithms. System S also leverages data-parallelism through a content-based load partitioning mechanism that spreads the processing of an input or intermediate stream across multiple downstream PEs.

Key Applications

Stream processing architectures have been motivated by, and used in, several domains, including:

- Financial services: automated trading, market feed processing (cleaning, smoothing, and translation), smart order routing, real-time risk management and compliance (MiFID, RegNMS)
- Government and Military: surveillance, intrusion detection and infrastructure monitoring, battlefield command and control
- Telecommunications: network management, quality of service (QoS)/service level agreement (SLA) management, fraud detection
- Web/E-business: click-stream analysis, real-time customer experience management (CEM)

URL to Code

Borealis: http://www.cs.brown.edu/research/borealis/public/
Stream: http://infolab.stanford.edu/stream/code/

Cross-references

► Continuous Query
► Data Stream
► Stream-oriented Query Languages and Operators
► Stream Processing
► Streaming Applications
► Windows

Recommended Reading

1. Abadi D., Ahmad Y., Balazinska M., Çetintemel U., Cherniack M., Hwang J.-H., Lindner W., Maskey A.S., Rasin A., Ryvkina E., Tatbul N., Xing Y., and Zdonik S. The design of the Borealis stream processing engine. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005.
2. Abadi D.J., Carney D., Çetintemel U., Cherniack M., Convey C., Lee S., Stonebraker M., Tatbul N., and Zdonik S. Aurora: A new model and architecture for data stream management. The VLDB J., 2003.
3. Arasu A., Babcock B., Babu S., McAlister J., and Widom J. Characterizing memory requirements for queries over continuous data streams. ACM Trans. Database Syst., 29 (1):162–194, 2004.
4. Babcock B., Babu S., Datar M., Motwani R., and Thomas D. Operator scheduling in data stream systems. VLDB J., 13 (4):333–353, 2004.
5. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data stream systems. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002.
6. Babu S., Motwani R., Munagala K., Nishizawa I., and Widom J. Adaptive ordering of pipelined stream filters. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 407–418.
7. Carney D., Çetintemel U., Rasin A., Zdonik S.B., Cherniack M., and Stonebraker M. Operator scheduling in a data stream manager. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 838–849.
8. Chandrasekaran S., Deshpande A., Franklin M., and Hellerstein J. TelegraphCQ: Continuous dataflow processing for an uncertain world. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
9. Chen J., DeWitt D.J., Tian F., and Wang Y. Niagaracq: A scalable continuous query system for internet databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 379–390.
10. Cranor C.D., Johnson T., Spatscheck O., and Shkapenyuk V. Gigascope: a stream database for network applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 647–651.
11. Data stream processing (Johannes Gehrke, ed.). IEEE Data Eng. Bull., 26(1), 2003.
12. Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and MyungCheol Doo. SPADE: The Systems S Declarative Stream Processing Engine. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2008.
13. Golab L. and Özsu M.T. Issues in data stream management. SIGMOD Rec., 32(2):5–14, 2003.
14. Hammad M.A., Mokbel M.F., Ali M.H., Aref W.G., Catlin A.C., Elmagarmid A.K, Eltabakh M.Y., Elfeky M.G., Ghanem T.M., Gwadera R., Ilyas I.F., Marzouk M.S., and Xiong X. Nile: a query processing engine for data streams. In Proc. 20th Int. Conf. on Data Engineering, 2004, p. 851.
15. Tatbul N., Çetintemel U., Zdonik S.B., Cherniack M., and Stonebraker M. Load shedding in a data stream manager. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 309–320.

Data Stream Processing

► Stream Processing

Data Suppression

► Data Compression in Sensor Networks

Data Swapping

► Data/Rank Swapping

Data Time

► Valid Time

Data Tracking

► Data Provenance

Data Transformation

► Data Exchange

Data Translation

► Data Exchange

Data Types for Moving Objects

► Spatio-Temporal Data Types

Data Types in Scientific Data Management

Amarnath Gupta
University of California San Diego, La Jolla, CA, USA

Synonyms

Data sorts; Many sorted algebra; Type theory

Definition

In mathematics, logic and computer science, the term "type" has a formal connotation. By assigning a variable to a type in a programming language, one implicitly defines constraints on the domains and operations on the variable. The term "data type" as used in data management derives from the same basic idea. A data type is a specification that concretely defines the "structure" of a data variable of that type, the operations that can be performed on that variable, and any constraints that might apply to them. For example, a "tuple" is a data type defined as a finite sequence (i.e., an ordered list) of objects, each of a specified type; it allows operations like "projection" popularly used in relational algebra.

In science, the term "data type" is sometimes used less formally to refer to a kind of scientific data. For example, one would say "gene expression" or "4D surface mesh of a beating heart" is a data type.

Foundations

Commonly Used Data Types in Science Applications

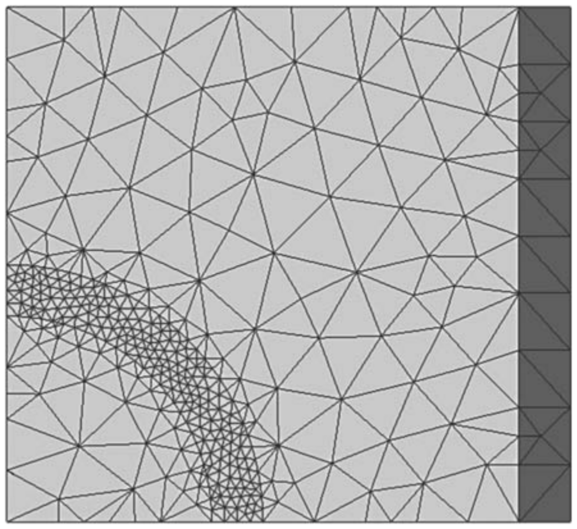
There is a very large variety of data types used in scientific domains. The following data types are commonly used in several different scientific disciplines.

Arrays Multidimensional arrays are heavily used in many scientific applications; they not only serve as natural representation for many kinds of scientific data, but they are supported by programming languages, object relational databases, many computational software libraries, as well as data visualization routines. The most common operation on arrays is index-based access to data values. However, more complex (and useful) operations can be defined on arrays. Marathe and Salem [6,7] defined an algebra on multidimensional arrays where a cell may contain a vector of values. The algebra derives from nested relational algebra, and allows one to perform value-based relational queries on arrays. Arrays are a very general data type and can be specialized with additional semantics. Baumann [1] defined a somewhat different array algebra for modeling spatiotemporal data for a system called RasDaMan. Reiner et al. [10] present a storage model for large scale arrays.

Time-Series Temporal data is a very important class of information for many scientific applications. Time-series data is a class of temporal data where the value of a variable may change with a roughly regular interval. On the other hand, the salary history of an employee is temporal data but not necessarily time-series data because the change in salary can happen at arbitrary frequencies. Data from any sensors (temperature, seismic, strain gages, electrocardiograms and so on) come in the form of a stream of explicitly or implicitly time-stamped sequence numbers (or characters). Time series data collection and storage is based on granularity, and often different data are collected at different granularity that need to be queries together [8]. All database management systems assume a discrete time line of various granularities. Granularity is a partition of a time line, for instance, years, days, hours, microseconds and so forth. Bettini et al have illustrated a formal notion of time granularities [2]. An interesting class of operations on time-series data is similarity between two time-series data. This operation can become complicated because one time series data can be recorded at a different time resolution than another and may show local variations, but still have overall similarity in the shape of the data envelope. This has prompted the investigation of efficient operators to perform this similarity. Popivanov and Miller [9] for example, has developed a measure of time-series similarity based on wavelet decomposition.

Finite Element Meshes Numerical modeling of a physical system is fundamental to many branches of science. A well known technique in this domain is called finite element analysis where a continuous domain (e.g., a 3D terrain) is partitioned into a mesh, and variables are recorded over the nodes of this mesh or regions covering multiple cells of the mesh. Assume that Fig. 1 shows the distribution of electric potential over a continuous surface. Every node of the mesh will have a (positive or negative) charge value, while a variable like "zero charge region" will be defined over regions of the mesh. Figure 1 also illustrates that mesh is not always regular – a region with a higher variation of data values will be partitioned into a finer mesh than a region will less variation.

Finite element meshes are used in many modeling as well as visualization software. In cases, where the size of the mesh is very large, and complex manipulation of data (like repartitioning based on some conditions) is needed over the elements of the mesh, the existing software do not offer robust and scalable solutions. Recently, the CORIE system [5] has developed a systematic approach to modeling a general data structure they call a *gridfield* to handle finite element meshes, and an algebra for manipulating arbitrary gridded datasets together with algebraic optimization techniques to improve efficiency of operations.


Data Types in Scientific Data Management. Figure 1. A finite element mesh.

Graphs Like arrays, graphs form a ubiquitous data type used in many scientific applications. Eckman and Brown [4] describes the use of graphs in molecular and cell biology. In their case, graphs denote relationships between biomolecular entities (A and B) that constitute molecular interaction networks, representing information like A is similar to B, A interacts with B, A regulates the expression of B, A inhibits the activity of B, A stimulates the activity of B, A binds to B and so forth. Operators on the graph data type include those that extract a subgraph from a large graph, compare one graph to another, transform one graph to another, decompose a graph into its nodes and edges, compute the intersection, union, or disjunction of two graphs, compute structural derivatives such as transitive closure and connected components and so on. In chemistry, data mining techniques are used to find most frequent subgraphs of a large number of graphs. Graphs play a dominant role in social sciences where social network analysts are interested in the analysis of the connectivity structure of the graphs. A class of operations of interest centers around the notion of aggregate properties of the graph structure. One such property is centrality, a quantity that measures for each node in a graph a value that denotes how well the node is connected to the rest of the nodes in the graph. This class of measures have been investigated in the context of data mining [11] where the task was to find the most likely subgraph "lying between" a set of query-defined nodes in the graph. While there are many alternate definitions of centrality, it should be clear that computing these aggregate values on the fly requires a different kind of data representation and operators than the previous case, where traversal and subgraph operations were dominant.

Some Basic Issues about Scientific Data Types

While database systems do not offer extensive support for scientific data types, there are many specialized software libraries that do, and hence are widely used by the scientific community. This leads to a fundamental problem as observed by the authors of [5]. On the one hand, the performance of SQL queries for manipulating large numeric datasets is not competitive with specialized tools. For example, database extensions for processing multidimensional discrete data can only model regular, rectilinear grids (i.e., arrays). On the other hand, specialized software products such as visualization software libraries are designed to process arbitrary gridded datasets efficiently. However, no algebra has been developed to simplify their use and afford optimization. In almost all cases, these libraries are data dependent – physical changes to data representation or organization break user programs. This basic observation about type specific scientific software holds for almost all of scientific data types. This calls for future research in developing storage and an algebraic manipulation for scientific data type as well as for an effort to incorporate in these techniques in scientific data management systems.

A second basic problem regarding scientific data types arises from the fact that the same data can be viewed differently for different forms of analysis and thus need to support multiple representations and storage or indexes. Consider the data type of character sequences often used in genomic studies. If S is a sequence, it is common to determine is S is "similar to" another sequence T, where T may have additional characters and missing characters with respect to S. It has been shown that a suffix tree like representation of sequences is suitable for operations of this category. However, in biology, scientists are also interested in an arbitrary number of subsequences of on the same sequences like S to which they would assign an arbitrary number of properties (called "annotations" in biology) to each subsequence. Finding similar subsequences is not a very common operation in this case. The focus is rather on interval operations like finding all subsequences overlapping a given interval that satisfies some conditions on their properties, and on finding the 1D spatial relationships among subsequences that satisfy some given properties. These operations possibly require a different storage and access structure such as an interval tree. Since a scientific application both kinds of operations would be necessary, it becomes important for the data management system to handle the multiplicity of representations and operations so that the right representations can be chosen as run time for efficient access.

Key Applications

Bioinformatics, cheminformatics, engineering databases.

Cross-references

► Graph Data Management in Scientific Applications
► Mining of Chemical Data
► Storage of Large Scale Data for Multidimensional Data

Recommended Reading

1. Baumann P. A database array algebra for spatio-temporal data and beyond. In Proc. Fourth Int. Workshop on Next Generation Information Technologies and Systems (NGITS 1999). 1999, pp. 76–93.
2. Bettini C., Jajodia S., and Wang S.X. Time Granularities in Database, Data Mining, and Temporal Reasoning. Springer, 2000.
3. Borgatti S.P. and Everett M.G. A graph-theoretic perspective on centrality. Soc. Netw., 28(4):466–484, 2006.
4. Eckman B.A. and Brown P.G. Graph data management for molecular and cell biology. IBM J. Res. Dev., 50(6):545–560, 2006.
5. Howe B. and Maier D. Algebraic manipulation of scientific data sets. VLDB J., 14(4):397–416, 2005.
6. Marathe A.P. and Salem K. A language for manipulating arrays. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 46–55.
7. Marathe A.P. and Salem K. Query processing techniques for arrays. SIGMOD Rec., 28(2):323–334, 1999.
8. Merlo I., Bertino E., Ferrari E., Gadia S., and Guerrini G. Querying multiple temporal granularity data. In Proc. Seventh Int. Conf. on Temporal Representation and Reasoning (TIME 2000), 2000, pp. 103–114.
9. Popivanov I. and Miller R.J. Similarity search over time-series data using wavelets. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 212–221.
10. Reiner B., Hahn K., Höfling G., and Baumann P. Hierarchical storage support and management for large-scale multidimensional array database management systems. In Proc. 13th Int. Conf. Database and Expert Syst. Appl., 2002, pp. 689 –700.
11. Tong H. and Faloutsos C. Center-piece subgraphs: problem definition and fast solutions. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006, pp. 404–413.

Data Types: Image, Video, Pixel, Voxel, Frame

► Biomedical Image Data Types and Processing

Data Uncertainty Management in Sensor Networks

Sunil Prabhakar[1], Reynold Cheng[2]
[1]Purdue University, West Lafayette, IN, USA
[2]The University of Hong Kong, Hong Kong, China

Synonyms

Imprecise data; Probabilistic data; Probabilistic querying

Definition

Data readings collected from sensors are often imprecise. The uncertainty in the data can arise from multiple sources, including measurement errors due to the sensing instrument and discrete sampling of the measurements. For some applications, ignoring the imprecision in the data is acceptable, since the range of the possible values is small enough not to significantly affect the results. However, for others it is necessary for the sensor database to record the imprecision and also to take it into account when processing the sensor data. This is a relatively new area for sensor data management. Handling the uncertainty in the data raises challenges in almost all aspects of data management. This includes modeling, semantics, query operators and types, efficient execution, and user interfaces. Probabilistic models have been proposed for handling the uncertainty. Under these models, data values that would normally be single values are transformed into groups of data values or even intervals of possible values.

Historical Background

The management of uncertain data in database management systems is a relatively new topic of research, especially for attribute-level uncertainty. Earlier work has addressed the case of tuple-level uncertainty and also node-level uncertainty for XML data. The earliest work on attribute-level uncertainty is in the area of moving object databases. In order to reduce the need for very frequent updates from moving objects, the frequency of the updates is reduced at the expense of uncertainty in the location of the object (in the database). For example, the notion of a dead-reckoning update policy allows an object to not report updates as long as it has not moved by more than a certain threshold from its last update.

In most of the earlier works, the use of probability distributions of values inside an uncertainty interval as a tool for quantifying uncertainty was not considered. Further, discussions of queries on uncertain data were often limited to the scope of aggregate functions or range queries. A model for probabilistic uncertainty was proposed for moving-objects and later extended to general numeric data for sensors in [2]. A probabilistic data model for data obtained from a sensor network was described in [6]. Past data are used to train the model through machine-learning techniques, and obtain information such as data correlation, time-varying functions of probability distributions, as well as how probability distributions are updated when new sensor values are acquired. Recently, new relational models have been proposed to manage uncertain data. These projects include MauveDB [9], Mystiq [6], Orion [15], and Trio [1]. Each of these projects aims to develop novel database systems for handling uncertain data.

Foundations

Modeling Uncertainty

Uncertainty in sensor data is often the result of either inherent limitations in the accuracy with which the sensed data is acquired or limitations imposed by concerns such as efficiency and battery life. Consider for example, a moving object application that uses GPS devices to determine the locations of people as they move about. Although GPS accuracy has improved significantly, it is well known that the location reported by a GPS sensor is really an approximation – in fact, the actual location is likely to be distributed with a Gaussian probability distribution around the reported location. This is an example of uncertainty due to the limitation of the measurement instrument.

Since most sensors are powered by batteries that can be quickly depleted, most sensor applications take great pains to conserve battery power. A common optimization is to not measure and transmit readings continuously. Instead, the data are sampled at some reasonable rate. In this case the exact values are only known at the time instances when samples are taken.

Between samples, the application can only estimate (based on the earlier samples) the values. For certain sensors, the battery overhead for taking certain types of measurements is much lower than that for others. Furthermore, the cheaper readings are correlated with more expensive reading. This allows the sensor to estimate the costlier reading by taking a cheaper reading and exploiting the correlation. However the estimate is not exact, which introduces some uncertainty.

Even when sensor readings are precise and frequently sampled, uncertainty can creep in. For example, if a given sensor is suspected of being faulty or compromised, the application may only partially trust the data provided by the sensor. In these cases, the data are not completely ignored but their reliability can be reduced. Alternatively, sensor input may be processed to generate other information – e.g. face detection on video data from a sensor. Post processing methods may not yield certain matches – the face detection algorithm may have a known degree of error or may give a degree of confidence with which it has detected a face (or a given person). In these cases, the unreliability of the raw or processed sensor data can be captured as uncertain data.

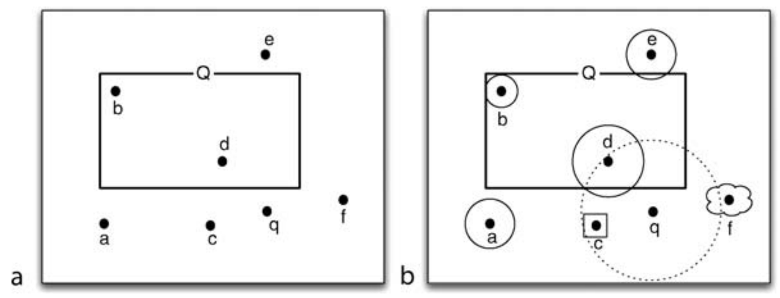
Each of these examples shows that sensor readings are not precise. Instead of data having a definite discrete value, data has numerous alternative values, possibly with associated likelihood (probabilities). The types of uncertainty in sensor data can be divided into two categories:

- *Discrete* uncertainty. Instead of a single value, a data item could take on one out of a set of alternative values. Each value in this set may further be associated with a probability indicating the likelihood of that particular value being the actual value.
- *Continuous* uncertainty. Instead of a single value, a data item can take on any one value within an interval. In addition, there may be an associated probability density function (pdf) indicating the distribution of probabilities over this interval.

In each of these cases, the total probability may or may not total to 1 for each data item. Several models for handling probabilistic data based upon the relational data model have been proposed in the literature. Most of these models can only handle discrete data wherein each alternative value for a given data time is stored in the database along with its associated probability. Extra rules are imposed over these records to indicate that only one of the alternative values for a given data time will actually occur. The Orion model is explicitly designed for handling continuous uncertainty. Under this model, uncertain attributes can be expressed as intervals with associated pdfs or as a discrete set. Representing probabilities symbolically as pdfs instead of enumerating every single alternative allows the model to handle continuous distributions.

Queries

As data becomes imprecise, there is a direct impact on the nature of query results. Figure 1 shows an example of points in two-dimensional space, a range query (Q), and a nearest-neighbor query (q) with two cases: (i) with no uncertainty; and (ii) with different types of uncertainty for different objects. Consider the two-dimensional range query Q shown in Fig. 1a. The result of the query are the identities of those points that fall within the range of the query – Points b and d in this example. If the data is imprecise (as in Fig. 1b), the data consist of regions of space (and possibly with associated probability distributions). Some of these



Data Uncertainty Management in Sensor Networks. Figure 1. A two-dimensional example: (a) exact points with no uncertainty; (b) points with uncertainty.

regions may clearly lie outside the query region and the corresponding moving objects are thus excluded from the answer (e.g. Point a). Those that lie completely within the query region are included in the answer, as in the case of precise point data (e.g. Point b). However, those objects that partially overlap the query region represent points that may or may not actually be part of the query answer (Points d and e). These points may be reported as a special subset of the answer. In [14] Future Temporal Logic (FTL) was proposed for processing location-based queries over uncertain data with no probability information. Thus an object is known to be located somewhere within a given spatial region. Queries are augmented with either `MUST` or `MAY` keywords. With the MUST keyword, objects that have even a small chance of not satisfying a query are not included in the results. On the other hand, with the MAY keyword, all objects that have even a remote chance of satisfying a query are included. FTL therefore provides some qualitative treatment for queries over uncertain data.

With the use of probability distributions, it is possible to give a more quantitative treatment to queries over uncertain data. In addition to returning the query answers, probability of each object satisfying the query can be computed and reported. In order to avoid reporting numerous low probability results, queries can be augmented with a *probability threshold*, τ. Only those objects that have a probability greater than τ of satisfying the query are reported. This notion of probabilistic queries was introduced in [2]. Most work on uncertain data management gives a quantitative treatment to queries. It should be noted that the MUST and MAY semantics can be achieved by choosing the threshold to be 1 or 0, respectively.

An important issue with regards to queries over uncertain data is the semantics of the query. What exactly does it mean to execute an arbitrary query over uncertain data? Most researchers have adopted the well-established *possible worlds* semantics (PWS) [10]. Under PWS, a database with uncertain (probabilistic) data consists of numerous probabilistic events. Depending upon the outcome of each of these events, the actual database is one out of an exponential number of possible worlds. For example, consider a single relation with two attributes: `Sensor_id` and `reading`. Assume there is a single tuple in this table, with Sensor_id S_1, and an uncertain reading which could be 1 with probability 0.3 and 2 with probability 0.7. This uncertain database consists of a single event, and there are two possible worlds: in one world (W_1), the relation consists of the single tuple $<S_1, 1>$; in world W_2, the relation consists of the single tuple $<S_1, 2>$. Furthermore, the probability of W_1 is 0.3 and that of W_2 is 0.7. In general, with multiple uncertain events, each world corresponds to a given outcome of each event and the probability of the world is given by the product of the probabilities of each event that appears in the world. It should be noted that there is no uncertainty in a given world. Each world looks like a regular database relation.

Under PWS, the semantics of a query are as follows. Executing a query over an uncertain data is conceptually composed of three steps: (i) Generate all possible worlds for the given data with associated probabilities; (ii) execute the query over each world (which has no uncertainty); and (iii) Collapse the results from all possible worlds to obtain the uncertain result to the original query. While PWS provides very clean semantics for any query over an uncertain database, it introduces challenges for efficient evaluation. First, if there is continuous uncertainty in the data, then there are an infinite number of possible worlds. Even when there are a finite number of possible worlds, the total number is exponential in the number of events. Thus it is impractical to enumerate all worlds and execute the query over each one. Techniques to avoid enumerating all worlds while computing the query correctly were proposed in [6]. They showed that there is a class of *safe* queries over uncertain data which can be computed using query plans similar to those for certain data.

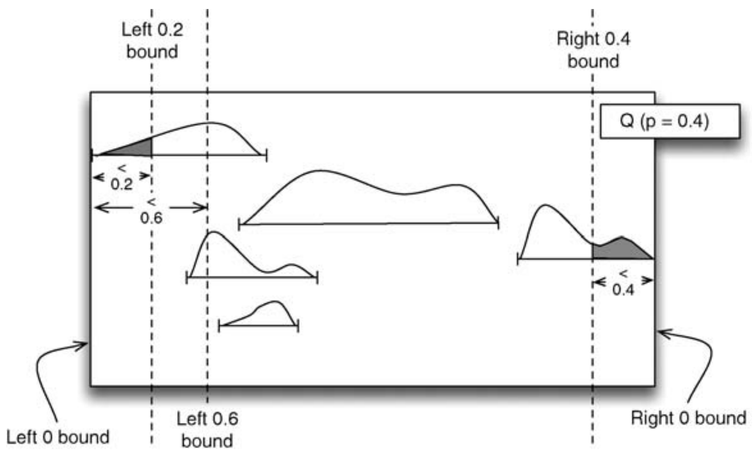
Implementation

With the goal of supporting PWS over uncertain data, systems that support uncertainty need to define probabilistic versions of database operators such as selection, projection, cross products, and comparison operators. Typically this involves operations over the probability distributions of the data, and tracking dependencies that are generated as a result of processing. Efficient management of dependencies between derived data is among the greatest challenges for uncertain data management. The base data in an uncertain database are assumed to be independent (with the exception of explicit dependencies that are expressed in the base data). However, as these data are used to produce other data, the derived data may no longer be independent of each other [6]. These dependencies affect the correct evaluation of query operators. To correctly handle dependencies, it is necessary to track them. Thus the model has to be augmented to store not only the data, but also dependencies among them. In the Trio system this information is called *Lineage*, the Orion model calls it *History*, and the MauveDB model handles dependencies using factor tables. As data is processed multiple times, the size and complexity of this dependency information can grow significantly. Efficient handling of this information is currently an active area of research.

Query processing algorithms over uncertain data have been developed for range queries [13], nearest-neighbor queries [2,11], and skyline queries [12]. Efficient join algorithms over uncertain data have been proposed in [3]. Despande et al. [7] studied the problem of answering probabilistic queries over data streams. They proposed algorithms to return results with minimum resource consumption. In [5], Cormode et al. proposed space- and time-efficient algorithms for approximating complex aggregate queries over probabilistic data streams. For queries that cannot be correctly processed using these modified operators and safe query plans, one alternative is to use approximation techniques based upon sampling. Samples of possible worlds can be drawn using the probabilities of the various events that make up the uncertain database. The query is then executed on these sample worlds and the results are aggregated to obtain an approximation of the true answer.

Indexing Indexing is a well known technique for improving query performance. Indexing uncertain data presents some novel challenges. First, uncertain data do not have a single value as is the case for traditional data. Consequently indexes such as B+-trees (and also hash indexes, since hashing requires exact matches) are inapplicable. By treating the uncertain intervals (regions) as spatial data, it is possible to use spatial indexes, such as R-trees or interval indexes, over uncertain attributes. These indexes can provide pruning based upon the extent and location of the uncertainty intervals alone. However, these index structures do not consider probability information, and are therefore incapable of exploiting probability for better evaluation. This is especially true in the case for probabilistic threshold queries.

There has been some recent work on developing index structures for uncertain data [4,11,13]. These index structures take the probability distribution of the underlying data into account. In particular, the *Probability Threshold Index* (PTI), is based on the modification of a one-dimensional R-tree. Each entry in this R-tree variant is augmented with multiple Minimum Bounding Rectangles (MBRs) to facilitate pruning. The extra MBRs are called *x*-bounds. Consider a one-dimensional data set. An MBR can been viewed as a pair of bounds: a left bound that is the right-most line that lies to the left of every object in the given node; and a right bound that is the left-most line that lies to the right of every object in the given node. The notion of *x*-bounds is similar, except that a left-x-bound is the right-most line that ensures that no object in the given node has a probability greater than *x* of lying to the left of this bound. The right-x-bound is similarly defined. Figure 2 shows an example of these bounds. Using these



Data Uncertainty Management in Sensor Networks. Figure 2. An example of *X*-Bounds for PTI.

bounds it is possible to achieve greater pruning as shown by the range query in the figure. This query has a threshold bound of 0.4. Even though the query intersects with overall MBR, using the right-0.4-bound it is clear that there is no need to visit this subtree for this query. Since the query does not cross the right-0.4-bound, there can be no objects under this node that have a probability greater than 0.4 of overlapping with the query.

Key Applications

Uncertainty in sensor data is found in virtually all applications of sensors. For many applications, however, it may be acceptable to ignore the uncertainty and treat a given value as a reasonable approximation of the sensor reading. For others, such approximations and the resulting errors in query answers are unacceptable. In order to provide correct answers for these applications it is necessary to handle the uncertainty in the data. Examples include location-based services and applications that introduce uncertainty in order to provide some degree of privacy.

Future Directions

Work on the problem of handling uncertain data in sensor databases has only just begun. Much remains to be done. A long-term goal of several current projects is the development of a full-fledged database management system with native support for uncertain data as first-class citizens. Examples of current systems include Orion, MauveDB, Mystiq, and Trio. Immediate steps in building such systems include the development of query optimization techniques. This includes cost estimation methods, query plan enumeration techniques, and approximate query evaluation methods. In addition, an important facet of system development is the user interface. Interesting issues for user interfaces include: How do users make sense of the probabilistic answers? How do they input probabilistic data and pose queries? Are new query language constructs needed? Should the probabilistic nature of the data be hidden from the user or not?

Cross-references

► Data Storage and Indexing in Sensor Networks
► Location-Based Services
► Moving Objects Databases and Tracking
► Probabilistic Databases
► R-Tree (and family)

Recommended Reading

1. Benjelloun O., Sarma A.D., Halevy A., and Widom J. ULDBs: databases with uncertainty and lineage. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 953–964.
2. Cheng R., Kalashnikov D., and Prabhakar S. Evaluating probabilistic queries over uncertain data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
3. Cheng R., Singh S., Prabhakar S., Shah R., Vitter J., and Xia Y. Efficient join processing over uncertain data. In Proc. ACM 15th Conf. on Information and Knowledge Management, 2006.
4. Cheng R., Xia Y., Prabhakar S., Shah R., and Vitter J. Efficient indexing methods for probabilistic threshold queries over uncertain data. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
5. Cormode G. and Garofalakis M. Sketching probabilistic data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 143–154.
6. Dalvi N. and Suciu D. Efficient query evaluation on probabilistic databases. In Proc. 30th Int. Conf. on Very Large Data Bases. 2004.
7. Despande A., Guestrin C., Hong W., and Madden S. Exploiting correlated attributes in acquisitional query processing. In Proc. 21st Int. Conf. on Data Engineering, 2005.
8. Deshpande A., Guestrin C., Madden S., Hellerstein J., and Hong W. Model-driven data acquisition in sensor networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
9. Deshpande A. and Madden S. MauveDB: supporting model-based user views in database systems. In Proc. ACM SIGMOD Int. Conf. Management of Data. 2006, pp. 73–84.
10. Halpern J.Y. Reasoning about uncertainty. MIT, Cambridge, USA, 2003.
11. Ljosa V. and Singh A. ALPA: indexing arbitrary probability distributions. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
12. Pei J., Jiang B., Lin X., and Yuan Y. Probabilistic skylines on uncertain data. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
13. Singh S., Mayfield C., Prabhakar S., Shah R., and Hambrusch S., Indexing uncertain categorical data. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
14. Sistla P.A., Wolfson O., Chamberlain S., and Dao S. Querying the uncertain positions of moving objects. Temporal databases: research and practice 1998.
15. The Orion Uncertain Database Management System. Available at: http://orion.cs.purdue.edu/

Data Utility Measures

► Information Loss Measures

Data Visualiyations

► Dense Pixel Displays

Data Visualization

Hans Hinterberger
ETH Zurich, Zurich, Switzerland

Synonyms

Graphic representation of data; Information visualization

Definition

Data Visualization: (i). Interpreting information in visual terms by forming a mental picture based on data. (ii). Applying suitable methods to put data into visible form.

This definition is consistent with the Oxford English Dictionary definitions of 'data': Facts, esp. numerical facts, collected together for reference or information and of 'visualization': (i) The action or fact of visualizing; the power or process of forming a mental picture or vision of something not actually present to the sight; a picture thus formed. (ii) The action or process of rendering visible.

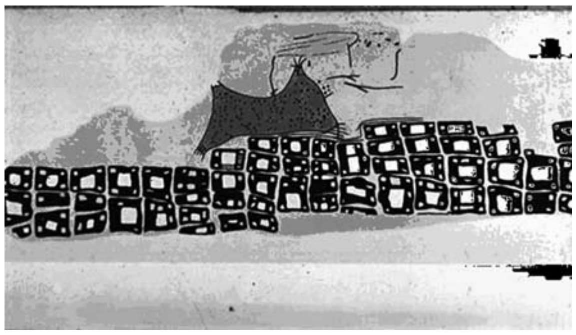
The first part of the definition refers to the human cognitive activity of forming a mental picture, independent of how something is represented. If this is the only activity of interest, then the term 'information visualization' is more commonly used. Similarly, 'data visualization' is often reduced to the second part of the definition.

Some authors explicitly include the computer and cognition in their definition of visualization: *The use of computer supported, interactive, visual representations of data to amplify cognition* [1]. Others emphasize how data visualization differs from information visualization: *data visualization is for exploration, for uncovering information, as well as for presenting information. It is certainly a goal of data visualization to present any information in the data, but another goal is to display the raw data themselves, revealing the inherent variability and uncertainty* [16].

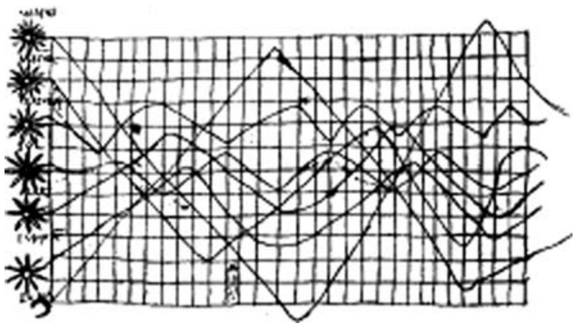
Historical Background

Up to the 15th Century. Over eight thousand year old maps, carved in stone, suggest that the visualization of information is as old as civilization (Fig. 1). The earliest known data visualization, a time series plot, depicting the changing values of several planets' positions, is estimated to have appeared in the tenth century (Fig. 2). In the middle of the fourteenth century, Nicole Oresme introduced an early form of coordinate graphing. He marked points in time along a horizontal line and for each of these points he drew a bar whose length represented the object's velocity at that moment.

1500–1800. Meteorological maps, showing the prevalence of winds on a geographical map, date back to 1686. In 1782, Marcellin du Carla-Boniface issues the first modern topographical maps (Fig. 3) and August Crome prints the first thematic map, showing economic production data across Europe. Also in this time period appear the first graphics used for descriptive statistics, for example Christian Huygens' plot of a function to graphically determine the years of life remaining given the current age, published in 1669. William Playfair, an English political economist, laid the ground for business graphics in 1786 with his Commercial and Political Atlas in which he documented commercial and political time series using curves, bar charts and column charts (Fig. 4). Playfair is also



Data Visualization. Figure 1. Ca. 6200 BC. The oldest known map, from a museum at Konya, Turkey.



Data Visualization. Figure 2. Ca. 950. First known graphic of a time series visualizing data of planetary orbits.

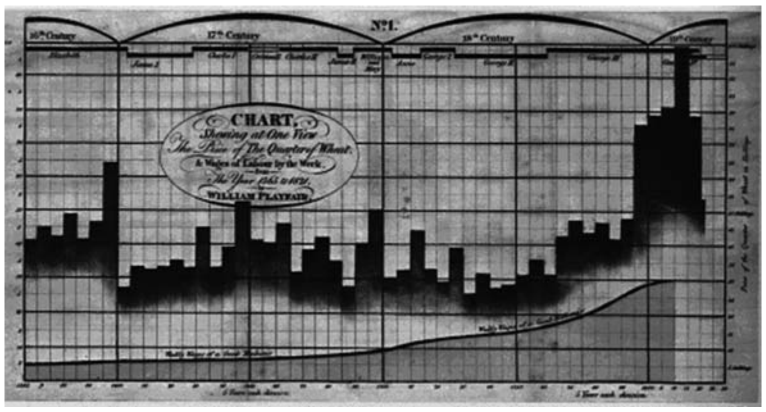
arguably the inventor of the pie chart (Statistical Breviary, 1801).

1800–1949. Scientists and civil servants are beginning to use thematic maps and statistical graphics to support their arguments. Famous examples are the dot map that Dr. John Snow drew by plotting the locations of deaths from cholera in central London during the 1854 epidemic (Fig. 5) and Florence Nightingale's comparison of deaths due to injuries in combat and deaths due to illness in an army hospital for which she invented her own graphic, the Polar-Area Diagram (Fig. 6). The second half of the nineteenth century saw a 50 year long debate on the standardization of statistical maps and diagrams which failed to produce concrete results. Early in the twentieth century there followed a 50 year period of consolidation where the accomplishments of the previous one hundred years became widely accepted.

Data Visualization. Figure 3. 1782. Detail of Marcellin du Carla-Boniface's topological map.

1950–1975. Researchers started to face enormous challenges when analyzing the deluge of data produced by electronic equipment that was put in use after the Second World War. In this environment, John Tukey led the way, established the field of Exploratory Data Analysis [15] (Fig. 8) and sparked a flurry of activities that have and still are producing many novel graphical methods [5,8], including techniques that marked the beginning of dynamic statistical graphics. In the 1960s, when signals from remote sensing satellites needed to be processed graphically, geographers started to combine spatially referenced data, spatial models and map based visualizations in geographic information systems. The French cartographer Jacques Bertin worked on a theory of graphics [3] and introduced with his reorderable matrix an elegant technique to graphically process quantitative data [2] (Fig. 7).

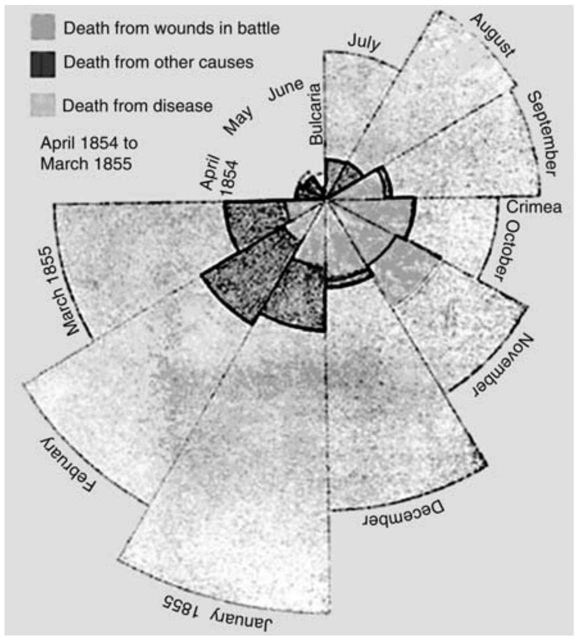
From 1975–present. Fast, interactive computers connected to a high resolution color graphic display created almost unlimited possibilities for scientific visualizations of data generated by imaging techniques, computational geometry or physics based models [10]. Event oriented programming made it easy to link different data displays, encouraging new techniques such as brushing [5]. Statisticians started to tackle high dimensional data by interactively 'touring' low dimensional projections. Large display areas encouraged graphic methods based on multiple plots [5], space filling techniques (e.g. mosaic plots) and graphics with high data densities. For an overview the reader is referred to [1,5,16]. In the early 1990s, virtual



Data Visualization. Figure 4. 1786. William Playfair' chart, depicting prices, wages, and the reigns of British kings and queens.

BROAD ST
PUMP
CAMBRIDGE ST
MARSHALL STREET
PLACE

Data Visualization. Figure 5. 1854. Detail of the pioneering statistical map drawn by John Snow to illustrate patterns of disease.



Data Visualization. Figure 6. 1858. Polar-Area diagram, invented by Florence Nightingale to convince authorities of the need to improve sanitary conditions in hospitals.

environments were introduced as methods to immersively investigate scientific data [4]. (Fig. 10) Another way to overcome the restrictions of two dimensional displays was shown by Alfred Inselberg with his concept of parallel coordinates [9], today a ubiquitous method to visualize multidimensional data (Fig. 9).

To further explore the history of data visualization the reader is referred to [7,14].

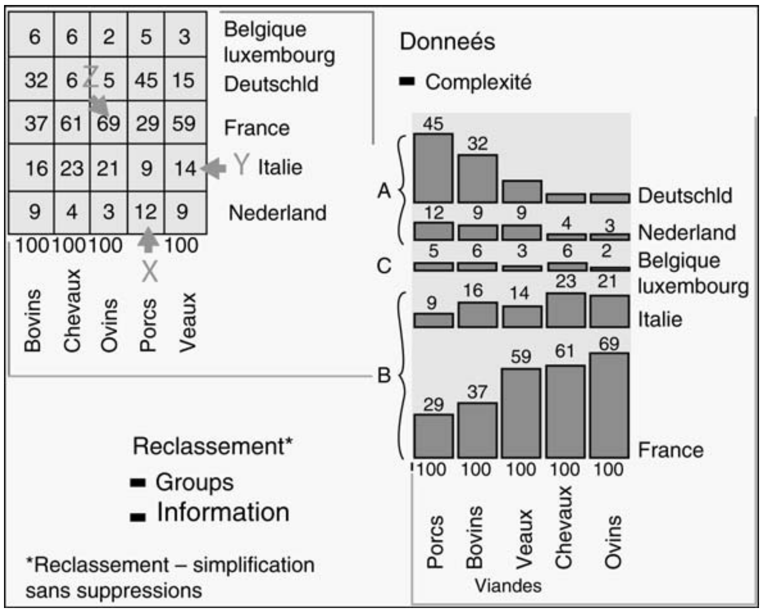
Foundations

The literature on scientific fundamentals of data visualizations fall into three independent but related fields: (i) computer graphics, (ii) presentation techniques, (iii) cognition.

Computer graphics, primarily the domain of computer scientists and mathematicians, builds on elementary principles in the following broad areas: visualization algorithms and data structures, modeling and (numerical) simulation, (volume) rendering, particle tracing, grid generation, wavelet transforms, multiscale and multiresolution methods as well as optics and color theory. A more exhaustive treatment of computer graphics fundamentals related to data visualization can be found in [10,12,17].

Most of the literature on presentation techniques can be found in statistics and computer science although economists and cartographers also made substantial contributions. The publications of John Tukey [15] and Andrew Ehrenberg [6] show how tables can be used as simple but effective presentation technique to organize data and demonstrate the method's usefulness for statistics and data analysis. In 1967 Jacques Bertin formulated a comprehensive theory for a graphical system [3] and subsequently applied parts of it to graphic information processing [2]. Other classics were published in 1983 when William Cleveland wrote an excellent methodological resource for the design of plots and Edward Tufte published his review on the graphical practice to visualize quantitative data. A reference, featuring perhaps the most complete listing of graphs, maps, tables, diagrams, and charts has been compiled by Robert Harris [8]. Parallel coordinates is one of the leading methodologies for multidimensional visualization [9]. Starting from geometric foundations, Al Inselberg explains how *n*-dimensional lines and planes can be represented in 2D through parallel coordinates. The most recent publications explain mostly dynamic, interactive methods. Antony Unwin concentrates on graphics for large datasets [16] while Robert Spence favors techniques that allow user interaction [13].

Ultimately, to be of any use, data visualization must support human cognition. The challenges this raises are of interest to cognitive scientists, psychologists, and computer scientists specializing in human-computer interaction. Rudolf Arnheim investigated the role of visual perception as a crucial cognitive activity of reasoning [1]. Also in the domain of 'visual thinking'



Data Visualization. Figure 7. 1967. Bertin's reorderable matrix, a visualization method embedded in a comprehensive theory of graphics.

is the work of Colin Ware [17] as he, among other contributions, proposes a foundation for a science of data visualization based on human visual and cognitive processing. Card et al. discuss topics of computer graphics as well as presentation techniques with a focus on how different methods support cognition. A more general approach worth mentioning is taken by Donald Norman when he argues that people deserve information appliances that fit their needs and lives [11].

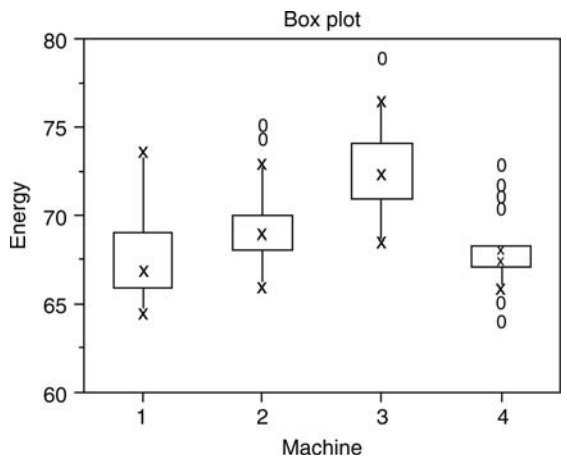
To summarize, most of the literature on data visualization describes the efforts of computer scientists and cognitive scientists to develop new techniques for people to interact with data, from small statistical datasets to large information environments.

Key Applications

There are few – if any – application areas that do not benefit from data visualization simply because graphical methods assist the fundamental human activity of cognition and because in an increasingly digital world people are flooded with data. In the following four areas, data visualization plays a key role:

Statistics

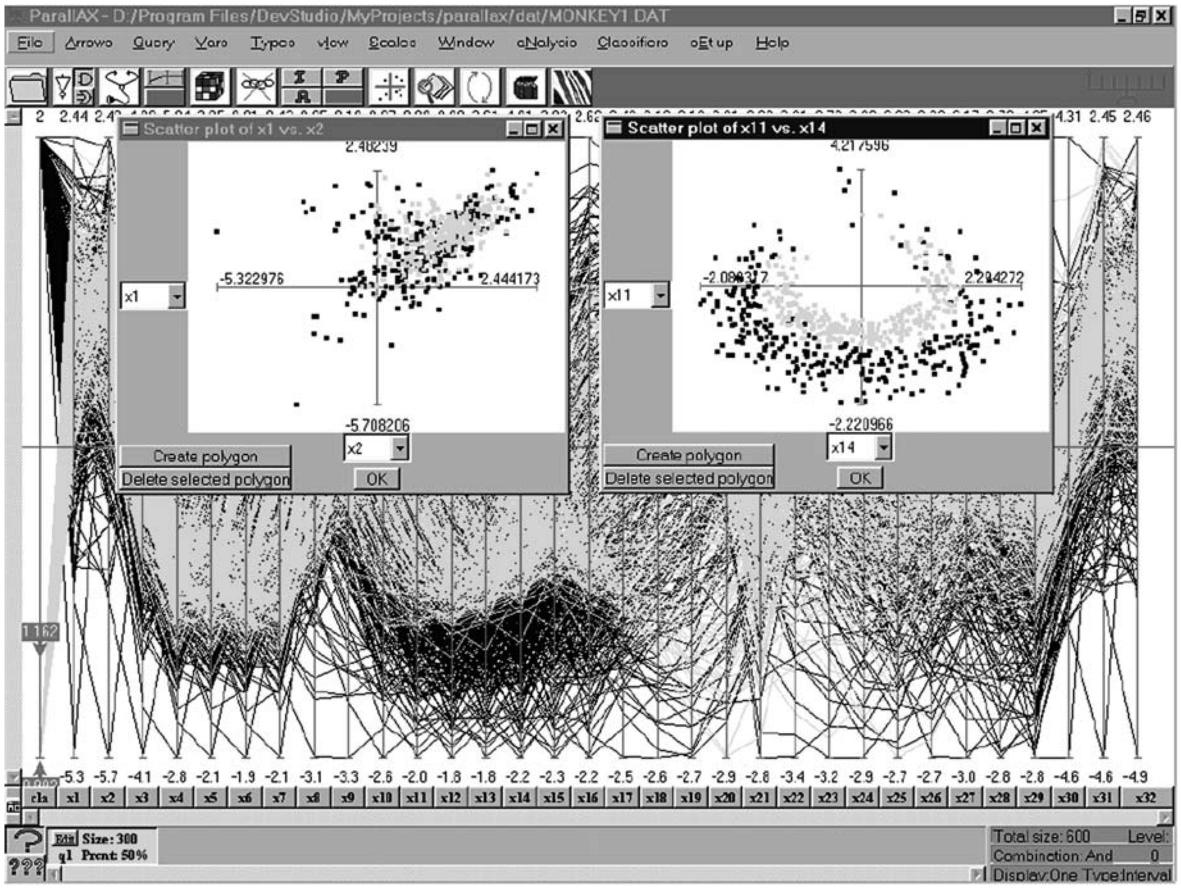
Descriptive statistics has traditionally been the strongest customer for data visualization, primarily through its application to support exploratory data analysis. The use of data visualization as part of descriptive statistics has become a matter of fact wherever data are being collected.



Data Visualization. Figure 8. 1977. Tukey's box-and-whisker plot graphically summarizes effectively key characteristics of the data's distribution.

Information Systems

Data visualization has become an important component in the interface to information systems simply


ParallAX - D:/Program Files/DevStudio/MyProjects/parallax/dat/MONKEY1.DAT
Scatter plot of x1 vs. x2
Scatter plot of x11 vs. x14
Create polygon
Delete selected polygon
OK
Size: 300
Prcnt: 50%
Total size: 600
Combination: And
Display: One Type: Interval

Data Visualization. **Figure 9.** 1999. A continued mathematical development of parallel coordinates led to software for 'visual data mining' in high dimensional data sets.

Data Visualization. Figure 10. 2006. Immersive Geovisualization at West Virginia University.

because information is stored as data. The process of recovering information from large and complex databases often depends on data mining techniques. Visual data mining – a new and rapidly growing field – supports people in their data exploration activity with graphical methods. Geographic information systems have traditionally been key applications, particularly for map-based visualizations.

Documentation

Ever since thematic maps and statistics graphics became popular with commerce, government agencies and the sciences, data visualization methods are being used routinely to illustrate that part of documents which deal with data.

Computational Science

Progress in solving scientific and engineering problems increasingly depends on powerful software for modeling and simulation. Nevertheless, success in the end often only comes with effective scientific visualizations.

Computational science as a key application for data visualization is a strong driving force behind the development of graphical methods for huge amounts of high dimensional data.

Cross-references

- ► Chart
- ► Comparative Visualization
- ► Dynamic Graphics
- ► Exploratory Data Analysis
- ► Graph
- ► Methods
- ► Multivariate Visualization
- ► Parallel Coordinates
- ► Result Display
- ► Symbolic Representation

Recommended Reading

1. Arnheim R. Visual Thinking. University of California Press, Berkeley, CA, 1969.
2. Bertin J. Graphics and Graphic Information-Processing. Walter de Gruyter, Berlin/New York, 1981.
3. Bertin J. Semiology of Graphics (translation by W.J. Berg). University of Wisconsin Press, USA, 1983.
4. Card S.K., MacKinlay J.D., and Shneiderman B. Readings in Information Visualization: Using Vision to Think. Morgan Kaufmann, San Francisco, CA, 1999.
5. Cleveland W.S. The Elements of Graphing Data (Revised Edition). Hobart Press, Summit, NJ, 1994.
6. Ehrenberg A.S.C. A Primer in Data Reduction. Wiley, Chichester, UK, 1982.
7. Friendly M. The History of Thematic Cartography, Statistical Graphics, and Data Visualization.
8. Harris R.L. Information Graphics: A Comprehensive Illustrated Reference. Oxford University Press, New York, 1999.
9. Inselberg A. The plane with parallel coordinates. The Visual Comput., 1(2):69–91, 1985.
10. Nielson G.M., Hagen H., and Müller H. Scientific Visualization: Overviews, Methodologies, Techniques. IEEE Computer Society Press, USA, 1997.
11. Norman D.A. The Invisible Computer. The MIT Press, 1998.
12. Post F.H., Nielson G.M., and Bonneau, G.-P. (eds.). Data Visualization: The State of the Art. Kluwer Academic, 2002.
13. Spence R. Information Visualization: Design for Interaction (2nd edn.). Pearson Education, 2007.
14. Tufte E.R. The Visual Display of Quantitative Information. Graphics Press, 1983.
15. Tukey J.W. Exploratory Data Analysis. Addison-Wesley, Reading, MA, 1977.
16. Unwin A., Theus M., and Hofmann H. Graphics of Large Datasets: Visualizing a Million. Springer Series in Statistics and Computing, Berlin, 2006.
17. Ware C. Information Visualization: Perception for Design (2nd edn.). Morgan Kaufmann, 2004.

Data Warehouse

IL-YEOL SONG
Drexel University, Philadelphia, PA, USA

Synonyms

Information repository
DW

Definition

A data warehouse (DW) is an integrated repository of data put into a form that can be easily understood, interpreted, and analyzed by the people who need to use it to make decisions. The most widely cited definition of a DW is from Inmon [2] who states that "a data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions."

The *subject-oriented* property means that the data in a DW are organized around major entities of interests of an organization. Examples of subjects are customers, products, sales, and vendors. This property allows users of a DW to analyze each subject in depth for tactical and strategic decision-making.

The *integrated* property means that the data in a DW are integrated not only from all operational database systems but also some meta-data and other related external data. When data are moved from operational databases to a DW, they are extracted, cleansed, transformed, and then loaded. This makes a DW a centralized repository of all the business data with common semantics and formats.

The *nonvolatile* property means that the data in a DW are not usually updated. Once the data are loaded into a DW, they are not deleted. Any change to the data that were already moved to a DW is recorded in the form of a snapshot. This allows a DW to keep track of the history of the data.

The *time-variant* property means that a DW usually contains multiple years of data. It is not uncommon for a DW to contain data for more than ten years. This allows users of a DW to analyze trends, patterns, correlations, rules, and exceptions from a historical perspective.

Key Points

DWs have become popular for addressing the needs of a centralized repository of business data in decision-making. An operational database system, also known as an online transaction processing (OLTP) system, supports daily business processing. On the other hand, a DW usually supports tactical or strategic business processing for business intelligence. While an OLTP system is optimized for short transactions, a DW system is optimized for complex decision-support queries. Thus, a data warehouse system is usually maintained separately from operational database systems. This distinction makes DW systems different from OLTP systems in many aspects.

The data in a DW are usually organized in formats for easy access and analysis in decision-making. The most widely used data model for DWs is called the dimensional model or the star schema [3]. A dimensional model consists of two types of entities – a fact table and many dimensions. A *fact* table stores transactional or factual data called measures that are analyzed. Examples of fact tables are order, sale, return, and claim. A dimension represents an axis that analyzes the fact data. Examples of dimensions are time, customer, product, promotion, store, and market. The dimensional model allows users of a data warehouse to analyze the fact data from any combination of dimensions. Thus, a dimensional model simplifies end-user query processing and provides a multidimensional analysis space within a relational database.

The different goals and data models of DWs need special access, implementation methods, maintenance, and analysis methods, different from those of OLTP systems [1]. Therefore, a data warehouse requires an environment that uses a blend of technologies.

Cross-references

► Active and Real-Time Data Warehousing
► Business Intelligence
► Data Mart
► Data Mining
► Data Warehouse Life-cycle and Design
► Data Warehouse Maintenance
► Data Warehouse Metadata
► Data Warehouse Security
► Data Warehousing and Quality Data Management for Clinical Practice
► Data Warehousing for Clinical Research
► Data Warehousing Systems: Foundations and Architectures
► Dimension
► Evolution and versioning
► Multidimensional Modeling
► On-Line Analytical Processing

Recommended Reading

1. Chaudhuri S. and Dayal U. An overview of data warehousing and OLAP technology, SIGMOD Rec., 26(1):65–74, 1997.
2. Inmon W.H. Building the Data Warehouse, 3rd edn. Wiley, New York, 2002.
3. Kimball R. and Ross M. The Data Warehouse Toolkit, 2nd edn. Wiley, New York, 2002.

Data Warehouse Back Stage

► Extraction, Transformation and Loading

Data Warehouse Design Methodology

► Data Warehouse Life-Cycle and Design

Data Warehouse Life-Cycle and Design

Matteo Golfarelli
University of Bologna, Bologna, Italy

Synonyms

Data Warehouse design methodology

Definition

The term *data warehouse life-cycle* is used to indicate the phases (and their relationships) a data warehouse system goes through between when it is conceived and when it is no longer available for use. Apart from the type of software, life cycles typically include the following phases: requirement analysis, design (including modeling), construction, testing, deployment, operation, maintenance, and retirement. On the other hand, different life cycles differ in the relevance and priority with which the phases are carried out, which can vary according to the implementation constraints (i.e., economic constraints, time constraints, etc.) and the software specificities and complexity. In particular, the specificities in the data warehouse life-cycle derive from the presence of the operational database that feeds the system and by the extent of this kind of system that must be considered in order to keep the cost and the complexity of the project under control.

Although the design phase is only a step within the overall life cycle, the identification of a proper life-cycle model and the adoption of a correct *design methodology* are strictly related since each one influences the other.

Historical Background

The *data warehouse* (DW) is acknowledged as one of the most complex information system modules, and its design and maintenance is characterized by several complexity factors, which determined, in the early stages of this discipline, a high percentage of project failures. A clear classification of the critical factors of Data Warehousing projects was already available in 1997 when three different risk categories were identified [1]:

- *Socio-technical:* DW projects have deep impact on the decisional processes and political equilibriums, thus reducing the power of some stakeholders who will be willing to interfere with the project. For example, data ownership is power within an organization. Any attempt to share or take control over somebody else's data is equivalent to a loss of power of this particular stakeholder. Furthermore, no division or department can claim to possess 100% clean, error-free data. The possibility of revealing the quality problems of data within the information system of the department is definitely frustrating for the stakeholders affected.
- *Technological:* DW technologies are continuously evolving and their features are hard to test. As a consequence, problems related to the limited scalability of the architecture, difficulty in sharing meta-data between different components and the inadequate expertise of the programmers may hamper the projects.
- *Design:* designing a DW requires a deep knowledge of the business domain. Some recurrent errors are related to limited involvement of the user communities in the design as well as the lack of a deep analysis of the quality of the source data. In both these cases, the information extracted from the DW will have a limited value for the stakeholders since they will turn out to be unreliable and outside the user focus.

The awareness of the critical nature of the problems and the experience accumulated by practitioners determined the development of different design methodologies and the adoption of proper life cycles that can increase the probability of completing the project and fulfill the user requirements.

Foundations

The choice of a correct life cycle for the DW must take into account the specificities of this kind of systems, which according to [2], are summarized as follows:

1. DWs rely on operational databases that represent the sources of the data.
2. User requirements are difficult to collect and usually change during the project.
3. DW projects are usually huge projects: the average time for their construction is 12–36 months and their average cost ranges from 0.5 to 10 million dollars.
4. Managers are demanding users that require reliable results in a time compatible with business needs.

While there is no consensus on how to address points (i) and (ii), the DW community has agreed on an approach that cuts down cost and time to make a satisfactory solution available to the final users. Instead of approaching the DW development as a whole in a top-down fashion, it is more convenient to build it bottom-up working on single data marts [3]. A *data mart* is part of a DW with a restricted scope of content and support for analytical processing, serving a single department, part of an organization, and/or a particular data analysis problem domain. By adopting a bottom-up approach, the DW will turn out to be the union of all the data marts.

This iterative approach promises to fulfill requirement (iii) since it cuts down development costs and time by limiting the design and implementation efforts to get the first results. On the other hand, requirement (iv) will be fulfilled if the designer is able to implement first those data marts that are more relevant to the stakeholders.

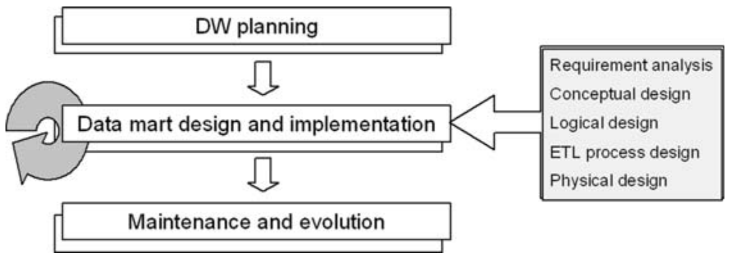
As stated by many authors, adopting a pure bottom-up approach presents many risks originating from the partial vision of the business domain that will be available at each design phase. This risk can be limited by first developing the data mart that plays a central role within the DW, so that the following can be easily integrated into the existing backbone; this kind of solution is also called *bus architecture*. The basis for designing coherent data marts and for achieving an integrated DW is the agreement of all the design teams on the classes of analysis that are relevant for the business. This is primarily obtained by the adoption of *conformed dimensions* of analysis [4]. A dimension is conformed when two copies of the dimensions are either exactly the same (including the values of the keys and all the attributes), or else one dimension is a proper subset of the other. Therefore, using the same time dimension in all the data marts implies that the data mart teams agree on a corporate calendar. All the data mart teams must use this calendar and agree on fiscal periods, holidays, and workdays. When choosing the first data mart to be implemented the designer will probably cope with the fact that the most central data mart (from a technical point of view) is not the most relevant to the user. In that case, the designer choice must be a trade-off between technical and political requirements.

Based on these considerations the main phases for the DW life-cycle can be summarized as follows:

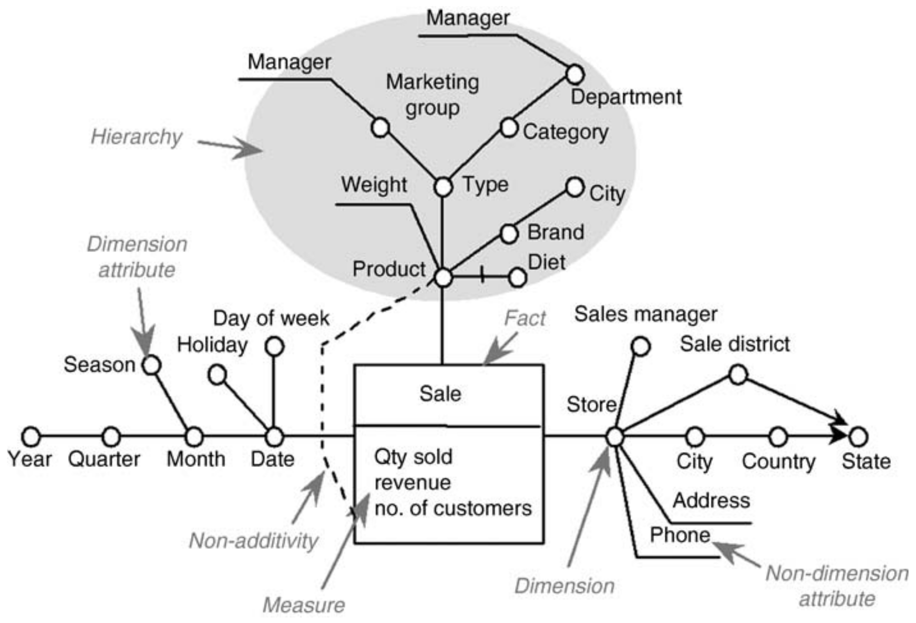
1. *DW planning*: this phase is aimed at determining the scope and the goals of the DW, and determines the number and the order in which the data marts are to be implemented according to the business priorities and the technical constraints [5]. At this stage the physical architecture of the system must be defined too: the designer carries out the sizing of the system in order to identify appropriate hardware and software platforms and evaluates the need for a reconciled data level aimed at improving data quality. Finally, during the project planning phase the staffing of the project is carried out.
2. *Data mart design and implementation*: this macro-phase will be repeated for each data mart to be implemented and will be discussed in more detail in the following. At every iteration, a new data mart is designed and deployed. Multidimensional modeling of each data mart must be carried out considering the available conformed dimensions and the constraints derived from previous implementations.
3. *DW maintenance and evolution*: DW maintenance mainly concerns performance optimization that must be periodically carried out due to user requirements that change according to the problems and the opportunities the managers run into. On the other hand, DW evolution concerns keeping the DW schema up-to-date with respect to the business domain and the business requirement changes: a manager requiring a new dimension of analysis for an existing *fact schema* or the inclusion of a new level of classification due to a change in a business process may cause the early obsolescence of the system (Fig. 1).

DW design methodologies proposed in the literature mainly concern phase 2 and thus should be better referred to as data mart design methodologies. Though a lot has been written about how a DW should be designed, there is no consensus on a design method yet. Most methods agree on the opportunity of distinguishing between the following phases:

- *Requirement analysis:* identifies which information is relevant to the decisional process by either considering the user needs or the actual availability of data in the operational sources.
- *Conceptual design*: aims at deriving an implementation-independent and expressive conceptual schema for the DW, according to the conceptual model chosen (see Fig. 2).
- *Logical design*: takes the conceptual schema and creates a corresponding logical schema on the chosen logical model. While nowadays most of the DW systems are based on the relational logical model (ROLAP), an increasing number of software vendors are proposing also pure or mixed multidimensional



Data Warehouse Life-Cycle and Design. Figure 1. The main phases for the DW life-cycle.



Data Warehouse Life-Cycle and Design. Figure 2. A conceptual representation for the SALES fact based on the DFM model [6].

solutions (MOLAP/HOLAP). Figure 3 reports the relational implementation of the SALE fact based on the well-known star schema [4].

- *ETL process design*: designs the mappings and the data transformations necessary to load into the logical schema of the DW the data available at the operational data source level.
- *Physical design*: addresses all the issues specifically related to the suite of tools chosen for implementation – such as indexing and allocation.

Requirement analysis and conceptual design play a crucial role in handling DW peculiarities (i) and (ii) described at the beginning of the present section. The lack of settled user requirements and the existence of operational data sources that fix the set of available information make it hard to develop appropriate multidimensional schemata that on the one hand fulfill user requirements and on the other can be fed from the operational data sources. Two different design principles can be identified: supply-driven and demand-driven [5].

- *Supply-driven* approaches [3,6] (also called *data-driven*) start with an analysis of operational data sources in order to reengineer their schemata and identify all the available data. Here user involvement is limited to select which chunks of the available data are relevant for the decision-making process. While supply-driven approaches simplify the design of the ETL because each piece of data in the DW corresponds to one or more attributes of the sources, they give user requirements a secondary role in determining the information contents for analysis as well as giving the designer little support in identifying facts, dimensions, and measures. Supply-driven approaches are feasible when all of the following are true: (i) detailed knowledge of data sources is available a priori or easily achievable; (ii) the source schemata exhibit a good degree of normalization; and (iii) the complexity of source schemata is not too high.
- *Demand-driven* approaches [7,8] start from determining the information requirements of business users. The emphasis is on the requirement analysis process and on the approaches for facilitating user participations. The problem of mapping these requirements onto the available data sources is faced only a posteriori, and may fail thus determining the users' disappointment as well as a waste of the designer's time.

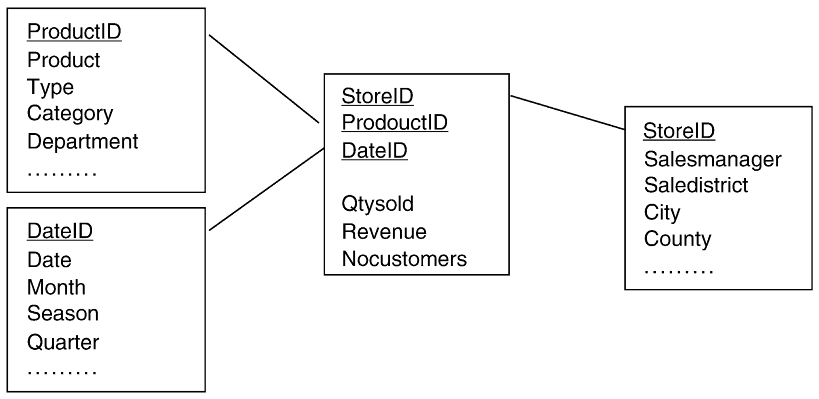
Based on the previous approaches some mixed modeling solutions have been proposed in the last few years in order to overcome the weakness of each pure solution.

Conceptual design is widely recognized to be the necessary foundation for building a DW that is well-documented and fully satisfies the user requirements. The goal of this phase is to provide the designer with a high level description of the data mart possibly at different levels of detail. In particular, at the DW level it is aimed at locating the data mart within the overall DW picture, basically characterizing the class of information captured, its users, and its data sources. At the data mart level, a conceptual design should identify the set of facts to be built and their conformed dimensions. Finally, at the fact level a nonambiguous and implementation-independent representation of each fact should be provided. If a supply driven approach has been followed for requirement analysis, the conceptual model at the schema level can be semi-automatically derived from the source schemata by identifying the many-to-one relationship [3,6]. Concerning the formalism to be adopted for representing information at this level, researchers and practitioners agreed that, although the E/R model has enough expressivity to represent most necessary concepts, in its basic form, it is not able to properly emphasize the key aspects of the multidimensional model. As a consequence many ad-hoc formalisms has been proposed in the last years (e.g. [6,9]) and a comparison of the different models done by [10] pointed out that, abstracting from their graphical form, the core expressivity is similar, thus proving that the academic community reached an informal agreement on the required expressivity.

Logical design is the phase that most attracted the interest of researchers in the early stage of Data Warehousing since it strongly impacts the system performance. It is aimed at deriving out of the conceptual schemata the data structure that will actually implement the data mart by considering some sets of constraints (e.g., concerning disk space or query answering time) [11]. Logical design is more relevant when a relational DBMS is adopted (ROLAP) while in the presence of a native multidimensional DBMS (MOLAP) the logical model derivation is straightforward. On the other hand, in ROLAP system, the choices concern, for example the type of schema to be adopted (i.e., star o snowflake), the specific solution for historicization of data (i.e., slowly changing dimensions) and schema.

ETL process design is considered to be the most complex design phase and usually takes up to 70% of the overall design time. Complexity arises from the need of integrating and transforming heterogeneous and inconsistent data coming from different data sources. This phase also includes the choice of the strategy for handling wrong and incomplete data (e.g. discard, complete). Obviously, the success of this phase impacts the overall quality of DW data. Different from other design phases little efforts have been made in the literature to organize and standardize this phase [12,13], and actually none of the formalisms proposed have been widely adopted in real projects that usually rely on the graphical representation obtained from the ETL tool for documentation purposes.

Finally, during physical design, the logical structure is optimized based on the means made available by the adopted suite of tools. Specialized DBMSs usually include ad hoc index types (e.g., bitmap index and join index) and can store the meta-knowledge



Data Warehouse Life-Cycle and Design. Figure 3. A relational implementation of the SALE fact using the well-known star schema.

necessary to automatically rewrite a given query on the appropriate materialized view. In DW systems, a large part of the available disk space is devoted to optimization purposes and it is a designer task to find out its assignment to the different optimization data structures in order to maximize the overall performance [14].

Despite the basic role played by a well-structured methodological framework in ensuring that the DW designed fully meets the user expectations, only a few of the cited papers cover all the design phases [6,13]. In addition, an influential book, particularly from the practitioners' viewpoint, is the one by Kimball [4], which discusses the major issues arising in the design and implementation of data warehouses. The book presents a case-based approach to data mart design that is bottom-up oriented and adopts a mixed approach for collecting user requirements.

Finally it should be noted that, though most vendors of DW technology propose their own CASE solutions (that are very often just wizards capable of supporting the designer during the most tedious and repetitive phases of design), the only tools that currently promise to effectively automate some phases of design are research prototypes. In particular, [3,15], embracing the supply-driven philosophy, propose two approaches for automatically deriving the conceptual multidimensional schema from the relational data sources. On the contrary the CASE tool proposed in [12] follows the demand-driven approach and allows the multidimensional conceptual schemata to be drawn from scratch and to be semi-automatically translated into the target commercial tool.

Key Applications

The adoption of an appropriate methodological approach during design phases is crucial to ensure the project success. People involved in the design must be skilled on this topic, in particular.

Designers

Designers should have a deep knowledge of the pros and cons of different methodologies in order to adopt the one that best fits the project characteristics.

Business Users

Users should be aware of the design methodology adopted and their role within it in order to properly support the designer's work and to provide the correct information at the right time.

Future Directions

Research on this topic should be directed to generalizing the methodologies discussed so far in order to derive a consensus approach that, depending on the characteristics of the project, will be made up of different phases. Besides, more generally, mechanisms should appear to coordinate all DW design phases allowing the analysis, control, and traceability of data and metadata along the project life-cycle. An interesting approach in this direction consists in applying the Model Driven Architecture to automate the inter schema transformations from requirement analysis to implementation [16]. Finally, the emergence of new applications for DW such as spatial DW [17], web DW, real-time DW [18], and business performance management [19] will have their side-effects on the DW life-cycle and inevitably more general design methodologies will be devised in order to allow their correct handling.

Cross-references

► Cube Implementations
► Data Mart
► Data Warehouse Maintenance evolution and versioning
► Data Warehousing Systems: Foundations and Architectures
► Multidimensional Modeling
► Optimization and Tuning in Data Warehouse
► Snowflake Schema
► Star Schema

Recommended Reading

1. Abello A., Samos J., and Saltor F.YAM2: a multidimensional conceptual model extending UML. Infor. Syst., 31(6):541–567, 2006.
2. Bimonte S., Towards S., and Miquel M.Towards a Spatial Multidimensional Model. In Proc. ACM 8th Int. Workshop on Data Warehousing and OLAP, 2005.
3. Demarest, M. The politics of data warehousing. Retrieved June 2007 from http://www.noumenal.com/marc/dwpoly.html.
4. Giorgini P., Rizzi S., and Garzetti M. GRAnD: A goal-oriented approach to requirement analysis in data warehouses. *Decision Support System*, 2008, 45(1):4–21.
5. Golfarelli M., Maio D., and Rizzi S. The dimensional fact model: a conceptual model for data warehouses. IJCIS 7(2–3): 215–247, 1998.
6. Golfarelli M. and Rizzi S. WAND: A CASE tool for data warehouse design. In Proceedings of *ICDE*, Bremen, Germany, 2001.
7. Golfarelli M., Rizzi S., and Cella I. Beyond data warehousing: What's next in business intelligence? In Proc. ACM 7th Int. Workshop on Data Warehousing and OLAP, 2004.

8. Golfarelli M., Rizzi S., and Saltarelli E. Index selection for data warehousing. In Proc. 4th Int. Workshop on Design and Management of Data Warehouses, 2002.
9. Hüsemann B., Lechtenbörger J., and Vossen G. Conceptual data warehouse design, Proc. 2nd Int. Workshop on Design and Management of Data Warehouses, 2000.
10. Jarke M., Lenzerini M., Vassiliou Y., and Vassiliadis P. Fundamentals of Data Warehouses. Springer, 2000.
11. Jensen M., Holmgren T., and Pedersen T. Discovering Multidimensional Structure in Relational Data. In Proc. 6th Int. Conf. Data Warehousing and Knowledge Discovery, 2004.
12. Kimbal R., Reeves L., Ross M., and Thornthwaite W. The Data Warehouse Lifecycle Toolkit. Wiley, New York, 1998.
13. Laender A., Freitas G., and Campos M. MD2 – Getting users involved in the development of data warehouse applications. In Proc. 14th Int. Conf. on Advanced Information Systems Eng., 2002.
14. Mazon J., Trujillo J., Serrano M., and Piattini M. Applying MDA to the development of data warehouses. In Proc. ACM 8th Int. Workshop on Data Warehousing and OLAP, 2005.
15. Theodoratos D. and Sellis T. Designing data Data warehouses. DKE, 31(3):279–301, 1999.
16. Tho N. and Tjoa A. Grid-Based Zero-Latency Data Warehousing for continuous data streams processing. In Proc. *IIWAS2004*, 2004.
17. Trujillo J. and Luján-Mora S.A. UML Based Approach for Modeling ETL Processes in Data Warehouses. In Proc. 22nd Int. Conf. on Conceptual Modeling, 2003.
18. Trujillo J., Luján-Mora S., and Medina E. The Gold model case tool: An environment for designing OLAP applications. In Proc. ACM 5th Int. Workshop on Data Warehousing and OLAP, 2002.
19. Vassiliadis P., Simitsis A., and Skiadopoulos S. Conceptual modeling for ETL processes. In Proc. ACM 5th Int. Workshop on Data Warehousing and OLAP, 2002.
20. Winter R. and Strauch B. A method for demand-driven information requirements analysis in data warehousing. In Proc. *HICSS*, Ciutad Real, Spain, 2003.

Data Warehouse Maintenance, Evolution and Versioning

JOHANN EDER[1], KARL WIGGISSER[2]
[1]University of Vienna, Vienna, Austria
[2]University of Klagenfurt, Klagenfurt, Austria

Synonyms

Temporal data warehousing

Definition

A multidimensional data warehouse consists of three different levels: The schema level (dimensions, categories), the instance level (dimension members, master data) and the data level (data cells, transaction data). The process and methodology of performing changes on the schema and instance level to represent changes in the data warehouse's application domain or requirements is called *Data Warehouse Maintenance*. *Data Warehouse Evolution* is a form of data warehouse maintenance where only the newest data warehouse state is available. *Data Warehouse Versioning* is a form of data warehouse maintenance where all past versions of the data warehouse are kept available. Dealing with changes on the data level, mostly insertion of new data, is not part of data warehouse maintenance, but part of a data warehouse's normal operation.

Historical Background

Data warehouses are supposed to provide functionality for storing and analyzing data over a long period of time. Since the world is changing, the need for applying changes to data warehouse structures arose. Kimball [8] was probably the first to describe the problem and propose solutions. Several more sophisticated proposals followed (see below).

Foundations

A multidimensional data warehouse consists of three different levels: The schema level, the instance level, and the data level. On the schema level a data warehouse is defined by a set of dimensions and corresponding dimension categories, which build up a category hierarchy. On the instance level a data warehouse is defined by a set of dimension members for each dimension. Dimension members build up a member hierarchy which corresponds to the category hierarchy of the respective dimension. Schema and instance level together define the structure of a data warehouse. Different multidimensional models deal with measures in different ways. If no particular measure dimension is defined, measures are modeled as attributes of the fact table, thus are seen as part of the schema. If there is a measure dimension existing, measures are members of this particular dimension and therefore seen as instances. On the data level, a data warehouse consists of a set of data cells, which hold the actual values to analyze. A data cell is defined by selecting one dimension member from each dimension.

Whereas changes on the data level, most of the time data inserts, are part of the daily business in data warehouse systems, modifications of the data warehouse structure need additional effort. Structural modifications can be implied by changes in the application domain of a data warehouse system or by changes in the requirements.

Levels of the Maintenance Problem

Data warehouse maintenance systems must provide means to keep track of schema modifications as well as of instance modifications. On the schema level one needs operations for the *Insertion*, *Deletion* and *Change* of dimensions and categories. Category changes are for instance adding or deleting user defined attributes. Also the hierarchical relations between categories may be modified. On the instance level operations for the *Insertion, Deletion* and *Change* of dimension members are needed, as well as operations for changing the hierarchical relations between dimension members. Whether changing measures is a schema or instance change depends on the underlying multidimensional model. Typically, schema changes happen rarely but need much effort to be dealt with, whereas modifications of instances may happen quite often, but need fewer effort.

Keeping track of the data warehouse structure is only one aspect of data warehouse maintenance. The structure of the cell data contained in a data warehouse is determined by the data warehouse's structure. Thus, if this structure changes, existing cell data may have to be adjusted to be consistent with the new structure. Such adjustments can range from simple reaggregation to complex data transformations because for instance some unit of a measure is changed. These data adaptations must not be mistaken for data change operations as mentioned above, for instance loading new data into the data warehouse.

Figure 1 shows an example for instance and schema changes. It contains three subsequent versions of one dimension of a car dealer's data warehouse structure together with the categories for this dimension. On top, the initial version is shown. The dealer sells different car models of different brands. Each model has an attribute which denotes the engine power. For traditional German models this is given in horsepower, for English models it is given in kilowatt. The outline in the middle shows the subsequent version, where two instance changes can be seen: a new model *(BMW 1)* is introduced, and one model *(Phantom V)* is discontinued. The bottom outline shows the current structure version. Here one can see a schema change: a new category *(Company)* is inserted into the category hierarchy. On the instance level there are a number of changes: one brand *(Puch)* is removed from the product portfolio. The model *(Modell G)* attached to this brand is now sold under another brand *(Mercedes)*. Furthermore a new brand *(Chrysler)* was added to the product portfolio, together with one model assigned to it. For the newly introduced category two dimension members *(BMW&Rolls-Royce* and *Daimler-Chrysler)* are added and the brands are connected to the respective company. The attribute denoting the power of a model is unified for all models to kilowatt. All the mentioned structure modifications are due to changes in the application domain. A requirements change leading to structure updates could for instance be that besides analyzing the number of car sells, the car dealer also wants to keep track of the resulting profit (insert measure).

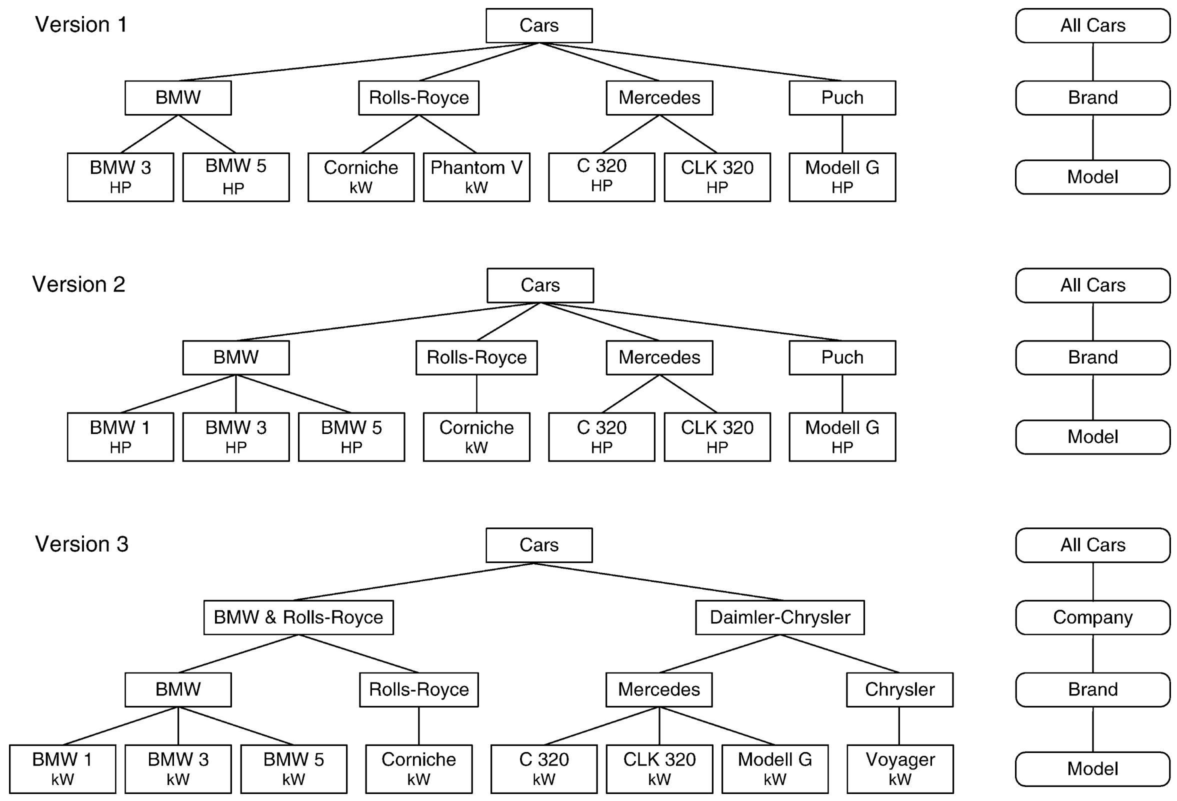
A data adjustment for this example would be the reaggregation to express that *Modell G* is now sold under the brand of *Mercedes*. Data transformation could for instance result from changing the currency from ATS to EUR, where every money-related value has to be divided by 13.7603.

Data Warehouse Versioning Versus Data Warehouse Evolution

In principle two methods of maintenance can be distinguished: Evolution and Versioning. Both of these techniques rely on the defined operations for structure changes but significantly vary in terms of query flexibility, query costs and data management effort. This distinction between versioning and evolution can be applied for both the schema and the instance level.

With *Data Warehouse Evolution*, every applied operation changes the structure of the data warehouse and the old structure is lost. The respective cell data is transformed to fit the new structure. As the old structure is lost, queries can only be done against the current structure. Queries spanning different structure versions are not possible. As the data follows one single structure, no adaptations have to be done during query runtime, which results in a better query performance compared to the versioning approach. Furthermore, no information about former versions has to be kept, which reduces the effort for data management.

With *Data Warehouse Versioning* every applied operation again leads to a new structure version. But in contrast to the evolutionary approach the old version is also kept available. Existing cell data does not need to be adapted, but can be stored further on following



Data Warehouse Maintenance, Evolution and Versioning. Figure 1. Changes in Data Warehouse Structure.

the respective structure version. This facilitates queries spanning multiple structure versions. When running such multiversion queries, data has to be either adapted in runtime, which reduces query performance, or pre-calculated and stored, which increases the required space and maintenance effort. Keeping track of structure version history is mandatory, which results in a considerable effort for the data management.

Approaches Addressing the Maintenance Problem

There are a set of approaches addressing the data warehouse maintenance problem. Kimball [8] is one of the first, discovering the need for evolving data warehouses and introducing three methods for dealing with "slowly changing dimensions". The first method proposes simply overwriting old instances with their new values. Tracking a change history is not possible. The second method consists in creating a new instance for each change. This will create a change history, but needs additional effort in data management. One has to introduce a surrogate key, because the natural primary keys may not be unique any longer. For relating the various instances for an object to each other, creating a time stamp for the validity of each version is proposed. The third method proposes creating a new attribute for the instance, such that the original and the current attribute value can be saved. This method can of course only handle two versions of an instance. All three methods are quite straightforward and only allow very basic modifications on the instance level.

With FIESTA [2] Blaschka, Sapia and Höfling present a schema design technique supporting schema evolution. Evolution for instances is not supported, but FIESTA provides an automatism to adapt existing instances after schema modification. For this adaptation two alternatives are proposed: adaption on the physical level (i.e., database changes) and adaption on the logical level (i.e., create a filter for accessing the instances). The authors define a rich set of schema changing operations, including the creation and deletion of dimensions, categories and attributes.

In [11] Ravat and Teste present their approach for dealing with changing instances. The authors define an object-oriented approach for data warehouse modeling, based on the class concept proposed by the Object Database Management Group. A warehouse object (instance) is defined by its current state and a set of historical and archived states. The difference between historical and archived states is that historical states can be exactly reestablished, whereas for archived states only aggregations are kept, for reducing data size. Mapping functions describe the building process from which the data warehouse classes are generated.

The approach of Hurtado, Mendelzon and Vaisman [14] allows data warehouse evolution on the schema and the instance level. Both schema and instances are modeled using a directed acyclic graph where the nodes represent levels and instances, respectively. The edges are labeled with their valid time intervals. Nodes connected to edges are only valid in the time interval where the edge is valid. Operations for inserting and deleting categories and instances are provided. Evolution of instances is not supported. Defining whether a specific instance is part of the current schema happens by timestamping the edge which connects the node to the graph. Additionally the temporal query language TOLAP is defined to enable queries over a set of temporal dimensions and temporal fact tables.

In [4,5] Eder and Koncilia present their COMET Metamodel for temporal data warehousing. Based on the principles of temporal databases, they introduce a system that supports data warehouse versioning on the schema and the instance level. COMET provides a rich set of maintenance operations, which comprise insertion, deletion, and update of schema elements and instances. Also the complex operations split member and merge members are defined. In contrast to other approaches these operations can also be applied on the time and fact dimensions. COMET furthermore defines so called transformation functions, which allow to transform the cell data between arbitrary versions of the data warehouse. This provides the functionality of queries spanning several structure versions.

In [6] Golfarelli et al. present their approach for schema versioning in data warehouse. Based on a graph model of the data warehouse schema they present their algebra for schema modifications. This approach supports versioning, therefore past versions are not lost. Based on those schema versions the authors describe a mechanism to execute cross-version queries, with the help of so called augmented schemas. For creating such an augmented schema, an old schema version is enriched with structure elements from a subsequent version, such that the data belonging to the old schema version can be queried as if it follows the new version.

Besides these research proposals there are also two commercial products which introduce basic means for data warehouse maintenance. SAP Inc. describes in a white paper [9] how to produces different types of reports over existing data. This can be a report using the current constellation, a report using an old constellation, a report showing the historical truth, and a report showing comparable results. This approach supports only basic operations on dimension data.

The KALIDO Dynamic Information Warehouse [7] also realizes some aspects of data warehouse maintenance. Their support for change is based on the so called generic data modeling. The data warehouse model consists of three categories of data, the transaction data (which describes the activities of the business and the measures associated with them), the business context data (which is the analog to the instances), and the metadata (which comprises among others, parts the schema). With evolving the business context data, instance evolution is supported.

There are a set of alternative approaches which have not been mentioned yet. The different techniques addressing the data warehouse maintenance problem can be classified by two features: First, by whether they support structure versioning or structure evolution, and second by the level of modifications they can handle. Table 1 shows this classification for some of the best known approaches in this area. So each of the mentioned approaches provides the features naming the respective row and column.

Besides the classical maintenance requirements of keeping track of changes in data warehouse, maintenance methodologies can also be used to facilitate so called what–if-analysis. In [1] Bebel et al. present their approach for the management of multiversion data warehouses. They differentiate between real versions and alternative versions. Real versions are used to historicize data warehouse modifications resulting from real world changes. Alternative versions provide the

Data Warehouse Maintenance, Evolution and Versioning. Table 1. Classification of data warehouse maintenance approaches

	Versioning	Evolution
Schema and instance maintenance	[4]	[14]
Schema maintenance only	[6]	[2,10]
Instance maintenance only	[9,11,13]	[7,3,8,15]

functionality to create several versions, each of them representing a possible future situation and then apply what–if-analysis on them. Additionally, alternative versions can be used to simulate data warehouse changes for optimization purposes.

Another instance of data warehouse maintenance is the so called view maintenance. Whereas the approaches presented above assume a data warehouse structure which is defined somehow independent from underlying data sources and is populated with data by ETL-processes, a data warehouse can also be seen as materialized view over a set of data sources. Such a materialized view is of course directly affected by changes in the sources. For instance, in [16] Zhuge et al. present their approach for view maintenance. But as these approaches most times only deal with data updates, they are out of scope for data warehouse maintenance. Rundensteiner et al. [12] present a view maintenance approach which can also deal with changing structures. Their evolvable view management is realized as middleware between the data sources and the data warehouse. A core feature is the so called evolvable SQL which allows to define preferences for view evolution. With these preferences it is possible to redefine the view after some source changes, such that the resulting view is possibly not equivalent the to original view any more, but still fulfills the user's needs.

Key Applications

Data warehouses are often used to efficiently support the decision making process in companies and public authorities. To fulfil this task they have to represent the application domain and users' requirements. To keep the analysis results accurate and correct over the time, data warehouse maintenance is a crucial issue. Application domains which are typically vulnerable to changing structures are among others statistic and geographic applications (for instance statistical data in the European Union), health care (for instance switching from International Classification of Deceases Version 9 to Version 10), or stock market (for instance splitting stocks). In each of these domains, traceability and comparability of data over long periods of time are very important, thus effective and efficient means to provide these capabilities have to be defined.

Future Directions

Current commercial systems assume the data warehouse structure to be constant, therefore their support for modifications is rather limited. On the other hand, in real-world applications the demand for changing structures is rather high, as the data warehouse has to be consistent with the application domain and the requirements. Despite the fact that more effort is put into integrating maintenance capabilities into commercial data warehouse systems [9,7], current products are still not well prepared for this challenge.

Whereas schema and instance maintenance is quite elaborated in current research papers, the efficient transformation of cell data between different versions is still subject to research. The main problems with data transformation are first of all defining semantically correct transformation functions, and second the oftentimes huge amount of cell data which has to be handled in an efficient way.

Related to data transformation is the problem of multiversion queries. The problem with such queries is defining the desired semantics and structure of the outcome, i.e., whether and how elements and cell values, which are not valid for all affected versions should be included in the result.

Cross-references

► Data Warehousing Systems: Foundations and Architectures
► On-line Analytical Processing
► Optimization and Tuning in Data Warehouses
► Quality of Data Warehouses
► Schema Versioning
► Temporal Database
► What–If Analysis

Recommended Reading

1. Bębel B., Eder J., Koncilia C., Morzy T., and Wrembel R. Creation and management of versions in multiversion data warehouse. In Proc. 2004 ACM Symp. on Applied computing, 2004, pp. 717–723.
2. Blaschka M., Sapia C., and Höfling G. On schema evolution in multidimensional databases. In Proc. Int. Conf. on Data Warehousing and Knowledge Discovery, 1999, pp. 153–164.
3. Chamoni P. and Stock S. Temporal structures in data warehousing. In Proc. Int. Conf. on Data Warehousing and Knowledge Discovery, 1999, pp. 353–358.
4. Eder J., Koncilia C., and Morzy T. The COMET Metamodel for Temporal Data Warehouses. In Proc. Int. Conf. on Advanced Information Systems Engineering, 2002, pp. 83–99.
5. Eder J., Koncilia C., and Wiggisser K. Maintaining temporal warehouse models. In Proc. Int. Conf. on Research and Practical Issues of Enterprise Information Systems, 2006, pp. 21–30.

6. Golfarelli M., Lechtenbörger J., Rizzi S., and Vossen G. Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation. Data & Knowledge Engineering, 59:435–459, 2006.
7. KALIDO Dynamic Information Warehouse: A Technical Overview. Tech. rep., Kalido, 2004.
8. Kimball R. Slowly Changing Dimensions. DBMS Magazine, 9(4):14, 1996.
9. Multi-Dimensional Modeling with BW: ASAP for BW Accelerator. Tech. rep., SAP Inc., 2000.
10. Quix C. Repository Support for Data Warehouse Evolution. In Proc. Int. Workshop on Design and Management of Data Warehouses, 1999.
11. Ravat F. and Teste O. A Temporal Object-Oriented Data Warehouse Model. In Proc. Int. Conf. on Database and Expert Systems Applications, 2000, pp. 583–592.
12. Rundensteiner E.A., Koeller A., and Zhang X. Maintaining data warehouses over changing information sources. Commun. ACM, 43(6):57–62, 2000.
13. Sarda N.L. Temporal Issues in Data Warehouse Systems. In Proc. Int. Symp. on Database Applications in Non-Traditional Environments, 1999.
14. Vaisman A. and Mendelzon A. A Temporal Query Language for OLAP: Implementation and a Case Study. In Proc. Int. Workshop on Database Programming Languages, 2001, pp. 78–96.
15. Yang J. and Widom J. Maintaining temporal views over non-temporal information sources for data warehousing. In Proc. Int. Conf. on Extending Database Technology. 1998, pp. 389–403.
16. Zhuge Y., Garcia-Molina H., Hammer J., and Widom J. View Maintenance in a Warehousing Environment. In Proc. ACM SIGMOD Int Conf. on Management of Data, 1995, pp. 316–327.

Data Warehouse Indexing

► Indexing of Data Warehouses

Data Warehouse Integration

► Interoperability in Data Warehouses

Data Warehouse Metadata

Panos Vassiliadis
University of Ioannina, Ioannina, Greece

Definition

Data warehouse metadata are pieces of information stored in one or more special-purpose *metadata repositories* that include (i) information on the contents of the data warehouse, their location and their structure, (ii) information on the processes that take place in the data warehouse back-stage, concerning the refreshment of the warehouse with clean, up-to-date, semantically and structurally reconciled data, (iii) information on the implicit semantics of data (with respect to a common enterprise model), along with any other kind of data that aids the end-user exploit the information of the warehouse, (iv) information on the infrastructure and physical characteristics of components and the sources of the data warehouse, and, (v) information including security, authentication, and usage statistics that aids the administrator tune the operation of the data warehouse as appropriate.

Historical Background

Data warehouses are systems with significant complexity in their architecture and operation. Apart from the central data warehouse itself, which typically involves an elaborate hardware architecture, several sources of data, in different operational environments are involved, along with many clients that access the data warehouse in various ways. The infrastructure complexity is only one part of the problem; the largest part of the problem lies in the management of the data that are involved in the warehouse environment. Source data with different formats, structure, and hidden semantics are integrated in a central warehouse and then, these consolidated data are further propagated to different end-users, each with a completely different perception of the terminology and semantics behind the structure and content of the data offered to them. Thus, the administrators, designers, and application developers that cooperate towards bringing clean, up-to-date, consolidated and unambiguous data from the sources to the end-users need to have a clear understanding of the following issues (see more in the following section):

1. The location of the data
2. The structure of each involved data source
3. The operations that take place towards the propagation, cleaning, transformation and consolidation of the data towards the central warehouse
4. Any audit information concerning who has been using the warehouse and in what ways, so that its performance can be tuned

5. The way the structure (e.g., relational attributes) of each data repository is related to a common model that characterizes each module of information

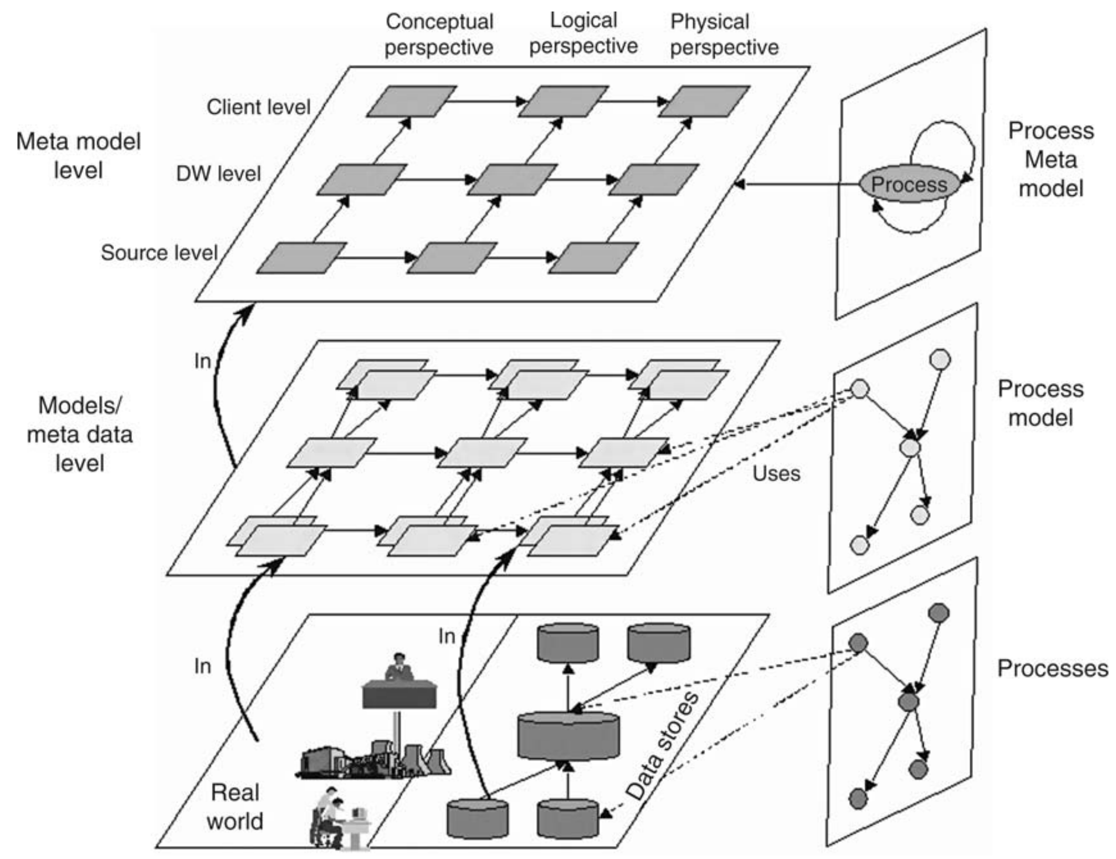
Data warehouse metadata repositories store large parts (if not all) of this kind of *data warehouse metadata* and provide a central point of reference for all the stakeholders that are involved in a data warehouse environment.

What happened was that all areas of data warehousing, ad-hoc solutions by industrial vendors and consultants were in place before the academic world provided a principled solution for the *problem of the structure and management of data warehouse metadata.* Early attempts of academic projects that related to wrapper-mediator schemes of information integration (Information Manifold, WHIPS, Squirrel, TSIMMIS – see [9] for a detailed discussion of the related literature), did not treat metadata as first-class concepts in their deliberations. At the same time, early standardization efforts from the industrial world (e.g., the MDIS standard [13]) were also poor in their treatment of the problem.

The first focused attempt towards the problem of data warehouse metadata management was made in the context of the European Project "Foundations of Data Warehouse Quality (DWQ)" [7,5]. In Fig. 1, the vertical links represent levels of abstraction: the data warehouse metadata repository, depicted in the middle layer, is an abstraction of the way the warehouse environment is structured in real life (depicted in the lowest layer of Fig. 1). At the same time, coming up with the appropriate formalism for expressing the contents of the repository (depicted in the upper layer of Fig. 1), provided an extra challenge that was tackled by [7] through the usage of the Telos language.

Foundations

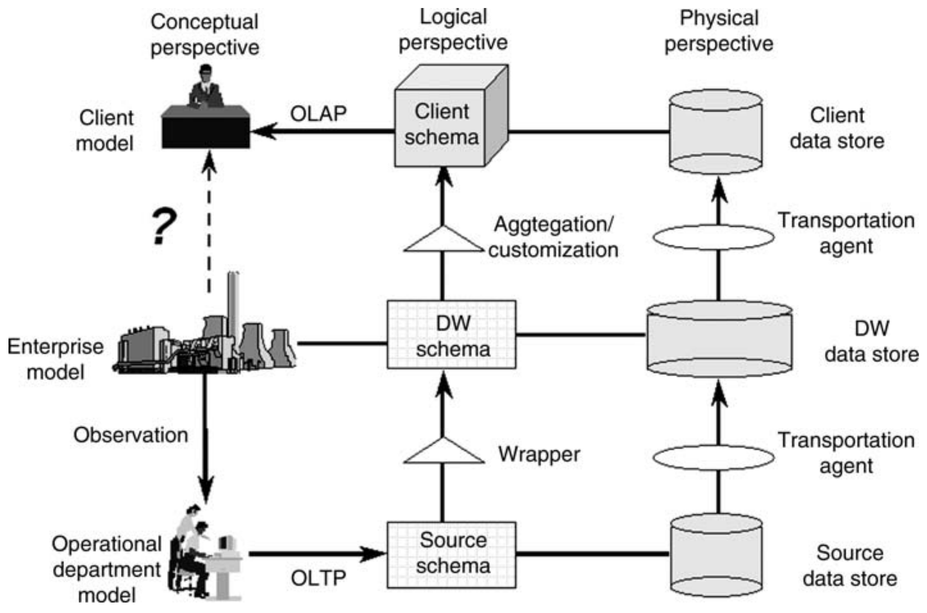
Structure of the data warehouse metadata repository. A principled approach towards organizing the structure of the data warehouse metadata repository was



Data Warehouse Metadata. Figure 1. Role and structure of a data warehouse metadata repository [12].

first offered by [7,8]. The ideas of these papers were subsequently refined in [9] and formed the basis of the DWQ methodology for the management of data warehouse metadata. The specifics of the DWQ approach are fundamentally based on the separation of data and processes and their classification in a grid which is organized in three *perspectives*, specifically the conceptual, the logical and the physical one and three *location levels,* specifically, the source, warehouse and client levels (thus the 3 × 3 contents of the middle layer of Fig. 1 and also the structure of Fig. 2). The proposal was subsequently extended to incorporate a *program versus data* classification (Fig. 1) that discriminates static architectural elements of the warehouse environment (i.e., stored data) from process models (i.e., software modules).

The *location* axis is straightforward and classifies elements as source, data warehouse and client elements. The data warehouse elements incorporate both the officially published data, contained in fact and dimension tables as well as any auxiliary data structures, concerning the Operational Data Store and the Data Staging Area. Similarly, any back-stage Extract-Transform-Clean (ETL) processes that populate the warehouse and the data marts with data are also classified according to the server in which they execute. The most interesting part of the DWQ method has to do with the management of the various *models* (a.k.a. *perspectives* in the DWQ terminology) of the system. Typically, in all DBMS's –and, thus, all deployed data warehouses- the system catalog includes both a *logical model* of the data structure (i.e., the database schema) as well as a *physical schema*, indicating the physical properties of the data (tablespaces, internal representation, indexing, statistics, etc) that are useful to the database administrator to perform his everyday maintenance and tuning tasks. The DWQ approach claimed that in a complicated and large environment like a data warehouse it is absolutely necessary to add a conceptual modeling perspective to the system that explains the role of each module of the system (be it a data or a software module). Clearly, due to the vast number of the involved information systems, each of them is accompanied by its own model, which is close enough to the perception of its users. Still, to master the complexity of all these submodels, it is possible to come up with a centralized, reference model of all the collected information (a.k.a., *enterprise model*) – exploiting, thus, the centralized nature of data warehouses. The interesting part of the method is the idea of expressing every other submodel of the warehouse as a "view" over this enterprise model. Thus, once an interested user understands the enterprise model, he/she can ultimately understand the



Data Warehouse Metadata. Figure 2. The DWQ proposal for the internal structure of the data warehouse metadata repository [4].

particularities of each submodel, independently of whether it concerns a source or client piece of data or software.

In [15], the authors discuss a coherent framework for the structuring of data warehouse metadata. The authors discriminate between back-stage *technical metadata*, concerning the structure and population of the warehouse and *semantic metadata*, concerning the front-end of the warehouse, which are used for querying purposes. Concerning the technical metadata, the proposed structure is based on (i) *entities*, comprising attributes as their structural components and (ii) an early form of schema mappings, also called *mappings* in the paper's terminology, that try to capture the semantics of the back-stage ETL process by appropriately relating the involved data stores through aggregations, joins etc. Concerning the semantic metadata, the authors treat the enterprise model as a set of *business concepts*, related to the typical OLAP metadata concerning cubes, dimensions, dimension levels and hierarchies. The overall approach is a coherent, UML-based framework for data warehouse metadata, defined at a high-level of abstraction. Specialized approaches for specific parts (like definitions of OLAP models, or ETL workflows) can easily be employed in a complementary fashion to the framework of [6] (possibly through some kind of specialization) to add more detail to the metadata representation of the warehouse. It is also noteworthy to mention that the fundamental distinction between *technical* and *business metadata* has also deeply influenced the popular, industrially related literature [11].

Contents of the data warehouse metadata repository (data warehouse metadata in detail). The variety and complexity of metadata information in a data warehouse environment are so large that giving a detailed list of all metadata classes that can be recorded is mundane. The reader who is interested in a detailed list is referred to [12] for a broader discussion of all these possibilities, and to [11] for an in depth discussion with a particular emphasis on ETL aspects (with the note that the ETL process is indeed the main provider of entries in the metadata repository concerning the technical parts of the warehouse). In the sequel, the discussion is classified in terms of data and processes.

Data. Figure 3 presents a summarized view of relevant metadata concerning the static parts of the warehouse architecture. The physical-perspective metadata are mostly related to (i) the location and naming of the information wherever data files are used and (ii) DBMS catalog metadata wherever DBMS's are used. Observe the need for efficiently supporting the end-user in his navigation through the various reports, spreadsheets and web pages (i.e., answering the question "where can I find the information I am looking for?") also observe the need to support the questions "what information is available to me anyway?" which is supported at the logical perspective for the client level. The rest of the logical perspective is also straightforward and mostly concerns the schema of data; nevertheless business rules are also part of any schema and thus data cleaning requirements and the related business rules can also be recorded at this level. The conceptual perspective involves a clear recording of the involved concepts and their intra-level mappings (source-to-DW, client-to-DW). As expected, academic efforts adopt rigorous approaches at this level [9], whereas industrial literature suggests informal, but simpler methods (e.g., see the discussion on "Business metadata" at [11]).

It is important to stress the need of tracing the mappings between the different levels and perspectives in the

Data	Source	DW	Client
Conceptual	Source model	Enterprise model	Business concepts
Logical	Schemata and/or data formats	– Schemata and/or data formats – Surrogate Key, Slowly Changing Dimension information – Data cleaning standards/specs and business rules	– Schemata of any data marts – List of available pre-canned reports and their definitions – User documentation – User profiles – Security, authentication profiles
Physical	Names of the involved data files or database installations physical properties like partitions, deployment/striping of data at disks, indexes, etc)		Map of available reports, spreadsheets, web pages

Data Warehouse Metadata. Figure 3. Metadata concerning the data of the warehouse.

warehouse. The physical-to-logical mapping is typically performed by the DBMS's and their administrative facilities; nevertheless, the logical-to-conceptual mapping is not. Two examples are appropriate in this place: (i) the developer who constructs (or worse, maintains) a module that processes a source file of facts, has to translate cryptic code-and-value pairs (e.g., CDS_X1 = 145) to data that will be stored in the warehouse and (ii) an end-user who should see data presented with names that relate to the concepts he is familiar with (e.g., see a description "Customer name" instead of the attribute name CSTR_NAME of a dimension table). In both cases, the logical-to-conceptual mappings are of extreme importance for the appropriate construction and maintenance of code and reports.

This is also the place to *stress the importance of naming conventions* in the schema of databases and the signatures of software modules: the huge numbers of involved attributes and software modules practically enforce the necessity of appropriately naming all data and software modules in order to facilitate the maintenance process (see [11] for detailed instructions).

Processes. When the discussion comes to the metadata that concern processes, things are not very complicated again, at the high level (Fig. 4). There is a set of ETL workflows that operate at the warehouse level, and populate the warehouse along with any pre-canned reports or data marts on a regular basis. The structure of the workflow, the semantics of the activities and the regular scheduling of the process form the conceptual and logical parts of the metadata. The physical locations and names of any module, along with the management of failures form the physical part of the metadata, concerning the design level of the software. Still, it is worth noting that the physical metadata can be enriched with information concerning the execution of the back-stage processes, the failures, the volumes of processed data, clean data, cleansed or impossible-to-clean data, the error codes returned by the DBMS and the time that the different parts of the process took. This kind of metadata is of statistical importance for the tuning and maintenance of the warehouse back-stage by the administration team. At the same time, the audit information is of considerable value, since the data lineage is recorded as every step (i.e., transformation or cleaning) in the path that the data follow from the sources to their final destination can be traced.

Standards. The development of standards for data warehouse metadata has been one of the holy grails in the area of data warehousing. The standardization of data warehouse metadata allows the vendors of all kinds of warehouse-related tools to extract and retrieve metadata in a standard format. At the same time, metadata interchange among different sources and platforms –and even migration from one software configuration to another – is served by being able to export metadata from one configuration and loading it to another.

The first standardization effort came from the *MetaData Coalition (MDC)*, an industrial, non-profitable consortium. The standard was named *MetaData Interchange Specification (MDIS)* [13] and its structure was elementary, comprising descriptions for databases, records, dimensions and their hierarchies and relationships among them. Some years after MDIS, the *Open Information Model (OIM)* [14] followed. OIM was also developed in the context of the MetaData Coalition and significantly extends MDIS by capturing core metadata types found in the operational

Processes	Source	DW	Client
Conceptual		Semantics of each activity of the workflow	
Logical	List of software modules related to the extraction task (and how)	– Structure of the ETL workflow – Scheduling for the execution of ETL workflows – Security settings	
Physical design	– Names & location of the involved scripts or software modules in the ETL process – Exception handling		
Physical execution	Execution statistics	– Execution statistics – Audit & data lineage logs – Time statistics	Usage statistics

Data Warehouse Metadata. Figure 4. Metadata concerning the process of the warehouse.

and data warehousing environment of enterprises. The MDC OIM uses UML both as a modeling language and as the basis for its core model. The OIM is divided into sub-models, or *packages*, which extend UML in order to address different areas of information management, including database schema elements, data transformations, OLAP schema elements and data types. Some years later, in 2001, the *Object Management Group (OMG)* initiated its own standard, named *Common Warehouse Metamodel (CWM)* [4]. CWM is built on top of other standard OMG notations (UML, MOF, XMI) also with the aim to facilitate the interchange of metadata between different tools and platforms. As of 2007, CWM appears to be very popular, both due to its OMG origin and as it is quite close to the parts concerning data warehouse structure and operation. Much like OIM, CWM is built around packages, each covering a different part of the data warehouse life-cycle. Specifically, the packages defined by CWM cover metadata concerning (i) static parts of the warehouse architecture like relational, multidimensional and XML data sources, (ii) back-stage operations like data warehouse processes and operations, as well as data transformations and (iii) front-end, user-oriented concepts like business concepts, OLAP hierarchies, data mining and information visualization tasks. A detailed comparison of earlier versions of OIM and CWM can be found in [19].

Key Applications

Data Warehouse Design. Typically, the data warehouse designers both populate the repository with data and benefit from the fact that the internal structure and architecture of the warehouse is documented in the metadata repository in a principled way. [17] implements a generic graphical modeling tool operating on top of a metadata repository management system that uses the IRDS standard. Similar results can be found in [3,18].

Data Warehouse Maintenance. The same reasons with data warehouse design explain why the data warehouse administrators can effectively use the metadata repository for tuning the operation of the warehouse. In [16], there is a first proposal for the extension of the data warehouse metadata with operators characterizing the evolution of the warehouse's structure over time. A more formal approach on the problem is given by [6].

Data Warehouse Usage. Developers constructing or maintaining applications, as well as the end-users interactively exploring the contents of the warehouse can benefit from the documentation facilities that data warehouse metadata offer (refer to [11] for an example where metadata clarify semantic discrepancies for synonyms).

Data Warehouse Quality. The research on the annotation of data warehouse metadata with annotations concerning the quality of the collected data (a.k.a. *quality indicators*) is quite large. The interested reader is referred to [10,9] for detailed discussions.

Model Management. Model management was built upon the results of having a principled structure of data warehouse metadata. The early attempts in the area [1,2] were largely based on the idea of mapping source and client schemata to the data warehouse schema and tracing their attribute inter-dependencies.

Design of large Information Systems. The mental tools developed for the management of large, intra-organizational environments like data warehouses can possibly benefit other areas –even as a starting point. The most obvious candidate concerns any kind of open agoras of information systems (e.g., digital libraries) that clearly need a common agreement in the hidden semantics of exported information, before they can interchange data or services.

Cross-references

- ► CWM
- ► Data Quality
- ► Data Warehouse Life-Cycle and Design
- ► Data Warehouses
- ► MDC
- ► Metadata
- ► Metadata Repository
- ► Model Management
- ► OIM

Recommended Reading

1. Bernstein P., Levy A., and Pottinger R. A Vision for management of complex models. SIGMOD Rec. 29(4):55–63, 2000.
2. Bernstein P.A. and Rahm E. Data warehouse scenarios for model management. In Proc. 19th Int. Conf. on Conceptual Modeling, 2000, pp. 1–15.
3. Carneiro L., and Brayner A. X-META: A methodology for data warehouse design with metadata management. In Proc. 4th Int. Workshop on Design and Management of Data Warehouses, 2002, pp. 13–22.
4. Common Warehouse Metamodel (CWM) Specification, version 1.1. OMG, March 2003.
5. Foundations of Data Warehouse Quality (DWQ) homepage. http://www.dblab.ece.ntua.gr/∼dwq/.

6. Golfarelli M., Lechtenbörger J., Rizzi S., and Vossen G. Schema versioning in data warehouses: enabling cross-version querying via schema augmentation. Data Knowl. Eng., 59(2):435–459, 2006.
7. Jarke M., Jeusfeld M.A., Quix C., and Vassiliadis P. 1998, Architecture and quality in data warehouses. In Proc. Tenth Conf. on Advanced Information Systems Engineering (CAiSE' 98), 1998. Lecture Notes in Computer Science, vol. 1413, Springer, 1998, pp. 93–113.
8. Jarke M., Jeusfeld M.A., Quix C., and Vassiliadis P. Architecture and quality in data warehouses. Inf. Syst., 24(3):229–253, 1999.
9. Jarke M., Lenzerini M., Vassiliou Y., and Vassiliadis P. (eds.). Fundamentals of Data Warehouses (2nd edn.). Springer, 2003, p. 207.
10. Jeusfeld M.A., Quix C., and Jarke M. Design and analysis of quality information for data warehouses. In Proc. 17th Int. Conf. on Conceptual Modeling, 1998, pp. 349–362.
11. Kimball R. and Caserta J. The Data Warehouse ETL Toolkit. Wiley, New York, NY, 2004.
12. Kimbal R., Reeves L., Ross M., and Thornthwaite W. The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses. Wiley, 1998.
13. Metadata Coalition: Proposal for version 1.0 metadata interchange specification, 1996.
14. MetaData Coalition. Open Information Model, version 1.0 (1999).
15. Müller R., Stöhr T., and Rahm E. An integrative and uniform model for metadata management in data warehousing environments. In Proc. Int. Workshop on Design and Management of Data Warehouses, DMDW'99, 1999.
16. Quix C. Repository support for data warehouse evolution. In Proc. Int. Workshop on Design and Management of Data Warehouses, 1999.
17. Sapia C., Blaschka M., and Höfling G. GraMMi: Using a standard repository management system to build a generic graphical modeling tool. In 33rd Annual Hawaii Int. Conf. on System Sciences (HICSS-33), Track 8: Software Technology, Maui, Hawaii, 2000.
18. Vaduva A, Kietz J-U, Zücker R. M4 - A metamodel for data preprocessing. In Proc. ACM 4th Int. Workshop on Data Warehousing and OLAP, 2001.
19. Vetterli T, Vaduva A, and Staudt M. Metadata standards for data warehousing: open information Model vs. Common warehouse metamodel. SIGMOD Rec., 29(3):68–75, 2000.

Data Warehouse Query Processing

► Query Processing in Data Warehouses

Data Warehouse Refreshment

► Extraction, Transformation and Loading

Data Warehouse Security

Carlos Blanco[1], Eduardo Fernández-Medina[1], Juan Trujillo[2], Mario Piattini[1]
[1]University of Castilla-La Mancha, Ciudad Real, Spain
[2]University of Alicante, Alicante, Spain

Synonyms

Secure data warehouses; Data warehouses confidentiality

Definition

Security, as is stated in the ISO/IEC 9126 International Standard, is one of the components of software quality. Information Security can be defined as the preservation of confidentiality, integrity and availability of information [5], in which confidentiality ensures that information is accessible only to those users with authorization privileges. Integrity safeguards the accuracy and completeness of information and process methods, and availability ensures that authorized users have access to information and associated assets when required. Other modern definitions of Information Security also consider properties such as authenticity, accountability, non-repudiation, and reliability. Therefore, Data Warehouse (DW) Security is defined as the mechanisms which ensure the confidentiality, integrity and availability of the data warehouse and its components. Confidentiality is especially important once the Data Warehouse has been deployed, since the most frequent operations that users perform are SQL and OLAP queries, and therefore the most frequent security attack is against the confidentiality of data stored in the data warehouse.

Historical Background

Considering that DWs are the basis of companies' decision making processes, and due to the fact that they frequently contain crucial and sensitive internal information, and that DWs are usually managed by OLAP tools, most of the initial approaches to data warehouse security were focused on the definition and enforcement of access control policies for OLAP tools [6,10], taking into consideration one of the most traditional access control models (Discretional Access Control) and also managing the concept of role defined as subject. Other approaches dealt with real implementation in specific commercial tools by using multidimensional elements [10]. Indirect access and cover channel problems have also been detected in Statistical Databases but an entirely satisfactory solution has not yet been found.

Moreover, data stores in DWs come from heterogeneous data sources, which must be integrated, thus provoking various security problems. However, few works dealing with the security defined in data sources (which tackle the problem of merging different security measures established in each source) appear to exist. This problem has, nevertheless, been addressed in the field of Federated Databases, and some authors have used this parallelism to propose an architecture for developing Data Warehouses through the integration of Multilevel Access Control (MAC) policies defined in data sources [12]. Furthermore, ETL processes have to load the information extracted and transformed from the data sources into the Data Warehouse, but these processes do not consider security issues and must use the security defined in the data source and add new security measures for the detected lacks of security. In this field, the proposed works focus solely upon ETL model processes, and do not consider security issues.

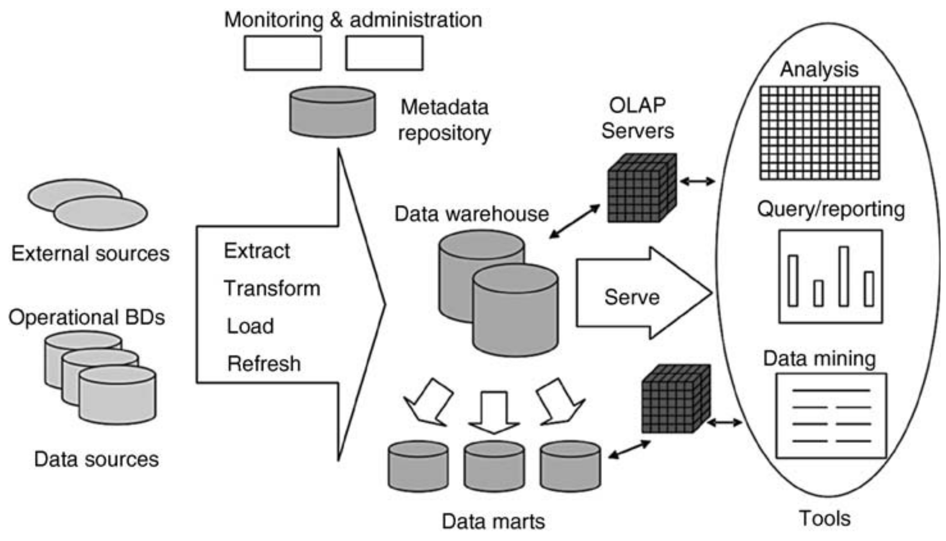
In recent decades, the development of DWs has evolved from being a handmade method, to being a more engineering-based method, and several approaches have been defined for the conceptual modeling of DWs, e.g., [4,8]. Unfortunately none of these proposals has considered security issues. However, one of these approaches has recently been extended to propose a Model Driven Multidimensional approach for developing secure DWs [1]. This approach permits the inclusion of security requirements (audit and access control) from the first stages of the DWs life cycle, and it is possible to automatically generate code for different target platforms through the use of model transformation. The scientific community demands the integration of security engineering and software engineering in order to ensure the quality and robustness of the final applications [9], and this approach fulfills this demand.

Foundations

The DW development process follows the scheme presented in Fig. 1. Therefore, security should be considered in all stages of this process by integrating the existing security measures defined in data sources, considering these measures in ETL processes, defining models that represent security constraints at a high level of abstraction and finally, enforcing these security constraints in the OLAP tools in which the DW is deployed.

Security in Data Sources

In DWs architecture, data coming from heterogeneous data sources are extracted, cleaned, debugged, and stored. Once this process is completed, the DW will be composed of these stored and integrated data, with which users will be able to discover information in strategic decision making processes. Data sources are heterogeneous, can use different representation models (relational databases, object-orientated databases, XML files, etc.), and may or may not have associated



Data Warehouse Security. Figure 1. Data warehouse architecture.

security policies. Although DW users will be different from data sources, these security policies should be considered and integrated into the DW security design.

Data source security can be defined by using various security policies such as Discretional Access Control (DAC) which restricts access to objects based on the identity of subjects with a certain access permission: Mandatory Access Control (MAC), which restricts access to objects based on the sensitivity of the information contained in the objects and the formal authorization of subjects to access information of such sensitivity; or Role-Based Access Control (RBAC), an approach which restricts system access to authorized users by assigning permissions to perform certain operations to specific roles. The integration of these policies presents a problem which has been studied in Federated Databases [12]. Some research efforts have been made to integrate different multilevel policies in a semi-automatic manner by using a schema integration process which obtains the ordered set and the translation functions between each ordered set belonging to each component database and the federated ordered set. In addition, the integration of different role-based policies has been dealt with by representing role configurations as role graphs and by applying techniques of graph integration to obtain the final role configuration. Other authors, such as Rosenthal and Sciore [11], have applied inference mechanisms to data sources in order to obtain access control policies and have used them to set up DW security.

After considering the parallelism between DW and Federated Information Systems (FIS), Saltor et al. [12] propose a seven layered architecture for preserving and integrating the multilevel security established in data sources. This architecture extends the five layered architecture developed for FIS, including two schemas: “authorization schemas” for each authorization level and “external schemas” with which to represent multilevel security information of the data sources in a Canonical Data Model (CDM). These “external schemas” with security information will later be used to obtain DW and Data Marts (DM) schemas.

Security in ETL Processes

ETL (Extraction-Transformation-Loading) processes participate in the first stage of acquisition and extract information from heterogeneous data sources, debug it, integrate it and finally, load it into data warehouses following their previously defined design.

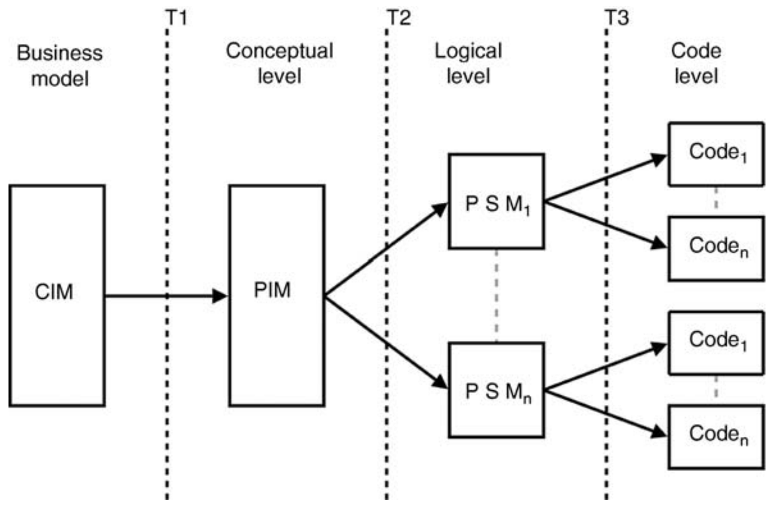
It is necessary to define security measures in ETL processes, in order to both use, adapt and integrate the security measures defined in the data sources and to add new security measures for the possibly detected lacks of security. At present, despite the existence of proposals with which to model ETL processes which can be extended to include security issues, none of the said proposals include the aforementioned concepts.

Some interesting works on the modeling of ETL processes exist, but they do not deal with security issues. Vassiliadis and Simitsis use their own graphic notation for modeling ETL processes at a conceptual level, propose how to transform these conceptual designs into logical designs [13], and define a framework for designing and maintaining ETL processes (ARKTOS). Trujillo and Luján-Mora [15] model ETL processes by using the UML notation and OCL to establish constraints. Their proposal does not take attributes into consideration but simplifies the design and maintenance processes, and the use of UML and OCL provides one with possibilities which greatly simplify the extension of this model with security.

Security in Data Warehouses Modeling

Multidimensional modeling is the foundation of DWs, Multidimensional Databases and On-Line Analytical Processing Applications (OLAP) and is different from traditional database modeling in that it is adapted to the characteristics of these approaches. Despite the quantity of interesting background on security measures and access control models specified for relational databases which is available, it cannot be directly applied as it is not appropriate for DWs. Both are models but they are based on different concepts. Relational security measures use terms of database tables, rows and columns, and DW security uses multidimensional terms of facts, dimensions or classification hierarchy. Several modeling proposals specifically created for DWs consider their properties, but none use standard notations or include security issues, e.g., [4,8].

A model driven multidimensional modeling approach for developing secure DWs has been proposed by Fernández-Medina et al. [1]. This approach proposes a Query/Views/Transformations (QVT) and Model-Driven Architecture (MDA) based approach (see Fig. 2). This aligns MDA with the DWs development process, considering multidimensional models as being PIM, logical models (such as ROLAP, MOLAP and HOLAP) as being Platform-Specific Model (PSM),



Data Warehouse Security. Figure 2. Model driven architecture.

and the DBMS and OLAP tools as being the target platforms. This proposal is made up of a security model (access control and audit model) for DW [2], an extension of UML for modeling secure multidimensional models [3] as Platform-Independent Models (PIM), and an extension of the Common Warehouse Metamodel (CWM) [14] as a Platform-Specific Model (PSM). This proposal is currently being extended within the extremes of MDA architecture: the Computational-Independent Model (CIM) level is being defined through an extension of i* which defines security goals and subgoals, and the code generation is being carried out by considering Oracle, SQL Server Analysis Services, and Pentaho as target platforms of the architecture.

Security in OLAP Tools

OLAP systems are mechanisms with which to discover business information and use a multidimensional analysis of data to make strategic decisions. This information is organized according to the business parameters, and users can discover unauthorized data by applying a set of OLAP operations to the multidimensional view. Therefore, it is of vital importance for the organization to protect its data from unauthorized accesses including security constraints in OLAP systems which take these OLAP operations (roll-up, drill-down, slice-dice and pivoting) into account, and from indirect accesses (inferences) which use parallel navigation, tracker queries, etc. The inference problem is an important security problem in OLAP which has yet to be solved and which can be studied by using the existing parallelism with Statistical Databases. Several solutions to the inference problem have been applied. Various solutions to the problem of controlling inference exist, such as the perturbation of data or the limitation of queries, but these imply a large amount of computational effort. On the other hand the establishment of security constraints at cell level allows one to control inferences without this lack of efficiency.

Several works attempting to include security issues in OLAP tools by implementing the previously defined security rules at a conceptual level have been proposed, but these works focus solely upon Discretional Access Control (DAC) and use a simplified role concept implemented as a subject. For instance, Katic et al. [6] proposed a DWs security model based on metamodels which provides one with views for each user group and uses Discretional Access Control (DAC) with classification and access rules for security objects and subjects. However, this model does not allow one to define complex confidentiality constraints. Kirkgöze et al. [7] defined a role-based security concept for OLAP by using a "constraints list" for each role, and this concept is implemented through the use of a discretional system in which roles are defined as subjects.

Priebe and Pernul later proposed a security design methodology, analyzed security requirements, classifying them into basic and advanced, and dealt with their implementation in commercial tools. First, in [10] they used adapted UML to define a Discretional Access Control (DAC) system with roles defined as subjects at a conceptual level. They then went on to implement this in Microsoft Analysis Services (SQL Server 2000) by using Multidimensional Expressions (MDX). They created a Multidimensional Security Constraint Language (MDSCL) based on MDX and put forward HIDE statements with which to represent negative authorization constraints on certain multidimensional elements: cube, measure, slice, and level.

Key Applications

DWs security is a highly important quality aspect of a DW, which must be taken into account at all stages of the development process. If security measures are not established, then unauthorized users may obtain the business information used for making strategic decisions which is vital to the survival of the organization. DWs security has to be considered in all the fields involved. These are, principally, the following: the application of techniques through which to integrate different kinds of security policies detected in the data sources; the definition of models, which permit the establishment of security constraints at upper abstraction levels; and the study of the final implementation of the defined security measures in OLAP tools in order to protect information from malicious operations such as navigations or inferences.

Cross-references

► Data Warehousing Systems: Foundations and Architectures
► Extraction
► Multidimensional Modeling
► On-Line Analytical Processing
► Transformation and Loading

Recommended Reading

1. Fernández-Medina E., Trujillo J., and Piattini M. Model driven multidimensional modeling of secure data warehouses. Eur. J. Inf. Syst., 16:374–389, 2007.
2. Fernandez-Medina E., Trujillo J., Villarroel R., and Piattini M. Access control and audit model for the multidimensional modeling of data warehouses. Decis. Support Syst., 42(3):1270–1289, 2006.
3. Fernandez-Medina E., Trujillo J., Villarroel R., and Piattini M. Developing secure data warehouses with a UML extension. Inf. Syst., 32(6):826–856, 2007.
4. Golfarelli M., Maio D., and Stefano R. The dimensional fact model: a conceptual model for data warehouses. Int. J. Coop. Inf. Syst. (IJCIS), 7(2–3):215–247, 1998.
5. ISO27001, ISO/IEC 27001 Information technology – Security techniques – Information security management systems – Requirements, 2005.
6. Katic N., Quirchmayr G., Schiefer J., Stolba M., and Tjoa A. 1A prototype model for DW security based on metadata. In Proc. Ninth Int. Workshop on Database and Expert Systems Applications (DEXA), 1998, p. 300.
7. Kirkgöze R., Katic N., Stolda M., and Tjoa A. A security concept for OLAP. In Proc. Eighth Int. Workshop on Database and Expert System Applications, 1997, p. 0619.
8. Lujan-Mora S., Trujillo J., and Song I.-Y. A UML profile for multidimensional modeling in data warehouses. Data Knowl. Eng., 59(3):725–769, 2006.
9. Mouratidis H. and Giorgini P. Integrating Security and Software Engineering: Advances and Future Visions. Idea Group, Hershey, PA, 2006.
10. Priebe T. and Pernul G. A pragmatic approach to conceptual modeling of OLAP security. In Proc. 20th Int. Conf. on Conceptual Modeling, 2001, pp. 311–324.
11. Rosenthal A. and Sciore E. View security as the basis for data warehouse security. In Proc. Second Int. Workshop on Design and Management of Data Warehouses (DMDW 2000), 2000, p. 8.
12. Saltor F., Oliva M., Abelló A., and Samos J. Building secure data warehouse schemas from federated information systems. In Heterogeneous Information Exchange and Organizational Hubs, D.T. Bestougeff (ed.). Kluwer Academic, 2002.
13. Simitsis A. and Vassiliadis P. A method for the mapping of conceptual designs to logical blueprints for ETL processes. Decis. Support Syst., 45(1):22–40, 2007.
14. Soler E, Trujillo J., Fernández-Medina E., and Piattini M. SECRDW: an extension of the relational package from CWM for representing secure data warehouses at the logical level. In Proc. Fifth Int. Workshop on Security in Information Systems. 2007, pp. 245–256.
15. Trujillo J. and Luján-Mora S. A UML based approach for modeling ETL processes in data warehouses. In Proc. 22nd Int. Conf. on Conceptual Modeling, 2003, pp. 307–320.

Data Warehousing for Clinical Research

Shawn Murphy
Massachusetts General Hospital, Boston, MA, USA

Synonyms

Clinical research chart

Definition

The clinical data warehouse allows rapid querying and reporting across patients. It is used to support the discovery of new relationships between the cause and effects of diseases, and to find specific patients that qualify for research studies.

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:23

Historical Background

In healthcare, the term "data warehouse" is generally reserved for those databases optimized for analysis and integrated queries across patient populations. This is as opposed to the transactional database, which is optimized for rapid updating and highly specific kinds of retrieval (like those based upon a specific patient identifier).

There appear to be three fundamentally different approaches to organizing the healthcare data warehouse. The first is to extract tables from the transaction systems of the healthcare organization and load them into the database platform of the data warehouse with minimal transformation of the data model. The codes present in the columns are usually transformed to make them compatible with codes from other systems. For example, an ICD9 diagnosis code stored as "27.60" in one system may be transformed to a common format of 02760. However, the tables are left in essentially the same schema as the transaction system [2].

The second approach is more ambitious, where not just the codes from different systems are transformed to look the same, but the data is transformed to look the same as well. The diverse data coming from different systems must be made to fit into new tables. This involves a considerable amount of data transformation, but queries against the warehouse are then much less complex [1]. This is the approach that will be described.

The third approach is to keep the data located at its source in a "distributed" data warehouse. Queries are distributed to the local databases across a network. This strategy can be successful when patients have all of their data contained within one of the local systems (such as when systems exist in distant cities). However, if a single patient's data is distributed across many of these local databases, detailed data would need to travel across the network to be accumulated in the internal processing structures of a central CPU to allow the execution of query plans. This will have a severe negative impact on the performance of these types of systems.

Foundations

Database Design for Clinical Research Data Warehouse

The clinical data warehouse allows rapid querying and reporting across patients, which unexpectedly is not available in most clinical transaction systems. Rather, transaction systems are optimized for lookups, inserts, updates, and deletes to a single patient in the database. Transactions usually occur in small packets during the day, such as when a patient's lab test is sent to the database. Transaction systems are usually updated by small bits of data at a time, but these bits come in at the rate of thousands per second. Therefore the typical clinical database used for patient care must be optimized to handle these transactions [2].

Because the clinical data warehouse does not need to handle high volumes of transactions all day long, the data warehouse can be optimized for rapid, cross patient searching. For optimal searching of a database it is best to have very large tables. These can be indexed such that a single index allows a global search. So when one designs a clinical data warehouse, one adopts a few tables that can hold nearly all the available data. The way to hold many forms of healthcare data in the same table is by the classical entity-attribute-value schema (or EAV for short) [4,5].

The EAV schema forces one to define the fundamental fact of healthcare [2]. The fundamental fact of healthcare will be the most detailed rendition possible of any healthcare observation as reported from the data warehouse. This can be defined as an observation on a patient, made at a specific time, by a specific observer, during a specific event. The fact may be accompanied by any number of values or modifiers. Each observation is tagged with a specific concept code, and each observation is entered as a row in a "fact table." This fact table can grow to billions of rows, each representing an observation on a patient. The fact table is complimented by a least an event table, a patient table, a concept table, and an observer table [4].

The Patient table is straightforward. Each row in the table represents a patient in the database. The table includes common fields such as gender, age, race, etc. Most attributes of the patient dimension table are discrete (i.e., Male/Female, Zip code, etc.) or relevant dates.

The Event table represents a "session" where observations were made. This "session" can involve a patient directly such as a visit to a doctor's office, or it can involve the patient indirectly such as running several tests on a tube of the patient's blood. Several observations can be made during a visit. Visits have a start and end date-time. The visit record also contains specifics about the location of the session, like which hospital or clinic the session occurred, and whether the patient was an inpatient or outpatient at the time of the visit.

The Observer table is a list of observers. Generally, each row in the observer dimension represents a provider at an institution, but more abstractly, it may be an observing machine, such as an Intensive Care Unit continuous blood pressure monitor.

The Concept table is the key to understanding how to search the fact table. A concept specifies exactly what observation was made on the patient and is being represented in a particular row of the fact table. A code is used to represent the concept in the fact table, and the concept table links if to a human-readable description of the code (Fig. 1).

Metadata Management in Clinical Research Data Warehouse

When looking at rows in the concept table, one is introduced to Metadata. Metadata is everywhere in a data warehouse. It represents data about the data, and is where medical knowledge is represented in the clinical data warehouse. The primary form of representation is in the groupings of terms so they can be queried as groups of similar concepts. The terms are grouped into hierarchies, each level up usually expressing a more general medical concept.

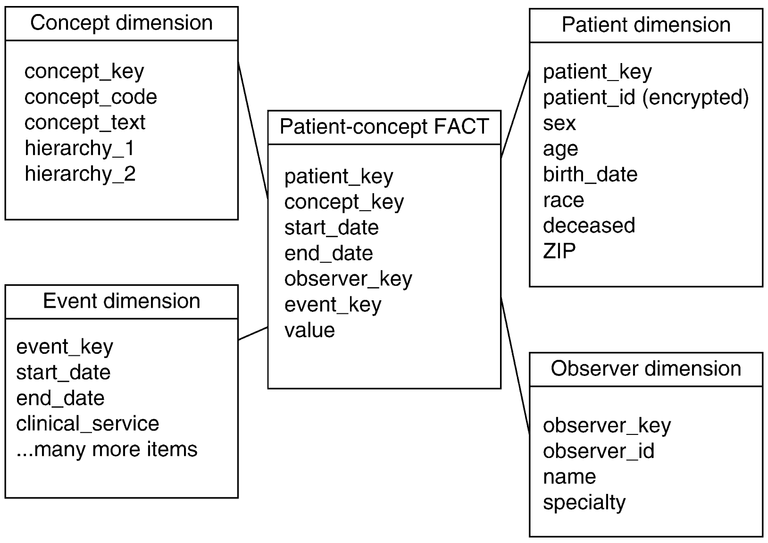
Many diverse concepts about a patient can exist in the fact table. In a clinical data warehouse, typically 100–500 thousand different concepts exist. All sorts of concepts including ICD-9 codes (International Classification of Diseases 9th Edition, most common codes used in hospitals to classify diagnoses), CPT codes (Current Procedural Terminology, most common codes used in hospitals to classify procedures), NDC codes (National Drug Codes, most common codes used in hospitals to classify medication), and LOINC codes (Logical Observation Identifiers Names and Codes, most common codes used in hospitals to classify laboratory tests) as well as numerous local coding systems are used to describe the patient. The challenge is maintaining and updating the classification of the concepts. This classification needs to seamlessly absorb new codes, and be back-compatible to old coding and classification systems.

The organization of concepts hierarchically allows the user to navigate and use the concepts in a query. Like a file path in the Windows Explorer, the path of the hierarchy indicates in which groups the concept belongs, with the most general group being listed on the far left and each group to the right of that growing more and more specific.

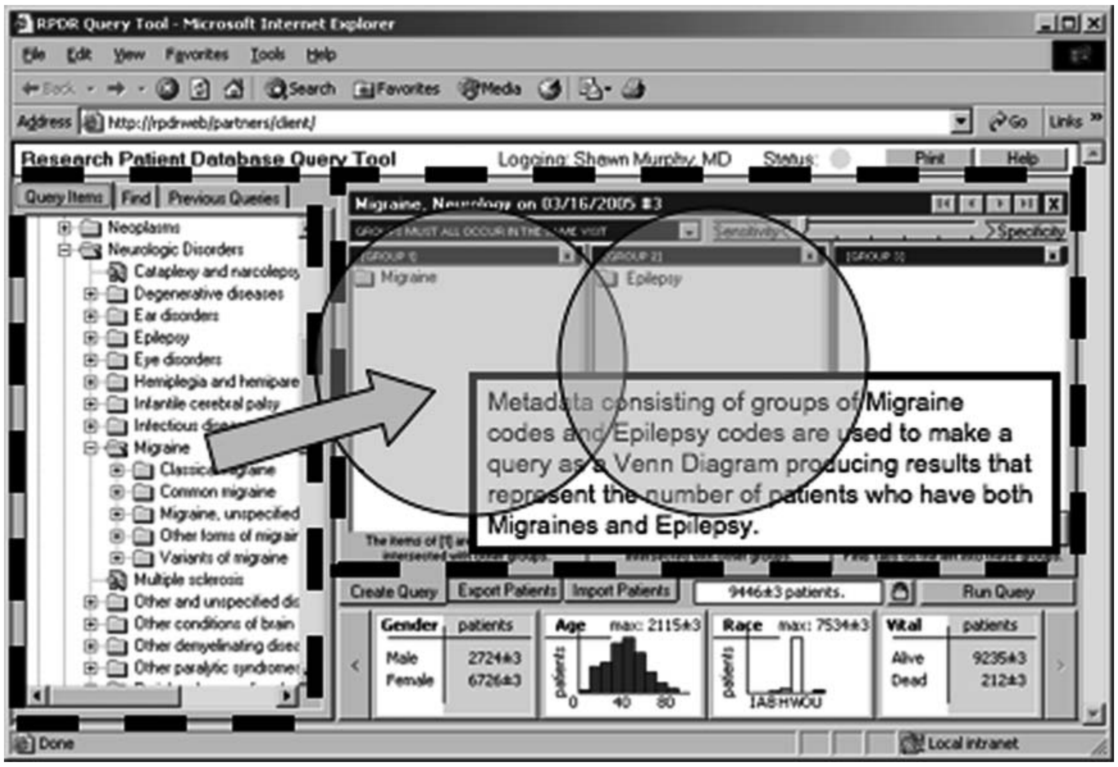
An interface to present this concept representation is shown below (Fig. 2). The use of this interface has been described in detail [3], but is essentially a way of building queries using concepts represented in the concept and provider dimension tables.

Privacy Management in the Clinical Research Data Warehouse

The clinical data warehouse should be built with patient privacy in mind. The most common strategy is to separate the data warehouse into two databases. The clinical



Data Warehousing for Clinical Research. Figure 1. Optimal star schema database design for healthcare data warehouse.



Metadata consisting of groups of Migraine codes and Epilepsy codes are used to make a query as a Venn Diagram producing results that represent the number of patients who have both Migraines and Epilepsy.

Data Warehousing for Clinical Research. Figure 2. Construction of query using the metadata from a healthcare data warehouse.

data goes into one database, and the identifiers of the patients go into a second database. Access to the second, identified, database is strictly controlled, and only accessed during data loading and the building of the data marts. The patients are given codes in the clinical database and these codes can only be looked up in the identified database. In this way, customers can use the clinical database and not have access to the patient identifiers.

Data Flow in Clinical Research Data Warehouse

Data generally flows into the data warehouse by loading it from the transaction systems, or by receiving a duplicate feed of data that are going into the transaction systems. Data are usually loaded from the transaction systems once it is provided as large "data dumps," or downloads. Transaction systems may contain many millions of records, but with current technology they can usually be written out in their entirety in just hours. Reloading all this data into the data warehouse similarly takes only a few hours, and the simplicity of this model, as opposed to the complexity of update models, often makes this a much more desirable process. The risk of an update process is that errors in update flags will cause the data warehouse to become desynchronized with the transaction system. To note many transaction systems do not have a way to provide updates and a full "data dump" is all that is possible from the transaction system.

When the data is loaded from the transaction systems, it is usually first loaded to a "staging area." As previously discussed, the data structure usually differs considerably between the transaction system and the data warehouse. Loading the transaction data into a staging area allows the data to be studied and quality assured before introducing the complexity of transforming the data into the format of the data warehouse. Because the teams from the transaction systems are usually very familiar with the data in this form, it is desirable to have the transaction team responsible for their corresponding staging area, and allow them to transfer and load the data into this area.

The data warehouse will usually distribute data back to the data consumers as a "data mart." These are subsets of the data from the data warehouse. The advantage of this approach is that the data can be prepared per request in a consumer-friendly format.

Attempting to allow customers to query the clinical data warehouse using Structured Query Language (SQL) is rarely successful. The EAV scheme is notoriously unfriendly to the causal user of data [5]. Furthermore, the metadata exists in tables that are not obviously connected to the patient data, so that tables in the data warehouse often contain no humanly readable content. Finally, the data in the data warehouse is often updated once every day and so analysis would need to go against constantly shifting data. The result is that the data is often exported into a user friendly data mart. This also limits the set of patients that a customer can view, which is important from the patient privacy point-of-view.

Key Applications

This clinical research data warehouse allows researchers to quickly obtain information that can be critical for winning corporate and government sponsored research grants, and easily gather data on patients identified for research studies. It allows clinical data to be available for research analysis where security and confidentiality are an integral part of the design, bringing clinical information to researchers' fingertips while controlling and auditing the distribution of patient data within the guidelines of the Institutional Review Boards. It also serves as a "building-block" that enables high-throughput use of patient data in some of the following applications:

1. *Bayesian inference engines.* Bayesian inference can be used to synthesize many diverse observations into fundamental atomic concepts regarding a patient. For example, a code may be assigned to a patient from several sources indicating that a patient has a disease such as diabetes. Some sources may indicate the patient has type I diabetes, while others indicate the patient has type II diabetes. Since these two types of diabetes are mutually exclusive, it is clear that one of the sources is in error. A determination of the true diagnosis can be estimated by assigning a prior probability to each source as to how often it contains correct information, and use these probabilities, to calculate the likelihood of each diagnosis.

2. *Clinical trials performed "in-silico."* Performing an observational phase IV clinical trial is an expensive and complex process that can be potentially modeled in a retrospective database using groups of patients available in the large amounts of highly organized medical data. This application would allow a formalized way of discovering new knowledge from medical databases in a manner that is well accepted by the medical community. For example, a prospective trial examining the potential harm of Vioxx would entail recruiting large numbers of patients and several years of observation. However, an in-silico clinical trial would entail setting up the database to enroll patients into a patient set automatically when they are given a prescription for Vioxx and watching them for adverse events as these events are entered in the course of clinical care. Besides requiring fewer resources, these trials could be set up for thousands of medications at a time and thereby provide a much greater scope of observational trials.

3. *Finding correlations within data.* When multiple variables are measured for each patient in a data set, there exists an underlying relationship between all pairs of variables, some highly correlated and some not. Correlations between pairs of variables may be discovered with this application, leading to new knowledge, or further insight into known relationships. Unsupervised techniques using Relevance Networks and Mutual Information algorithms can generate hypothesis from secondary observed correlations in the data. This is a way to exploit existing electronic databases for unsupervised medical knowledge discovery without a prior model for the information content. Observations collected within labs, physical examinations, medical histories, and gene expressions can be expressed as continuous variables describing human physiology at a point in time. For example, the expression of RNA found within a tumor cell may be found to correlate with the dose of effective chemotherapy for that tumor. This would allow future tumors to have their RNA expression determined and matched to various chemotherapies, and the chemotherapy found to correlate most with that gene expression would be chosen as the agent for that individual.

Cross-references

► Bioinformatics and Health Informatics Databases
► Data Integration Systems
► Data Mining
► Data Models

Recommended Reading

1. Inmon W.H. Building the Data Warehouse, 2nd edn. Wiley, NY, 1996.
2. Kimball R. The Data Warehousing Toolkit. Wiley, NY, 1997.
3. Murphy S.N., Gainer V.S., and Chueh H. A visual interface designed for novice users to find research patient cohorts in

a large biomedical database. In: Proc. AMIA Annu. Fall Symp., 489–493, 2003.
4. Murphy S.N., Morgan M.M., Barnett G.O., and Chueh H.C. Optimizing healthcare research data warehouse design through past COSTAR query analysis. In: Proc. AMIA Fall Symp., 892–896, 1999.
5. Nadkarni P.M. and Brandt C. Data extraction and ad hoc query of an entity-attribute-value database. J. Am. Med. Inform. Assoc., 5:511–517, 1998.

Data Warehousing Systems: Foundations and Architectures

Il-Yeol Song
Drexel University, Philadelphia, PA, USA

Definition

A data warehouse (DW) is an integrated repository of data for supporting decision-making applications of an enterprise. The most widely cited definition of a DW is from Inmon [3] who states that "a data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions."

Historical Background

DW systems have evolved from the needs of decision-making based on integrated data, rather than an individual data source. DW systems address the two primary needs of enterprises: data integration and decision support environments. During the 1980s, relational database technologies became popular. Many organizations built their mission-critical database systems using the relational database technologies. This trend proliferated many independent relational database systems in an enterprise. For example, different business lines in an enterprise built separate database systems at different geographical locations. These database systems improved the operational aspects of each business line significantly. Organizations, however, faced the needs of integrating the data which were distributed over different database systems and even the legacy database systems in order to create a central knowledge management repository. In addition, during the 1990s, organizations faced increasingly complex challenges in global environments. Organizations realized the need for decision support systems that can analyze historical data trends, generate sophisticated but easy-to-read reports, and react to changing business conditions in a rapid fashion. These needs resulted in the development of a new breed of database systems that can process complex decision-making queries against integrated, historical, atomic data. These new database systems are now commonly called data warehousing systems because they store a huge amount of data – much more than operational database systems – and they are kept for long periods of time. A data warehousing system these days provides an architectural framework for the flow of data from operational systems to decision-support environments. With the rapid advancement in recent computing technologies, organizations build data warehousing systems to improve business effectiveness and efficiency. In a modern business environment, a data warehousing system has emerged as a central component of an overall business intelligence solution in an enterprise.

Foundations

OLTP vs. Data Warehousing Systems

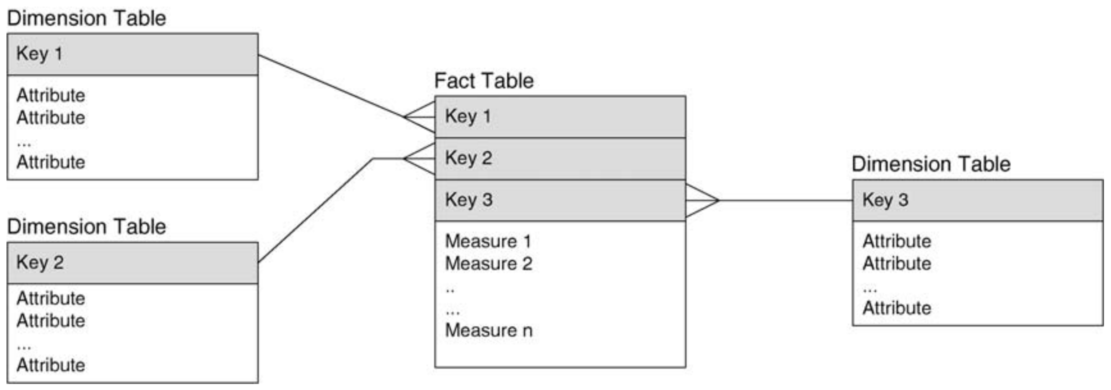
Data warehousing systems contain many years of integrated historical data, ending up storing a huge amount of data. Directly storing the voluminous data in an operational database system and processing many complex decision queries would degrade the performance of daily transaction processing. Thus, DW systems are maintained separately from operational databases, known as online transaction processing (OLTP) systems. OLTP systems support daily business operations with updatable data. In contrast, data warehousing systems provide users with an environment for the decision-making process with read-only data. Therefore, DW systems need a query-centric view of data structures, access methods, implementation methods, and analysis methods. Table 1 highlights the major differences between OLTP systems and data warehousing systems.

Rolap and Molap

The data in a DW are usually organized in formats made for easy access and analysis in decision-making. The most widely used data model for DWs is called the dimensional model or the star schema [6]. A dimensional model consists of two types of entities–a fact table and many dimensions. A *fact* table stores transactional or factual data called *measures* that get analyzed. Examples of fact tables are *Order*, *Sale*, *Return*, and *Claim*. A dimension represents an axis that analyzes the fact data. Examples of

Data Warehousing Systems: Foundations and Architectures. Table 1. A comparison between OLTP and data warehousing systems

	OLTP	Data warehouse & OLAP
Purpose	Daily business support	Decision support
	Transaction processing	Analytic processing
User	Data entry clerk, administrator, developer	Decision maker, executives
DB design	Application oriented	Subject-oriented
DB design model	ER model	Star, snowflake, Multidimensional model
Data structures	Normalized, Complex	Denormalized
		Simple
Data redundancy	Low	High
Data contents	Current, up-to-date operational data	Historical
	Atomic	Atomic and summarized
Data integration	Isolated or limited integration	Integrated
Usage	Repetitive, Routine	Ad-hoc
Queries	Predictable, predefined	Unpredictable, Complex, long queries
	Simple joins	
	Optimized for small transactions	Optimized for complex queries
Update	Transactions constantly generate new data	Data is relatively static
		Often refreshed weekly, daily
Access type	Read/update/delete/insert	Read/append mostly
Number of Records per access	Few	Many
Concurrency level	High	Low
Data retention	Usually less than a year	3–10 years or more
Response time	Subsecond to second	Seconds, minutes, worse
Systems Requirements	Transaction throughput, Data consistency	Query throughput, Data accuracy



Data Warehousing Systems: Foundations and Architectures. Figure 1. The typical structure of the star schema.

dimensions are *Time, Customer, Product, Promotion, Store*, and *Market*. Since a DW contains time-variant data, the Time dimension is always included in dimensional schemas and the data in a fact table are organized by a unit of time. An extensive list of dimensions commonly found in DWs including those dimensions used in [1,6] are presented in [4]. A typical structure of the dimensional model is illustrated in Fig. 1.

Syntactically, all the dimensions are connected with the fact table by one-to-many relationships. Thus, when a dimension has a many-to-many relationship with the fact table, a special technique such as an intersection table should be used. All the dimensions have a surrogate key, which establishes an identifying relationship with the fact table. In a star schema, all the dimensions are usually denormalized to simplify the query structure in order to minimize the number of joins. When dimensions are normalized into the third normal form, the schema is called a snowflake schema [6].

A dimensional model simplifies end-user query processing by simplifying the database structure with a few well-defined join paths. Conceptually, a dimensional model characterizes a business process with the fact table, the dimensions, and the measures involved in the business process. The dimensional model allows users of a DW to analyze the fact data from any combination of dimensions. The structure provides a multidimensional analysis space within a relational database.

Interactive data analysis of the data in a DW environment is called online analytic processing (OLAP). When the data in a dimensional model is stored in a relational database, the analysis is called relational online analytic processing (ROLAP). ROLAP engines extend SQL to support dimensional model schema and advanced OLAP functions.

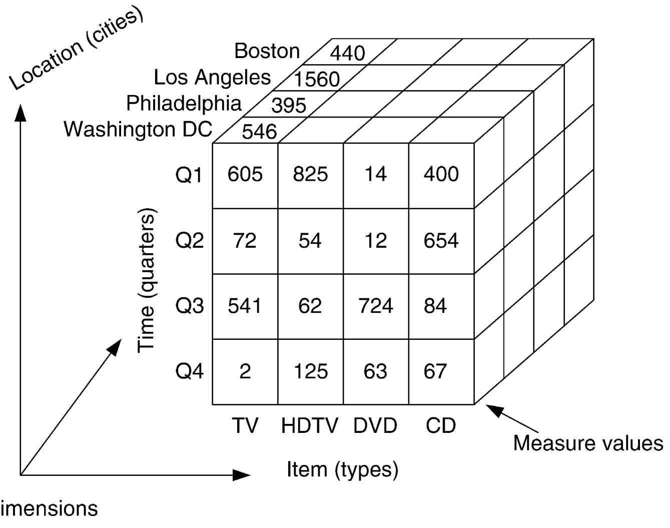
DW data can also be stored in a specialized multidimensional structure called a data cube or a hypercube. Data analysis of the data stored in a data cube is called multidimensional OLAP (MOLAP). Compared with ROLAP engines, MOLAP engines are usually limited in data storage, but provide more efficient OLAP processing by taking advantage of the multidimensional data cube structure. A typical structure of a data cube is illustrated in Fig. 2.

Hybrid OLAP (HOLAP) servers take advantage of both ROLAP and MOLAP technologies. They usually store large volumes of detailed data in a ROLAP server and store aggregated data in a MOLAP server.

Data Warehousing Architecture

A data warehousing system is an environment that integrates diverse technologies into its infrastructure. As business data and analysis requirements change, data warehousing systems need to go through an evolution process. Thus, DW design and development must take growth and constant change into account to maintain a reliable and consistent architecture. A DW architecture defines an infrastructure by which components of DW environments are organized. Figure 3 depicts the various components of a typical DW architecture that consists of five layers – data source systems, ETL management services, DW storage and metadata repository, data marts and OLAP engines, and front-end tools.

Data Source Systems The data source system layer represents data sources that feed the data into the DW. An enterprise usually maintains many different databases or information systems to serve different OLTP



Data Warehousing Systems: Foundations and Architectures. Figure 2. A three dimensional data cube having dimensions Time, Item, and Location for MOLAP.

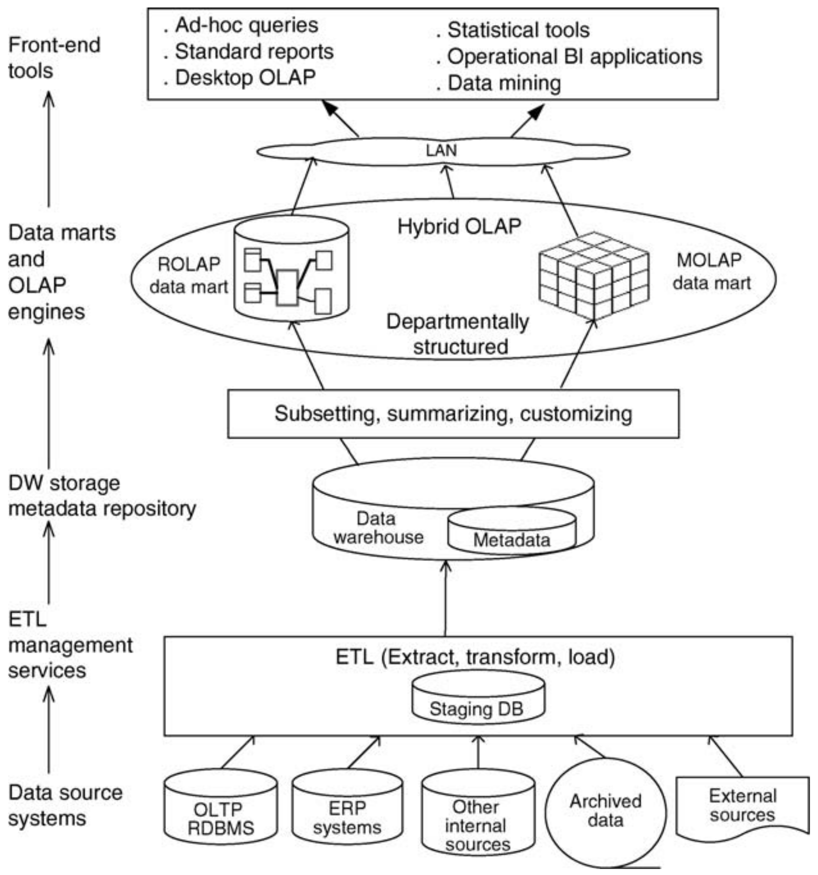
Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:25

functions. Since a DW integrates all the important data for the analysis requirements of an enterprise, it needs to integrate data from all disparate sources. Data could include structured data, event data, semi-structured data, and unstructured data. The primary source for data is usually operational OLTP databases. A DW may also integrate data from other internal sources such as legacy databases, spreadsheets, archived storages, flat files, and XML files. Frequently, a DW system may also include any relevant data from external sources. Examples of such data are demographic data purchased from an information vendor to support sales and marketing analysis and standard reference data from the industry or the government. In order to analyze trends of data from a historical perspective, some archived data could also be selected. Thus, data warehousing systems usually end up with huge amounts of historical data.

These data are regularly fed into the second layer for processing. The interval between each feed could be monthly, weekly, daily, or even real-time, depending on the frequency of changes in the data and the importance of up-to-datedness of the data in the DW.

ETL Management Services The second layer extracts the data from disparate data sources, transforms the data into a suitable format, and finally loads them to a DW. This process is known as ETL processing.

A DW does not need all the data from the data source systems. Instead, only those data that are necessary for data analysis for tactical and strategic decision-making processes are extracted. Since these data come from many different sources, they could come in heterogeneous formats. Because a DW contains integrated data, data need to be kept in a single standard format by removing syntactic and semantic variations from different data source systems. Thus, these data are standardized for the data model used in the DW in terms of data type, format, size, unit of data, encoding of values, and semantics. This process ensures that the warehouse provides a "single version of the truth" [3].



Data Warehousing Systems: Foundations and Architectures. Figure 3. An enterprise data warehousing system architecture with ROLAP/MOLAP/Hybrid OLAP.

Only cleaned and conformed data are loaded into the DW. The storage required for ETL processing is called a staging database.

The ETL process is usually the most time-consuming phase in developing a data warehousing system [7]. It normally takes 60–80% of the whole development effort. Therefore, it is highly recommended that ETL tools and data cleansing tools be used to automate the ETL process and data loading.

Data Warehouse Storage and Metadata Repository
The third layer represents the enterprise DW and metadata repository. The enterprise DW contains all the extracted and standardized historical data at the atomic data level. A DW addresses the needs of cross-functional information requirements of an enterprise. The data will remain in the warehouse until they reach the limit specified in the retention strategy. After that period, the data are purged or archived.

Another component of this layer is the metadata repository. Metadata are data about the data. The repository contains information about the structures, operations, and contents of the warehouse. Metadata allows an organization to track, understand, and manage the population and management of the warehouse. There are three types of metadata – business metadata, technical metadata, and process metadata [7]. *Business metadata* describe the contents of the DW in business terms for easy access and understanding. They include the meaning of the data, organizational rules, policies, and constraints on the data as well as descriptive names of attributes used in reports. They help users in finding specific information from the warehouse. *Technical metadata* define the DW objects such as tables, data types, partitions, and other storage structures, as well as ETL information such as the source systems, extraction frequency, and transformation rules. *Process metadata* describe events during ETL operations and query statistics such as begin time, end time, CPU seconds, disk reads, and rows processed. These data are valuable for monitoring and troubleshooting the warehouse.

Metadata management should be carefully planned, managed, and documented. OMG's Common Warehouse Metamodel [9] provides the metadata standard.

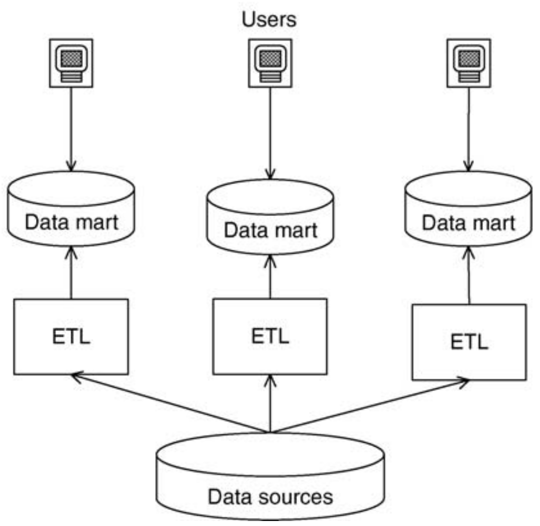
Data Mart and OLAP Engines The fourth layer represents the data marts and OLAP engines. A data mart is a small-sized DW that contains a subset of the enterprise DW or a limited volume of aggregated data for the specific analysis needs of a business unit, rather than the needs of the whole enterprise. This definition implies three important features of a data mart, different from a DW system. First, the data for a data mart is fed from the enterprise DW when a separate enterprise DW exists. Second, a data mart could store lightly aggregated data for optimal analysis. Using aggregated data improves query response time. Third, a data mart contains limited data for the specific needs of a business unit. Conceptually, a data mart covers a business process or a group of related business processes of a business unit. Thus, in a fully-developed DW environment, end-users access data marts for daily analysis, rather than the enterprise DW.

An enterprise usually ends up having multiple data marts. Since the data to all data marts are fed from the enterprise DW, it is very important to maintain the consistency between a data mart and the DW as well as among data marts themselves. A way to maintain the consistency is to use the notion of conformed dimension. A *conformed dimension* is a standardized dimension or a master reference dimension that is shared across multiple data marts [6]. Using conformed dimensions allows an organization to avoid repeating the "silos of information" problem.

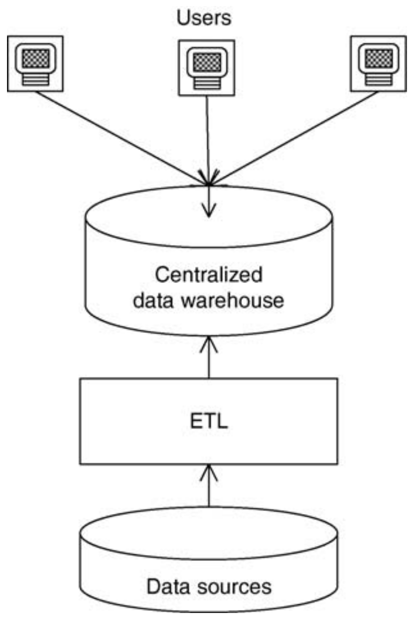
Data marts are usually implemented in one or more OLAP servers. OLAP engines allow business users to perform data analysis using one the underlying implementation model – ROLAP, MOLAP, or HOLAP.

Front-end Tools The fifth layer represents the front-end tools. In this layer, end-users use various tools to explore the contents of the DW through data marts. Typical analyses include standard report generations, ad-hoc queries, desktop OLAP analysis, CRM, operational business intelligence applications such as dashboards, and data mining.

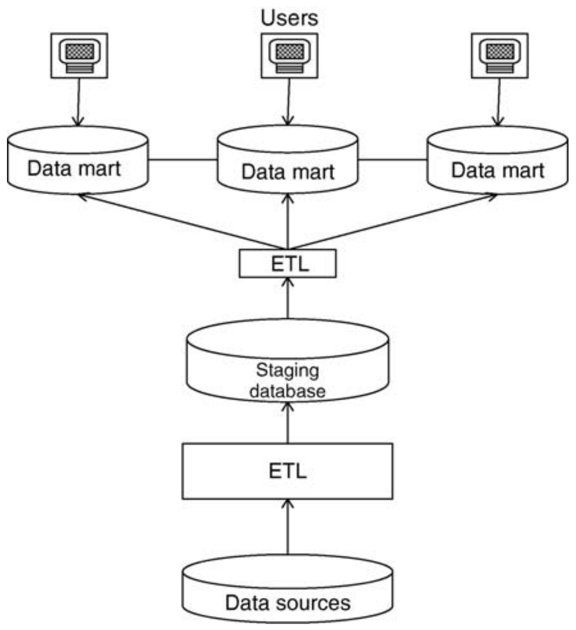
Other DW Architectures
Figure 3 depicts the architecture of a typical data warehousing system with various possible components. The two primary paradigms for DW architectures are enterprise DW design in the top-down manner [3] and data mart design in the bottom-up manner [6]. A variety of architectures based on the two paradigms and other options exists [3,6,8,10,12]. In this section, seven different architectures are outlined. Figures 4–9 illustrate those architectures.



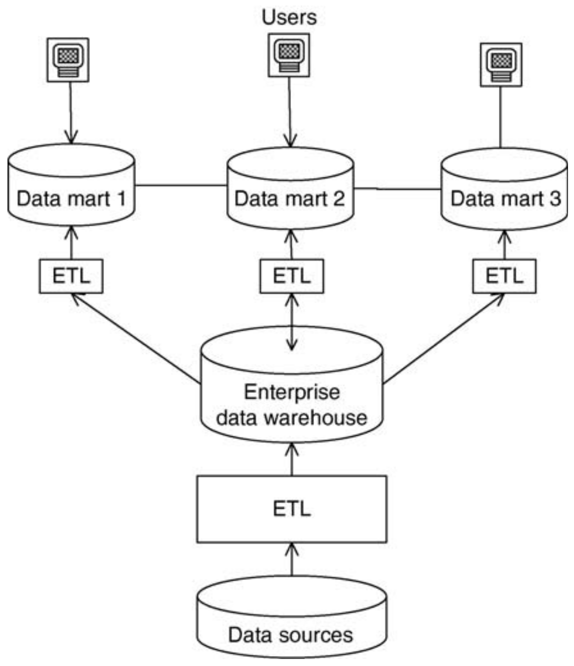
Data Warehousing Systems: Foundations and Architectures. Figure 4. Independent data marts.



Data Warehousing Systems: Foundations and Architectures. Figure 6. Centralized DW architecture with no data marts.

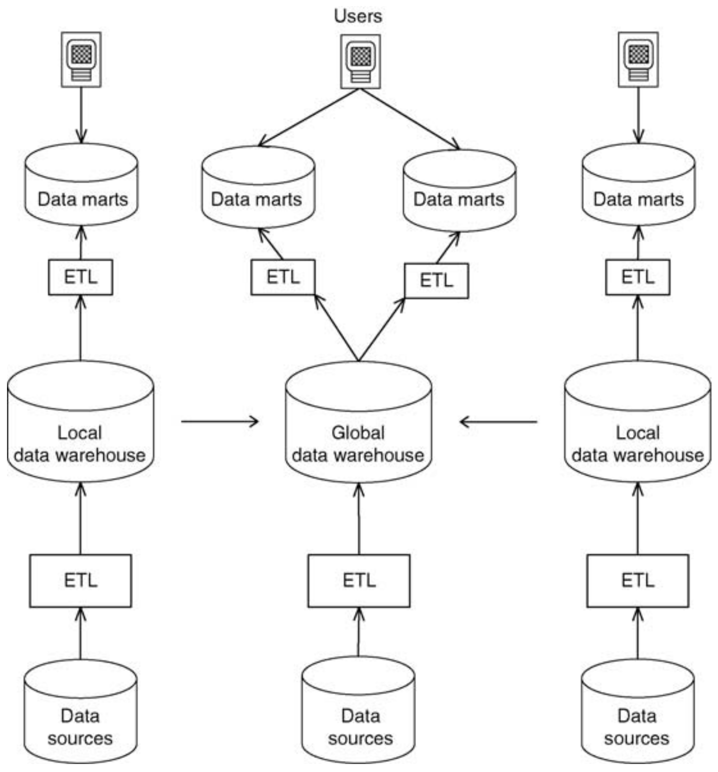


Data Warehousing Systems: Foundations and Architectures. Figure 5. Data mart bus architecture with conformed dimensions.



Data Warehousing Systems: Foundations and Architectures. Figure 7. Hub-and-spoke architecture.

Independent Data Marts Architecture In this architecture, multiple data marts are created independently of each other. The data marts do not use conformed dimensions and measures. Thus, there is no unified view of enterprise data in this architecture. As the number of data marts grows, maintenance of consistency among data marts are difficult. In the long run, this architecture is likely to produce "silos of data marts."



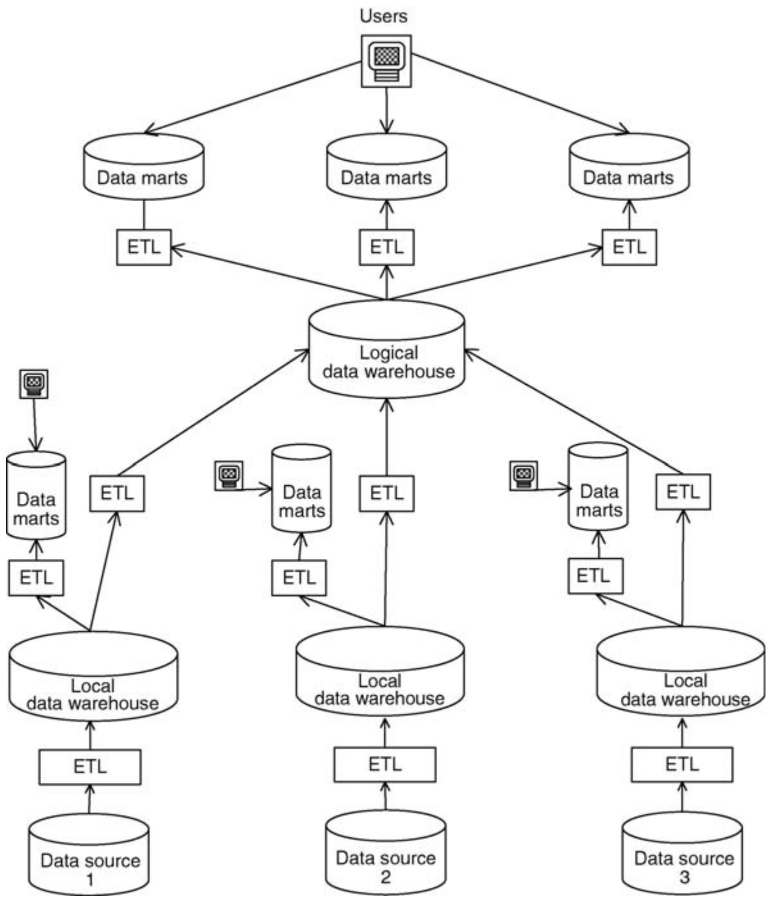
Data Warehousing Systems: Foundations and Architectures. Figure 8. Distributed DW architecture.

Data Mart Bus Architecture with Conformed Dimensions In this architecture, instead of creating a single enterprise level DW, multiple dimensional data marts are created that are linked with conformed dimensions and measures to maintain consistency among the data marts [6,7]. Here, an enterprise DW is a union of all the data marts together with their conformed dimensions. The use of the conformed dimensions and measures allows users to query all data marts together. Data marts contain either atomic data or summary data. The strength of the architecture is that data marts can be delivered quickly, and multiple data marts can be delivered incrementally. The potential weaknesses are that it does not create a single physical repository of integrated data and some data may be redundantly stored in multiple data marts.

Centralized Data Warehouse Architecture In this architecture, a single enterprise level DW is created for the entire organization without any dependent data marts. The warehouse contains detailed data for all the analytic needs of the organization. Users and applications directly access the DW for analysis.

Hub-and-Spoke Architecture (Corporate Information Factory) In this architecture, a single enterprise DW, called the hub, is created with a set of dimensional data marts, called spokes, that are dependent on the enterprise DW. The warehouse provides a single version of truth for the enterprise, and each data mart addresses the analytic needs of a business unit. This architecture is also called the corporate information factory or the enterprise DW architecture [3]. The warehouse contains data at the atomic level, and the data marts usually contain either atomic data, lightly summarized data, or both, all fed from the warehouse. The enterprise warehouse in this architecture is usually normalized for flexibility and scalability, while the data marts are structured in star schemas for performance. This top-down development methodology provides a centralized integrated repository of the enterprise data and tends to be robust against business changes. The primary weakness of this architecture is that it requires significant up-front costs and time for developing the warehouse due to its scope and scale.

Distributed Data Warehouse Architecture A distributed DW architecture consists of several local DWs



Data Warehousing Systems: Foundations and Architectures. Figure 9. Federated DW architecture.

and a global DW [3]. Here, local DWs have mutually exclusive data and are autonomous. Each local warehouse has its own ETL logic and processes its own analysis queries for a business division. The global warehouse may store corporate-wide data at the enterprise level. Thus, either corporate-level data analysis at the enterprise level or global data analyses that require data from several local DWs will be done at the global DW. For example, a financial analysis covering all the business divisions will be done at the global DW. Depending on the level of data and query flows, there could be several variations in this architecture [3]. This architecture supports multiple, geographically distributed business divisions. The architecture is especially beneficial when local DWs run on multiple vendors.

Federated Data Warehouse Architecture A federated DW architecture is a variation of a distributed DW architecture, where the global DW serves as a logical DW for all local DWs. The logical DW provides users with a single centralized DW image of the enterprise. This architecture is a practical solution when an enterprise acquires other companies that have their own DWs, which become local DWs. The primary advantage of this architecture is that existing environments of local DWs can be kept as they are without physically restructuring them into the global DW. This architecture may suffer from complexity and performance when applications require frequent distributed joins and other distributed operations. The architecture is built on an existing data environment rather than starting with a "clean slate."

Virtual Data Warehouses Architecture In a virtual DW architecture, there is no physical DW or any data mart. In this architecture, a DW structure is defined by a set of materialized views over OLTP systems. End-users directly access the data through the materialized views. The advantages of this approach are that it is easy to build and the additional storage requirement is minimal. This approach, however, has many disadvantages in that it does not allow any historical data; it does not contain a centralized metadata repository; it does not create cleansed standard data items across source systems; and it could severely affect the performance of the OLTP system.

Key Applications

Numerous business applications of data warehousing technologies to different domains are found in [1,6]. Design and development of clickstream data marts is covered in [5]. Applications of data warehousing technologies to customer relationship management (CRM) are covered in [2,11]. Extension of data warehousing technologies to spatial and temporal applications is covered in [8].

URL to Code

Two major international forums that focus on data warehousing and OLAP research are International Conferences on Data Warehousing and Knowledge Discovery (DaWaK) and ACM International Workshop on Data Warehousing and OLAP (DOLAP). DaWaK has been held since 1999, and DOLAP has been held since 1998. DOLAP papers are found at http://www.cis.drexel.edu/faculty/song/dolap.htm. A collection of articles on industrial DW experience and design tips by Kimball is listed in http://www.ralph-kimball.com/, and the one by Inmon is listed in www.inmoncif.com.

Cross-references

► Active and Real-time Data Warehousing
► Cube
► Data Mart
► Data Mining
► Data Warehouse
► Data Warehouse Life-Cycle and Design
► Data Warehouse Maintenance
► Data Warehouse Metadata
► Data Warehouse Security
► Dimension
► Evolution and Versioning
► Extraction
► Materialized Views
► Multidimensional Modeling
► On-line analytical Processing
► Optimization and Tuning in Data Warehouses
► Transformation and Loading
► View Maintenance

Recommended Reading

1. Adamson C. and Venerable M. Data Warehouse Design Solutions. Wiley, New York, 1998.
2. Cunningham C., Song I.-Y., and Chen P.P. Data warehouse design for customer relationship management. J. Database Manage., 17(2):62–84, 2006.
3. Inmon W.H. Building the Data Warehouse, 3rd edn., Wiley, New York, 2002.
4. Jones M.E. and Song I.-Y. Dimensional modeling: identification, classification, and evaluation of patterns. Decis. Support Syst., 45(1):59–76, 2008.
5. Kimball R. and Merz R. The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse. Wiley, New York, 2000.
6. Kimball R. and Ross M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd edn., Wiley, 2002.
7. Kimball R., Ross M., Thorntwaite W., Munday J., and Becker B. 1The Data Warehouse Lifecycle Toolkit, 2nd edn., Wiley, 2008.
8. Malinowski E. and Zimanyi E. Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications. Springer, 2008.
9. Poole J., Chang D., Tolbert D., and Mellor D. Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration. Wiley, 2002.
10. Sen A. and Sinha P. A comparison of data warehousing methodologies. CACM, 48(3):79–84, 2005.
11. Todman C. Designing a Data Warehouse Supporting Customer Relationship Management. Prentice-Hall, 2000.
12. Watson H.J. and Ariyachandra T. Data Warehouse Architectures: Factors in the Selection, Decision, and the Success of the Architectures. 2005.

Data, Text, and Web Mining in Healthcare

Elizabeth S. Chen
Partners HealthCare System, Boston, MA, USA

Synonyms

Data mining; Text data mining; Web mining; Web data mining; Web content mining; Web structure mining; Web usage mining

Definition

The healthcare domain presents numerous opportunities for extracting information from heterogeneous sources ranging from structured data (e.g., laboratory results and diagnoses) to unstructured data (e.g., clinical documents such as discharge summaries) to usage data (e.g., audit logs that record user activity for clinical applications). To accommodate the unique characteristics of these disparate types of data and support the subsequent use of extracted information, several existing techniques have been adapted and applied including *Data Mining*, *Text Mining*, and *Web Mining* [7]. This entry provides an overview of each of these mining techniques (with a focus on Web usage mining) and example applications in healthcare.

Historical Background

Given the exponential growth of data in all domains, there has been an increasing amount of work focused on the development of automated methods and techniques to analyze data for extracting useful information. Data mining is generally concerned with large data sets or databases; several specialized techniques have emerged such as text mining and Web mining that are focused on text data and Web data, respectively. Early applications were in the domains of business and finance; however, the past decade has seen an increasing use of mining techniques in the life sciences, biomedicine, and healthcare. In the healthcare domain, data mining techniques have been used to discover medical knowledge and patterns from clinical databases, text mining techniques have been used to analyze unstructured data in the electronic health record, and Web mining techniques have been used for studying use of healthcare-related Web sites and systems.

Foundations

Data Mining

Knowledge Discovery in Databases (KDD) and data mining are aimed at developing methodologies and tools, which can automate the data analysis process and create useful information and knowledge from data to help in decision-making [9,11]. KDD has been defined as "the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data." This process is interactive and iterative and consists of several steps: data selection, preprocessing, transformation, data mining, and interpretation. Data mining is considered one step in the KDD process and is concerned with the exploration and analysis of large quantities of data in order to discover meaningful patterns and rules [9,11]. Two primary goals of data mining are prediction and description.

Text Mining

While data mining focuses on algorithmic and database-oriented methods that search for previously unsuspected structure and patterns in data, text mining is concerned with semi-structured or unstructured data found within text documents [5,12]. A narrower definition of text mining follows that of data mining in that it aims to extract useful information from text data or documents; a broader definition includes general text processing techniques that deal with search, extraction, and categorization [17]. Example applications include document classification, entity extraction, and summarization.

Web Mining

Web mining is the application of data mining techniques to automatically discover and extract information from data related to the World Wide Web [9,24,25]. Three categories of Web mining have been defined [18,6]:

- *Web content mining*: involves the discovery of useful information from Web content. These techniques involve examining the content of Web pages as well as results of Web searching.
- *Web structure mining*: obtains information from the organization of pages on the Web. These techniques seek to discover the model underlying link structures of the Web.
- *Web usage mining*: discovers usage patterns from Web data. These techniques involve analyzing data derived from the interactions of users while interacting with the Web.

Web usage mining seeks to understand the behavior of users by automatically discovering access patterns from their Web usage data. These data include Web server access logs, proxy server logs, browser logs, user sessions, and user queries. The typical Web usage mining process has three phases: preprocessing, pattern discovery, and pattern analysis [6,18,25].

Key Applications

Data Mining in Healthcare

Several studies have discussed the use of structured and unstructured data in the electronic health record for understanding and improving health care processes [5]. Applications of data mining techniques for structured clinical data include extracting diagnostic rules, identifying new medical knowledge, and discovering relationships between different types of clinical data. Using association rule generation, Doddi et al. discovered relationships between procedures performed on a patient and the reported diagnoses; this knowledge could be useful for identifying the effectiveness of a set of procedures for diagnosing a particular disease [8]. To identify factors that contribute to perinatal outcomes, a database of obstetrical patients was mined for the goal of improving the quality and cost effectiveness of perinatal care [23]. Mullins et al. explored a set of data mining tools to search a clinical data repository for novel disease correlations to enhance research capabilities [21].

Text Mining in Healthcare

Natural language processing and text mining techniques have been applied in healthcare for a range of applications including coding and billing, tracking physician performance and resource utilization, improving provider communication, monitoring alternate courses of treatment, and detecting clinical conditions and medical errors [15]. Several studies have focused on the development of text mining approaches for identifying specific types of co-occurring concepts (e.g., concept pairs such as disease-drug or disease-finding) in clinical documents (e.g., discharge summaries) and biomedical documents (e.g., Medline articles). In one study, associations between diseases and findings (extracted from discharge summaries using a natural language processing tool) were identified and used to construct a knowledge base for supporting an automated problem list summarization system [2]. Another study discusses the mining of free-text medical records for the creation of disease profiles based on demographic information, primary diseases, and other clinical variables [14].

Web Usage Mining in Healthcare

Major application areas of Web usage mining include personalization, system improvement, site modification, business intelligence, and usage characterization [25]. Web usage mining is viewed as a valuable source of ideas and methods for the implementation of personalized functionality in Web-based information systems [10,22]. Web personalization aims to make Web-based information systems adaptive for the needs and interests of individual users. The four basic classes of personalization functions are: *memorization, guidance, customization,* and *task performance support*. A number of research projects have used Web usage mining techniques to add personalization functionality in Web-based systems [20].

There are several reports of applying advanced techniques such as Web usage mining to study healthcare-related Web sites and systems. Malin has looked at correlating medical status (represented in health insurance claims as ICD-9 codes) with how information is accessed in a health information Web site [19]. The value of log data for public health surveillance has been explored for detecting possible epidemics through usage logs that record accesses to disease-specific on-line health information [16,13]. Zhang et al. used Web usage data to study users' information-seeking patterns of MyWelch, a Web-based medical library portal system [27]. Rozic-Hristovski et al. have used data warehouse and On-Line Analytical Processing (OLAP) techniques to evaluate use of the Central Medical Library (CMK) Web site. They found that existing Web log analysis tools only provided a set of predefined reports without any support for interactive data exploration, while their data warehouse and OLAP techniques would allow for dynamic generation of different user-defined reports that could be used to restructure the CMK Web site [26].

```
Oct  1  00:19:47  server1  syslog:  |WebCIS|user1||||login
Oct  1  00:20:19  server1  syslog:  |WebCIS|user1|111.222.33.77|mrn2|lab^2002-09-30-12.15.00.000000|view
Oct  1  00:21:43  server1  syslog:  |WebCIS|user1|111.222.33.77|mrn2|rad|view
Oct  1  00:21:55  server1  syslog:  |WebCIS|user1|111.222.33.77|mrn2|rad^2002-09-04-20.27.00.000000|view
Oct  1  00:26:03  server1  syslog:  |WebCIS|user1|111.222.33.77|||logout
```

Data, Text, and Web Mining in Healthcare. Figure 1. WebCIS log file records. The WebCIS log files record details for users' (e.g., clinicians) interactions with patient data. Log file lines provide information on who, what, when, where, and how information was accessed in a patient's record. Each line has seven fields: timestamp, application name, userID, IP address, Medical Record Number (MRN), data type, and action. Data types may have subtypes (delimited by "^"). For example, the subtype "2002–09–30–12.15.00.000000" for the data type "lab" refers to a specific laboratory result (e.g., Basic Metabolic Panel) for the patient.

(a) lab^Basic Metabolic panel -> rad -> rad^ X-ray of chest

(b) If user viewed a basic metabolic panel laboratory result and is viewing the listing of radiology results for a patient, then user will view an X-ray of chest radiology result next

(c) If user viewed a basic metabolic panel laboratory result, then user will view an X-ray of chest radiology result next

Data, Text, and Web Mining in Healthcare. Figure 2. Transforming usage patterns to rules to shortcut rules. Each usage pattern (mined from the CIS log files (a) can be converted to a rule (b) and some patterns can be transformed to shortcut rules that exclude viewing of the department listings such as a listing of radiology results (c).

Another study explored regression analysis as a Web usage mining technique to analyze navigational routes used to access the gateway pages of the Arizona Health Sciences Library Web site. Bracke concluded that this technique could supplement Web log analysis for improving the design of Web sites [1].

Experimental Results

Depending on the clinical task, often only subsets of data are of interest to clinicians. Identifying these data, and the patterns in which they are accessed, can contribute to the design of efficient clinical information systems. At NewYork-Presbyterian Hospital (NYP), a study was performed to learn the patient-specific information needs (need for data in the patient record) of clinicians from the log files of WebCIS (a Web-based clinical information system at NYP) and subsequently apply this knowledge to enhance PalmCIS (a wireless handheld extension to WebCIS) [3,4].

Based on existing mining techniques (i.e., data mining and Web usage mining), "CIS Usage Mining" was developed as an automated approach for identifying patterns of usage for clinical information systems through associated log files (CIS log files). The CIS usage mining process consists of four phases: Data Collection – identify sources of CIS log files and obtain log file data (Fig. 1); Preprocessing – perform various tasks to prepare data for pattern discovery techniques including de-identification, data cleaning, data enrichment, and data transformation; Pattern Discovery – apply techniques for discovering statistics, patterns, and relationships such as descriptive statistical analysis, sequential pattern discovery, classification, and association rule generation; and, Pattern Analysis – filter out uninteresting patterns and determine how the discovered knowledge can be used through visualization techniques or query mechanisms.

The CIS usage mining techniques were applied to the log files of WebCIS to obtain usage statistics and patterns for all WebCIS users as well as particular classes of users (e.g., role-based groups such as physicians or nurses or specialty-based groups like pediatrics and surgery). A subset of the patterns were transformed into rules and stored in a knowledge base for enhancing PalmCIS with context-sensitive "shortcuts", which seek to anticipate what patient data the clinician may be interested in viewing next and provide automated links to those data (Fig. 2). Preliminary evaluation results indicated that shortcuts may have a positive impact and that CIS usage mining techniques may be valuable for detecting clinician information needs in different contexts.

Cross-references

► Association Rules
► Data Mining
► Text Mining
► Text Mining of Biological Resources
► Visual Data Mining

Recommended Reading

1. Bracke P.J. Web usage mining at an academic health sciences library: an exploratory study. J. Med. Libr. Assoc., 92(4): 421–428, 2004.
2. Cao H., Markatou M., Melton G.B., Chiang M.F., and Hripcsak G. Mining a clinical data warehouse to discover disease-finding associations using co-occurrence statistics. In Proc. AMIA Annual Symposium, 2005, pp. 106–110.
3. Chen E.S. and Cimino J.J. Automated discovery of patient-specific clinician information needs using clinical information system log files. In Proc. AMIA Symposium, 2003, pp. 145–149.
4. Chen E.S. and Cimino J.J. Patterns of usage for a web-based clinical information system. In Medinfo, 2004, pp. 18–22.
5. Chen H., Fuller S., Friedman C., and Hersh W. Knowledge Management and Data Mining in Biomedicine. Springer, 2005.
6. Cooley R., Mobasher B., and Srivastava J. Web mining: information and pattern discovery on the World Wide Web. In Proc.

Nineth IEEE Int. Conf. on Tools with Artificial Intelligence, 1997, pp. 558–567.
7. Data Mining, Web Mining, Text Mining, and Knowledge Discovery. wwwkdnuggetscom.
8. Doddi S., Marathe A., Ravi S.S., and Torney D.C. Discovery of association rules in medical data. Med. Inform. Internet Med., 26(1):25–33, 2001.
9. Dunham M. Data Mining Introductory and Advanced Topics. Prentice-Hall, Englewood, Cliffs, NJ, 2003.
10. Eirinaki M. and Vazirgiannis M. Web mining for web personalization. ACM Trans. Internet Techn., 3(1):1–27, 2003.
11. Fayyad U., Piatetsky-Shapiro G., Smyth P., and Uthurusamy R. Advances in Knowledge Discovery and Data Mining. AAAI/MIT, 1996.
12. Hearst M. Untangling text data mining. In Proceedings of ACL. 1999.
13. Heino J. and Toivonen H. Automated detection of epidemics from the usage logs of a physicians' reference database. In Principles of Data Mining and Knowledge Discovery, 7th European Conf, 2003, pp. 180–191.
14. Heinze D.T., Morsch M.L., and Holbrook J. Mining free-text medical records. In Proc. AMIA Symposium, 2001, pp. 254–258.
15. Hripcsak G., Bakken S., Stetson P.D., and Patel V.L. Mining complex clinical data for patient safety research: a framework for event discovery. J. Biomed. Inform. 36(1–2):120–30, 2003.
16. Johnson H.A., Wagner M.M., Hogan W.R., Chapman W., Olszewski R.T., and Dowling J. et al. Analysis of web access logs for surveillance of influenza. In Medinfo, 2004, p. 1202.
17. Konchady M. Text Mining Application Programming. Charles River Media. 2006, p. 2.
18. Kosala R. and Blockeel H. Web mining research: a survey. SIGKDD Explor., 2(1):1–15, 2000.
19. Malin B.A. Correlating web usage of health information with patient medical data. In Proc. AMIA Symposium. 484–488, 2002.
20. Mobasher B., Cooley R., and Srivastava J. Automatic personalization based on web usage mining. Commun. ACM, 43(8):142–151, 2000.
21. Mullins I.M., Siadaty M.S., Lyman J., Scully K., Garrett C.T., and Greg Miller W. et al. Data mining and clinical data repositories: insights from a 667,000 patient data set. Comput. Biol. Med., 36 (12):1351–77, 2006.
22. Pierrakos D., Paliouras G., Papatheodorou C., and Spyropoulos C. Web usage mining as a tool for personalization: a survey. User Model. User-Adap., 13(4):311–372, 2003.
23. Prather J.C., Lobach D.F., Goodwin L.K., Hales J.W., Hage M.L., and Hammond W.E. Medical data mining: knowledge discovery in a clinical data warehouse. In Proc. AMIA Annual Fall Symposium. pp. 101–105, 1997.
24. Scime A. Web mining: applications and techniques. Idea Group Inc. 2005.
25. Srivastava J., Cooley R., Deshpande M., and Tan P. Web usage mining: discovery and applications of usage patterns from web data. SIGKDD Explor., 1(2):12–23, 2000.
26. Rozic-Hristovski A., Hristovski D., and Todorovski L. Users' information-seeking behavior on a medical library Website. J. Med. Libr. Assoc., 90(2):210–217, 2002.
27. Zhang D., Zambrowicz C., Zhou H., and Roderer N. User information seeking behavior in a medical web portal environment: a preliminary study. J. Am. Soc. Inform. Sci. Tech., 55(8): 670–684, 2004.

Database Adapter and Connector

Changqing Li
Duke University, Durham, NC, USA

Synonyms

Database connectivity

Definition

A database connector is a software that connects an application to any database. A database adapter is an implementation of a database connector. The connector is more at the conceptual level, while the adapter is at the implementation level, though they refer to the same thing. For simplicity, in the remaining parts of this entry, a database adapter will not be explicitly distinguished from a database connector, i.e. they are used to have the same meaning in the rest sections. Unlike the way to access data with a fixed schema, stored procedures, or queues, one can access table data directly and transparently with a database adapter.

Open Database Connectivity (ODBC) [2] and Java Database Connectivity (JDBC) [4] are two main database adapters to execute Structured Query Language (SQL) statements and retrieve results.

Historical Background

Before the universal database adapters, one has to write code that talks to a particular database using an appropriate language. For example, if a program needs to talk to an Access database and an Oracle database, the program has to be coded with two different database languages. This can be a quite daunting task, therefore uniform database adapters emerged.

Here the histories of the two main universal database adapters, i.e. ODBC and JDBC, are introduced.

ODBC enables applications connect to any database for which an ODBC driver is available. ODBC was created in 1992 by Microsoft, in partnership with Simba Technologies, by adapting the Call Level Interface (CLI) from the SQL Access Group (SAG). Later ODBC was aligned with the CLI specification making its way through X/Open (a company name) and International

Organization for Standardization (ISO), and SQL/CLI became part of the international SQL standard in 1995.

JDBC is similar to ODBC, but is designed specifically for Java programs. JDBC was firstly developed by JavaSoft, a subsidiary of Sun Microsystems, then developed under the Java Community Process. JDBC is part of the Java Standard Edition and the Java package java.sql contains the JDBC classes.

Foundations

A data architecture defines a data source interface to an application through connectors, and also by commands. Thus, a configurable request for data is issued through commands to the adapters of the data sources. This architecture provides the ability to create custom connectivity to disparate backend data sources.

Universal connectors enable rapid access to heterogeneous data and allow a broad range of seamless connectivity to file systems, databases, web applications, business applications and industry-standard protocols on numerous platforms. Business connectors allow customers to participate in collaboration, while web database adapters allow direct access to the database from web services.

Relational Database (RDB) adapters efficiently provide access to RDB data and systems. Standard SQL statements may be used to access RDB data via connectors including ODBC, OLE DB (Object Linking and Embedding, Database), JDBC, XML, iWay Business Services (Web services), MySQL Connector/ODBC and Connector/NET driver, and others.

Due to the longer history, ODBC offers connectivity to a wider variety of data sources than other new data-access Application Programming Interfaces (APIs) such as OLE DB, JDBC, and ADO.NET (ADO stands for ActiveX Data Objects).

Before the information from a database can be used by an application, an ODBC data source name must be defined, which provides information about how to connect the application server to a database, such as Microsoft SQL Server, Sybase, Oracle, or IBM DB2.

The implementations of ODBC can run on different operating systems such as Microsoft Windows, Unix, Linux, OS/2, and Mac OS X. Hundreds of ODBC drivers exist for different database products including Oracle, DB2, Microsoft SQL Server, Sybase, MySQL, PostgreSQL, Pervasive SQL, FileMaker, and Microsoft Access.

The first ODBC product was released by Microsoft as a set of Dynamic-Link Libraries (DLLs) for Microsoft Windows. In 2006, Microsoft ships its own ODBC with every supported version of Windows.

Independent Open Database Connectivity (iODBC) offers an open source, platform-independent implementation of both the ODBC and X/Open specifications. iODBC has been bundled into Darwin and Mac OS X, and it has also been ported by programmers to several other operating systems and hardware platforms, including Linux, Solaris, AIX, HP-UX, Digital UNIX, Dynix, FreeBSD, DG-UX, OpenVMS, and others.

Universal Database Connectivity (UDBC), laid the foundation for the iODBC open source project, is a cross-platform fusion of ODBC and SQL Access Group CLI, which enables non-Windows-based DBMS-independent (Database Management System independent) application development when shared-library implementations on Unix occurred only sporadically.

Headed, maintained and supported by Easysoft Director Nick Gorham, unixODBC has become the most common driver-manager for non-Microsoft Windows platforms and for one Microsoft platform, Interix. In advance of its competitors, unixODBC fully supports ODBC3 and Unicode. Most Linux distributions including Red Hat, Mandriva and Gentoo, now ship unixODBC. unixODBC is also used as the drivers by several commercial database vendors, including IBM (DB2, Informix), Oracle and SAP (Ingres). Many open source projects also make use of unixODBC. unixODBC builds on any platform that supports most of the GNU (a computer operating system composed entirely of free software) autoconf tools, and uses the LGPL (Lesser General Public License) and the GPL (General Public License) for licensing.

ODBC provides the standard of ubiquitous connectivity and platform-independence because hundreds of ODBC drivers exist for a large variety of data sources.

However, ODBC has certain drawbacks. Writing ODBC code to exploit DBMS-specific features requires more advanced programming. An application needs to use introspection to call ODBC metadata functions that return information about supported features, available types, syntax, limits, isolation levels, driver capabilities and more. Even when adaptive techniques are used, ODBC may not provide some advanced DBMS features. Important issues can also be raised by differences between drivers and driver maturity. Compared with drivers deployed and tested for years which may contain fewer bugs, newer ODBC drivers do not always have the stability.

Developers may use other SQL APIs if ODBC does not support certain features or types but these features are required by the applications. Proprietary APIs can be used if it is not aiming for platform-independence; whereas if it is aiming to produce portable, platform-independent, albeit language specific code, JDBC API is a good choice.

Sun's (a company name) Java (a programming language) 2 Enterprise Edition (J2EE) Connector Architecture (JCA) defines a standard architecture for connecting the Java 2 Platform to heterogeneous Enterprise Information Systems (EISs). The JCA enables an EIS vendor to provide a standard resource adapter (connector). The JDBC Connector is used to connect relational data sources. DataDirect technology is a pioneer in JDBC which provides resource adapters as an installable option for JDBC. The JDBC Developer Center provides the most current, developer-oriented JDBC data connectivity information available in the industry.

Multiple implementations of JDBC can exist and be used by the same application. A mechanism is provided by the API to dynamically load the correct Java packages and register them with the JDBC Driver Manager, a connection factory for creating JDBC connections.

Creating and executing statements are supported by JDBC connections. These statements may either be update statements such as SQL CREATE, INSERT, UPDATE and DELETE or query statements with SELECT.

Update statements e.g. INSERT, UPDATE and DELETE return how many rows are affected in the database, but do not return any other information.

Query statements, on the other hand, return a JDBC row result set, which can be walked over. Based on a name or a column number, an individual column in a row can be retrieved. Any number of rows may exist in the result set and the row result set has metadata to describe the names of the columns and their types.

To allow for scrollable result sets and cursor support among other things, there is an extension to the basic JDBC API in the javax.sql package.

Next the bridging configurations between ODBC and JDBC are discussed:

ODBC-JDBC bridges: an ODBC-JDBC bridge consists of an ODBC driver, but this ODBC driver uses the services of a JDBC driver to connect to a database. Based on this driver, ODBC function calls are translated into JDBC method calls. This bridge is usually used when an ODBC driver is lacked for a particular database but access to a JDBC driver is provided.

JDBC-ODBC bridges: a JDBC-ODBC bridge consists of a JDBC driver, but this JDBC driver uses the ODBC driver to connect to the database. Based on this driver, JDBC method calls are translated into ODBC function calls. This bridge is usually used when a particular database lacks a JDBC driver. One such bridge is included in the Java Virtual Machine (JVM) of Sun Microsystems. Sun generally recommends against the use of its bridge. Far outperforming the JVM built-in, independent data-access vendors now deliver JDBC-ODBC bridges which support current standards.

Furthermore, the OLE DB [1], the Oracle Adapter [3], the iWay [6] Intelligent Data Adapters, and MySQL [5] Connector/ODBC and Connector/NET are briefly introduced below:

OLE DB (Object Linking and Embedding, Database), maybe written as OLEDB or OLE-DB, is an API designed by Microsoft to replace ODBC for accessing different types of data stored in a uniform manner. While supporting traditional DBMSs, OLE DB also allows applications to share and access a wider variety of non-relational databases including object databases, file systems, spreadsheets, e-mail, and more [1].

The Oracle Adapter for Database and Files are part of the Oracle Business Process Execution Language (BPEL) Process Manager installation and is an implementation of the JCA 1.5 Resource Adapter. The Adapter is based on open standards and employs the Web Service Invocation Framework (WSIF) technology for exposing the underlying JCA Interactions as Web Services [3].

iWay Software's Data Adapter can be used for ALLBASE Database, XML, JDBC, and ODBC-Based Enterprise Integration. The Intelligent Data Adapters of iWay Work Together; each adapter contains a communication interface, a SQL translator to manage adapter operations in either SQL or iWay's universal Data Manipulation Language (DML), and a database interface to translate standard SQL into native SQL syntax [6].

MySQL supports the ODBC interface Connector/ODBC. This allows MySQL to be addressed by all the usual programming languages that run under Microsoft Windows (Delphi, Visual Basic, etc.). The ODBC interface can also be implemented under Unix, though that is seldom necessary [5]. The Microsoft .NET Framework, a software component of Microsoft Windows operating system, provides a programming interface to Windows services and APIs, and manages the execution of programs written for this framework [7].

Key Applications

Database adapters and connectors are essential for the current and future Web Services and Service Oriented Architecture, Heterogeneous Enterprise Information Systems, Data Integration and Data Interoperability, and any other applications to access any data transparently.

URL To Code

The catalog and list of ODBC Drivers can be found at: http://www.sqlsummit.com/ODBCVend.htm and http://www.unixodbc.org/drivers.html.

The guide about how to use JDBC can be found at: http://java.sun.com/javase/6/docs/technotes/guides/jdbc/.

Cross-references

► Data Integration
► Interface
► Java Database Connectivity
► .NET Remoting
► Open Database Connectivity
► Web 2.0/3.0
► Web Services

Recommended Reading

1. Blakeley J. OLE DB: a component dbms architecture. In Proc. 12th Int. Conf. on Data Engineering, 1996.
2. Geiger K. Inside ODBC. Microsoft, 1995.
3. Greenwald R., Stackowiak R., and Stern J. Oracle Essentials: Oracle Database 10g. O'Reilly, 2004.
4. Hamilton G., Cattell R., and Fisher M. JDBC Database Access with Java: A Tutorial and Annotated Reference. Addison Wesley, USA, 1997.
5. Kofler M. The Definitive Guide to MySQL5. A press, 2005.
6. Myerson J. The Complete Book of Middleware. CRC, USA, 2002.
7. Thai T., Lam H., .NET Framework Essentials. O'Reilly, 2003.

Database Clustering Methods

Xue Li
The University of Queensland, Brisbane, QLD, QLD, Australia

Synonyms

Similarity-based data partitioning

Definitions

Given a database $D = \{t_1, t_2, \ldots, t_n\}$, of tuples and a user defined similarity function s, $0 \leq s(t_i, t_j) \leq 1$, $t_i, t_j \in D$, the database clustering problem is defined as a partitioning process, such that D can be partitioned into a number of (such as k) subsets (k can be given), as C_1, $C_2, \ldots, C_k$, according to s by assigning each tuple in D to a subset C_i. C_i is called a cluster such that $C_i = \{t_i \mid s(t_i, t_r) \geq s(t_i, t_s)$, *if* $t_i, t_r \in C_j$ *and* $t_s \notin C_j\}$.

Key Points

Database clustering is a process to group data objects (referred as tuples in a database) together based on a user defined similarity function. Intuitively, a cluster is a collection of data objects that are "similar" to each other when they are in the same cluster and "dissimilar" when they are in different clusters. Similarity can be defined in many different ways such as Euclidian distance, Cosine, or the dot product. For data objects, their membership belonging to a certain cluster can be computed according to the similarity function. For example, Euclidian distance can be used to compute the similarity between the data objects with the numeric attribute values, where the geometric distance is used as a measure of the similarity. In a Euclidian space, the data objects are to each other, the more similar they are. Another example is to use the Euclidian distance to measure the similarity between a data object and a central point namely centroid of the cluster. The closer to the centroid the object is, the more likely it will belong to the cluster. So in this case, the similarity is decided by the radius of the points to their geometric centre.

For any given dataset a challenge question is how many natural clusters that can be defined. The answer to this question is generally application-dependent and can be subjective to user intentions.

In order to avoid specifying k for the number of clusters in a clustering process, a hierarchical method can be used. In this case, two different approaches, either agglomerative or divisive, can be applied. Agglomerative approach is to find the clusters step-by-step through a bottom-up stepwise merging process until the whole dataset is grouped as a single cluster. Divisive approach is to find the clusters step-by-step through a top-down stepwise split process until every data object becomes a single cluster.

Although hierarchical approaches have been widely used in many applications such as biomedical researches and experimental analysis in life science, they suffer from the problems of unable to undo the intermediate results in order to approach a global optimum solution. In an agglomerative approach, once two objects are merged, they will be together for all following merges and cannot be reassigned. In a divisive approach, once a cluster is split into two sub-clusters, they cannot be re-grouped into the same cluster for the further split.

In addition to hierarchical approaches, which do not need to specify how many clusters to be discovered, a user may specify an integer *k* for clustering data objects. In general, the task of finding a global optimal *k* partitions belongs to the class of NP-hard problem. For this reason, heuristics are used in many algorithms to achieve a balance between the efficiency and effectiveness as much as possible to close to the global optimum. Two well-known algorithms are the k-means and k-medoids.

One important feature of database clustering is that a dataset tends to be very large, high-dimensional, and coming at a high speed. By using a balanced tree structure, BIRCH algorithm [3] makes a single scan on the incoming data stream. BIRCH algorithm consists of two phases: (i) a summary of historical data is incrementally maintained in main memory as a clustering tree (CF tree). A node in CF tree gives the cardinality, centre, and radius of the cluster. Based on some heuristics, each new arriving data object is assigned to a sub-cluster, which leads to the update of its cluster feature in the CF tree. (ii) The clustering process is then applied on the leaf nodes of the CF tree. When the final cluster needs to be generated, the sub-clusters are treated as weighted data points and various traditional clustering algorithms can be applied in phase two computation without involving I/O operations.

DBSCAN [1] is a density based approach considering the coherence of the data objects. As a result, the nonconvex shapes clusters can be found based on the density that connect the data objects forming any kind of shapes in a Euclidian space. Spatial data indexes such as R* tree can be used to improve the system performance. STING [2] is another hierarchical approach that uses a grid structure to stores density information of the objects.

The key features of database clustering approaches are that (i) they are designed to deal with a large volume of data so a trade-off of accuracy and efficiency often needs to be considered. (ii) They are not able to see the complete dataset before the objects are clustered. So a progressive resolution refinement is used to approach the optimal solutions. (iii) They are designed to deal with constant data streams and so the incremental maintenances of the clustering results are required.

Cross-references

► Data Partitioning
► K-Means and K-Medoids Clustering
► Unsupervised Learning

Recommended Reading

1. Ester M., Kriegel H.P., Sander J., and Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise, In Proc. Second Int. Conf. on Knowledge Discovery and Data Mining, pp. 226–231.
2. Han J., Kamber M., and Tung A.K.H. *1*Spatial clustering methods in data mining: a survey, In Geographic Data Mining and Knowledge Discovery, H. Miller, J. Han (eds.). Taylor and Francis, UK, 2001.
3. Zhang T., Ramakrishnan R., and Livny M. Birch: An efficient data clustering method for very large databases. In Proc. 1996 ACM SIGMOD Int. Conf. on Management of Data. Quebec, Canada, 1996, pp. 103–114.

Database Clusters

Marta Mattoso
Federal University of Rio de Janeiro, Rio de Janeiro, Brazil

Synonyms

DBC

Definition

A database cluster (DBC) is as a standard computer cluster (a cluster of PC nodes) running a Database Management System (DBMS) instance at each node. A DBC middleware is a software layer between a database application and the DBC. Such middleware is responsible for providing parallel query processing on top of the DBC. It intercepts queries from applications and coordinates distributed and parallel query execution by taking advantage of the DBC. The DBC term comes from an analogy with the term PC cluster, which is a solution for parallel processing by assembling sequential PCs. In a PC cluster there is no need for special hardware to provide parallelism as opposed to parallel machines or supercomputers. A DBC takes advantage of off-the-shelf sequential DBMS to run parallel queries. There is no need for special software or hardware as opposed to parallel database systems. The idea is to offer a high performance and cost-effective solution based on a PC cluster, without needing to change the DBMS or the application and its database.

Historical Background

Traditionally, high-performance of database query processing has been achieved with parallel database systems [7]. Parallel processing has been successfully used to improve performance of heavy-weight queries, typically by replacing the software and hardware platforms with higher computational capacity components (e.g. tightly-coupled multiprocessors and parallel database systems). Although quite effective, this solution requires the database system to have full control over the data, requiring an efficient database partitioning design. It also requires adapting applications from the sequential to the parallel environment. Migrating applications is complex (sometimes impossible), since it may require modifications to the source code. In addition, often it requires the expansion of the computational environment and the application modification, which can be very costly. A cheaper hardware alternative is to use parallel database systems for PC clusters. However, the costs can still be high because of a new database partitioning design and some solutions require specific software (DBMS) or hardware (e.g. SAN – Storage Area Network).

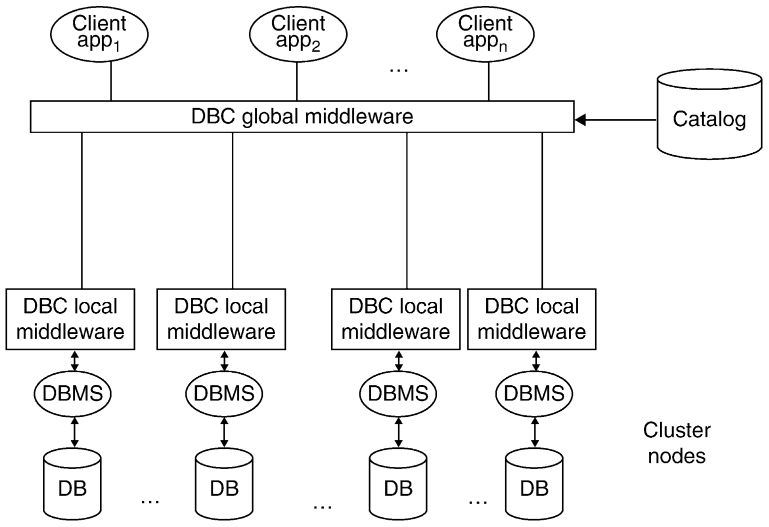
The DBC approach has been initially proposed by the database research group from ETH Zurich through the PowerDB project [10] to offer a less expensive and cost-effective alternative for high performance query processing. Thus, DBC is based on clusters of PC servers and pre-existing DBMS and applications. However, PowerDB is not open-source nor available for download. Several open-source DBC systems (e.g. RepDB*, C-JDBC, ParGRES, and Sequoia) have been proposed to support database applications by using different kinds of database replication on the DBC to obtain inter- and intra-query parallelism and fault tolerance.

Foundations

While many techniques are available for high performance query processing in parallel database systems, the main challenge of a DBC is to provide parallelism from outside the DBMS software.

A typical DBC architecture is a set of PC servers interconnected by a dedicated high-speed network, each one having its own processor(s) and hard disk (s), and running an off-the-shelf DBMS all coordinated by the DBC software middleware (Fig. 1). The DBC middleware is responsible for offering a single external view of the whole system, like a virtual DBMS. Applications need not be modified when database servers are replaced by their cluster counterparts. The DBC approach is considered to be non-intrusive since it does not require changes on the current application, its queries, its DBMS and its database.

Typically, the application is on the client side while the DBMS and the database is fully replicated at the PC cluster nodes. The DBC software middleware



Database Clusters. Figure 1. DBC architecture.

intercepts the application queries at the moment they are sent to the DBMS through the database driver. The DBC middleware then defines the best strategy to execute this query on the DBC to obtain the best performance from the DBC configuration. The DBC software middleware is typically divided on a global component which orchestrates the parallelism and a local component which tunes the local execution to participate on load balancing.

High performance in database applications can be obtained by increasing the system throughput, i.e. improving the number of transactions processed per second, and by speeding-up the query execution time for long running queries. The DBC query execution strategy varies according to the type of transactions being submitted and the DBC load. To improve system throughput, the DBC uses inter-query parallelism. To improve queries with long time execution the DBC implements intra-query parallelism. Inter- and intra-query parallelism can be combined. The query execution strategy is based on available database replicas.

Inter-query parallelism consists of executing many queries at the same time, each at a different node. Inter-query parallelism is implemented in DBC by transparently distributing queries to nodes that contain replicas of the required database. When the database is replicated at all nodes of the DBC, read-only inter-query parallelism is almost straightforward. Any read query can be sent to any replica node and execute in parallel. However, in the presence of updates the DBC must ensure the ACID transaction properties. Typically, a DBC global middleware has a component that manages a pool of connections to running DBMSs. Each request received by the DBC is submitted to a scheduler component that controls concurrent request executions and makes sure that update requests are executed in the same order by all DBMSs. Such scheduler should be able to be configured to enforce different parallel levels of concurrency.

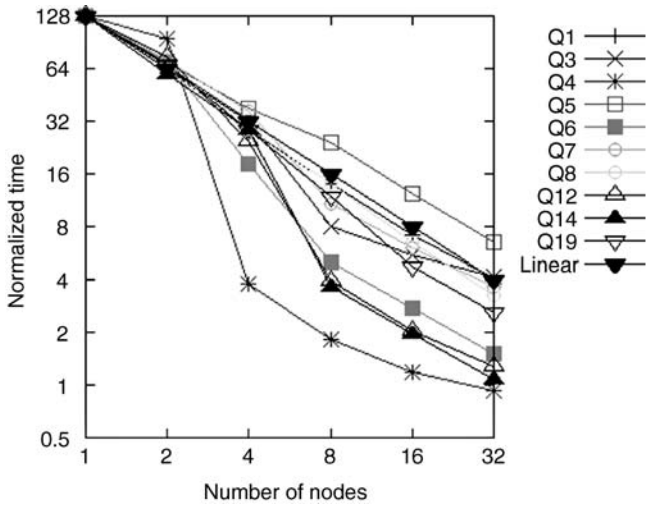
Intra-query parallelism consists of executing the same query in parallel, using sub-queries that scan different parts of the database (i.e. a partition), each at a different node. In a DBC, scanning different partitions without hurting the database autonomy is not simple to implement. In DBC, independent DBMSs are used by the middleware as "black-box" components. It is up to the middleware to implement and coordinate parallel execution. This means that query execution plans generated by such DBMSs are not parallel. Furthermore, as "black-boxes," they cannot be modified to become aware of the other DBMS and generate cooperative parallel plans. Physically partitioning the database relies on a good distribution design which may not work for several queries. An interesting solution to implement intra-query parallelism in DBC is to keep the database replicated and design partitions using *virtual partitioning* (VP) as proposed by Akal et al. [1]. VP is based on replication and dynamically designs partitions. The basic principle of VP is to take one query, rewrite it as a set of sub-queries "forcing" the execution of each one over a different subset of the table. Then the final query result is obtained through a composition of the partial results generated by the sub-queries.

Key Applications

DBC obtained much interest for various database applications like OLTP, OLAP, and e-commerce. Such applications can be easily migrated from sequential environments to the low cost DBC solution and obtain high performance in query processing. Different DBC open source solutions are available to cost-effective parallelism for various database applications. Since the high-performance requirements vary according to the typical queries of the applications, different DBC parallel techniques are provided. C-JDBC [3] and Sequoia [11] are DBC focused on e-commerce and OLTP applications. They use inter-query parallelism and are based on fault tolerance and load balancing in query distribution. RepDB* [8] is a DBC focused on throughput, which offers HPC for OLTP transactions. It uses inter-query parallelism and it is based on replica consistency techniques. ParGRES [6] is the only open-source DBC to provide for intra-query parallel processing [5], thus it is focused on OLAP applications. All these solutions have shown significant speedup through high performance query processing. Experimental results using the TPC series of benchmarks can be found for each one of the specific DBC software middlewares, for example TPC-W with C-JDBC and Sequoia, TPC-C with RepDB* and TPC-H with ParGRES.

Future Directions

Grid platforms can be considered a natural extension of PC clusters. They are also an alternative of high performance computing with large volumes of data. Several challenges in grid data management are discussed in [9]. An extension of the DBC approach to



Database Clusters. Figure 2. ParGRES DBC – TPC-H query execution times.

grids is proposed [4]. However, communication and data transfer can become a major issue.

Experimental Results

The graphic in Fig. 2 shows query execution time decreasing as more processors are included to process queries from the TPC-H benchmark. Query execution times in the graphic are normalized. These experiments have used a 32 PC cluster from Grid′5000 [2]. The graphic also shows the execution time that should be obtained if linear speedup was achieved. The speedup achieved by ParGRES while processing isolated queries with different number of nodes (from 1 to 32) is superlinear for most queries. A typical OLAP transaction is composed by a sequence of such queries, where one query depends on the result of the previous query. The user has a time frame to take his decisions after running a sequence of queries. Since OLAP queries are time consuming, running eight queries can lead to a four hour elapsed time, according to these tests using one single node for an 11 GB database. These eight queries can have their execution time reduced from four hours of elapsed time to less than one hour, just by using a small four nodes cluster configuration. With 32 nodes these queries are processed in a few minutes.

Data Sets

"TPC BenchmarkTM H – Revision 2.1.0", url: www.tpc.org.

URL to Code

url: cvs.forge.objectweb.org/cgi-bin/viewcvs.cgi/pargres/pargres/

Cross-references

- ▶ Data Partitioning
- ▶ Data Replication
- ▶ Data Warehouse Applications
- ▶ Distributed Database Design
- ▶ Grid File (and family)
- ▶ JDBC
- ▶ ODBC
- ▶ On-line Analytical Processing
- ▶ Parallel Database Systems
- ▶ Parallel Query Processing
- ▶ Storage Area Network

Recommended Reading

1. Akal F., Böhm K., and Schek H.J. OLAP query evaluation in a database cluster: a performance study on intra-query parallelism. In Proc. Sixth East-European Conference on Advances in Databases and Information Systems (ADBIS), 2002, pp. 218–231.
2. Cappello F., Desprez F., and Dayde, M., et al. Grid5000: a large scale and highly reconfigurable grid experimental testbed. In International Workshop on Grid Computing, IEEE 2005, pp. 99–106.
3. Cecchet E. C-JDBC: a middleware framework for database clustering. IEEE Data Eng. Bull., 27:19–26, 2004.
4. Kotowski N., Lima A.A., Pacitti E., Valduriez P., and Mattoso M., Parallel Query Processing for OLAP in Grids. Concurrency and Computation: Practice & Experience, 2008, p. 1303.

5. Lima A.A.B., Mattoso M., and Valduriez P. Adaptive virtual partitioning for OLAP query processing in a database cluster. In Proc. XIX Brazilian Symp. on Database Systems, 2004, pp. 92–105.
6. Mattoso M. et al. ParGRES: a middleware for executing OLAP queries in parallel. COPPE-UFRJ Technical Report, ES-690, 2005.
7. Özsu T. and Valduriez P. Principles of Distributed Database Systems (2nd edn.). Prentice Hall, Englewood Cliffs, NJ, 1999.
8. Pacitti E., Coulon C., Valduriez P., and Özsu M.T. Preventive replication in a database cluster. Distribut. Parallel Databases, 18(3):223–251, 2005.
9. Pacitti E., Valduriez P., and Mattoso M. Grid data management: open problems and new issues. J. Grid Comput., 5(3):273–281, 2007.
10. Röhm U., Böhm K., Scheck H.-J., and Schuldt H. FAS - A freshness-sensitive coordination middleware for a cluster of OLAP components. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 754–768.
11. Sequoia Project, http://sequoia.continuent.org.

Database Connectivity

► Database Adapter and Connector

Database Constraints

► Database Dependencies

Database Dependencies

Marc Gyssens
Hasselt University, Diepenbeek, Belgium

Synonyms

Database constraints; Data dependency

Definition

For a relational database to be valid, it is not sufficient that the various tables of which it is composed conform to the database schema. In addition, the instance must also conform to the intended meaning of the database [15]. While many aspects of this intended meaning are inherently informal, it will generally induce certain formalizable relationships between the data in the database, in the sense that whenever a certain pattern is present among the data, this pattern can either be extended or certain data values must be equal. Such a relationship is called a database dependency. The vast majority of database dependencies in the literature are of the following form [5]:

$$(\forall x_1)\ldots(\forall x_n)\varphi(x_1,\ldots,x_n) \Rightarrow (\exists z_1)\ldots(\exists z_k)\psi(y_1,\ldots,y_m,z_1,\ldots,z_k).$$

Here, $\{y_1,\ldots,y_m\} \subseteq \{x_1,\ldots,x_n\}$, φ is a (possibly empty) conjunction of relation atoms using all the variables $x_1,\ldots,x_n$, and ψ is either a single equality atom involving universally quantified variables only (in which case the dependency is called *equality-generating*); or ψ is a non-empty conjunction of relation atoms involving all the variables $y_1,\ldots,y_m$, $z_1,\ldots,z_k$ (in which case the dependency is called *tuple-generating*. A tuple-generating dependency is called *full* if it has no existential quantifiers: In the other case, it is called *embedded*.

Historical Background

The theory of database dependencies started with the introduction of *functional dependencies* by Codd in his seminal paper [8]. They are a generalization of (super) keys. A relation satisfies a functional dependency $X \to Y$ (where X and Y are sets of attributes) if, whenever two tuples agree on X, they also agree on Y. For example, if in a employee relation of a company database with schema

$$\Omega = \{\text{EMP-NR},\ \text{EMP-NAME},\ \text{JOB},\ \text{SALARY}\},$$

the functional dependencies

$$\begin{aligned}\{\mathit{EMP\text{-}NR}\} &\to \{\mathit{EMP\text{-}NAME}, \mathit{DEPT}, \\ &\qquad \mathit{JOB}, \mathit{SALARY}\}; \\ \{\mathit{DEPT}, \mathit{JOB}\} &\to \{\mathit{SALARY}\}\end{aligned}$$

hold, this means that *EMP-NR* is a key of this relation, i.e., uniquely determines the values of the other attributes, and that *JOB* in combination with *DEPT* uniquely determines *SALARY*.

Codd also noticed that the presence of a functional dependency $X \to Y$ also allowed a lossless decomposition of the relation into its projections onto $X \cup Y$ and $X \cup \overline{Y}$ ($\overline{Y}$ denoting the complement of Y). In the example above, the presence of $\{DEPT, JOB\} \to \{SALARY\}$ allows for the decomposition of the original relation into its projections onto $\{DEPT, JOB, SALARY\}$ and $\{EMP\text{-}NR, EMP\text{-}NAME, DEPT\}$.

Hence, the identification of constraints was not only useful for integrity checking but also for more efficient representation of the data and avoiding update anomalies through redundancy removal.

Subsequent researchers (e.g., [18]) noticed independently that the presence of the functional dependency $X \to Y$ is a sufficient condition for decomposability of the relation into its projection onto $X \cup Y$ and $X \cup \overline{Y}$, but not a necessary one. For example,

Drinker	*Beer*	*Bar*
Jones	*Tuborg*	*Tivoli*
Smith	*Tuborg*	*Far West*
Jones	*Tuborg*	*Tivoli*
Smith	*Tuborg*	*Tivoli*

can be decomposed losslessly into its projections onto $\{DRINKER, BEER\}$ and $\{BEER, BAR\}$, but neither $\{BEER\} \to \{DRINKER\}$ nor $\{BEER\} \to \{BAR\}$ holds. This led to the introduction of the *multivalued dependency*: a relation satisfies the multivalued dependency $X \twoheadrightarrow Y$ exactly when this relation can be decomposed losslessly into its projections onto $X \cup Y$ and $X \cup \overline{Y}$. Fagin [10] also introduced *embedded multivalued dependencies*: A relation satisfies the embedded multivalued dependency $X \twoheadrightarrow Y|Z$ if its projection onto $X \cup Y \cup Z$ can be decomposed losslessly into its projections onto $X \cup Y$ and $X \cup Z$. Sometimes, however, a relation be decomposed losslessly into three or more of its projections but not in two. This led Rissanen [17] to introduce a more general notion: a relation satisfies a *join dependency* $X_1 \bowtie \cdots \bowtie X_k$ if it can be decomposed losslessly into its projections onto $X_1, \ldots, X_k$.

Quite different considerations led to the introduction of *inclusion dependencies* [6], which are based on the concept of referential integrity, already known to the broader database community in the 1970s. As an example, consider a company database in which one relation, *MANAGERS*, contains information on department managers, in particular, *MAN-NAME*, and another, *EMPLOYEES*, contains general information on employees, in particular, *EMP-NAME*. As each manager is also an employee, every value *MAN-NAME* in *MANAGERS* must also occur as a value of *EMP-NAME* in *EMPLOYEES*. This is written as the inclusion dependency $MANAGERS[MAN\text{-}NAME] \subseteq EMPLOYEES[EMP\text{-}NAME]$. More generally, a database satisfies the inclusion dependency $R[A_1, \ldots, A_n] \subseteq S[B_1, \ldots, B_m]$ if the projection of the relation R onto the sequence of attributes $A_1, \ldots, A_n$ is contained in the projection of the relation S onto the sequence of attributes $B_1, \ldots, B_n$.

The proliferation of dependency types motivated researchers to propose subsequent generalizations, eventually leading to the tuple- and equality-generating dependencies of Beeri and Vardi [5] defined higher. For a complete overview, the reader is referred to [14] or the bibliographic sections in [1]. For the sake of completeness, it should also be mentioned that dependency types have been considered that are not captured by the formalism of Beeri and Vardi. An example is the *afunctional dependency* of De Bra and Paredaens (see, e.g., Chap. 5 of [15]).

Foundations

The development of database dependency theory has been driven mainly by two concerns. One of them is solving the inference problem, and, when decidable, developing tools for deciding it. The other is, as pointed out in the historical background, the use of database dependencies to achieve decompositions of the database contributing to more efficient data representation, redundancy removal, and avoiding update anomalies. Each of these concerns is discussed in some more detail below.

Inference

The inference problem is discussed here in the context of tuple- and equality-generating dependencies. The question that must be answered is the following: given a subtype of the tuple- and equality generating dependencies, given as input a set of constraints $\mathcal{C}$ and a single constraint c, both of the given type, is it decidable whether $\mathcal{C}$ logically implies c In other words, is it decidable if each database instance satisfying $\mathcal{C}$ also satisfies c? Given that database dependencies have been defined as first-order sentences, one might be inclined to think that the inference problem is just an instance of the implication problem in mathematical logic. However, for logical implication, one must consider all models of the given database scheme, also those containing infinite relations, while database relations are by definition finite. (In other words, the study of the inference of database dependencies lies within finite model theory.) To separate both notions of inference, a distinction is made between *unrestricted* *implication* (denoted $\mathcal{C} \models c$) and *finite implication* (denoted $\mathcal{C} \models_f c$) [5]. Since unrestricted implication is recursively enumerable and finite implication is co-recursively enumerable, their coincidence yields that the finite implication problem is decidable. The opposite, however, is not true, as is shown by the following counterexample. Consider a database consisting of a single relation R with scheme $\{A, B\}$. Let $\mathcal{C} = \{B \rightarrow A, R[B] \subseteq R[A]\}$ and let c be the inclusion dependency $R[A] \subseteq R[B]$. One can show that $\mathcal{C} \models_f c$, but $\mathcal{C} \not\models c$, as illustrated by the following, necessarily infinite, counterexample:

A	B
0	1
1	2
2	3
3	4
⋮	⋮

As will be pointed out later, the finite implication problem for functional dependencies and so-called *unary* inclusion dependencies (i.e., involving only one attribute in each side) is decidable.

An important tool for deciding (unrestricted) implication is the *chase*. In the chase, a table is created for each relation in the database. For each relation atom in the left-hand side of the dependency c to be inferred, its tuple of variables is inserted in the corresponding table. This set of tables is then *chased* with the dependencies of $\mathcal{C}$: in the case of a tuple-generating dependency, new tuples are added in a minimal way until the dependency is satisfied (in each application, new variables are substituted for existential variables); in the case of an equality-generating dependency, variables are equated until the dependency is satisfied. The result, *chase*($\mathcal{C}$), which may be infinite, can be seen as a model for $\mathcal{C}$. It is the case that $\mathcal{C} \models c$ if and only if the right-hand side of c is subsumed by some tuple of *chase*($\mathcal{C}$) (in the case of a tuple-generating dependency) or the required equality has been applied during the chase procedure.

In the case where only *full* tuple-generating dependencies and equality-generating dependencies are involved, the chase procedure is bound to end, as no existential variables occur in the dependencies, hence no new values are introduced. In particular, the unrestricted implication problems coincides with the finite implication problem, and is therefore decidable. Deciding this inference problem is EXPTIME-complete, however.

The inference problem for all tuple- and equality-generating dependencies is undecidable, however (hence unrestricted and finite implication do not coincide). In 1992, Herrmann [13] solved a longstanding open problem by showing that the finite implication problem is already undecidable for embedded multivalued dependencies.

Another approach towards deciding inference of dependency types is trying to find an *axiomatization*: a finite set of inference rules that is both sound and complete. The existence of such an axiomatization is also a sufficient condition for the decidability of inference. Historically, Armstrong [2] was the first to propose such an axiomatization for functional dependencies. This system of inference rules was eventually extended to a sound and complete axiomatization for functional and multivalued dependencies together [3]:

(F1) $\emptyset \models X \rightarrow Y$ if Y ⊆ X (reflexivity)

(F2) $\{X \rightarrow Y\} \models XZ \rightarrow YZ$ (augmentation)

(F3) $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$ (transitivity)

(M1) $\{X \twoheadrightarrow Y\} \models X \twoheadrightarrow \bar{Y}$ (complementation)

(M2) $\emptyset \models X \twoheadrightarrow Y$ if Y ⊆ X (reflexivity)

(M3) $\{X \twoheadrightarrow Y\} \models XZ \twoheadrightarrow YZ$ (augmentation)

(M4) $\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow Z - Y$ (pseudo−transitivity)

(FM1) $\{X \rightarrow Y\} \models X \twoheadrightarrow Y$ (conversion)

(FM2) $\{X \twoheadrightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z - Y$ (interaction)

Moreover, (F1)–(F3) are sound and complete for the inference of functional dependencies alone, and (M1)–(M4) are sound and complete form the inference of multivalued dependencies alone. The above axiomatization is at the basis of an algorithm to decide inference of functional and multivalued dependencies in low polynomial time.

Of course, the inference problem for join dependencies is also decidable, as they are full tuple-generating dependencies. However, there does not exist a sound and complete axiomatization for the inference of join dependencies [16], even though there does exist an axiomatization for a larger class of database dependencies.

There also exists a sound and complete axiomatization for inclusion dependencies [6]:

(I1) $\emptyset \models R[X] \subseteq R[X]$ (reflexivity)

(I2) $\{R[A_1, \dots, A_m] \subseteq S[B_1, \dots, B_m]\} \models R[A_{i1}, \dots A_{ik}] \subseteq S[B_{i1}, \dots, B_{ik}]$
if $i_1, \dots, i_k$ is a sequence of integers in $\{1, \dots, m\}$ (projection)

(I3) $\{R[X] \subseteq S[Y], S[Y] \subseteq T[Z]\} R[X] \subseteq T[Z]$ (transitivity)

Above, X, Y, and Z represent sequences rather than sets of attributes.

Consequently, the implication problem for inclusion dependencies is decidable, even though inclusion dependencies are embedded tuple-generating dependencies. However, deciding implication of inclusion dependencies is PSPACE-complete.

It has already been observed above that the unrestricted and finite implication problems for functional dependencies and unary inclusion dependencies taken together do no coincide. Nevertheless, the finite implication problem for this class of dependencies is decidable. Unfortunately, the finite implication problem for functional dependencies and general inclusion dependencies taken together is undecidable (e.g., [7]).

Decompositions

As researchers realized that the presence of functional dependencies yields the possibility to decompose the database, the question arose as to how far this decomposition process ought to be taken. This led Codd in follow-up papers to [8] to introduce several *normal forms*, the most ambitious of which is *Boyce-Codd Normal Form (BCNF)*. A database is in BCNF if, whenever one of its relations satisfies a nontrivial functional dependency $X \rightarrow Y$ (i.e., where Y is not a subset of X), X must be a superkey of the relation (i.e., the functional dependency $X \rightarrow U$ holds, where U is the set of all attributes of that relation). There exist algorithms that construct a lossless BCNF decomposition for a given relation. Unfortunately, it is not guaranteed that such a decomposition is also dependency-preserving, in the following sense: the set of functional dependencies that hold in the relations of the decomposition and that can be inferred from the given functional dependencies is in general not equivalent with the set of the given functional dependencies. Even worse, a dependency-preserving BCNF decomposition of a given relation does not always exist. For that reason, *Third Normal Form (3NF)*, historically a precursor to BCNF, is also still considered. A datatabase is in 3NF if, whenever one of its relations satisfies a nontrivial functional dependency $X \rightarrow \{A\}$ (A being a single attribute), the relation must have a minimal key containing A. Every database in BCNF is also in 3NF, but not the other way around. However, there exists an algorithm that, given a relation, produces a dependency-perserving lossless decomposition in 3NF. Several other normal forms have also been considered, taking into account multivalued dependencies or join dependencies besides functional dependencies.

However, one can argue that, by giving a join dependency, one actually already specifies how one wants to decompose a database. If one stores this decomposed database rather than the original one, the focus shifts from integrity checking to *consistency checking*: can the various relations of the decompositions be interpreted as the projections of a universal relation? Unfortunately, consistency checking is in general exponential in the number of relations. Therefore, a lot of attention has been given to so-called *acyclic join dependencies* [4]. There are many equivalent definitions of this notion, one of which is that an acyclic join dependency is equivalent to a set of multivalued dependencies. Also, global consistency of a decomposition is already implied by pairwise consistency if and only if the join dependency defining the decomposition is acyclic, which explains in part the desirability of acyclicity. Gyssens [12] generalized the notion of acyclicity to k-cyclicity, where acyclicity corresponds with the case $k = 2$. A join dependency is k-cyclic if it is equivalent to a set of join dependencies each of which has at most k components. Also, global consistency of a decomposition is already implied by k-wise consistency if and only if the join dependency defining the decomposition is k-cyclic.

Key Applications

Despite the explosion of dependency types during the latter half of the 1970s, one must realize that the dependency types most used in practice are still functional dependencies (in particular, key dependencies) and inclusion dependencies. It is therefore unfortunate that the inference problem for functional and inclusion dependencies combined is undecidable.

At a more theoretical level, the success of studying database constraints from a logical point view and the awareness that is important to distinguish between unrestricted and finite implication certainly contributed to the interest in and study and further development of finite model theory by theoretical computer scientists.

Finally, decompositions of join dependencies led to a theory of decompositions for underlying hypergraphs, which found applications in other areas as well, notably in artificial intelligence (e.g., [9,11]).

Cross-references

► Boyce-Codd Normal Form (BCNF)
► Chase
► Equality-Generating Dependencies
► Fourth Normal Form (4NF)
► Functional Dependency
► Implication of Constraints
► Inconsistent Databases
► Join Dependency
► Multivalued Dependency
► Normal Forms and Normalization
► Relational Model
► Second Normal Form (2NF)
► Third Normal Form (3NF)
► Tuple-Generating Dependencies

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of databases. Addison-Wesley, Reading, Mass., 1995. (Part C).
2. Armstrong W.W. Dependency structures of data base relationships. In Proc. IFIP Congress 74. J.L. Rosenfeld (ed). North-Holland, 1974, pp. 580–583.
3. Beeri C., Fagin R., and Howard J.H. A complete axiomatization for functional and multivalued dependencies. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1978, pp. 47–61.
4. Beeri C., Fagin R., Maier D., and Yannakakis M. On the desirability of acyclic database schemes. J. ACM, 30 (3):479–513, 1983.
5. Beeri C. and Vardi M.Y. The implication problem for data dependencies. In Proc. Int. Conf. on Algorithms, Languages, and Programming, 1981. In Even Kariv Lect. Notes Comput. Sci., 115:73–85, Springer, 1981.
6. Casanova M.A., Fagin R., and Papadimitriou C.H. Inclusion dependencies and their interaction with functional dependencies. J. Comput. Syst. Sci., 28(1):29–59, 1984.
7. Chandra A.K. and Vardi M.Y. The implication problem for functional and inclusion dependencies is undecidable. SIAM J. Comput., 14(3):671–677, 1985.
8. Codd E.F. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387, 1970.
9. Cohen D.A., Jeavons P., and Gyssens M. A unified theory of structural tractability for constraint satisfaction problems. J. Comput. Syst. Sci., 74(5):721–743, 2008.
10. Fagin R. Multivalued dependencies and a new normal form for relational databases. ACM Trans. Database Syst., 2(3):262–278, 1977.
11. Gottlob G., Miklós Z., and Schwentick T. Generalized hypertree decompositions: NP-hardness and tractable variants. In Proc. 26th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2007, pp. 13–22.
12. Gyssens M. On the complexity of join dependencies. Trans. Database Syst., 11(1):81–108, 1986.
13. Herrmann C. On the undecidability of implications between embedded multivalued dependencies. Inform. Comput., 122 (2):221–235, 1995.
14. Kanellakis P.C. Elements of relational database theory. In: Van Leeuwen J. (ed.). Handbook of theoretical computer science, Elsevier, 1991, pp. 1074–1156.
15. Paredaens J., De Bra P., Gyssens M., and Van Gucht D. The structure of the relational database model. In EATCS Monographs on Theoretical Computer Science, Vol. 17. Brauer W., Rozenberg G., and Salomaa A., (eds.). Springer, 1989.
16. Petrov S.V. Finite axiomatization of languages for representation of system properties. Inform. Sci., 47(3):339–372, 1989.
17. Rissanen J. Independent components of relations. ACM Trans. Database Syst., 2(4):317–325, 1977.
18. Zaniolo C. Analysis and design opf relational schemata for database systems. Ph. D. thesis, University of California at Los Angeles, 1976. Technical Report UCLA-Eng-7669.

Database Design

John Mylopoulos
University of Trento, Trento, Italy

Definition

Database design is a process that produces a series of database schemas for a particular application. The schemas produced usually include a conceptual, logical and physical schema. Each of these is defined using a different data model. A conceptual or semantic data model is used to define the conceptual schema, while a logical data model is used for the logical schema. A physical schema is obtained from a logical schema by deciding what indexes and clustering to use, given a logical schema and an expected workload for the database under design.

Key Points

For every existing database, there is a design team and a design process that produced it. That process can make or break a database, as it determines what information it will contain and how will this information be structured.

The database design process produces a conceptual, a logical and a physical database schema. These schemas describe the contents of a database at different levels of abstraction. The conceptual schema focuses on the entities and relationships about which information is to be contained in the database. The Entity-Relationship Model is the standard model for defining conceptual schemas, though there have been many other proposals. UML class diagrams can also be used for this design phase. The logical schema describes the logical structure of the database. The Relational Model is the standard model for this phase, which views a database as a collection of tables. Alternative data models include the Hierarchical and the Network Data Models, but also object-oriented data models that view a database as a collection of inter-related objects instantiating a collection of classes.

The need to create different schemas that describe the contents of a database at different levels of abstraction was noted as far back as 1975 in a report by the American National Standards Institute (ANSI) [1], but has also evolved since. The report proposed a three-level architecture consisting of several external schemas representing alternative user views of a database, a conceptual schema whose information content subsumed that of external schemas, and an internal schema that represented database content in terms of a particular database technology (such as a relational Database Management System). For database design purposes, conceptual schemas have to be built up-front, whereas external schemas can be created dynamically according to user needs. Moreover, the notion of an internal schema has been refined to that of a logical and a physical schema.

The database design process often consists of four phases: requirements elicitation, conceptual schema design, logical schema design, and physical schema design. Requirements elicitation gathers information about the contents of the database to be designed from those who have a stake (a.k.a. *stakeholders*) This information is often expressed in natural language and may be ambiguous and/or contradictory. For example, two stakeholders may differ on what information about customers or patients is useful and should be included in the database-to-be. A conceptual schema is extracted from a given set of requirements through a series of steps that focus on noun phrases to identify entities, verb phrases to identify important relationships among entities, and other grammatical constructions to identify attributes about which information is useful to include in the database.

A conceptual schema is then transformed to a logical one through a series of well-defined transformations that map collections of entities and relationships into a relation whose attributes and keys are determined by the source entities and relationships. The logical schema design phase often includes a normalization step where an initial logical schema with associated functional dependencies is transformed into a normalized schema using one of several well-studied normal forms.

Physical schema design starts with a logical schema and determines the index to be used for each relation in the logical schema. This decision is based on the expected workload for the database-to-be, defined by the set of most important queries and updates that will be evaluated against the database. In addition, physical design determines the clustering of tuples in physical storage. This clustering plays an important role in the performance of the system as it evaluates queries that return many tuples (for example, queries that include joins). Physical schema design may dictate the revision of the logical schema by splitting/merging relations to improve performance. This step is known as *denormalization.*

As suggested by denormalization, the database design process should not be viewed as a sequential process that begins with requirements elicitation and proceeds to generate a conceptual, logical and physical schema in that order. Rather, the process consists of four linearly ordered phases and is iterative: after completing any one phase, the designer may return to earlier ones to revise the schemas that have been produced so far, and even the requirements that have been gathered.

Cross-references

► Conceptual Data Models
► Normalization Theory
► Physical Database Design for Relational Databases
► Semantic Data Models

Recommended Reading

1. American National Standards Institute. Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems. FDT – Bull. ACM SIGMOD, 7(2):1–140, 1975.

2. Atzeni P., Ceri S., Paraboschi S., and Torlone R. Database Systems: Concepts, Languages and Architectures. McGraw Hill, New York, 1999.

Database Design Recovery

► Database Reverse Engineering

Database Engine

► Query Processor

Database Implementation

► Physical Database Design for Relational Databases

Database Interaction

► Session

Database Languages for Sensor Networks

Samuel Madden
Massachusetts Institute of Technology, Cambridge, MA, USA

Synonyms

Acquisitional query languages; TinySQL

Definition

Sensor networks – collections of small, inexpensive battery-powered, wirelessly networked devices equipped with sensors (microphones, temperature sensors, etc.) – offer the potential to monitor the world with unprecedented fidelity. Deploying software for these networks, however, is difficult, as they are complex, distributed, and failure prone. To address these complexities, several sensor network database systems, including TinyDB [7], Cougar [12], and SwissQM [8] have been proposed. These systems provide a high level SQL-like query language that allows users to specify what data they would like to capture from the network and how they would like that data processed without worrying about low-level details such power management, network formation, and time synchronization. This entry discusses the main features of these languages, and their relationship to SQL and other database languages.

Historical Background

Cougar and TinyDB were the first sensor network databases with the bulk of their development occurring between 1999 and 2003. They emerged as a result of rising interest in wireless sensor networks and other tiny, embedded, battery powered computers. TinyDB was co-developed as a part of the TinyOS operating system [2] for Berkeley Mote-based sensor networks. Initial versions of the motes used Atmel 8-bit microprocessors and 40 kbit s^{-1} radios; newer generations, developed by companies like Crossbow Technologies (http://www.xbow.com) and Moteiv Technologies (http://www.moteiv.com) use Zigbee (802.15.4) radios running at 250 kbit s^{-1} and Atmel or Texas Instruments 8 or 16 bit microprocessors running at 4–8 MHz. Nodes typically are very memory constrained (with 4–10 Kbytes of RAM and 48–128 Kbytes of non-volatile flash-based program memory.) Most nodes can be interfaced to sensors that can capture a variety of readings, including light, temperature, humidity, vibration, acceleration, sounds, or images. The limited processing power and radio bandwidth of these devices constrains sample rates to at most a few kilosamples/s. Using such tiny devices does allow power consumption to be quite low, especially when sample rates are kept down; for example, networks that sample about once a second from each node can provide lifetimes of a month or longer on coin-cell batteries or a year or more on a pair of AA batteries [4].

The promise of sensor network databases is that they provide a very simple way to accomplish one of the most common goals of sensor networks: data collection. Using a simple, high level declarative language, users specify what data they want and how fast they want it. The challenge of building a sensor network database lies in capturing the required data in a power-efficient and reliable manner. The choice of programming language for these systems – the main topic of this entry – is essential to meeting that challenge. The language must be expressive enough to allow users to get the data they want, but also implementable in a way that is power-efficient, so that the network lasts as long as possible.

To understand how sensor network querying works, it is important to understand how sensor network databases are used. The typical usage model is as follows: a collection of static sensor nodes is placed in some remote location; each node is pre-programmed with the database software. These nodes report data wirelessly (often over multiple radio hops) to a nearby "basestation" – typically a laptop-class device with an Internet connection, which then relays data to a server where data is stored, visualized, and browsed.

Users interact with the system by issuing queries at the basestation, which in turn broadcasts queries out into the network. Queries are typically disseminated via flooding, or perhaps using some more clever gossip based dissemination scheme (e.g., Trickle [3]). As nodes receive the query, they begin processing it. The basic programming model is data-parallel: each node runs the same query over data that it locally produces or receives from its neighbors. As nodes produce query results, they send them towards the basestation.

When a node has some data to transmit, it relays it to the basestation using a so-called *tree-based routing protocol*. These protocols cause the nodes to arrange themselves into a tree rooted at the basestation. This tree is formed by having the basestation periodically broadcast a beacon message. Nodes that hear this beacon re-broadcast it, indicating that they are one hop from the basestation; nodes that hear those messages in turn re-broadcast them, indicating that they are two hops from the basestation, and so on. This process of (re)broadcasting beacons occurs continuously, such that (as long as the network is connected) all nodes will eventually hear a beacon message. When a node hears a beacon message, it chooses a node from which it heard the message to be its *parent*, sending messages through that parent when it needs to transmit data to the basestation (In general, parent selection is quite complicated, as a node may hear beacons from several candidate parents. Early papers by Woo and Culler [11] and DeCouto et al. [1] provide details.). Note that this ad hoc tree-based network topology is significantly different than the any-to-any routing networks that are used in traditional parallel and distributed database systems. As discussed below, this imposes certain limitations on the types of queries that are feasible to express efficiently in sensor network database systems.

Foundations

Most sensor network databases systems provide a SQL-like query interface. TinySQL, the query language used in TinyDB, for example, allows users to specify queries (through a GUI or command line interface) that appear as follows:

```
SELECT <select list>
  FROM <table list>
  WHERE <condition list>
  GROUP BY <field list>
  HAVING <condition list>
  SAMPLE PERIOD <duration>
   <additional clauses>
```

Most of these clauses behave just as they do in standard SQL. There are a few differences, however. First, the SAMPLE PERIOD clause requests that a data reading be produced once every <*duration*> seconds. This means that unlike most databases, where each query produces a single result set, in most sensor databases, each query produces a continuous stream of results. For example, to request the temperature from every node in a sensor network whose value is greater than 25°C once per sec, a user would write:

```
SELECT nodeid, temperature
  FROM sensors
  WHERE temperature > 25°C
  SAMPLE PERIOD 1s
```

Besides the continuous nature of sensor network query languages, this example illustrates that the data model provided by these systems is also somewhat unusual. First, the nodeid attribute is a unique identifier assigned to each sensor node and available in every query. Second, the table sensors is *virtual table* of sensor readings. Here, *virtual* means that it conceptually contains one row for every sensor type (light, temperature, etc.) from every sensor node at every possible instant, but all of those rows and columns are not actually materialized. Instead, only the sensor readings needed to answer a particular query are actually generated. The research literature refers to such languages as *acquisitional*, because they specify the rate and location where data should be acquired by the network, rather that simply querying a pre-existing table of data stored in the memory of the device [6].

Note also that although this table appears to be a single logical table its rows are actually produced by different, physically disjoint sensors. This suggests that some elements of query processing may be done inside of the network, before nodes transmit their data. For example, in the TAG system [5] a method for efficiently computing aggregates inside of a sensor network was proposed.

Restricted Expressiveness

It is important to note that sensor network query languages are less expressive than more general languages like SQL. By restricting expressiveness of their query languages, sensor network databases are able to ensure reasonably efficient query execution. For example, the TinyDB system imposes the following restrictions on queries:

- Arbitrary self-joins with the `sensors` table are not allowed. Such joins would require propagation of the entire table to all other nodes in the system. Queries with an equality predicate on `nodeid` can be evaluated efficiently and may be allowed.
- Nested queries are not allowed. Such queries potentially require disseminating query state throughout the network. For example, the query:

```
SELECT nodeid, temp
  FROM sensors
  WHERE temp >
  (SELECT AVG(temp)
  FROM sensors)
  SAMPLE PERIOD 1s
```

requires disseminating the average throughout the network in order to compute the query in a distributed fashion. Centralized implementations – where all of the data is sent to the basestation – are likely the only feasible implementation but can be quite inefficient due to the large amount of required data transmission.

Not all nested queries are inefficient to implement. For example, queries that compute aggregates over local state and then compute global aggregates over those local values (e.g., the average of the last five minutes temperatures at each node) have a natural distributed implementation. To avoid a confusing language interface where some nested queries are allowed and some are not, the designers of TinyDB chose to support certain classes of nested queries through two additional syntactic clauses: *temporal aggregates* and the *storage points*.

- *Temporal aggregates* allow users to combine a series of readings collected on a single node over time. For example, in a building monitoring system for conference rooms, users may detect occupancy by measuring the maximum sound volume over time and reporting that volume periodically; this could be done with the following query:

```
SELECT nodeid, WINAVG(volume,
30s, 5s)
  FROM sensors
  GROUP BY nodeid
  SAMPLE PERIOD 1s
```

This query will report the average volume from each sensor over the last 30 seconds once every 5 seconds, sampling once per second. The `WINAVG` aggregate is an example of a *sliding-window* operator. The final two parameters represent the window size, in seconds, and the sliding distance, in seconds, respectively.

- *Storage points* add a simple windowing mechanism to TinyDB that can be used to compute certain classes of locally nested queries. A storage point simply defines fixed-size materialization buffer of local sensor readings that additional queries can be posed over. Consider, as an example:

```
CREATE
  STORAGE POINT recentLight SIZE
  8 seconds
  AS (SELECT nodeid, light FROM
  sensors
  SAMPLE PERIOD 1s)
```

This statement provides a shared, local (i.e., single-node) location called `recentLight` to store a streaming view of recent data.

Users may then issue queries which insert into or read from storage points, for example, to insert in the `recentLight` storage point a user would write:

```
SELECT nodeid, light
  INTO recentLight
  SAMPLE PERIOD 1s
```

And to read from it, he might write:

```
SELECT AVG(light)
  FROM recentLight
  SAMPLE PERIOD 5s
```

Joins are also allowed between two storage points on the same node, or between a storage point and the `sensors` relation. When a `sensors` tuple arrives, it is joined with tuples in the storage point at its time of arrival.

The SwissQM [8] system does allow some forms of nested queries to be specified, but like TinyDB's STORAGE POINT syntax, these queries can operate only on a node's local state. For example, the internal query can compute each node's minimum temperature over the last 5 minutes, and then a global aggregate query can be used to compute the global minimum temperature over all nodes.

Specialized Language Constructs

Sensor network query languages usually include a number of specialized features designed to allow them to take advantage of the special hardware available on the sensor nodes they run on. For example, a user may wish to actuate some piece of attached hardware in response to a query. In TinyDB queries may specify an OUTPUT ACTION that will be executed when a tuple satisfying the query is produced. This action can take the form of a low-level operating system command (such as "Turn on the red LED"), or the instantiation of another query. For example, the query:

```
SELECT nodeid, temp
  WHERE temp > 100˚ F
  OUTPUT ACTION alarm()
  SAMPLE PERIOD 1 minute
```

will execute the command `alarm()` whenever a tuple satisfying this query is produced. This command is an arbitrary piece of C code that is written by the user and stored in a system catalog.

A related feature that is important in sensor networks is *event handling*. The idea is to initiate data collection when a particular external event occurs – this event may be outside of the database system (for example, when a switch on the physical device is triggered.) Events are important because they allow the system to be dormant until some external condition occurs, instead of continually polling or blocking, waiting for data to arrive. Since most microprocessors include external interrupt lines than can wake a sleeping device to begin processing, efficient implementations of event processing are often possible.

As an example, the TinyDB query:

```
ON EVENT switch-pressed(loc):
  SELECT AVG(light), AVG(temp),
  event.loc
  FROM sensors AS s
  WHERE dist(s.loc, event.loc) < 10m
  SAMPLE PERIOD 2 s FOR 30 s
```

reports the average light and temperature level at sensors when a switch on the device is pressed. Every time a `switch-pressed` event occurs, the query is issued from the detecting node and the average light and temperature are collected from nearby nodes once every 2 seconds for 30 seconds. The `switch-pressed` event is signaled to TinyDB by a low piece of driver-like C code that interfaces to the physical hardware.

Key Applications

Sensor network query languages have applications in any wireless sensing domain. They are particularly designed for environments where relatively simple programs that capture and process data are needed. Systems that implement such languages are often comparably efficient to hand-coded programs that require hundreds of times more code to implement.

Applications where sensor network databases have been deployed to good effect include environmental [10] and industrial [9] monitoring.

Cross-references

► In-Network Query Processing
► SQL
► Stream Processing

Recommended Reading

1. Couto D.S.J.D., Aguayo D., Bicket J., and Morris R. A high-throughput path metric for multi-hop wireless routing. In Proc. 9th Annual Int. Conf. on Mobile Computing and Networking, 2003.
2. Hill J., Szewczyk R., Woo A., Hollar S., Culler D., and Pister K. System architecture directions for networked sensors. In Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 2000.
3. Levis P., Patel N., Culler D., and Shenker S. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor network. In Proc. 1st USENIX Symp. on Networked Systems Design & Implementation, 2004.
4. Madden S. The Design and Evaluation of a Query Processing Architecture for Sensor Networks. Ph.D. thesis, UC Berkeley, 2003.
5. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002.
6. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. The design of an acquisitional query processor for sensor networks.

In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
7. Madden S., Hong W., Hellerstein J.M., and Franklin M. TinyDB Web Page.
8. Müller R., Alonso G., and Kossmann D. SwissQM: next generation data processing in sensor networks. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 1–9.
9. Stoianov I., Nachman L., Madden S., and Tokmouline T. PIPENET: a wireless sensor network for pipeline monitoring. In Proc. 6th Int. Symp. Inf. Proc. in Sensor Networks, 2007, pp. 264–273.
10. Tolle G., Polastre J., Szewczyk R., Culler D.E., Turner N., Tu K., Burgess S., Dawson T., Buonadonna P., Gay D., and Hong W. A macroscope in the redwoods. In Proc. 3rd Int. Conf. on Embedded Networked Sensor Systems, 2005, pp. 51–63.
11. Woo A., Tong T., and Culler D. Taming the underlying challenges of reliable multihop routing in sensor networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003.
12. Yao Y. and Gehrke J. Query processing in sensor networks. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.

Database Machine

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Definition

A database machine is a computer system which has special hardware designed and/or tuned for database accesses. Database machines may sometimes be coupled with a frontend server and, in this case, the database machines are called backend processors.

Key Points

The basic idea behind database machines was to put database computation closer to disk drives so as to achieve significant performance improvements. Database machines were actively studied in the 1970's and 1980's.

Early researchers explored filter processors which could efficiently examine data transferred from disk drives to a frontend server. Filter processors were categorized into four groups by D. DeWitt et al. [2]. A Processor-per-Track (PPT) machine is a system which consists of a number of cells (a set of tracks) and cell processors. As the data track rotates, the cell processor can scan the data and process search operations on the fly. A Processor-per-Head (PPH) machine is a system in which a processor is coupled with each head. Data is transferred in parallel from a set of heads and then processed in a set of processors. Thus, a whole cylinder is searched in a single rotation. In contrast to the PPT and PPH machines which need special disk hardware, a Processor-per-Disk (PPD) machine couples a processor with each disk drive. PPD can be seen as a compromising design, which has less performance advantage but can be realized at lower cost. A Multi-Processor Cache (MPC) machine is a system which couples multiple processors and multiple disk cache modules with each disk drive. The cache space is used for the processors to exchange the ability of selection operation.

When it came to the 1980s, researches of database machines were shifted to massive parallel computing. General-purpose processors and disk drives were tightly coupled into a node, and such nodes were then combined by a high-speed interconnect. Some of these types of database machines attained much success in the industry.

Cross-references

► Active Storage
► Intelligent Storage Systems

Recommended Reading

1. DeWitt D.J. and Gray J. Parallel database systems: The future of high performance database systems. Comm. ACM, 36(6):85–98, 1992.
2. DeWitt D.J. and Hawthorn P.B. A performance evaluation of data base machine architectures. In Proc. Very Large Data Base (VLDB). 1981, pp. 199–214.
3. Hurson A.R., Miller L.L., and Pakzad S.H. Parallel architectures for database systems. IEEE CS Press, 1989.

Database Management System

Pat Helland
Microsoft Corporation, Redmond, WA, USA

Synonyms

DBMS

Definition

A database management system is a software-based system to provide application access to data in a controlled and managed fashion. By allowing separate definition of the structure of the data, the database

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:29

management system frees the application from many of the onerous details in the care and feeding of its data.

Historical Background

The first general purpose database systems emerged in the 1960s and represented their data in the *network data model* which presumes that portions of the data (typically called *records*) would explicitly reference any related data items in a graph (or network). The model was standardized as the CODASYL (Conference on Data Systems Languages) model and remained a strong influence on database systems into the 1980s.

In the late 1960s, the *hierarchical data model* emerged as exemplified by IBM's IMS (Information Management System). In the hierarchical data model, data was oriented in a hierarchy and the most natural form of navigation was via a parent-child relationship.

Starting in the 1970s, the *relational data model* emerged based on theoretical work done by Ted Codd and the System R research project at IBM's San Jose Research Lab. Out of System R emerged the SQL language. Concurrent with the System R effort, the Ingres project at UC Berkeley contributed strongly to both the development of the concepts and implementation techniques of the relational data model.

In the relational data model, data is represented in tables which can be related to each other by value in an operation called a join. Seminal to the relational data model is the absence of any explicit navigational links in the data. Relational systems continue to dominate what most people think of as a database system.

Other systems such as Object-Relational (O-R) systems and Entity-Relationship (E-R) systems either augment or compete with pure relational systems by formalizing the representation of relationships and objects (or entities) in the abstractions formally managed by the system. Industry opinions vary as to whether the O-R and E-R functionality should be included in the database management system itself or provided by the application layered on top of the database management system.

The notion of a database management system is under pressure to evolve in many different ways:

- In the 1980s, stored procedures and triggers were introduced into some system allowing the execution of application logic within the database itself. Stored procedures, combined with referential integrity and other declarative forms of business rules, pushed portions of the application into the database management system itself, blurring the traditional delineation between "app and data."
- As the single mainframe evolved into distributed systems, databases evolved to span many computers while attempting to provide the same behavior (on a larger scale) as provided by the centralized system. As the scope of the systems grows to thousands of machines, the semantics of the access to the data can no longer be identical to the smaller systems. Hence, the meaning of access to data is evolving.
- As huge numbers of devices and sources of data have arrived, it is no longer always enough to consider data as a passive collection. Consequently, innovations are seen in streaming databases wherein questions are posed about data which has not yet been completely gathered.
- With the arrival of the Internet, data is frequently sent outside of the database and then is returned back into it (potentially with changes). Database management systems have been designed to have complete control over their data and the effect of the Internet and other widely distributed systems pose challenges.
- As multiple applications (and their data) are independently created and then brought together in an increasingly connected world, the concept of a centralized definition of the data is under increasing pressure and forcing new innovation.

Database management systems focus on the data as separated from the application. While the concepts and implementations have evolved, the emphasis on data has remained at the center.

Foundations

The basic charter of a database management system is to focus on the separate management of data to reduce the costs and increase the functionality. Key to database management systems is the creation of higher level abstractions around how the application is separated from the data.

Today's database management systems are dominated by the relational data model. With the relational model, the high level abstraction is expressed as the DDL (Data Definition Language) which defines the schema for the data. In DDL, the data is laid out in tables and rows. Above the DDL abstraction, the application can manipulate the data. Below the DDL abstraction, the database management system can handle the storage, protection, high-performance reading and writing, and many other services. In so many ways, it is the existence of the DDL abstraction layer that is the essence of a database management system.

The high-level abstractions that separate the data from the application allows for a number of valuable characteristics:

- *Independent Evolution*

With the existence of the schema, the application can evolve based on the schema itself. Indeed, in most modern database management systems, the schema itself can be evolved while maintaining continuous support for existing applications.

In addition, the schema as seen by the application (the conceptual or logical schema) is typically separated from the physical schema which defines the actual layout of the storage and its associated indices. Leveraging the separation provided by the schema, the actual representation of the data may evolve to respond to changes in technology.

An essential part of a database management system is the protection of the investment in the application (which is typically very large) to allow for changes in technology. It is the higher level abstraction captured in the schema (via the DDL definition) that enable the protection of the investment in the application.

- *Multiple Applications Sharing the Same Data*

As the data is represented in a fashion based on its underlying semantics, it is possible to write new applications that modify the same shared data. By capturing the high-level conceptual schema, the underlying access to the physical storage is managed by the database management system. The combination of clearly separated meaning and the delegation of physical access to the intermediary provided by the database management system allows for new applications to be written against the same data.

- *Ad-Hoc Access to Data*

An important usage of database management systems has emerged in the form of *business intelligence*. Users are allowed to directly query and/or modify the contents of the data using direct access to the database management system itself. Ad-hoc access to data is made possible by the existence of the higher-level abstraction of the conceptual schema which describes the data independently of its physical schema and of its applications.

Business intelligence has, on its own, grown to a multi-billion dollar industry. Many enterprises find that the knowledge extracted from rapid and ad-hoc queries against their data can dramatically influence the business.

Essential to providing these abstractions are three application visible mechanisms, schema definition, data manipulation language (DML), and transactions.

- *Schema Definition*

Schema definition is typically done at two levels, the conceptual (or logical) schema and the physical schema.

The conceptual schema definition is the expression of the shape and form of the data as viewed by the application. In relational database management systems, the conceptual schema is described as tables, rows, and columns. Each column has a name and a data type. A collection of columns comprise a row, and a collection of rows, a table. The goal of the conceptual schema is to express the data organization as concisely as possible with each piece of knowledge represented in a single way.

The physical schema definition maps the conceptual schema into the underlying access methods which store the data onto disk. Both the storage of the underlying records and the capturing of various indices used for locating the records must be declared in the physical schema.

- *Data Manipulation*

Data manipulation refers to the mechanism by which the application extracts data from the database and changes data within the database. In relational systems, data manipulation is expressed as set oriented operations known as queries or updates. The SQL language, originally defined as a part of IBM's System R project in the late 1970s, has emerged as an ANSI standard and is, by far, the most commonly used DML (Data Manipulation Language) in modern relational database management systems.

The means for expressing access to and manipulation of data is one of the most important interfaces in computing. As innovations in data representation (such as XML) arrive, there are frequent debates about how to codify the access to those representations. There are competing forces in that the preservation of the existing interfaces is essential to the industry-wide investment in applications and in programmer expertise. Applying pressure to that is the desire to innovate in the representation and usage of data as system evolve. The evolution of database management systems is currently driven by the pressures of the blurring of data versus application, Internet scale distribution, streaming databases, and independently defined systems coming together.

The data manipulation portion of database management systems is a vibrant and lively space for innovation both in academia and industry.

- *Transactions*

The ability to process data using relations within the relational model of database management is dependent on *transactions* to combine the data in a meaningful and predictable way.

The transaction is a unit of work with four properties whose first letters spell ACID:

Atomic – The changes made within the transaction are all or nothing (even when failures occur).
Consistent – Rules enforced within the database (e.g. don't lose money in a bank) remain in effect when many transaction are concurrently executing.
Isolated – An ongoing transaction cannot see the effects of any other ongoing transaction.
Durable – When a transaction is committed, it remains committed.

It is the combination of Atomic, Isolated, and Durable which, when provided by the database management system, allow the application to provide its own notion of consistency.

Underlying these application visible mechanisms lays a lot of technology. Consider, in turn, query processing, access methods, and concurrency control and recovery.

- *Query Processing*

Since the advent of relational systems, set oriented expressions of data and updates to data have proven to be extraordinarily powerful. Application developers are given mechanisms for describing large amounts of data to be examined or modified and the system determines the best approach to accomplishing the intent expressed by the application. SQL DML allows for these powerful queries and updates as do some newly arriving other representations such as XQuery. There is an entire discipline within database management systems called *query processing* which applies set oriented operations to the underlying data and leverages whatever optimizations are appropriate to efficiently perform the application's work.

Efficient query processing must consider many factors over and above the expressed desires of the application. The available indices on the data, potential distribution of the data, the expected result sizes of the various sets of data that must be created as intermediaries, the performance characteristics of the processors, disks, networks, and remote computers all play a role in deciding a strategy by which the work will be accomplished.

One of the most important aspects to modern query processing is that these performance concerns are removed from the application programmer in most cases. Separating performance concerns from the application program's intent allows for an investment in applications which can survive many changes in the machines and data set sizes. The query processor may change its optimization strategies based upon new knowledge while the application program remains intact. While the ideal of performance independence is not completely realized in complex cases, it is true for a large number of applications.

Similar to the pressures on DML, query processing remains a vibrant discipline in a changing world.

- *Access Methods*

Access methods provide mechanisms for storing and retrieving the data. The dominant semantics for access methods is called a key-value pair. A key-value pair provides the ability to insert a blob of data associated with a unique key and then subsequently retrieve that blob. Some access methods allow searching for adjacent keys within a sort order of the keys. Other access methods only allow reading and writing based exclusively on exact matches of the key.

Much of the work in the 1970s and early 1980s in access method was dominated by methods for rapidly determining the disk address for the data. Initially, the records in the network and hierarchical schemes included direct disk addresses and it was straightforward for the system to retrieve the record. Soon, hashing schemes were employed wherein a single primary key, not a direct pointer, could be used locate a bucket of records and do so with high probability of accessing the record with a single disk operation. Separate indices where needed to find records based on nonprimary key values.

Originally introduced in 1971 by Rudolf Bayer and Ed McCreight, B-Trees have emerged as the standard mechanism for self-organizing access methods. A B-Tree keeps an ordered list of keys in a balanced fashion which ensures a fixed depth from the base of the tree to the root even in the face of tremendous changes and churn. Since the mid-1990s, most modern database systems use a variant called a B+Tree in which the payload (the blob described above in the key-blob pair) is always kept in the collection of leaf nodes and a balanced structure of keys and pointers to other pages in the B+tree is kept in the non-leaf pages of the B+Tree.

Access methods (intertwined with the Concurrency Control and Recovery mechanism described below) are responsible for managing the storage of data within DRAM and when that data must be written out to disk. As the sizes of DRAM have increased more rapidly than most databases, there is an increasing trend towards in-memory databases. Indeed, the tipping point towards B+Trees as the dominant form of access method occurred when DRAM memories became large enough that the upper levels of the tree were essentially guaranteed to be in memory. That meant climbing the B+Tree did not require extra disk I/Os as it did when used with a smaller memory footprint.

B+Trees offer exceptional advantages for database management systems. In addition to access times that are uniform due to the uniform depth of the tree, each operation against the tree can be performed in a bounded (functional to O(log-n) of the size of the tree) time. Perhaps most importantly, B+Trees are self-managing and do not face challenges with empty space and garbage collection.

- *Concurrency Control and Recovery*

The goal of concurrency control is to provide the impression to the application developer that nothing else is happening to the contents of the database while the application's work proceeds. It should appear as if there is some serial order of execution even when lots of concurrent activity is happening. Practitioners of concurrency control speak of serializability. The effects of tightly controlling the concurrency are to make the execution behave as if it were within a serial order even when lots of work is happening concurrently. The ability to make a serial order is serializeabilty. While there are other more relaxed guarantees, serializability remains an important concept.

Recovery includes all the mechanisms used to ensure the intact recreation of the database's data even when things go wrong. These potential problems span system crashes, the destruction of the disks holding the database, and even the destruction of the datacenter holding the database.

Concurrency control and recovery systems use locking and other techniques for ensuring isolation. They are also responsible for managing the cached pages of the database as they reside in memory. Sophisticated techniques based on logging have allowed for high-performance management of caching, transactional recovery, and concurrency control.

Key Applications

Database management systems are widely in use to capture most of the data used in today's businesses. Almost all enterprise applications are written using database management systems. While many client-based applications are based on file system representations of data, most server-based applications use a DBMS.

Cross-references

► Abstraction
► Access Control
► ACID Properties
► Active and Real-Time Data Warehousing
► Atomicity
► B+-tree
► Conceptual Schema Design
► Concurrency Control-Traditional Approaches
► Concurrency Control Manager
► Distributed DBMS
► Entity Relationship Model
► Generalization of ACID Properties
► Hierarchical Data Model
► Logical Schema Design
► Network Data Model
► QUEL
► Query Language
► Query Processing
► Query Processing (in relational databases)
► Relational Model
► Schema Evolution
► Secondary Index
► Serializability
► Stream Models
► Stream Processing
► System R(R*) Optimizer

► Transaction
► Transaction Management
► Transaction Manager
► XML
► XML Query Processing
► XPath/XQuery

Recommended Reading

1. Bayer R. Binary B-Trees for Virtual Memory, ACM-SIGFIDET Workshop, 1971, pp. 219–235.
2. Blasgen M.W., Astrahan M.M., Chamberlin D.D., Gray J., King W.F., Lindsay B.G., Lorie R.A., Mehl J.W., Price T.G., Putzolu G.F., Schkolnick M., Slutz D.R., Selinger P.G., Strong H.R., Traiger I.L., Wade B., and Yost B. SystemR – an Architectural Update. IBM RJ IBM RJ 2581, IBM Research Center, 95193, 7/17/1979. 42 pp.
3. Gray J. Data management: past, present, and future. IEEE Comput., 29(10):38–46, 1996.
4. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques, Morgan Kaufmann, San Mateo, CA, 1992.
5. Stonebraker M., Wong E., Kreps P., and Held G. The Design and Implementation of INGRES.

Database Materialization

► Physical Database Design for Relational Databases

Database Middleware

Cristiana Amza
University of Toronto, Toronto, ON, Canada

Synonyms

Database scheduling; Load balancing; Mediation and adaptation

Definition

Database middleware is a generic term used to refer to software infrastructure that supports (i) functionality, such as, interoperability between software components, or distributed transaction execution, (ii) improved database service, such as, performance scaling or fault tolerance of a database back-end in a larger system, or (iii) adaptations to workloads e.g., through the use of adaptive queuing middleware or of a scheduler component for adaptive reconfiguration of a database back-end.

Historical Background

Historically, TP Monitors were the first recognized database middleware components. TP Monitors, thus database middleware, was originally run on mainframes to connect different applications. Later, with the advent of e-business applications and modern multi-tier architectures that supported them, similar functionality as in the original TP Monitors became integrated in software components within the software stack used in these infrastructures, software components known as: "application servers," middleware components for "enterprise application integration," "enterprise service bus," and "transactional middleware." Transactional middleware supported the execution of distributed electronic transactions, and often provided much more functionality than just transactions. Modern e-business architectures consist of multiple tiers, such as client, application server, and database server. In these architectures typically replication in the database back-end is used for scaling. In these architectures, middleware components, such as, schedulers and load balancers are interposed in front of a database back-end for the purposes of scheduling database transactions, maintaining fault tolerance or providing data availability.

Foundations

Software for database middleware is very diverse, and serves a variety of purposes, e.g., integration of various data sources, load balancing for scaling, fault tolerance, etc. This entry distinguishes between the following middleware classes: *middleware for integration*, *middleware for performance scaling and availability*, *transactional and messaging middleware*, and *middleware for adaptation and reconfiguration*. This section will review these main types of database middleware and their uses, providing a brief survey of each of these areas.

Database Middleware for Integration

Database middleware, such as, Oracle9i, Informix Universal Server, MOCHA [12], DISCO [11], and Garlic [13] support integration of possibly heterogeneous, distributed data sources. Specifically, database middleware of this type may commonly consist of a data integration server, accessing the data at the remote data sources through a *gateway*, which provides client applications with a uniform view and access mechanism to the data available in each source. Several existing commercial database servers use this approach to access remote data sources through a *database gateway*, which

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:29

provides an access method to the remote data, e.g., the IBM DB2 DataJoiner, and Sybases's Direct Connect, and Open Server Connect products. These products enable viewing multi-vendor, relational, nonrelational, and geographically remote data, as if it were local data.

The alternative is to use a *mediator* server for the same purpose. In this approach, the mediator server is capable of performing distributed query processing for the purposes of integration of the various data sources. Both methods superimpose a global data model on top of the local data models used at each data source. For example, the mediator uses wrappers to access and translate the information from the data sources into the global data model. Furthermore, in both of these middleware solutions, specialized libraries defining application data types and query operators are used by the clients, integration servers, gateways or wrappers deployed in the system.

Database Middleware for Scaling and Availability

Middleware components, such as, schedulers, load balancers and optimizers have been used for performance scaling of workloads on LAN-based database clusters, and/or for data availability. Schedulers and optimizers have been used for maintaining data consistency in replicated database clusters, and for minimizing the data movement in shared-nothing database clusters employing data partitioning, respectively. Shared-nothing database cluster architectures have been traditionally used for scaling classic database applications, such as, on-line transaction processing (OLTP) workloads. Data partitioning across the cluster [4,6] was absolutely necessary to alleviate the massive I/O needs of these applications through in-memory data caching. Data partitioning implied using rather complex optimizers to minimize reconfigurations and data movement between machines [5].

In contrast, more recently, due to the advent of larger memories, and the impact of modern e-commerce applications with more localized access patterns, scheduling applications for performance scaling on a cluster, using database replication has gained more attention [2,3]. For example, for the usual application sizes, there is little disk I/O in dynamic content applications [1], due to the locality exhibited by these applications. For example, in on-line shopping, bestsellers, promotional items and new products are accessed with high frequency. Similarly, the stories of the day and items on auction are hot objects in bulletin board and on-line bidding applications, respectively. This makes replication much more promising, and considerably easier to use than data partitioning.

However, replication for scaling incurs the cost of replicating the execution of update queries for maintaining the table replicas consistent. Fortunately, in dynamic content applications, queries that update the database are usually lightweight compared to read-only requests. For instance, in e-commerce, typically, only the record pertaining to a particular customer or product is updated, while any given customer may *browse* the product database using complex search criteria. More importantly, the locality in access patterns of dynamic content applications may imply higher conflict rates relative to traditional applications, given a sufficiently high fraction of writes. For instance the probability that a "best seller" book is being bought concurrently by two different customers, incurring a conflict on that item's stock is much higher than the probability that two customers access their joint account at the same time. Thus, intuitively, e-commerce applications have potentialy higher conflict rates than traditional OLTP applications. This trend has motivated more recent schemes on middleware support for scheduling transactions using a combination of load balancing and *conflict-aware replication* [2,3,10]. These techniques have shown good scaling in the tens of database engines range for the most common e-commerce workloads, e.g., on experiments using the TPC-W industry standard e-commerce benchmark. Middleware for caching query results has also been used in isolation or in combination with replication, e.g., through caching query results within the scheduler component corrdinating replication on a cluster, as an orthogonal technique for performance scaling of web sites supporting these workloads.

Finally, replication brings with it fault tolerance and high data availability as a bonus. Middleware components for fault tolerance support different degrees of failure transparency to the client. In the most common case, transaction aborts due to failures are exposed to the client when a failure of a replica occurs. More sophisticated fail-over schemes focus on precise error semantics, and on hiding failures from the client. Moreover, providing fault tolerance and data availability in a multi-tier architecture consisting of web-servers, application servers and database servers that interact with each other raises important trade-offs in terms of architecture design.

Middleware for precise failure semantics, such as exactly-once transactions [7], provide an automated end-to-end solution involving the client. Such middleware tracks client transactions through the software stack e.g., composed of client, application server, and database server, and can be used to automatically handle client duplicate requests, and reissue aborted transactions, thus seamlessly hiding failures in the database back-end from the client.

Transactional and Messaging Middleware

Transactional middleware provides control and the transfer of data between clients and a set of distributed databases. The main purpose is to run distributed transactions in an efficient manner. Transactional middleware systems, such as BEA Tuxedo, typically support a subset of the ACID properties, such as atomicity (by running a 2-phase-commit protocol over the distributed databases), durability or isolation. Transactional middleware is especially important in three-tier architectures that employ load balancing because a transaction may be forwarded to any of several servers based on their load and availability.

Message Queueing systems offer analogous functionality to TP Monitors, such as, improving reliability and scalability, although they typically offer weaker consistency guarantees than TP Monitors. Several messages may be required to complete an overall transaction, and those messages will each tend to reflect the latest system state rather than consistently looking back to the state of the system at the time the transaction started.

Messaging-oriented middleware provides an interface between client and server applications, allowing them to send data back and forth intermittently. If the target computer isn't available, the middleware stores the data in a message queue until the machine becomes available.

Middleware for Adaptation and Reconfiguration

Recent systems investigate adaptive reconfiguration in two classic middleware scenarios: database replication and message queuing systems.

In the context of database replication, dynamic adaptation has been used for reconfiguration of a database cluster to adapt to workload changes. Specifically, recent work adapts the configuration of a database cluster dynamically in response to changing demand by (i) adapting the placement of primary replicas and the degree of multi-programming at each replica [9] or by (ii) changing the number of replicas allocated to a workload [14]. For example, recent techniques for dynamic replica allocation in a database cluster employ an on-line technique based on middleware or group communication [8,14] for bringing a new replica up to date with minimal disruption of transaction processing on existing replicas in the application's allocation.

Key Applications

Database middleware is widely used in practice. All database vendors also offer a suite of middleware solutions for data integration, load balancing, scheduling, data replication, etc. Transactional middleware, message queuing systems, and middleware for fault tolerance, availbility and reconfiguration of the database back-end are commonly used in all modern e-business solutions, and in particular in multi-tier dynamic content web sites, such as, amazon.com and e-bay.com.

Cross-references

► Adaptive Middleware for Message Queuing Systems
► Mediation
► Message Queuing Systems
► Middleware Support for Caching And Replication
► Middleware Support for Precise Failure Semantics
► Replication in Multi-tier Architectures
► Transactional Middleware

Recommended Reading

1. Amza C., Cecchet E., Chanda A., Cox A., Elnikety S., Gil R., Marguerite J., Rajamani K., and Zwaenepoel W. Specification and implementation of dynamic web site benchmarks. In Fifth IEEE Workshop on Workload Characterization, 2002.
2. Amza C., Cox A., and Zwaenepoel W. Conflict-aware scheduling for dynamic content applications. In Proc. Fifth USENIX Symp. on Internet Technologies and Systems, March 2003, pp. 71–84.
3. Amza C., Cox A.L., and Zwaenepoel W. Distributed versioning: Consistent replication for scaling back-end databases of dynamic content web sites. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., volume 2672 of Lecture Notes in Computer Science, 2003, pp. 282–304.
4. Boral H., Alexander W., Clay L., Copeland G., Danforth S., Franklin M., Hart B., Smith M., and Valduriez P. Prototyping Bubba, a highly parallel database system. IEEE Trans. Knowl. Data Eng., 2:4–24, 1990.
5. Chaudhuri S. and Weikum G. Rethinking database system architecture: Towards a self-tuning RISC-style database system. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 1–10.
6. Copeland G., Alexander W., Boughter E., and Keller T. Data placement in Bubba. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988, pp. 99–108.

7. Frolund S. and Guerraoui R. e-transactions: End-to-end reliability for three-tier architectures. IEEE Trans. Software Eng., 2002, pp. 378–395.
8. Liang W. and Kemme B. Online recovery in cluster databases. In Advances in Database Technology, Proc. 11th Int. Conf. on Extending Database Technology, 2008.
9. Milan-Franco J.M., Jimenez-Peris R., Patio-Martnez M., and Kemme B. Adaptive middleware for data replication. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004.
10. Plattner C. and Alonso G. Ganymed: scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004.
11. Rashid L., Tomasic A., and Valduriez P. Scaling heterogeneous databases and the design of DISCO. In Proc. 16th Int. Conf. on Distributed Computing Systems, 1996.
12. Rodriguez-Martinez M. and Roussopoulos N. MOCHA: a self-extensible database middleware system for distributed data sources. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000.
13. Roth M.T. and Schwarz P. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997.
14. Soundararajan G. and Amza C. Reactive provisioning of back-end databases in shared dynamic content server clusters.. ACM Trans. Auto. Adapt. Syst. (TAAS), 1(2):151–188, 2006.

Database Physical Layer

► Storage Access Models

Database Profiling

► Data Profiling

Database Protection

► Database Security

Database Provisioning

► Autonomous Replication

Database Redocumentation

► Database Reverse Engineering

Database Repair

Leopoldo Bertossi
Carleton University, Ottawa, ON, Canada

Definition

Given an inconsistent database instance, i.e. that fails to satisfy a given set of integrity constraints, a repair is a new instance over the same schema that is consistent and is obtained after performing minimal changes on the original instance with the purpose of restoring consistency.

Key Points

Database instances may be inconsistent, in the sense that they may not satisfy certain desirable integrity constraints. In order to make the database consistent, certain updates can be performed on the database instance. However, it is natural to expect that any new consistent instance obtained in this way does not differ too much from the original instance. The notion of repair of the original instance captures this intuition: it is an instance of the same schema that does satisfy the integrity constraints and differs from the original instance by a minimal set of changes. Depending on what is meant by minimal set of changes, different repair semantics can be obtained.

The notion of *repair*, also called *minimal repair*, was introduced in [1]. Database instances can be seen as finite sets of ground atoms. For example, *Students(101, joe)* could be a database atom representing an entry in the relation *Students*. In order to compare two instances of the same schema, it is possible to consider their (set-theoretic) symmetric difference. A repair, as introduced in [1], will make the symmetric difference with the original instance minimal under set inclusion. That is, no other consistent instance differs from the original instance by a proper subset of database tuples. It is implicit in this notion of repair that changes on the original instance are obtained through insertions or deletions of complete database atoms. This notion of repair was used in [1] to characterize the consistent data in an inconsistent database as the data that is invariant under all possible repairs.

In the same spirit, other repairs semantics have also been investigated in the literature. For example, an alternative definition of repair might minimize the cardinality of the symmetric difference. There are also repairs that are obtained via direct updates of attribute values (as opposed to deletions followed by insertions, which might not represent a minimal change). In this case, the number of those local changes could be minimized. A different, more general aggregation function of the local changes could be minimized instead (cf. [2,3] for surveys).

Cross-references

► Consistent Query Answering
► Inconsistent Databases

Recommended Reading

1. Arenas M., Bertossi L., and Chomicki J. Consistent query answers in inconsistent databases. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 68–79.
2. Bertossi L. Consistent query answering in databases. ACM Sigmod Rec., 35(2):68–76, 2006.
3. Chomicki J. Consistent query answering: five easy pieces. In Proc. 11th Int. Conf. on Database Theory. Springer, LNCS, 4353:1–17, 2007.

Database Replication

► Data Replication
► Replica Control

Database Reverse Engineering

Jean-Luc Hainaut, Jean Henrard, Vincent Englebert, Didier Roland, Jean-Marc Hick
University of Namur, Namur, Belgium

Synonyms

Database redocumentation; Database design recovery

Definition

Database reverse engineering is the process through which the logical and conceptual schemas of a legacy database, or of a set of files, are reconstructed from various information sources such as DDL code, data dictionary contents, database contents or the source code of application programs that use the database.

Basically, database reverse engineering comprises three processes, namely physical schema extraction, logical schema reconstruction, and schema conceptualization. The first process consists in parsing the DDL code or the contents of an active data dictionary in order to extract the physical schema of the database. Reconstructing the logical schema implies analyzing additional sources such as the data and the source code of the application programs to discover implicit constraints and data structures, that is, constructs that have not been declared but that are managed by the information system or by its environment. The conceptualization process aims at recovering the conceptual schema that the logical schema implements.

Database reverse engineering is often the first step in information system maintenance, evolution, migration and integration.

Historical Background

Database reverse engineering has been recognized to be a specific problem for more than three decades, but has been formally studied since the 1980's, notably in [3,6,12]. The first approaches were based on simple rules, that work nicely with databases designed in a clean and disciplined way. A second generation of methodologies coped with physical schemas resulting from empirical design in which practitioners tend to apply non standard and undisciplined techniques. More complex design rules were identified and interpreted [2], structured and comprehensive approaches were developed [11,7] and the first industrial tools appeared (e.g., Bachman's Reengineering Tool). Many contributions were published in the 1990's, addressing practically all the legacy technologies and exploiting such sources of information as application source code, database contents or application user interfaces. Among synthesis publications, it is important to mention [5], the first tentative history of this discipline.

These second generation approaches were faced with two kinds of problems induced by empirical design [8]. The first problem is the recovery of *implicit constructs*, that is, structures and constraints that have not been explicitly declared in the DDL code. The second problem is that of the *semantic interpretation* of logical schemas that include non standard data structures.

Foundations

The ultimate goal of reverse engineering a piece of software is to recover its functional and technical specifications, starting mainly from the source code of the

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:30

programs [4]. The problem is particularly complex with old and ill-designed applications. In this case, there is no documentation to rely on; moreover, the lack of systematic methodologies for designing and maintaining them have led to tricky and obscure code. Therefore, reverse engineering has long been regarded as a complex, painful and failure-prone activity, in such a way that it is simply not undertaken most of the time, leaving huge amounts of invaluable knowledge buried in legacy software, lost for all practical purposes.

In most software engineering cases, analysts have to content themselves with the extraction of abstract and/or partial information, such as call graphs, dependency graphs or program slices in order to ease the maintenance and evolution of the software. The result of reverse engineering a database is more satisfying, in that reconstructing the logical and conceptual schemas of an undocumented database is achievable with reasonable effort.

Database Design Revisited

To understand the problems, challenges and techniques specific to database reverse engineering, it is necessary to reexamine the way databases are developed, both in theory and in practice.

Standard Database Design Methodology Standard database design comprises four formal processes, namely conceptual analysis, logical design, physical design and coding.

Conceptual analysis produces the conceptual schema of the database, that is, an abstract description of the concepts that structure the application domain, of the relationships between these concepts and of the information to be collected and kept about theses classes and relationships. This schema is independent of the application programs that will use the database and is expressed in an abstract formalism such as some variant of the Entity-relationship model. It must be readable, maintainable, normalized and independent of any implementation technology.

Logical design translates the conceptual schema into data structures compliant with the data model of a family of DBMSs. This process is best described by a transformation plan, according to which the constructs (or components) of the conceptual schema that cannot be directly translated into the target DDL are first *transformed* into constructs of the DBMS model. For instance, a single-valued atomic attribute is directly translated into a column. On the contrary, a N:N relationship type cannot be expressed in the relational DDL. Therefore, it is first transformed into a relationship entity type and two N:1 relationship types, which in turn are translated into a relationship table and two foreign keys. The resulting logical schema is the basis for program development. It must be clear, simple and devoid of any performance concern. Denoting the conceptual and logical schemas respectively by `CS` and `LS`, this process can be synthesized by the functional expression `LS = logical-design(CS)`, that states that the logical schema results from the transformation of the conceptual schema.

Physical design enriches and potentially reshapes the logical schema to make it meet technical and performance requirements according to a specific technology (DBMS). Physical design can be expressed by `PS = physical-design(LS)`, where `PS` denotes the physical schema.

Coding expresses the physical schema in the DDL of the DBMS. Some of the data structures and integrity constraints can be translated into explicit DDL statements. Such is the case, in relational databases, for elementary data domains, unique keys, foreign keys and mandatory columns. However, the developer must resort to other techniques to express all the other constructs. Most relational DBMSs offer check and trigger mechanisms to control integrity, but other servers do not include such facilities, so that many constraints have to be coped with by procedural code distributed and duplicated in the application programs. The derivation of the code can be expressed by `code = coding(PS)`. The code itself can be decomposed into the DDL code in which some constructs are explicitly expressed and the external code that controls and manages all the other constructs: $\texttt{code} = \texttt{code}_{\texttt{ddl}} \cup \texttt{code}_{\texttt{ext}}$. Similarly, the coding function can be developed into a sequence of two processes ($\texttt{coding}_{\texttt{ddl}}\texttt{(PS)};\ \texttt{coding}_{\texttt{ext}}\texttt{(PS)}$).

The production of the database code from the conceptual schema (forward engineering or `FE`) can be written as `code = FE(CS)`, where function `FE` is the composition `coding` o `physical-design` o `logical-design`.

Empirical Database Design Actual database design and maintenance do not always follow a disciplined approach such as that recalled above. Many databases have been built incrementally to meet the evolving needs of application programs. Empirical design relies on the experience of self-taught database designers, who often ignore the basic database design theories and best practices. This does not mean that all these databases are badly designed, but they may include many non-standard patterns, awkward constructs and idiosyncrasies that make them difficult to understand [2]. Since no disciplined approach was adopted, such databases often include only a logical schema that integrates conceptual, logical, physical and optimization constructs. Quite often too, no up-to-date documentation, if any, is available. An important property of the functional model of database design evoked in previous section is that it is still valid for empirical design. Indeed, if empirical design rules of the designer are sorted according to the criteria of the three processes, functions `logical-design'`, `physical-design'` and `coding'` can be reconstructed into an idealized design that was never performed, but that yields the same result as the empirical design.

Database Reverse Engineering Processes

Broadly speaking, reverse engineering can be seen as the reverse of forward engineering [1], that is, considering the function $\mathtt{RE} = \mathtt{FE}^{-1}$, `CS = RE(code)`. Since most forward engineering processes consist of *schema transformations* [9], their reverse counterparts should be easily derivable by inverting the forward transformations.

Unfortunately, forward engineering is basically a lossy process as far as conceptual specifications are concerned. On the one hand, it is not unusual to discard bits of specifications, notably when they prove difficult to implement. On the other hand, the three processes are seldom injective functions in actual situations. Indeed, there is more than one way to transform a definite construct and several distinct constructs can be transformed into the same target construct. For instance, there are several ways to transform an is-a hierarchy into relational structures, including the use of primary-foreign keys (forward engineering). However, a primary-foreign key can also be interpreted as the implementation of a 1:1 relationship type, as the trace of entity type splitting or as the translation of an is-a relation (reverse engineering). Clearly, the transformational interpretation of these processes must be refined.

Nevertheless it is important to study and to model the reverse engineering as the inverse of `FE`, at least to identify and describe the pertinent reverse processes. Decomposing the initial relation `CS = RE(code)`, one obtains:

$\mathtt{CS} = \mathtt{conceptualization(LS)}$
$\mathtt{LS} = \mathtt{logical\text{-}reconstruction(PS,\ code}_{\mathtt{ext}}\mathtt{)}$
$\mathtt{PS} = \mathtt{physical\text{-}extraction(code}_{\mathtt{ddl}}\mathtt{)}$
$\mathtt{RE} = \mathtt{conceptualization} \circ \mathtt{logical\text{-}reconstruction} \circ \mathtt{physical\text{-}extraction}$

where

$\mathtt{Conceptualization} = \mathtt{logical\text{-}design}^{-1}$
$\mathtt{Logical\text{-}reconstruction} = \mathtt{physical\text{-}design}^{-1} \,||\, \mathtt{coding}_{\mathtt{ext}}{}^{-1}$
$\mathtt{Physical\text{-}extraction} = \mathtt{coding}_{\mathtt{ddl}}{}^{-1}$

This model emphasizes the role of program code as a major source of information. As explained below, other sources will be used as well.

Physical Schema Extraction

This process recovers the physical schema of the database by parsing its DDL code ($\mathtt{code}_{\mathtt{ddl}}$) or, equivalently, by analyzing the contents of its active data dictionary, such as the system tables in most relational systems. This extraction makes visible the *explicit constructs* of the schema, that is, the data structures and constraints that have been explicitly declared through DDL statements and clauses. Such is the case for primary keys, unique constraints, foreign keys and mandatory fields. Generally, this process is fairly straightforward. However, the analysis of sub-schemas (e.g., relational views, CODASYL sub-schemas or IMS PCBs) can be more intricate. Indeed, each sub-schema brings a partial, and often refined view of the global schema. In addition, some data managers, such as standard file managers, ignore the concept of global schema. A COBOL file for instance, is only described in the source code of the programs that use it. Each of them can perceive its data differently. Recovering the global physical schema of a COBOL file requires a potentially complex schema integration process.

Logical Schema Reconstruction

This process addresses the discovery of the *implicit constructs* of the schema. Many logical constructs have not been declared by explicit DDL statements and clauses. In some favorable situations, they have been translated into programmed database components such as SQL checks, triggers and stored procedures.

However, most of them have been translated into application program fragments (nearly) duplicated and scattered throughout millions of lines of code (`code`$_{\texttt{ext}}$). For instance, a popular way to check a referential constraint consists in accessing the target record before storing the source record in its file. Recovering these *implicit constructs*, in contrast with *explicit constructs*, which have been expressed in DDL, requires a precise analysis of various pieces of procedural code. Though program source code is the richest information source, the database contents (the *data*), screen layout, report structure, program execution, users interview and, of course, (possibly obsolete) documentation will be analyzed as well. As a final step of the reconstruction, physical components are discarded inasmuch as they are no longer useful to discover logical constructs.

Implicit Constructs All the structures and constraints that cannot be expressed in the DDL are implicit by nature. However, many database schemas include implicit constructs that could have been declared at design time but that were not, for such reasons as convenience, standardization, inheritance from older technology or simply by ignorance or bad design. Two popular examples can be mentioned. In network and hierarchical databases, some links between record types are translated into implicit foreign keys despite the fact that relationship types could have been explicitly declared through set types or parent-child relationship types. In many legacy relational databases, most foreign keys are not declared through `foreign key` clauses, but are managed by an appropriate set of triggers. The most important implicit constructs are the following [8].

Exact field and record structure. Compound and multivalued fields are often represented by the concatenation of their elementary values. Screen layout and program analysis are major techniques to discover these structures.

Unique keys of record types and multivalued fields. This property is particularly important in strongly structured record types and in sequential files.

Foreign keys. Each value of a field is processed as a reference to a record in another file. This property can be discovered by data analysis and program analysis.

Functional dependencies. The values of a field can depend on the values of other fields that have not been declared or elicited as a candidate key. This pattern is frequent in older databases and file systems for performance reasons.

Value domains. A more precise definition of the domain of a field can be discovered by data and program analysis. Identifying enumerated domains is particularly important.

Meaningful names. Proprietary naming standards (or, worse, the absence thereof) may lead to cryptic component names. However, the examination of program variables and electronic form fields in/from which field values are moved can suggest more significant names.

Sources and Techniques Analytical techniques applied to various sources can all contribute to a better knowledge of the implicit components and properties of a database schema.

Schema analysis. Spotting similarities in names, value domains and representative patterns may help identify implicit constructs such as foreign keys.

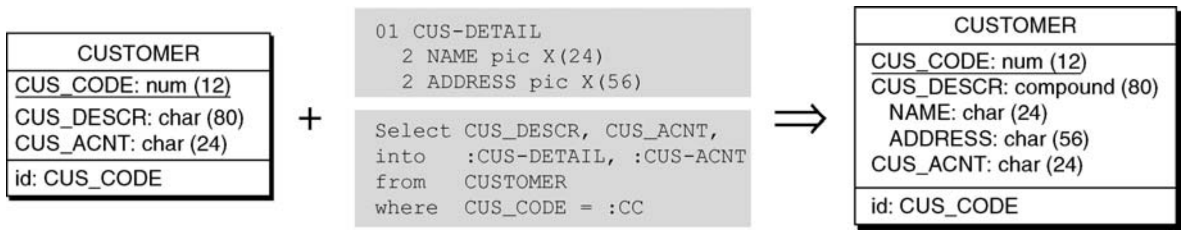
Data analysis. Mining the database contents can be used in two ways. First, to discover implicit properties, such as functional dependencies and foreign keys. Second, to check hypothetic constructs that have been suggested by other techniques. Considering the combinatorial explosion that threaten the first approach, data analysis is most often applied to check the existence of formerly identified patterns.

Program analysis. Understanding how programs use the data provides crucial information on properties of these data. Even simple analysis, such as dataflow graphs, can bring valuable information on field structure (Fig. 1) and meaningful names. More sophisticated techniques such as dependency analysis and program slicing can be used to identify complex constraint checking or foreign keys.

Screen/report layout analysis. Forms, reports and dialog boxes are user-oriented views on the database. They exhibit spatial structures (e.g., data aggregates), meaningful names, explicit usage guidelines and, at run time, data population that, combined with dataflow analysis, provide much information on implicit data structures and properties.

Schema Conceptualization

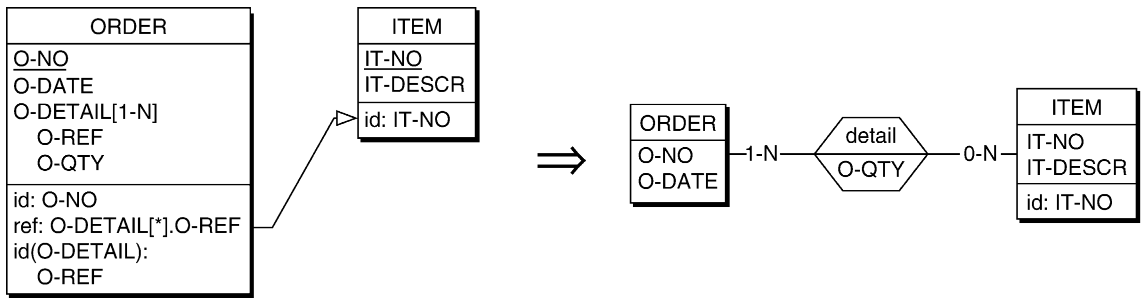
The goal of this process is to interpret the logical schema semantically by extracting a conceptual schema that represents its intended meaning. It mainly relies on transformational techniques that undo the effect of the logical design process. This complex process is decomposed in three subprocesses, namely



```
01 CUS-DETAIL
  2 NAME pic X(24)
  2 ADDRESS pic X(56)
```

```
Select CUS_DESCR, CUS_ACNT,
into   :CUS-DETAIL, :CUS-ACNT
from   CUSTOMER
where  CUS_CODE = :CC
```

Database Reverse Engineering. Figure 1. Illustration of the physical schema extraction and logical schema reconstruction processes.



Database Reverse Engineering. Figure 2. Conceptualization of a complex field.

untranslation, de-optimization and conceptual normalization. The *untranslation* process consists of reversing the transformations that have been used to draw the logical schema from the conceptual schema. For instance, each foreign key is interpreted as the implementation of a N:1 relationship type. This process relies on a solid knowledge of the rules and heuristics that have been used to design the database. Those rules can be standard, which makes the process fairly straightforward, but they can also be specific to the company or even to the developer in charge of the database (who may have left the company), in which case reverse engineering can be quite tricky. The main constructs that have to be recovered are relationship types (Fig. 2), super-type/subtype hierarchies, multivalued attributes, compound attributes and optional attributes. The *de-optimization* process removes the trace of all the optimization techniques that have been used to improve the performance of the database. Redundancies must be identified and discarded, unnormalized data structures must be decomposed and horizontal and vertical partitioning must be identified and undone. Finally, *conceptual normalization* improves the expressiveness, the simplicity, the readability and the extendability of the conceptual schema. It has the same goals and uses the same techniques as the corresponding process in Conceptual analysis.

Tools

Reverse engineering requires the precise analysis of huge documents such as programs of several millions of lines of code and schemas that include thousands of files and hundreds of thousands of fields. It also requires repeatedly applying complex rules on thousands of patterns. In addition, many reverse processes and techniques are common with those of forward engineering, such as transformations, validation and normalization. Finally, reverse engineering is only a step in larger projects, hence the need for integrated environments that combine forward and reverse tools and techniques [11].

Examples

Figure 1 illustrates the respective roles of the physical schema extraction and logical schema reconstruction processes. Parsing the DDL code identifies column CUS_DESCR as a large atomic field in the physical schema (left). Further dataflow analysis allows this column to be refined as a compound field (right).

The conceptualization of a compound field as a complex relationship type is illustrated in Figure 2. The multivalued field O-DETAIL has a component (O-REF) that serves both as an identifier for its values (the values of O-DETAIL in an ORDER record have distinct values of O-REF) and as a reference to an

ITEM record. This construct is interpreted as an N:N relationship type between ORDER and ITEM.

Key Applications

Database reverse engineering is most often the first step in information system maintenance, evolution [10], migration and integration. Indeed, such complex projects cannot be carried out when no complete, precise and up-to-date documentation of the database of the information system is available.

The scope of data reverse engineering progressively extends to other kinds of information such as web sites, electronic forms, XML data structures and any kind of semi-structured data. Though most techniques and tools specific to database reverse engineering remain valid, additional approaches are required, such as linguistic analysis and ontology alignment.

Cross-references

► Database Design
► Entity-Relationship Model
► Hierarchical Data Model
► Network Data Model
► Relational Data Model

Recommended Reading

1. Baxter I. and Mehlich M. Reverse engineering is reverse forward engineering. Sci. Comput. Programming, 36:131–147, 2000.
2. Blaha M.R. and Premerlani W.J. Observed idiosyncrasies of relational database designs. In Proc. 2nd IEEE Working Conf. on Reverse Engineering, 1995, p. 116.
3. Casanova M.A. and Amaral de Sa J.E. Mapping uninterpreted schemes into entity-relationship diagrams: two applications to conceptual schema design. IBM J. Res. Develop., 28(1):82–94, 1984.
4. Chikofsky E.J. and Cross J.H. Reverse engineering and design recovery: a taxonomy. IEEE Softw., 7(1):13–17, 1990.
5. Davis K.H. and Aiken P.H. Data reverse engineering: a historical view. In Proc. Seventh Working Conf. on Reverse Engineering (WCRE'00). 2000, pp. 70–78.
6. Davis K.H. and Arora A.K. A methodology for translating a conventional file system into an entity-relationship model. In Proc. Fourth Int. Conf. on Entity-Relationship Approach (ERA). 1985, pp. 148–159.
7. Edwards H.M. and Munro M. Deriving a logical model for a system using recast method. In Proc. Second IEEE Working Conf. on Reverse Engineering, 1995, pp. 126–135.
8. Hainaut J.-L. Introduction to database reverse engineering, LIBD lecture notes, Pub. University of Namur, Belgium, 2002, pp. 160.
9. Hainaut J-L. The transformational approach to database engineering. In Generative and Transformational Techniques in Software Engineering, R. Lämmel, J. Saraiva, J. Visser (eds.), LNCS 4143. Springer-Verlag, 2006, pp. 89–138.
10. Hainaut J-L., Clève A., Henrard J., and Hick J.-M. Migration of Legacy Information Systems. In Software Evolution, T. Mens, S. Demeyer (eds.). Springer-Verlag, 2007, pp. 107–138.
11. Hainaut J-L., Roland D., Hick J-M., Henrard J., and Englebert V. Database reverse engineering: from requirements to CARE tools. J. Automated Softw. Eng., 3(1/2):9–45, 1996.
12. Navathe S.B. and Awong A. Abstracting relational and hierarchical data with a semantic data model. In Proc. Entity-Relationship Approach: a Bridge to the User. North-Holland, 1987, pp. 305–333.

Database Scheduling

► Database Middleware

Database Security

Elena Ferrari
University of Insubria, Varese, Italy

Synonyms

Database protection

Definition

Database security is a discipline that seeks to protect data stored into a DBMS from intrusions, improper modifications, theft, and unauthorized disclosures. This is realized through a set of *security services*, which meet the security requirements of both the system and the data sources. Security services are implemented through particular processes, which are called *security mechanisms*.

Historical Background

Research in database security has its root in operating system security [6], whereas its developments follow those in DBMSs. Database security has many branches, whose main historical developments are summarized in what follows:

Access control. In the 1970s, as part of the research on System R at IBM Almaden Research Center, there was a lot of work on access control for relational DBMSs [3]. About the same time, some early work on Multilevel Secure Database Management Systems (MLS/DBMSs) was reported, whereas much of the development on MLS/DBMSs began [9] only after the Air Force Summer Study in 1982 [1]. This has resulted in different research prototypes, such as for instance those developed at MITRE, SRI International and Honeywell Inc. Access control models developed for relational databases have then been extended to cope with the new requirements of advanced DBMSs, such as object-oriented, object-relational, multimedia and active DBMSs [9], GIS [7], and XML DBMSs [5], and other advanced data management systems and applications, including digital libraries, data warehousing systems, and workflow management systems [9]. Role-based access control has been proposed in the 1990s [8] as a way to simplify authorization management within companies and organizations.

Privacy protection. Given the vast amount of personal data that is today collected by DBMSs, privacy is becoming a primary concern, and this has resulted in various research activities that have been started quite recently. A first research direction is related to *privacy-preserving data mining*, that is, how to apply data mining tools without compromising user privacy. All approaches developed so far are based on modifying or perturbing the data in some way. One of the issues is therefore how to maintain the quality of the modified data in such a way that they can be useful for data mining operations. Another line of research deals with the support of privacy policies, within the DBMS. In this direction, one of the most mature and promising proposal is the concept of Hippocratic database recently proposed by Agrawal et al. [3].

Auditing. Research on this issue has mainly focused on two directions: organization of the audit data and use of these data to discover possible security breaches. Another important research area is how to protect the audit data to prevent their malicious tampering.

Authentication. The simplest form of authentication is the one based on password. Throughout the years, several efforts have been made to make this scheme more robust against security threats or to develop schemes more suited to distributed and web-based environments. This is the case, for instance, of token-based schemes, that is, schemes based on biometric information, or *single sign-on* (SSO) schemes, particularly suited for distributed environments since they allow a user to authenticate once and gain access to the resources of multiple DBMSs. Recently, an innovative form of authentication has been proposed, based on user social relationships [4].

Foundations

Today data are one of the most crucial assets of any organization and, as such, their protection from intrusions, improper modifications, theft, and unauthorized disclosures is a fundamental service that any DBMS must provide [3,9,13]. Since data in a database are tightly related by semantic relationships, a damage of a data portion does not usually affect a single user or application, but the entire information system. Security breaches are typically categorized into the following categories: *unauthorized data observation*, *incorrect data modification*, and *data unavailability*. Security threats can be perpetrated either by outsiders or by users legitimately entitled to access the DBMS. The effect of unauthorized data observation is the disclosure of information to users not entitled to access such information. All organizations one may think of, ranging from commercial organizations to social or military organizations, may suffer heavy losses from both financial view and human point of view upon unauthorized data observation. Incorrect modifications of data or incorrect data deletion, either intentional or unintentional, results in an inconsistent database state. As a result, the database is not any longer correct. Any use of incorrect data may again result in heavy losses for the organization. Think, for instance, of medical data, where different observations of the same vital parameter may be used to make a clinical diagnosis. For the correct diagnosis it is crucial that each observation has not been incorrectly modified or deleted. When data are unavailable, information that are crucial for the proper functioning of the organization may not be readily accessible when needed. For instance, consider real-time systems, where the availability of some data may be crucial to immediately react to some emergency situations.

Therefore, data security requires to address three main issues:

1. *Data secrecy or confidentiality.* It prevents improper disclosure of information to unauthorized users. When data are related to personal information, *privacy* is often used as a synonym of data confidentiality. However, even if some techniques to protect confidentiality can be used to enforce privacy, protecting privacy requires some additional countermeasures. More precisely, since information privacy relates to an individual's right to determine how, when, and to what extent his or her personal

information will be released to another person or to an organization [11], protecting privacy requires to deal with additional issues with regard to confidentiality, such as, for instance, verifying that data are used only for the purposes authorized by the user and not for other purposes, or obtaining and recording the consents of users.

2. *Data integrity.* It protects data from unauthorized or improper modifications or deletions.
3. *Data availability.* It prevents and recovers data from hardware and software errors and from malicious data denials making the database or some of its portions not available.

Today, access to databases is mainly web-based and, in this context, two additional issues, besides the above-mentioned one, should be addressed, in order to provide strong guarantees about data contents to users:

Data authenticity. It ensures that a user receiving some data can verify that the data have been generated by the claimed source and that they have not been maliciously modified.
Data completeness. The user can verify whether he or she has received all the data he or she requested.

A DBMS exploits the services of the underlying operating system to manage its data (for instance to store data into files). This applies also to security. However, protecting data stored into a DBMS is different from protecting data at the operating system level, and therefore it requires additional and more sophisticated mechanisms. The main reasons are the following:

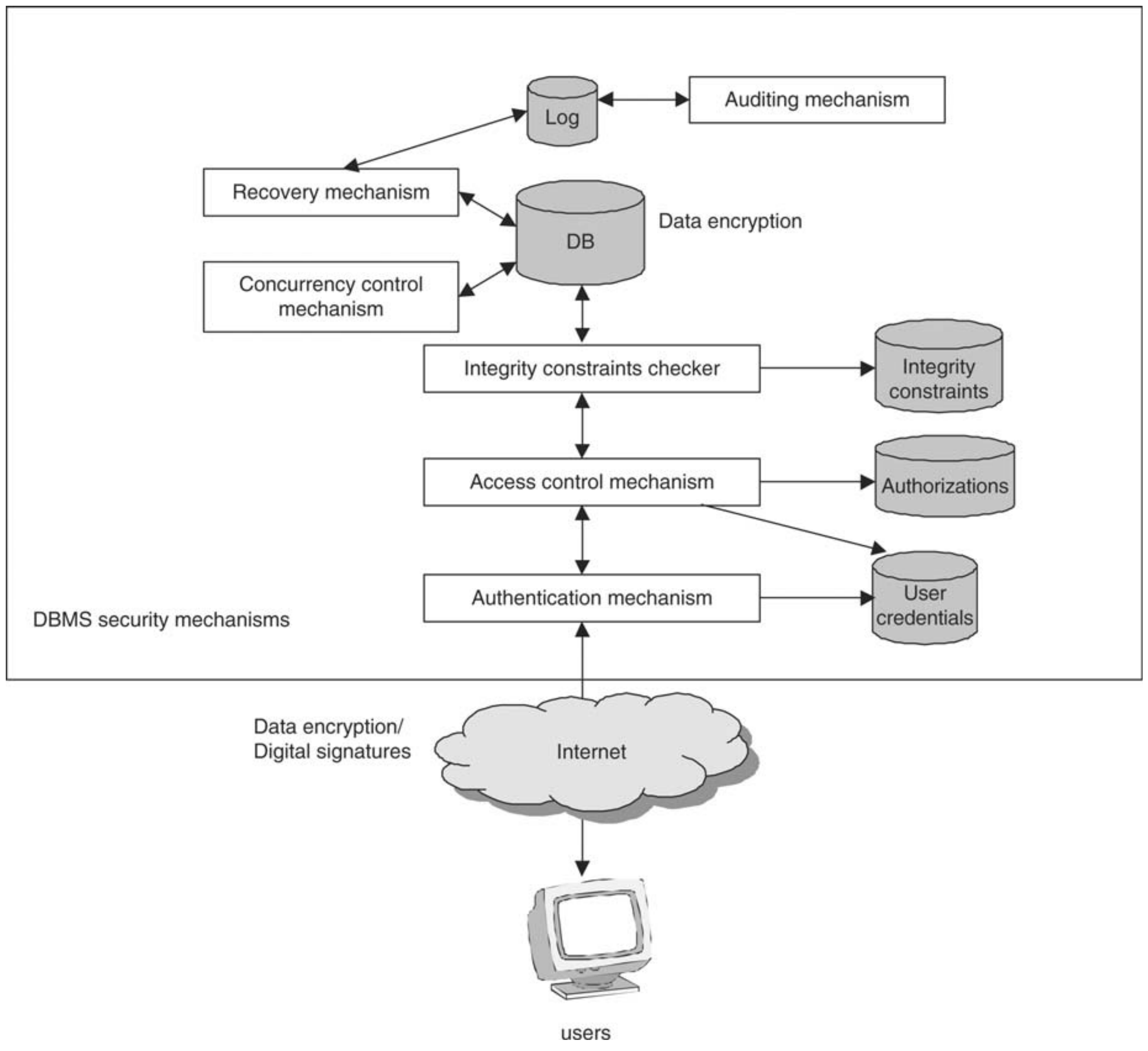
1. DBMSs and operating systems adopt different data models. In particular, data in a DBMS can be represented at different level of abstraction (physical, logical, view level), whereas an operating system adopts a unique representation of data (i.e., data are stored into files) and this simplifies data protection.
2. DBMSs are characterized by a variety of granularity levels for data protection. For instance, in a relational database, data can be protected at the relation or view level. However, sometimes finer granularity levels are needed, such as for instance selected attributes or selected tuples within a table. In contrast, in an operating system data protection is usually enforced at the file level.
3. In a database, objects at the logical level may be semantically related and these relations must be carefully protected. Moreover, several logical objects (e.g., different views) may correspond to the same physical object (e.g., the same file). These issues do not have to be considered when protecting data in an operating system.

Therefore, it is necessary that a DBMS is equipped with its own security services. Of course it can also exploit the security services provided by the underlying operating system, as well as those enforced at the hardware and network level. Generally, each security property is ensured by more than one DBMS service. In particular, the *access control mechanism* ensures data secrecy. Whenever a subject tries to access an object, the access control mechanism checks the right of the subject against a set of *authorizations*, stated usually by some Security Administrators or users. The access is granted only if it does not conflict with the stated authorizations. An *authorization* states which subject can perform which action on which object and, optionally, under which condition. Authorizations are granted according to the security policies of the organization. Data confidentiality is also obtained through the use of encryption techniques, either applied to the data stored on secondary storage or when data are transmitted on a network, to avoid that an intruder intercepts the data and accesses their contents. Data integrity is jointly ensured by the access control mechanism and by semantic integrity constraints. Similarly to confidentiality enforcement, whenever a subject tries to modify some data, the access control mechanism verifies that the subject has the right to modify the data, according to the specified authorizations and, only in this case, it authorizes the update request. Additionally, current DBMSs allow one to specify a set of *integrity constraints*, using SQL in case of an RDBMS, that expresses correctness conditions on the stored data, and therefore avoids incorrect data updates and deletions. These constraints are automatically checked by the constraint checker subsystem upon the request for each update operation. Furthermore, digital signature techniques can be applied to detect improper data modifications. They are also used to ensure data authenticity. Finally, the recovery subsystem and the concurrency control mechanism ensure that data are available and correct despite hardware and software failures and despite data accesses from concurrent application programs. In particular, to properly recover the correct state of the database after a failure, all data accesses are logged. Log files can then be further used for *auditing* activities,

Database Security. Table 1. Security requirements and enforcement techniques

Security properties	Techniques
Secrecy	Access control mechanism, data encryption
Integrity	Access control mechanism, semantic integrity constraints, digital signatures
Availability	Recovery subsystem, concurrency control mechanism,
	techniques preventing DoS attacks
Authenticity	Digital signatures
Completeness	Non standard digital signatures

that is, they can be analyzed by an *intrusion detection system* to discover possible security breaches and their causes. Data availability, especially for data that are available on the web, can be further enhanced by the use of techniques avoiding query floods [2] or other Denial-of-Service (DoS) attacks. Completeness enforcement is a quite new research direction, which is particularly relevant when data are outsourced to (untrusted) publishers for their management [12]. It can be ensured through the use of nonstandard digital signature techniques, like Merkle trees and aggregation signatures. Table 1 summarizes the security properties discussed so far and the techniques for their enforcement.



Database Security. Figure 1. Security mechanisms.

The above described security services must rely for their proper functioning on some *authentication mechanism*, which verifies whether a user wishing to connect to the DBMS has the proper credentials. Such a mechanism identifies users and confirms their identities. Commonly used authentication mechanisms are based on the use of login and password, whereas more sophisticated schemes include those using biometric information, or token-based authentication.

The security mechanisms discussed so far and their interactions are graphically summarized in Fig. 1. Additionally, security mechanisms should be devised when data are accessed through web applications. These applications can be exploited to perpetrate one of the most serious threats to web-accessible databases, that is, *SQL-Injection Attacks* (SQLIAs) [10]. In the worst case, these attacks may cause the intruder to gain access to all the data stored into the database, by by-passing the security mechanisms, and, therefore, it gives him or her the power to leak, modify, or delete the information that is stored in the database. SQLIAs are very widespread, as reported by a study by Gartner Group over 300 Internet websites, which has shown that most of them could be vulnerable to SQLIAs. SQLIAs basically exploit insufficient input validation in the application code. One of the simplest form of SQLJ attack is that of inserting SQL meta-characters into web-based input fields to manipulate the execution of the back-end SQL queries. Although several techniques to prevent such attacks have been recently proposed [10], there are so many variants of SQLIAs that finding a complete solution to these threats is still a challenging research issue.

Key Applications

Database security services are nowadays used in any application environment that exploits a DBMS to manage data, including Healthcare systems, Banking and financial applications, Workflow Management Systems, Digital Libraries, Geographical and Multimedia management systems, E-commerce services, Publish-subscribe systems, Data warehouses.

Cross-references

- ► Access Control
- ► Auditing and Forensic Analysis
- ► Authentication
- ► Concurrency Control-Traditional Approaches
- ► Data Encryption
- ► Digital Signatures
- ► Intrusion Detection Technologies
- ► Merkle Trees
- ► Privacy
- ► Privacy-Enhancing Technologies
- ► Privacy-Preserving Data Mining
- ► Secure Data Outsourcing
- ► Secure Database Development
- ► Security Services

Recommended Reading

1. Air Force Studies Bord and Committee on Multilevel data management security. Multilevel data management security. National Academy, WA, USA, 1983.
2. Bertino E., Laggieri D., and Terzi E. Securing DBMS: characterizing and detecing query flood. In Proc. Ninth Information Security Conference (ISC'04). Springer, 2004, pp. 195–206.
3. Bertino E. and Sandhu R.S. Database security: concepts, approaches, and challenges. IEEE Trans. Depend. Secure Comput., 2(1):2–19, 2005.
4. Brainard J., Juels A., Rivest R.L., Szydlo M., and Yung M. Fourth-factor authentication: somebody you know. In Proc. 13th ACM Conf. on Computer and Communications Security. USA, 2006.
5. Carminati B., Ferrari E., and Thuraisingham B.M. Access control for web data: models and policy languages. Annals Telecomm., 61(3–4):245–266, 2006.
6. Castano S., Fugini M.G., Martella G., and Samarati P. Database security. Addison-Wesley, Reading, MA, 1995.
7. Damiani M.L. and Bertino E. Access control systems for geo-spatial data and applications. In Modelling and Management of Geographical Data over Distributed Architectures, A. Belussi, B. Catania, E. Clementini, E. Ferrari (eds.). Springer, 2007, pp. 189–214.
8. Ferraiolo D.F., Sandhu R.S., Gavrila S.I., Kuhn D.R., and Chandramouli R. Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur., 4(3):224–274, 2001.
9. Ferrari E. and Thuraisingham B.M. Secure database systems. In Advanced Databases: Technology and Design, O. Diaz, M. Piattini (eds.). Artech House, London, 2000.
10. Halfond W.G., Viegas J., and Orso A. A classification of SQL-injection attacks and countermeasures. Int. Symp. on Secure Software Engineering (ISSSE 2006). 2006.
11. Leino-Kilpi H., Valimaki M., Dassen T., Gasull M., Lemonidou C., Scott A., and Arndt M. Privacy: a review of the literature. Int. J. Nurs. Stud., (38):663–671, 2001.
12. Pang H. and Tan K.L. Verifying completeness of relational query answers from online servers. ACM Trans. Inf. Syst. Secur., 11(2), 2008, article no. 5.
13. Pfleeger C.P. and Pfleeger S.L. Security in computing, 3rd edn. Prentice-Hall, Upper Saddle River, NJ, USA, 2002.

Database Socket

► Connection

Database Storage Layer

► Storage Access Models

Database Techniques to Improve Scientific Simulations

Biswanath Panda, Johannes Gehrke,
Mirek Riedewald
Cornell University, Ithaca, NY, USA

Synonyms

Indexing for online function approximation

Definition

Scientific simulations approximate real world physical phenomena using complex mathematical models. In most simulations, the mathematical model driving the simulation is computationally expensive to evaluate and must be repeatedly evaluated at several different parameters and settings. This makes running large-scale scientific simulations computationally expensive. A common method used by scientists to speed up simulations is to store model evaluation results at some parameter settings during the course of a simulation and reuse the stored results (instead of direct model evaluations) when similar settings are encountered in later stages of the simulation. Storing and later retrieving model evaluations in simulations can be modeled as a high dimensional indexing problem. Database techniques for improving scientific simulations focus on addressing the new challenges in the resulting indexing problem.

Historical Background

Simulations have always been an important method used by scientists to study real world phenomena. The general methodology in these application areas is similar. Scientists first understand the physical laws that govern the phenomenon. These laws then drive a mathematical model that is used in simulations as an approximation of reality. In practice scientists often face serious computational challenges. The more realistic the model, the more complex the corresponding mathematical equations. As an example, consider the simulation of a combustion process that motivated the line of work discussed in this entry. Scientists study how the composition of gases in a combustion chamber changes over time due to chemical reactions. The composition of a gas particle is described by a high-dimensional vector (10–70 dimensions). The simulation consists of a series of time steps. During each time step some particles in the chamber react, causing their compositions to change. This reaction is described by a complex high-dimensional function called the reaction function, which, given the current composition vector of a particle and other simulation properties, produces a new composition vector for the particle. Combustion simulations usually require up to 10^8–10^{10} reaction function evaluations each of which requires in the order tens of milliseconds of CPU time. As a result, even small simulations can run into days.

Due to their importance in engineering and science, many algorithms have been developed to speed up combustion simulations. The main idea is to build an approximate model of the reaction function that is cheaper to evaluate. Early approaches were offline, where function evaluations were collected from simulations and used to learn multivariate polynomials approximating the reaction function [8]. These polynomials were then used later in different simulations instead of the reaction function. Recently, more sophisticated models like neural networks and self organizing maps have also been used [3]. The offline approaches were not very successful because a single model could not generalize to a large class of simulations. In 1997, Pope developed the In Situ Adaptive Tabulation (ISAT) algorithm [7]. ISAT was an online approach to speeding up combustion simulations. The algorithm cached reaction function evaluations at certain frequently seen regions in the composition space and then used these cached values to approximate the reaction function evaluations at compositions encountered later on in the simulation. The technique was a major breakthrough in combustion simulation because it enabled scientists to run different simulations without having to first build a model for the reaction function. Until today, the Algorithm remains the state of the art for combustion simulations. Several improvements to ISAT have been proposed. DOLFA [9] and PRISM [1] proposed alternative methods of caching reaction function evaluations. More recently, Panda et al. [6] studied the storage/retrieval problem arising out of caching/reusing reaction function evaluations in ISAT and this entry mainly discusses their observations and findings.

Foundations

Even though the ISAT algorithm was originally proposed in the context of combustion simulations the algorithm can be used for building an approximate model for any high dimensional function (f), that is expensive to compute. This section begins with a discussion on local models, that represent the general class of models built by ISAT (section "Local Models"). This is followed by a description of the ISAT Algorithm that uses selective evaluations of the expensive function f to build a local model (section "ISAT Algorithm"). The algorithm introduces a new storage/retrieval, and hence indexing, problem. The section then discusses the indexing problem in detail: its challenges and solutions that have been proposed (sections "Indexing Problem" and "An Example: Binary Tree") and concludes with some recent work on optimizing long running simulations (section "Long Running Simulations").

Local Models

Local models are used in many applications to approximate complex high dimensional functions. Given a function $f : \mathbf{R}^m \rightarrow \mathbf{R}^n$; a local model defines a set of high dimensional regions in the function domain: $\mathcal{R} = \{R_1 \ldots R_n | R_i \subseteq \mathbf{R}^m\}$. Each region R_i is associated with a function $\hat{f}_{R_i} : R_i \rightarrow \mathbf{R}^n$; such that $\forall \mathbf{x} \in R_i : \ ||\hat{f}_{R_i}(\mathbf{x}) - f(\mathbf{x})\ || \leq \epsilon$; where ε is a specified error tolerance in the model and || is some specified error metric such as the Euclidean distance. Using a local model to evaluate f at some point $\mathbf{x}$ in the function domain first involves finding a region $(R \in \mathcal{R})$ that contains $\mathbf{x}$ and then evaluating $\hat{f}_R(\mathbf{x})$ as an approximation to $f(\mathbf{x})$.

ISAT Algorithm

Main algorithm: ISAT is an online algorithm for function approximation; its pseudocode is shown in Fig. 1. The algorithm takes as input a query point $\mathbf{x}$ at which the function value must be computed and a search structure S that stores the regions in $\mathcal{R}$. S is empty when the simulation starts. The algorithm first tries to compute the function value at $\mathbf{x}$ using the local model it has built so far (Lines 2–3). If that fails the algorithm computes $f(\mathbf{x})$ using the expensive model (Line 5) and uses $f(\mathbf{x})$ to update existing or add new regions in the current local model (Line 6). The algorithm is online because it does not have access to all query points when it builds the model.

```
   ISAT Algorithm
1: Require: Query Point x, inde x structure $S$
2: if $\exists\langle R, \hat{f}_R\rangle \in S$ such that x $\in R$
3:     Compute y = $\hat{f}$ (x)
4: else
5:     Compute y = $f$(x)
6:     Update($S$, x, $f$(x))
7: end if
8: return y
```

Database Techniques to Improve Scientific Simulations. Figure 1. Pseudocode of the ISAT algorithm.

Model updating: ISAT updates the local model using a strategy outlined in Fig. 2. In general it is extremely difficult to exactly define a region R and an associated $\hat{f}_R$, such that $\hat{f}_R$ approximates f in all parts of R. ISAT proposes a two step process to discover regions. It initially starts with a region that is very small and conservative but where it is known that a particular $\hat{f}_R$ approximates f well. It then gradually grows the conservative approximations over time. More specifically the update process first searches the index S for regions where $\mathbf{x}$ lies outside the region but $||\hat{f}_R(\mathbf{x}) - f(\mathbf{x})|| \leq \epsilon$. Such regions are grown to contain $\mathbf{x}$ (Lines 2–7). If no existing regions can be grown a new conservative region centered around $\mathbf{x}$ and associated $\hat{f}_R$ is added to the local model (Line 9). The grow process described is a heuristic that works well in practice for functions that are locally smooth. This assumption holds in combustion and in most other applications.

Instantiation: While the shape of the regions and associated functions can be anything, the original ISAT algorithm proposed high dimensional ellipsoids as regions and used a linear model as the function in a region. The linear model is initialized by computing the f value and estimating the derivative of f at the center of the ellipsoidal region:

$$\hat{f}_R(\mathbf{x}) = f(\mathbf{a}) + F'_{\mathrm{a}} \times (\mathbf{x} - \mathbf{a}),$$

where $\mathbf{a}$, is the center of the region R and $F'_{\mathbf{a}}$ is the derivative at $\mathbf{a}$.

ISAT performs one of the following high level operations for each query point $\mathbf{x}$. **Retrieve**: Computing the function value at $\mathbf{x}$ using the current local model by searching for a region containing $\mathbf{x}$. **Grow**: Searching for regions that can be grown to contain $\mathbf{x}$ and updating these regions in S. **Add**: Adding a new region (R) and an associated $\hat{f}_R$ into S.

```
Updating a local model
1:  Require: $S$,x,$f$(x)
2:  if $\exists\langle R, \hat{f}_R\rangle \in S$ : x can be included in $R$
3:      for all $\langle R, \hat{f}_R\rangle \in S$
4:          if x can be included in $R$
5:              Update $\langle R, \hat{f}_R\rangle$ to includex
6:          end if
7:      end for
8:  else
9:      Add new $\langle R, \hat{f}_R\rangle$ to $S$
10: end if
```

Database Techniques to Improve Scientific Simulations. Figure 2. Pseudocode for updating a local model.

Indexing Problem

The indexing problem in function approximation produces a challenging workload for the operations on index S in Figs. 1 and 2. The retrieve requires the index to support fast lookups. The grow requires both a fast lookup to find growable ellipsoids and then an efficient update process once an ellipsoid is grown. Finally, an efficient insert operation is required for the add step. There are two main observations which make this indexing problem different from traditional indexing [2,4]:

- The regions that are stored in the index are not predefined but generated by the add and grow operations. Past decisions about growing and adding affect future performance of the index, therefore the algorithm produces an uncommon query/update workload.
- The framework in which indexes must be evaluated is very different. Traditionally, the performance of index structures has been measured simply in terms of the cost of a search and in some cases update. There are two distinct cost factors in the function approximation problem. First, there are the costs associated with the search and update operations on the index. Second, there are costs of the function approximation application which include function evaluations and region operations. Since the goal of function approximation is to minimize the total cost of the simulation, all these costs must be accounted for when evaluating the performance of an index.

In the light of these observations a principled analysis of the various costs in the function approximation algorithm leads to the discovery of novel tradeoffs. These tradeoffs produce significant and different effects on different index structures. Due to space constraints a high-level discussion of the various costs in the algorithm and the associated tradeoffs are discussed here. The remainder of this section briefly describes the tradeoffs and the tuning parameters that have been proposed to exploit the different tradeoffs. The indexing problem is studied here using the concrete instantiation of the ISAT algorithm using ellipsoidal regions with linear models. Therefore, regions are often referred to as ellipsoids in the rest of the section. However, it is important to note that the ideas discussed are applicable to any kind of regions and associated functions.

Tuning Retrieves

In most high dimensional index structures the ellipsoid containing a query point is usually not the first ellipsoid found. The index ends up looking at a number of ellipsoids before finding “the right one.” The additional ellipsoids that are examined by the index are called false positives. For each false positive the algorithm pays to search and retrieve the ellipsoid from the index and to check if the ellipsoid contains the query point. In traditional indexing problems, if an object that satisfies the query condition exists in the index, then finding this object during search is mandatory. Therefore, the number of false positives is a fixed property of the index. However, the function approximation problem provides the flexibility to tune the number of false positives, because the expensive function can be evaluated if the index search was not successful. The number of false positives can be tuned by limiting the number of ellipsoids examined during the retrieve step. This parameter is denoted by Ellr. Ellr places an upper bound on the number of false positives for a query. Tuning Ellr controls several interesting effects.

- *Effect 1:* Decreasing Ellr reduces the cost of the retrieve operation as less number of ellipsoids are retrieved and examined.
- *Effect 2:* Decreasing Ellr decreases the chances of finding an ellipsoid containing the query point thereby resulting in expensive function evaluations.
- *Effect 3:* Misses that result from decreasing Ellr can grow and add other ellipsoids. These grows and adds index new parts of the domain and also change the overall structure of the index. Both of these affect the probability of retrieves for future queries. This is a more subtle effect unique to this problem.

Tuning Grows and Adds

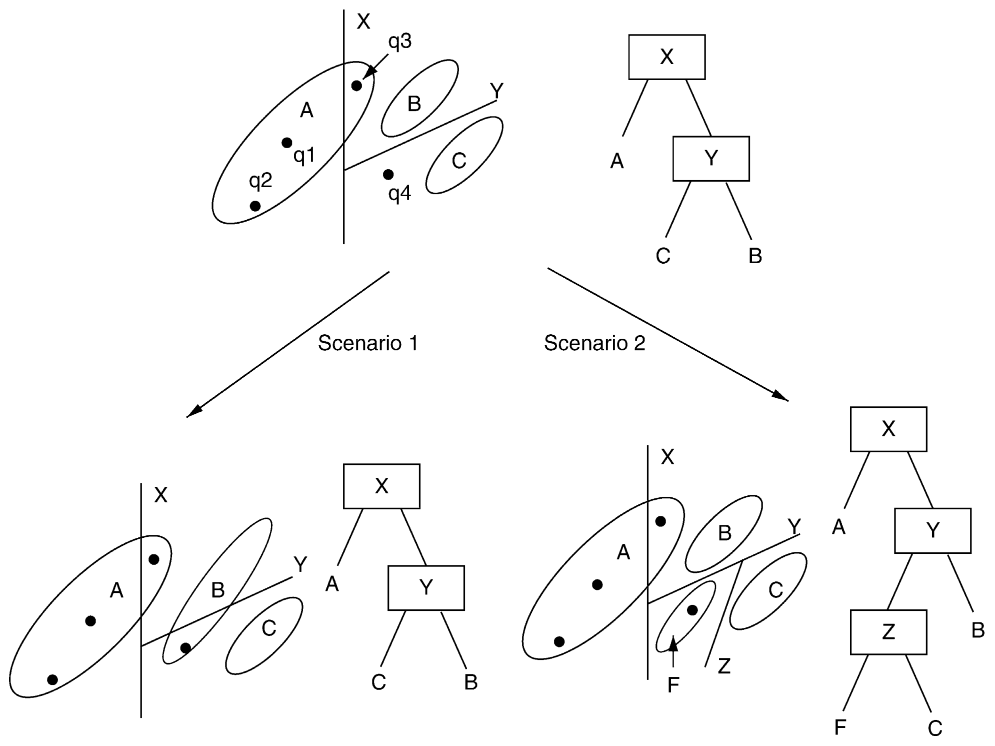
Just like the retrieve, the grow and add operations can be controlled by the number of ellipsoids examined for growing denoted as Ellg. Since an add is performed only if a grow fails, this parameter controls both the operations. Ellg provides a knob for controlling several effects that affect performance of the index and the algorithm.

- *Effect 4:* The first part of the grow process involves traversing the index to find ellipsoids that can be grown. Decreasing Ellg reduces the time spent in the traversal.
- *Effect 5:* Decreasing Ellg decreases the number of ellipsoids examined for the grow and hence the number of ellipsoids actually grown. This results in the following effects.
  - *Effect 5a:* Reducing the number of ellipsoids grown reduces index update costs that can be significant in high dimensional indexes.
  - *Effect 5b:* Growing a large number of ellipsoids on each grow operation indexes more parts of the function domain, thereby improving the probability of future retrieves.
  - *Effect 5c:* Growing a large number of ellipsoids on each grow results in significant overlap among ellipsoids. Overlap among objects being indexed reduces search selectivity in many high dimensional indexes.
- *Effect 6:* Decreasing Ellg increases the number of add operations. Creating a new region is more expensive than updating an existing region since it involves initializing the function $\hat{f}_R$ in the new region.

In summary, the two tuning parameters have many different effects on index performance and the cost of the simulation. What makes the problem interesting is that these effects often move in opposite directions. Moreover, tuning affects indexes differently and to varying degrees, which makes it necessary to analyze each index individually.

An Example: Binary Tree

The previous section presented a qualitative discussion of the effects that tuning Ellr and Ellg can have on index performance and simulation cost. This section makes the effects outlined in the previous section more concrete using an example index structure, called the Binary Tree. The tree indexes the centers of the ellipsoids by recursively partitioning the space with cutting



Database Techniques to Improve Scientific Simulations. Figure 3. Binary tree.

planes. Leaf nodes of the tree correspond to ellipsoid centers and non-leaf nodes represent cutting planes. Figure 3 shows an example tree for three ellipsoids *A*, *B*, *C* and two cutting planes *X* and *Y*. Next, this entry focuses on the tree in the top part of Fig. 3 and describe the operations supported by the index.

Retrieve

There are two possible traversals in the index that result in a successful retrieve.

Primary Retrieve. The first called a Primary Retrieve is illustrated with query point $\mathbf{q}_2$. The retrieve starts at the root, checking on which side of hyperplane *X* the query point lies. The search continues recursively with the corresponding subtree, the left one in the example. When a leaf node is reached, the ellipsoid in the leaf is checked for the containment of the query point. In the example, *A* contains $\mathbf{q}_2$, and hence, a successful retrieve.

Secondary Retrieve. Since the binary tree only indexes centers, ellipsoids can straddle cutting planes, e.g., *A* covers volume on both sides of cutting plane *X*. If ellipsoids are straddling planes, then the Primary Retrieve can result in a false negative. For example, $\mathbf{q}_3$ lies to the right of *X* and so the Primary Retrieve fails even though there exists an ellipsoid *A* containing it. To overcome this problem the Binary Tree performs a more expensive Secondary Retrieve if the Primary fails. The main idea of the Secondary Retrieve is to explore the "neighborhood" around the query point by examining "nearby" subtrees. In the case of $\mathbf{q}_3$, the failed Primary Retrieve ended in leaf *B*. Nearby subtrees are explored by moving up a level in the tree and exploring the other side of the cutting plane. Specifically, *C* is examined first(after moving up to *Y*, *C* is in the unexplored subtree). Then the search would continue with *A* (now moving up another level to *X* and accessing the whole left subtree). This process continues until a containing ellipsoid is found, or Ellr ellipsoids have been examined unsuccessfully.

Update

Scenario 1 (Grow) and 2 (Add) of Fig. 3 illustrate the update operations on the index.

Grow. The search for growable ellipsoids proceeds in exactly the same way as a Secondary Retrieve, starting where the failed Primary Retrieve ended. Assume that in the example in Fig. 3, ellipsoid *B* can be grown to include $\mathbf{q}_4$, but *C* and *A* cannot. After the retrieve failed, the grow operation first attempts to grow *C*. Then it continues to examine *B*, then *A* (unless Ellg < 3). *B* is grown to include $\mathbf{q}_4$, as shown on the bottom left (Scenario 1). Growing of *B* made it straddle hyperplane *Y*. Hence, for any future query point near $\mathbf{q}_4$ and "below" *Y*, a Secondary Retrieve is necessary to find containing ellipsoid *B*, which is "above" *Y*.

Add. The alternative to growing *B* is illustrated on the bottom right part of Fig. 3 (Scenario 2). Assume Ellg = 1, i.e., after examining *C*, the grow search ends unsuccessfully. Now a new ellipsoid *F* with center $\mathbf{q}_4$ is added to the index. This is done by replacing leaf *C* with an inner node , which stores the hyper-plane that best separates *C* and *F*. The add step requires the expensive computation of *f*, but it will enable future query points near $\mathbf{q}_4$ to be found by a Primary Retrieve.

Tuning parameter Ellg affects the Binary Tree in its choice of scenario 2 over 1. This choice, i.e., performing an add instead of a grow operation, reduces false positives for future queries, but adds extra-cost for the current query. Experiments on real simulation workloads have shown that this tradeoff has a profound influence on the overall simulation cost [6].

Long Running Simulations

When ISAT is used in long running combustion simulations ($\geq 10^8$ time steps), updates to the local model are unlikely after the first few million queries and the time spent in building the local model is very small compared to the total simulation time. Based on these observations, Panda et al. have modeled a long running combustion simulation as a traditional supervised learning problem [5]. They divide a combustion simulation into two phases. During the first phase, the ISAT algorithm is run and at the same time $(\mathbf{x},f(\mathbf{x}))$ pairs are sampled uniformly from the composition space accessed by the simulation. At the end of the first phase, the sampled $(\mathbf{x},f(\mathbf{x}))$ pairs are used as training data for a supervised learning algorithm that tries to find a "new" local model with lower retrieve cost than the model built using ISAT. This new model is then used for the remainder of the simulation. Their experiment shows that the algorithm adds little overhead and that the new model can reduce retrieve costs by up to 70% in the second phase of the simulation.

Key Applications

The ISAT algorithm and its optimizations have primarily been applied to combustion simulation workloads.

However, the ideas are applicable to any simulation setting where repeated evaluations in a fixed domain of a function that is locally smooth and expensive to compute are required.

Cross-references

► Spatial and Multidimensional Databases

Recommended Reading

1. Bell J.B., Brown N.J., Day M.S., Frenklach M., Grcar J.F., Propp R.M., and Tonse S.R. Scaling and efficiency of PRISM in adaptive simulations of turbulent premixed flames. In Proc. 28th Int. Combustion Symp., 2000.
2. Böhm C., Berchtold S., and Keim D.A. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. ACM Comput. Surv., 33(3):322–373, 2001.
3. Chen J.Y., Kollmann W., and Dibble R.W. A self-organizing-map approach to chemistry representation in combustion applications. Combustion Theory and Modelling, Vol. 4. 2000, pp. 61–76.
4. Gaede V. and Günther O. Multidimensional access methods. ACM Comput. Surv., 30(2):170–231, 1998.
5. Panda B., Riedewald M., Gehrke J., and Pope S.B. High speed function approximation. In Proc. 2007 IEEE Int. Conf. on Data Mining, 2007.
6. Panda B., Riedewald M., Pope S.B., Gehrke J., and Chew L.P. Indexing for function approximation. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.
7. Pope S.B. Computationally efficient implementation of combustion chemistry using in situ adaptive tabulation. Combust. Theory Model., 1997, 1:41–63.
8. Turanyi T. Application of repro-modeling for the reduction of combustion mechanisms. In Proc. 25th Symp. on Combustion, 1994, pp. 949–955.
9. Veljkovic I., Plassmann P., and Haworth D.C. A scientific on-line database for efficient function approximation. In Proc. Int. Conf. on Computational Science and its Applications, 2003, pp. 643–653.

Database Trigger

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Synonyms

Triggers

Definition

A database trigger is code that is executed in response to events that have been generated by database commands such as INSERT, DELETE, or UPDATE.

Key Points

Triggers are code that are executed in response to events that have been generated before or after a database operation. They are sometimes separated as pre- and post-triggers in the literature. A pre-trigger can be used as an extra validation check before the database command is executed, whereas a post-trigger can be used as a notification that the database command has been executed.

Triggers can be classified according to trigger granularity: *row-level triggers* or *statement-level triggers*. In case of row-level triggers, each row will generate an event, whereas statement-level triggers occur only once per database command.

Overviews of database triggers can be found in [1,2].

Cross-references

► Active Database (aDB)
► Active Database (Management) System (aDBS/aDBMS)
► ECA-rules
► Event

Recommended Reading

1. Kulkarni K.G., Mattos N.M., and Cochrane R. Active Database Features in SQL3. In Active Rules in Database Systems. 1999, pp. 197–219.
2. Sudarshan S., Silberschatz A., and Korth H. Triggers, chap. 8.6. 2006, pp. 329–334.

Database Tuning using Combinatorial Search

Surajit Chaudhuri[1], Vivek Narasayya[1], Gerhard Weikum[2]
[1]Microsoft Corporation, Redmond, WA, USA
[2]Max-Planck Institute for Informatics, Saarbrücken, Germany

Definition

Some database tuning problems can be formulated as combinatorial search, i.e. the problem of searching over a large space of discrete system configurations to find an appropriate configuration. One tuning problem where feasibility of combinatorial search has been demonstrated is physical database design. As part of the *self-management* capabilities of a database system, it is desirable to develop techniques for automatically recommending an appropriate physical design configuration to optimize database system performance. This entry describes the application of combinatorial search techniques to the problem of physical database design.

Historical Background

Combinatorial search (also referred to as combinatorial optimization) [8] is branch of optimization where the set of feasible solutions (or configurations) to the problem is *discrete*, and the goal is to find the "best" possible solution. Several well-known problems in computer science such as the Traveling Salesman Problem, Minimum Spanning Tree Problem, Knapsack Problem etc. can be considered as examples of combinatorial search. Several combinatorial search problems have been shown to be NP-Hard, and exact algorithms that guarantee optimality are not scalable. In such cases, heuristic search algorithms such as greedy search, simulated annealing, genetic algorithms etc. are often used to ensure scalability.

In the area of database tuning problems, the combinatorial search paradigm has been successfully used for problems such as query optimization [2] and physical database design tuning [5]. These are described in more details below.

Foundations

Some of the key aspects of combinatorial search are:

- The *search space*, i.e. the space of discrete configurations from which the solution is picked.
- A *metric* for evaluating the "goodness" of a configuration in the search space. This is essential for being able to quantitatively compare different configurations.
- A *search algorithm* for efficiently searching the space to find a configuration with the minimum (or maximum) value of the goodness metric.

One early example of use of combinatorial search for tuning in database systems is query optimization (see [2] for an overview on query optimization in relational database systems). The goal of query optimization is to produce an efficient execution plan for a given query using the physical operators supported by the underlying execution engine. The above key aspects of combinatorial search are now illustrated for the problem of query optimization. The *search space* of execution plans considered by a query optimizer depends on: (i) The *set of algebraic transformations* that preserve equivalence of query expressions (e.g. commutativity and associativity of joins, commutativity of group by and join, rules for transforming a nested subquery into a single-block query, etc.) (ii) The *set of physical operators* supported in the system (e.g. three different physical operators Index Nested Loops Join, Hash Join, Merge Join for implementing a join). Query optimizers use a *cost model* that defines the goodness *metric* for evaluating the "goodness" of an execution plan. In particular for a given execution plan, the cost model computes an overall number based on estimates of the CPU, I/O and communication costs of physical operators in the execution plan. Finally, different kinds of *search algorithms* are used in today's query optimizers including bottom-up approaches (e.g. in the Starburst query optimizer [7]) as well as top-down approaches (e.g. optimizers based on the Volcano/Cascades [6] framework).

Example: Physical Database Design using Combinatorial Search

An in-depth example is now considered, the problem of physical database design. A crucial property of a relational DBMS is that it provides physical data independence. This allows physical structures such as indexes and materialized views to be added or dropped without affecting the output of the query; but such changes do impact efficiency. Thus, together with the capabilities of the execution engine and the optimizer, the physical database design determines how efficiently a query is executed on a DBMS. Over the years, the importance of physical design has become amplified as query optimizers have become sophisticated to cope with complex decision support queries. Since query execution and optimization techniques have become more sophisticated, database administrators (DBAs) can no longer rely on a simplistic model of the engine. Thus, tools for automating physical database design can ease the DBA's burden and greatly reduce the total cost of ownership. For an overview of work in the area of physical database design, refer to [5].

The role of the *workload*, including SQL queries and updates, in physical design is widely recognized. Therefore, the problem of physical database design can be stated as: For a given workload, find a configuration, i.e. a set of indexes and materialized views that minimize the cost of the given workload. Typically, there is a constraint on the amount of storage space that the configuration is allowed to take. Since the early 1970s, there has been work on trying to automate physical database design. However, it is only in the past decade that automated physical database design tools have become part of all major commercial DBMS vendors [5].

Search Space: The search space of configurations (i.e. set of physical design structures) for a given query (and hence the given workload) can be very large. First, the set of physical design structures that are *relevant* to a query i.e., can potentially be used by the query optimizer to answer the query, is itself large. For example, consider a query with *n* selection predicates. An index defined on any subset of the columns referenced in the predicates is relevant. The order of columns in the index is also significant; thus, in principle any permutation of the columns of the subset also defines a relevant index. The space of relevant materialized views is larger than the space of indexes since materialized views can be defined on any subset of tables referenced in the query. Finally, the search space of *configurations* for the physical database design problem is the power set of all *relevant indexes and materialized views*.

Today's physical database design tools approach the issue of large search space by adopting heuristics for restricting the search space. For example in [3], a key decision is to define the search space as consisting of the union of (only) the *best* configurations for each query in the workload, where the best configuration itself is the one with lowest optimizer estimated cost for the query. Intuitively, this *candidate selection step* leverages the idea that an index (or materialized view) that is not part of an optimal (or close to optimal) configuration for at least one query, is unlikely to be optimal for the entire workload. To improve the quality of the solution in the presence of constraints (e.g. a storage bound), the above space is augmented with an additional set of indexes (and materialized views) derived by "merging" or "reducing" structures from the above space (e.g. [1]). These additional candidates exploit commonality across queries in the workload, and even though they may not be optimal for any individual query in the workload, they can be optimal for the workload as a whole in the presence of the constraint.

Metric: It is not feasible to estimate goodness of a configuration for a workload by actual creation of physical design structures and then executing the queries and updates in the workload. Early papers on physical design tuning used an external model to estimate the cost of a query for a given configuration. However, this has the fundamental problem that the decisions made by the physical design tool could be "out-of-sync" with the decisions made by the query optimizer. This can lead to a situation where the physical design tool recommends an index that is never used by the query optimizer to answer any query in the workload.

In today's commercial physical design tools, the goodness of a configuration for a query is measured by the *optimizer estimated cost* of the query for that configuration. Unlike earlier approaches that used an external cost model, this approach has the advantage that the physical design tool is "in-sync" with the query optimizer.

One approach for enabling this measure of goodness is by making the following key server-side enhancements: (i) Efficient creation of a hypothetical (or "what-if") index. This requires metadata changes to signal to the query optimizer the presence of a what-if index (or materialized view). (ii) An extension to the "Create Statistics" command to efficiently generate the statistics that describe the distribution of values of the column(s) of a what-if index via the use of sampling. (iii) A query optimization mode that enabled optimizing a query for a selected subset of indexes (hypothetical or actually materialized) and ignoring the presence of other access paths. This is important as the alternative would have been repeated creation and dropping of what-if indexes, a potentially costly solution. For more details, refer to [4].

Search algorithm: Given a workload and a set of candidate physical design structures (e.g. obtained as described above using the candidate selection step), the goal of the search algorithm is to efficiently find a configuration (i.e., subset of candidates) with the smallest total optimizer cost for the workload. Note that the problem formulation allows the specification of a set of constraints that the enumeration step must respect, (e.g. to respect a storage bound). Since the index selection problem has been shown to be NP-Hard [9], the focus of most work has been on developing heuristic solutions that give good quality recommendations and can scale well.

One important observation is that solutions that naively stage the selection of different physical design structures (e.g., select indexes first followed by materialized views, select partitioning for table first followed by indexes etc.) can result in poor recommendations. This is because: (i) The choices of these structures interact with one another (e.g., optimal choice of index can depend on how the table is partitioned and vice versa). (ii) Staged solutions can lead to redundant recommendations. (iii) It is not easy to determine a priori how to partition the storage bound across different kinds of physical design structures. Thus, there is a need for integrated recommendations that search the combined space in a scalable manner.

Broadly the search strategies explored thus far can be categorized as bottom-up (e.g. [3]) or top-down [1] search, each of which has different merits. The bottom up strategy begins with the empty (or pre-existing configuration) and adds structures in a greedy manner. This approach can be efficient when available storage is low, since the best configuration is likely to consist of only a few structures. In contrast, the top-down approach begins with a globally optimal configuration but it could be infeasible if it exceeds the storage bound. The search strategy then progressively refines the configuration until it meets the storage constraints. The top-down strategy has several key desirable properties and this strategy can be efficient in cases where the storage bound is large.

Future Directions

The paradigm of combinatorial search has been effectively used in database tuning problems such as query optimization and physical database design. It is future research to consider if this paradigm can also be effectively applied to other database tuning problems such as capacity planning and optimizing database layout.

Cross-references

► Administration Wizard
► Index Tuning
► Physical Layer Tuning
► Self-Management Technology in Databases

Recommended Reading

1. Bruno N. and Chaudhuri S. Automatic physical design tuning: a relaxation based approach. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
2. Chaudhuri S. An overview of query optimization in relational systems. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 1998.
3. Chaudhuri S. and Narasayya V. An efficient cost driven index selection tool for microsoft SQL server. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997.
4. Chaudhuri S. and Narasayya V. AutoAdmin "What-If" index analysis utility. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998.
5. Chaudhuri S. and Narasayya V. Self-tuning database systems: a decade of progress. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
6. Graefe G. The Cascades framework for query optimization. Data Eng. Bull., 18(3), 1995.
7. Haas L., Freytag C., Lohman G., and Pirahesh H. Extensible query processing in Starburst. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989.
8. Papadimitriou C.H. and Steiglitz K. Combinatorial optimization: algorithms and complexity. Dover, July 1998. ISBN 0-486-40258-4.
9. Piatetsky-Shapiro G. The optimal selection of secondary indices is NP-complete. SIGMOD Rec., 13(2):72–75, 1983.

Database Tuning using Online Algorithms

Nicolas Bruno[1], Surajit Chaudhuri[1],
Gerhard Weikum[2]
[1]Microsoft Corporation, Redmond, WA, USA
[2]Max-Planck Institute for Informatics, Saarbrücken, Germany

Definition

A self-managing database system needs to gracefully handle variations in input workloads by adapting its internal structures and representation to changes in the environment. One approach to cope with evolving workloads is to periodically obtain the best possible configuration for a hypothetical "average" scenario. Unfortunately, this approach might be arbitrarily suboptimal for instances that lie outside the previously determined average case. An alternative approach is to require the database system to continuously tune its internal parameters in response to changes in the workload. This is the *online tuning paradigm*. Although solutions for different problems share the same underlying philosophy, the specific details are usually domain-specific. In the context of database systems, online tuning has been successfully applied to issues such as buffer pool management, statistics construction and maintenance, and physical design.

Historical Background

Database applications usually exhibit varying workload characteristics over time. Moreover, changes in workloads cannot be easily modeled beforehand. As a

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:32

consequence, database systems traditionally provided *offline* tools to make corrections to the system's current configuration. Examples include physical design tools that take a representative workload and return a new database design that would be beneficial in the future, or the possibility of refreshing the statistical information about values stored in database columns. These approaches allow a database administrator to react to environmental changes but only after they had happened (and potentially disrupted a previously well-tuned system).

Although offline tuning tools have been successfully introduced in several commercial database systems, there has been a growing need for additional functionality that is outside the scope of such tools. As database applications increase in size and complexity, it becomes more difficult to even decide when offline tools should be called. Additionally, offline tools are sometimes resource intensive and assume that there are idle periods of time on which they can be applied, which is not necessarily the case. To cope with these new requirements, a new set of algorithms emerged (e.g., [1,2,6,8,11,12]), which are based on a different principle. The idea is to monitor the database system as queries are processed, and in the background reorganize its internal state to cope with changes in the workload. In this way, the database system is continuously modifying itself in a closed "monitor-diagnose-tune" loop.

Foundations

A requirement towards understanding online tuning is to conceptualize the transition from scenarios in which full information is known about the system in consideration (and can therefore identify the optimal solution). Within the online tuning paradigm, only partial information is known as time passes (and therefore it is necessary to approximate optimal solutions at all times without knowing the future). For illustration purposes, some specific examples of online tuning are reviewed briefly, and one such example is provided in more detail.

An example of online tuning is that of automatically managing memory allocation across different memory pools [2,12]. Complex queries generally use memory-intensive operators (e.g., hash-joins) whose performance depends on the amount of memory that is given to each operator at runtime. However, only a finite amount of memory is available at any time, and it has to be distributed among all the competing operators. This problem is further complicated by the fact that new queries are continually being served by the database system, and therefore any static policy to distribute available memory might be inadequate as workloads change. Reference [7] presents an online algorithm to manage memory that is based on the ability of operators to dynamically grow and shrink their own memory pools. By using a feedback loop while statements are being executed, this technique is able to incrementally model the optimal resource allocation and adapt the distribution of memory to the current operators to maximize performance.

Another example of online tuning is physical database design, which is concerned with identifying the best set of redundant access paths (e.g., indexes or materialized views) that would accelerate workloads [5]. While there has been work on offline tools that assume that the representative input workload would repeat indefinitely in the future, many scenarios exhibit unpredictable changes in workload patterns. Online physical design tuning takes a different approach: it monitors incoming queries and measures the relative cost/benefit of each of the present and hypothetical (i.e., not part of the current design) structures. By carefully measuring the impact of creating and dropping structures (e.g., indexes), the system is able to change the underlying database configuration in response to changes in workload characteristics [3].

Expanded Example: Self Tuning Histograms

Consider, as an in-depth example, the problem of statistics management in database systems. Relational query optimization has traditionally relied on single- or multi-column histograms to model the distribution of values in table columns. Ideally, histogram buckets should enclose regions of the data domain with approximately uniform tuple density (i.e., roughly the same number of tuples per unit of space in a bucket), to accurately estimate the result size of range queries. At the same time, histograms (especially multi-column ones) should be sufficiently compact and efficiently computable. Typically, histogram construction strategies inspect the data sets that they characterize without considering how the histograms will be used (i.e., there is an offline algorithm that builds a given histogram, possibly as a result of bad performance of some workload query). The implicit assumption while building such histograms is that all queries are equally likely. This assumption, however, is rarely true in practice, and certain data regions might be much more heavily queried than others. By analyzing workload queries and their results, one could detect buckets that do not have uniform density and "split" them into smaller and more accurate buckets, or realize that some adjacent buckets are too similar and "merge" them, thus recuperating space for more critical regions.

In other words, rather than computing a given histogram at once, without knowledge of how it is going to be used, one can instead incrementally refine the statistical model based on workload information and query feedback. This is the essence of *self-tuning histograms*. Intuitively, self-tuning histograms exploit query workload to zoom in and spend more resources in heavily accessed areas, thus allowing some inaccuracy in the rest. These histograms also exploit query feedback as truly multidimensional information to identify promising areas to enclose in histogram buckets. As a result, the resulting histograms are more accurate for the expected workload than traditional workload-independent ones. Self-tuning histograms can also gracefully adapt to changes in the data distribution they approximate, without the need to periodically rebuild them from scratch.

Reference [1] presents STGrid, the first multidimensional histogram that uses query feedback to refine buckets. STGrid histograms greedily partition the data domain into disjoint buckets that form a grid, and refine their frequencies using query feedback by adjusting the expected cardinality of buckets based on observed values. After a predetermined number of queries have been executed, the histogram is restructured by merging and splitting rows of buckets at a time. Efficiency in histogram tuning is the main goal of this technique, at the expense of accuracy. Since STGrid histograms need to maintain the grid structure at all times, and also due to the greedy nature of the technique, some locally beneficial splits and merges have the side effect of modifying distant and unrelated regions, hence decreasing the overall accuracy of the resulting histograms.

To alleviate this problem, STHoles histograms, introduced in [4], are based on a novel partitioning strategy that is especially well suited to exploit workload information. STHoles histograms allow inclusion relationships among buckets, i.e., some buckets can be completely included inside others. Specifically, each bucket in an *STHoles* histogram identifies a rectangular range in the data domain, similar to other histogram techniques. However, unlike traditional histograms, *STHoles* histograms identify bucket sub-regions with different tuple density and "pull" them out from the corresponding buckets. Hence a bucket can have *holes*, which are themselves first-class histogram buckets. In this way, these histograms implicitly relax the requirement of rectangular regions while keeping rectangular bucket structures. By allowing bucket nesting, the resulting histograms do not suffer from the problems of STGrid histograms and can model complex shapes (not restricted to rectangles anymore); by restricting the way in which buckets may overlap, the resulting histograms can be efficiently manipulated and updated incrementally by using workload information.

STHoles histograms exploit query feedback in a truly multidimensional way to improve the quality of the resulting representation. Initially, an STHoles histogram consists of a single bucket that covers the whole data domain. For each incoming query from the workload, the query optimizer consults existing histograms and produces a query execution plan. The resulting plan is then passed to the execution engine, where it is processed. A build/refine histogram module intercepts the stream of tuples that are returned, and tunes the relevant histogram buckets so that the resulting histogram becomes more accurate for similar queries. Specifically, to refine an *STHoles* histogram, one first intercepts the stream of results from the corresponding query execution plan and counts how many tuples lie inside each histogram bucket. Next, one determines which regions in the data domain can benefit from using this new information, and refines the histogram by "drilling holes," or zooming into the buckets that cover the region identified by the query plan. Finally, to adhere to the budget constraint, it is possible to consolidate the resulting histogram by merging similar buckets.

Recently, [10] introduces ISOMER (Improved Statistics and Optimization by Maximum-Entropy Refinement), a new algorithm for feedback-driven histogram construction. ISOMER uses the same partitioning strategy as STHoles, but it is based on a more efficient algorithm to restructure the histogram, which does not require counting the number of tuples that lie within each histogram bucket. In contrast, ISOMER uses the information-theoretic principle of maximum entropy to approximate the true data distribution by a histogram distribution that is as "simple" as possible while being consistent with all the previously observed cardinalities. In this manner, ISOMER avoids incorporating extraneous (and potentially erroneous) assumptions into the histogram. ISOMER's approach amounts to imposing uniformity assumptions (as made by traditional optimizers) when, and only when no other statistical information is available. ISOMER can be seen as combining the efficient refinement characteristics from STGrid histograms with the accuracy of STHoles histograms. A related piece of work is [9], which addresses the problem of combining complementary selectivity estimations from multiple sources (which themselves can be computed using ISOMER histograms) to obtain a consistent selectivity estimation using the idea of maximum entropy. Similar to the approach in ISOMER, this work exploits all available information and avoids biasing the optimizer towards plans for which the least information is known.

Future Directions

Online tuning is dramatically gaining importance as more and more applications exhibit unpredictable workload evolution. In particular, this is a requirement for cloud-based data services. In such scenarios, the backend data services are shared by multiple applications and must be able to cope with changing access patterns. It should be noted that feedback-driven control is another paradigm that has been applied for continuous incremental tuning of systems, e.g., in the context of automated memory management [7]. A detailed discussion of feedback-driven control and its applications to computing systems can be found in [8].

Cross-references

► Histograms
► Self-Management Technology in Databases

Recommended Reading

1. Aboulnaga A. and Chaudhuri S. Self-tuning histograms: building histograms without looking at data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
2. Brown K.P., Mehta M., Carey M.J., and Livny M. Towards automated performance tuning for complex workloads. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 72–84.
3. Bruno N. and Chaudhuri S. An online approach to physical design tuning. In Proceedings of ICDE. Istanbul, Turkey, 2007.
4. Bruno N., Chaudhuri S., and Gravano L. STHoles: a multidimensional workload-aware histogram. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.
5. Chaudhuri S. and Narasayya V.R. Self-tuning database systems: a decade of progress. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
6. Chen C.-M. and Roussopoulos N. Adaptive selectivity estimation using query feedback. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 161–172.
7. Dageville B. and Zait M. SQL memory management in Oracle9i. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
8. Diao Y., Hellerstein J.L., Parekh S.S., Griffith R., Kaiser G.E., and Phung D.B. Self-managing systems: a control theory foundation. In ECBS. Greenbelt, Maryland, USA, 2005, pp. 441–448.
9. Markl V., Haas P.J., Kutsch M., Megiddo N., Srivastava U., and Tran T.M., Consistent selectivity estimation via maximum entropy. VLDB J., 16(1):55–76, 2007.
10. Srivastava U. et al. ISOMER: consistent histogram construction using query feedback. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
11. Stillger M., Lohman G.M., Markl V., and Kandil M. LEO - DB2's LEarning Optimizer. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 19–28.
12. Weikum G., König A.C., Kraiss A., and Sinnwell M. Towards self-tuning memory management for data servers. IEEE Data Eng. Bull., 22(2):3–11, 1999.

Database Tuning using Trade-off Elimination

Surajit Chaudhuri[1], Gerhard Weikum[2]
[1]Microsoft Corporation, Redmond, WA, USA
[2]Max-Planck Institute for Informatics, Saarbrücken, Germany

Definition

Database systems need to be prepared to cope with trade-offs arising from different kinds of workloads that different deployments of the same system need to support. To this end, systems offer tuning parameters that allow experienced system administrators to tune the system to the workload characteristics of the application(s) at hand. As part of the *self-management* capabilities of a database system, it is desirable to eliminate these tuning parameters and rather provide an algorithm for parameter settings such that near-optimal performance is achieved across a very wide range of workload properties. This is the trade-off elimination paradigm. The nature of the solution for trade-off elimination depends on specific tuning problems; its principal feasibility has been successfully demonstrated on issues such as file striping and cache management.

Historical Background

To cope with applications that exhibit a wide *variety of workload characteristics*, database systems have traditionally provided a repertoire of alternative algorithms for the same or overlapping functionalities, and have also exposed a rich suite of quantitative *tuning parameters.* Examples include a choice of page sizes and a choice of striping units at the underlying storage level, a choice of different data structures for single-dimensional indexing, and various tuning options for cache management. This approach prepares the system for *trade-offs* that arise across different workloads and within mixed workloads.

More than a decade ago, exposing many options and tuning parameters to system administrators had been widely accepted, but has eventually led to prohibitive costs for skilled staff. In the last 10 years, many trade-offs have become much better understood and analytically or experimentally characterized in a systematic manner. In some cases, the analysis led to the insight that a specific criteria for parameter settings could yield satisfactory performance across a wide range of workloads. Typically, this insight was derived hand in hand with progress on the underlying algorithmics or hardware technology. This can form the basis for eliminating various tuning options such as page sizes, striping units, data structures for single-dimensional indexing, and cache management strategies. Some of these research results have been adopted by commercial engines for self-management; in other cases, tuning parameters are still exposed.

Foundations

The first step towards trade-off elimination is to better understand the nature of the trade-off. The key questions to consider and analyze are the following: Why are different alternatives needed for the given parameter or function? Is the variance across (real or conceivable) workloads really so high that tuning options are justified? Do different options lead to major differences in performance? Do some options result in very poor performance? Are there any options that lead to acceptably good performance across a wide spectrum of workloads?

For illustration consider the following specific tuning issues:

1. *Page sizes:* There is a trade-off between *disk I/O efficiency* and *efficient use of memory* [4]. Larger page sizes lead to higher disk throughput because larger sequential transfers amortize the initial disk-arm seeks. Smaller page sizes can potentially make better use of memory because they contain exactly the actual requested data and there are more of such small pages that fit into memory. The impact of this trade-off was large more than 10 years ago with much smaller memory sizes. Today, memory sizes are usually at comfortable level, and disk controllers always fetch entire tracks anyway. Thus, a page size of one disk track (e.g., 100 Kilobytes) is almost always a good choice, and neither hurts disk efficiency nor memory usage too much.
2. *Striping units:* When files or tablespaces are partitioned by block or byte ranges and striped (i.e., partitions being allocated in round-robin manner) across multiple disks, the partition size, aka. striping unit, leads to a trade-off regarding I/O parallelism versus disk throughput [2,9]. Small striping units lead to high parallelism when reading large amounts of data from the disk-array, but they consume much more disk-arm time for all involved disks together, compared to larger striping units with lower degree of parallelism. Thus, larger striping units can achieve higher I/O throughput. In some applications, it may still be worthwhile to tune striping units of different files according to their request size distributions. But in database systems, there is usually only a mix of single-block random accesses and sequential scans. For such workloads, large striping units in the order of one Megabyte achieve near-optimal throughput while still allowing I/O parallelism for scans.
3. *Single-dimensional indexing:* Many commercial systems offer the (human or automated) tuning wizard a choice between a B+-tree or a hash index, for each single-attribute index that is to be created. B+-trees provide high efficiency for both random key lookups and sequential scans, and have proven their high versatility. Hash indexes offer randomization to counter access skew (with some keys being looked up much more frequently than others) and even better worst-case performance for lookups. Hashing guarantees exactly one page access, whereas B+-tree indexes have to access a logarithmic number of pages in their descent towards the requested key. This was an important trade-off to consider for tuning a decade ago. But today, the disadvantages of B+-trees for certain workloads have become minor: randomization can be achieved as well by using hash-transformed keys in the tree (at the inherent expense of penalizing range queries); and the extra costs of the tree descent

are negligible given the high fan-out and resulting low depth of index trees and the fact that all index levels other than leaf pages virtually always reside in memory. So if one were to build a lean, largely self-managing database engine today, it would have to support only B+-tree indexes (but, of course, have hash-based algorithms in its repertoire for joins and grouping).

These examples demonstrate the kinds of insights and simplifications towards self-managing systems that may be achieved by means of trade-off analysis. The analyses may be based on mathematical modeling (e.g., for estimating response times and throughput of different striping units), draw from comprehensive experimentation or simulation, or assess strengths and weaknesses qualitatively (e.g., functionality and computational costs of index implementations). Often a combination of different methodologies is needed. None of these approaches can be automated; while automatic tool support is standard (e.g., for evaluating analytic models and for sensitivity analyses) the final assessment requires human analytics and judgment. Thus, the trade-off elimination paradigm is a "thinking tool" for system-design time, providing guidance towards self-tuning systems.

Example: Cache Management with Trade-off Elimination
As a more detailed example, consider the management of a *shared page cache* in a database system and the tuning issue that underlies the page replacement strategy. It is discussed in more depth here, as it is not only another illustration of eliminating trade-offs, but also an important performance issue by itself.

For choosing a cache replacement victim, cache managers assess the "*worthiness*" of the currently cache-resident pages and drop the least worthy page. Ideally, one would predict the future access patterns: the least worthy page is the one whose next access is farthest in the future [3]. But the cache manager only knows the past accesses, and can remember only a bounded amount of information about the previous patterns. In this regard, a page shows evidence of being potentially worthy if it exhibits a history frequent accesses or recent accesses. The traditional replacement strategies give priority to either *frequency* or *recency*, but neither is optimal and the co-existence of the two criteria presents a trade-off for the cache manager.

Frequency-based worthiness would be optimal if access patterns were in steady state, i.e., if page-access frequencies had a time-invariant distribution with merely stochastic fluctuation. The algorithm of choice would then be *LFU*, which always replaces the least-frequently-used page. However, if the workload evolves and the distributions of access frequencies undergo significant changes, LFU is bound to perform poorly as it takes a long time to adjust to new load characteristics and re-estimate page-access probabilities. Therefore, the practically prevalent method is actually *LRU*, which always replaces the least-recently-used page. The LRU strategy automatically adapts itself to evolving access patterns and is thus more robust than LFU.

Despite its wide use and salient properties, LRU shows significantly sub-optimal behavior under various workloads. One issue is the co-existence of *random accesses* and *sequential scans*. Pages that are accessed only once during a scan will be considered by LRU although they may never be referenced again in the near future. This idiosyncrasy is typically fixed in industrial-strength database system by allowing the query processor to pass hints to its cache manager and advise it to give low priority to such a read-once page. However, there are further situations where the specifics of the access patterns reveal shortcomings of the LRU strategy.

Consider a workload with random accesses only, but with very high variance of the individual pages' access probabilities. Assume that there is a sequence of primary-key lookups on a database table that results in alternating accesses to uniformly selected index and data pages. As there are usually many more data pages than index pages (for the same table), the individual index pages have much higher access frequencies than each of the data pages. This indicates that index pages are more worthy for staying in the cache, but LRU considers only recency and is inherently unable to discriminate the two kinds of pages. In steady state, LRU would keep the same numbers of index pages and data pages in the cache, but the optimal behavior would prioritize index pages. Ironically, LFU would be optimal for this situation (but fails in others).

The *LRFU* algorithm [6] has addressed this issue by defining the worthiness of a page as a linear combination of its access recency and access frequency. Unfortunately, the performance of this method highly depends on proper tuning of the weighting coefficients for the two aspects and on additional parameters that govern the aging of frequency estimates. Another sophisticated approach that at least one commercial system had taken is to support multiple caches with configurable sizes and allow the administrator to assign different tablespaces to different caches. This way, the cache management for data vs. index pages or pages of different tables and indexes can be tuned extremely well. But this requires very good human skills; automating this approach would be a major challenge for databases with hundreds of tables and highly diverse workloads.

An approach that aims at eliminating all such tuning parameters is the *LRU-k algorithm* [8]. Its main principle is to dynamically estimate the access frequency of "interesting" pages by tracking the time points of the last k accesses to each page. It can be viewed as a maximum-likelihood estimator for page-access probabilities, and uses a sliding-window technique for built-in aging of the estimates. Worthiness of a page is now defined as the reciprocal of the backward distance to its kth last access, and this has been shown to be optimal among all replacement algorithms that have the same information about the access history. For $k = 1$, LRU-k behaves exactly like LRU; for $k \geq 2$, it has enough information to properly handle sequentially scanned read-once pages and to automatically discriminate page classes with very different access frequencies and patterns.

LRU-k introduces additional bookkeeping that incurs overhead compared to the best, extremely lightweight, implementations of standard LRU. In particular, it needs to track the last k accesses of more pages than the cache currently holds. There is a robust heuristics, based on the "5-min rule of thumb" [4], for deciding which pages should be tracked at all. But the bookkeeping uses a non-negligible amount of memory – the very resource whose usage the cache manager aims to optimize with so much scrutiny. Thus, LRU-k invests some memory and effectively reduces the size of the cache by that amount, in order to improve the overall caching benefit. It has been experimentally shown that this *cost/benefit ratio* is indeed profitable: even if the cache size is reduced by the bookkeeping memory, LRU-k (for $k \geq 2$) still significantly outperforms LRU (but one would typically limit k to 2 or 3).

LRU-k also needs more CPU time than LRU because its replacement decision criterion, the backward distance to the k^{th} last accesses of the currently cache-resident pages, requires a priority queue rather than a simple linked list. Even with the best possible implementation techniques, this leads to logarithmic rather than constant cost per page access. However, there are excellent ways of implementing approximate versions of the LRU-k principle without this overhead. The *2Q algorithm* [5] and the *ARC algorithm* [7] use LRU-like linked lists but separate pages in two lists: one for pages with at least k accesses and one for pages with less accesses in the current bookkeeping. By devising smart rules for migrating pages between lists, these algorithms achieve cache hit ratios that are as good as LRU-k while keeping the implementation overhead as low as that of LRU. Other extensions of the LRU-k principle are cache replacement algorithms for variable-size data items such as files or Web pages [1,10].

The best algorithms of the LRU-k family have successfully eliminated the recency-frequency trade-off and provide self-tuning cache management. The insights from this line of research provide several, more general or potentially generalizable, lessons:

- It is crucial to analyze the nature of the trade-off that led to the introduction of tuning options or alternative algorithms for the same function.
- It is beneficial to unify the modeling and treatment of different classes of access patterns (workloads), thus providing a basis for simplifying algorithms and systems.
- Additional bookkeeping to better capture the workload can be worthwhile even if it consumes the very same resource that is to be optimized. But the overhead needs to be carefully limited.
- To eliminate a "troublesome" tuning option, self-managing algorithms may introduce additional second-order parameters (e.g., for bookkeeping data structures). The art is to ensure that the new parameters must be such that it should be easy to find a robust setting that leads to near-optimal behavior under almost all workloads.

Future Directions

Trade-off elimination is a general paradigm, but not a directly applicable recipe for self-management. Thus, future research should consider studying more trade-offs and tuning problems in view of this paradigm, bearing in mind both its potential benefits and intricacies.

Cross-references

► Memory Hierarchy
► Self-Management Technology in Databases

Recommended Reading

1. Cao P. and Irani S. Cost-aware WWW proxy caching algorithms. In Proc. 1st USENIX Symp. on Internet Tech. and Syst., 1997.
2. Chen P.M., Lee E.L., Gibson G.A., Katz R.H., and Patterson D.A. RAID: high-performance, reliable secondary storage. ACM Comput. Surv., 26(2):145–185, 1994.
3. Coffman E.G. Jr. and Denning P.J. Operating Systems Theory. Prentice-Hall, Englewood, Cliffs, NJ, 1973.
4. Gray J. and Graefe G. The five-minute rule ten years later, and other computer storage rules of thumb. SIGMOD Rec., 26 (4):63–68, 1997.
5. Johnson T. and Shasha D. 2Q: a low overhead high performance buffer management replacement algorithm. In Proc. 20th Int. Conf. on Very Large Data Bases. 1994, pp. 439–450.
6. Lee D., Choi J., Kim J.-H., Noh S.H., Min S.L., Cho Y., and Kim C.-S. LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Trans. Comput., 50(12):1352–1361, 2001.
7. Megiddo N. and Modha D.S. Outperforming LRU with an adaptive replacement cache algorithm. IEEE Comput., 37 (4):58–65, 2004.
8. O'Neil E.J., O'Neil P.E., and Weikum G. The LRU-K page replacement algorithm for database disk buffering. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 297–306.
9. Scheuermann P., Weikum G., and Zabback P. Data partitioning and load balancing in parallel disk systems. VLDB J., 7(1):48–66, 1998.
10. Young N.E. On-line file caching. Algorithmica, 33(3):371–383, 2002.

Databases for Biomedical Images

► Image Management for Biological Data

Dataguide

► Structure Indexing

Datalog

GRIGORIS KARVOUNARAKIS
University of Pennsylvania, Philadelphia, PA, USA

Synonyms
Deductive databases

Definition
An important limitation of relational calculus/algebra is that it cannot express queries involving "paths" through an instance, such as taking the transitive closure over a binary relation. *Datalog* extends conjunctive queries with recursion to support such queries. A Datalog program consists of a set of rules, each of which is a conjunctive query. Recursion is introduced by allowing the same relational symbols in both the heads and the bodies of the rules. A surprising and elegant property of Datalog is that there are three very different but equivalent approaches to define its semantics, namely the *model-theoretic*, *proof-theoretic*, and *fixpoint* approaches. Datalog inherits these properties from logic programming and its standard language Prolog. The main restriction that distinguishes Datalog from Prolog is that function symbols are not allowed.

Several techniques have been proposed for the efficient evaluation of Datalog programs. They are usually separated into two classes depending on whether they focus on *top-down* and *bottom-up* evaluation. The ones that have had the most impact are centered around *magic sets rewriting*, which involves an initial preprocessing of the Datalog program before following a bottom-up evaluation strategy. The addition of negation to Datalog rules yields highly expressive languages, but the semantics above do not extend naturally to them. For Datalog¬, i.e., Datalog with negated atoms in the body of the rules, *stratified* semantics, which impose syntactic restrictions on the use of negation and recursion, is natural and relatively easy to understand. The present account is based primarily on the material in [1]. Each of [1,2,9] has an excellent introduction to Datalog (Capitalization of the name follows the convention used in [2,8] (rather than [1,9]).). An informal survey can be found in [8]. The individual research contributions to Datalog are cited in the union of the bibliographies of these textbooks.

Historical Background
Datalog is a restriction of the paradigm of *logic programming (LP)* and its standard programming language, Prolog, to the field of databases. What makes logic programming attractive is its *declarative* nature, as opposed to the more operational flavor of other programming paradigms, be they imperative, object-oriented, or functional. In the late 1970's and into the 1980's, this led to much LP-related activity in Artificial Intelligence and even supercomputing (The

Fifth Generation Project) which has later subsided dramatically. In databases this remains a useful paradigm, since the relational calculus is also a declarative language and LP provides a mechanism for extending its expressiveness with so-called *recursive queries*.

The name "Datalog" was coined by David Maier [1]. Research on recursive queries in databases picked up in the 1980's and eventually led to several prototype *deductive database systems* [8,9] whose data is organized into relations, but whose queries are based on Datalog.

Datalog has not quite made it as a practical query language due to the preeminence of SQL. When the need for recursive queries was recognized by RDBMS vendors, they preferred to extend SQL with some limited forms of recursion [8]. Nonetheless, more recent research on data integration has found Datalog to be a useful conceptual specification tool.

Foundations

Datalog Syntax

The syntax of Datalog follows that of the logic programming language Prolog with the proviso that only constants and relational symbols are allowed (no function symbols).

Definition 1

Fix a relational schema. (The use of the symbol :- has its roots in Prolog, but some texts, e.g., [1], use the symbol ←instead, to convey the fact that each rule is closely related to a logical implication, as explained in the discussion of model-theoretic semantics.) *A* Datalog rule *has the form:*

$$T(\mathbf{x}) :- \ q(\mathbf{x}, \mathbf{y})$$

where $\mathbf{x} = x_1,\ldots,x_n$ is a tuple of *distinguished* variables, $\mathbf{y} = y_1,\ldots,y_m$ is a tuple of "existentially quantified" variables, T is a relation and q is a conjuction of relational atoms. The left-hand side is called the head of the rule and corresponds to the output/result of the query and the right-hand side is called the body of the rule. Note that all distinguished variables in the head need to appear in at least one atom in the body, i.e., the rules are *range restricted*. A Datalog rule is identical to a conjunctive query in rule-based syntax, except that the latter does not always have a name for the head relation symbol. A Datalog program is a finite set of Datalog rules over the same schema. Relation symbols (a.k.a. predicates) that appear only in the body of the program's rules are called *edb* (extensional database) predicates, while those that appear in the head of some rule are called *idb* (intensional database) predicates. A Datalog program defines a Datalog query when one of the idb predicates is specified as the *output*.

For example, if G is a relation representing edges of a graph, the following Datalog program P_{TC} computes its transitive closure in the output predicate T:

$$T(x, y) :- \ G(x, y)$$
$$T(x, y) :- \ G(x, z), T(z, y)$$

Semantics

Three different but equivalent definitions can be given for the semantics of Datalog programs, namely the *model-theoretic*, *proof-theoretic* and *fixpoint* semantics.

A countably infinite set $\mathbb{D}$ of constants is fixed as the sole universe for structures/instances. Since there are no function symbols, any relational instance over $\mathbb{D}$ is an *Herbrand interpretation* in the sense used in logic programming.

In the model-theoretic semantics of Datalog, each rule is associated with a first-order sentence as follows. First recall that as a conjunctive query, $T(\mathbf{x})$:- q ($\mathbf{x}$, $\mathbf{y}$) corresponds to the first-order query $T \equiv \{\mathbf{x} \mid \exists \mathbf{y}\ q(\mathbf{x}, \mathbf{y})\}$. To this, one associates the sentence $\forall \mathbf{x}\ (\exists \mathbf{y}\ q(\mathbf{x}, \mathbf{y}) \rightarrow T(\mathbf{x}))$ which is clearly satisfied in a structure in which T is interpreted as the answer to the query. Note that this sentence is a *definite Horn clause*. More generally, given a Datalog program P, let Σ_P be the set of Horn clauses associated to the rules of P.

Let I be an input database instance, in this case an instance of the schema consisting only of edb predicates. A *model* of P is an instance of the entire schema (both edb and idb relation symbols) which coincides with I on the edb predicates and which satisfies Σ_P. However, there can be infinitely many instances that satisfy a given program and instance of the edb relations. Thus, logic programming, and consequently Datalog use a *minimal* model, i.e., one such that no subset of it is also a model. This is usually understood as a manifestation of the *closed world assumption*: don't assume more than you need! It can be shown that for Datalog, there is exactly one minimal model, which is also the *minimum* model.

In the *proof-theoretic* approach of defining the semantics of Datalog, note first that a tuple of constants in a relation can be seen as the head of a rule with empty body. Such rules are called *facts*. As previously seen, Datalog rules can be associated with first-order sentences. Facts correspond to just variable-free relational atoms. Now, the main idea of the proof-theoretic semantics is that the answer of a Datalog program consists of the set of facts that can be proven from the edb facts using the rules of the program as *proof rules*. More precisely, a *proof tree* of a fact A is a labeled tree where (i) each vertex of the tree is labeled by a fact; (ii) each leaf is labeled by a fact in the base data; (iii) the root is labeled by A; and (iv) for each internal vertex, there exists an instantiation A_1 :- $A_2,\ldots,A_n$ of a rule r such that the vertex is labeled A_1 and its children are respectively labeled $A_2,\ldots,A_n$ and the edges are labeled r.

Example 1. *Consider the program*:

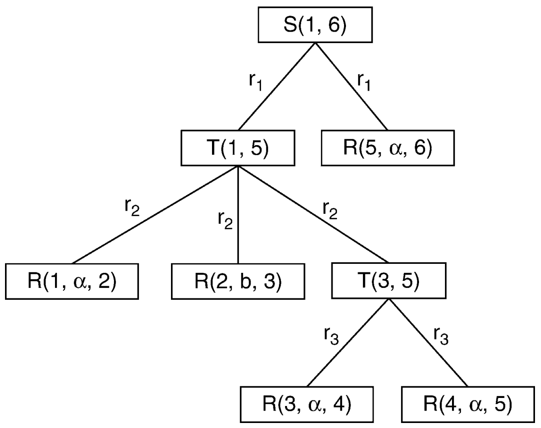
$$
\begin{aligned}
&(r_1)\ S(x_1, x_3) \text{ :- } T\ (x_1, x_2), R(x_2, a, x_3)\\
&(r_2)\ T(x_1, x_4) \text{ :- } R\ (x_1, a, x_2), R(x_2, b, x_3), T(x_3, x_4)\\
&(r_3)\ T(x_1, x_4) \text{ :- } R\ (x_1, a, x_2), R(x_2, a, x_3)
\end{aligned}
$$

and the instance

$$\{R(1, a, 2), R(2, b, 3), R(3, a, 4), R(4, a, 5), R(5, a, 6)\}$$

A proof tree of S(1, 6) is shown in Fig. 1.

Because rule instantiation and application correspond to standard first-order inference rules (substitution and modus ponens), the proof trees are actually rearrangements of first-order proofs. This connects Datalog, through logic programming, to automated theorem-proving. One technique for constructing proofs such as the one above in a *top-down* fashion (i.e., starting from the fact to be proven) is *SLD resolution* [1]. Alternatively, one can start from base data and apply rules on them (and subsequently on facts derived this way) to create proof trees for new facts.



Datalog. Figure 1. Proof tree.

The third approach is an operational semantics for Datalog programs stemming from *fixpoint* theory. The main idea is to use the rules of the Datalog program to define the *immediate consequence* operator, which maps idb instances to idb instances. Interestingly, the immediate consequence operator can be expressed in relational algebra, in fact, in the SPCU (no difference) fragment of the relational algebra, enriched with edb relation names. For example, the immediate consequence operator $\mathcal{F}$ for the transitive closure above is:

$$\mathcal{F}(T) = G \bowtie T \cup G$$

One way to think about this operator is that it applies rules on existing facts to get new facts according to the head of those rules. In general, for a recursive Datalog program, the same operator can be repeatedly applied on facts produced by previous applications of it. It is easy to see that the immediate consequence operator is *monotone*. Another crucial observation is that it will not introduce any constants beyond those in the edb instance or in the heads of the rules. This means that any idb instance constructed by iteration of the immediate consequence operator is over the *active domain* of the program and the edb instance. This active domain is finite, so there are only finitely many possible idb instances. They are easily seen to form a finite poset ordered by inclusion. At this point one of several technical variants of fixpoint theory can be put to work. The immediate consequence operator has a *least fixpoint* which is an idb instance and which is the semantics of the program. It can be shown that this idb instance is the same as the one in the minimal model semantics and the one in the proof tree semantics. It can also be shown that this least fixpoint can be reached after finitely many iterations of the immediate consequence operator which gives a Datalog evaluation procedure called *bottom-up*.

Evaluation and Optimization of Datalog

The simplest bottom-up evaluation strategy, also called *naive* evaluation, is based directly on fixpoint Datalog semantics. The main idea is to repeatedly apply the immediate consequence operator on results of all previous steps (starting from the base data in the first step) until some step doesn't yield any new data. It is clear that naive evaluation involves a lot of redundant computation, since every step recomputes all facts already computed in previous steps. *Seminaive* evaluation tries to overcome this deficiency, by producing at every step only facts that can be derived using at least one of the new facts produced in the last step (as opposed to all previous steps).

In some cases, bottom-up evaluation can produce a lot of "intermediate" tuples that are not used in derivations of any facts in the output relation of the query. The top-down approach avoids this problem by using heuristic techniques to focus attention on relevant facts, i.e., ones that appear in some proof tree of a query answer, especially for Datalog programs with constants appearing in some atoms. The most common approach in this direction is called the query-subquery (QSQ) framework. QSQ generalizes the SLD resolution technique, on which the proof-theoretic semantics are based, by applying it in sets, as opposed to individual tuples, as well as using constants to select only relevant tuples as early as possible. In particular, if an atom of an idb relation appears in the body of a rule with a constant for some attribute, this constant can be pushed to rules producing this idb. Similarly, "sideways information passing" is used to pass constant binding information between atoms in the body of the same rule. Such constant bindings are expressed using *adornments* or *binding patterns* on atoms in the rules, to indicate which attributes are *bound* to some constant and which are *free*.

Magic set techniques simulate the pushing of constants and selections that happens in top-down evaluation to optimize bottom-up evaluation. In particular, they rewrite the original Datalog program into a new program whose seminaive bottom-up evaluation produces the same answers as the original one, as well as producing the same intermediate tuples as the top-down approaches such as QSQ.

Datalog with Negation

The language Datalog¬ extends Datalog by allowing negated atoms in the body of the rules. Unfortunately, the semantics described above do not extend naturally to Datalog¬ programs. For example, if the fixpoint semantics are followed, there are programs that do not have a fixpoint or have multiple least fixpoints, or even if there is a least fixpoint the constructive method described above does not converge or its limit is not the least fixpoint. For model-theoretic semantics, uniqueness of the minimal model is not guaranteed. For these reasons, the common approach is to only consider a syntactically restricted use of negation in Datalog¬ programs, called *stratification*, for which natural extensions of the usual Datalog semantics do not have these problems. A stratification of Datalog¬ is a partition of its rules into subprograms that can be ordered in *strata*, so that for each relation R in the program, all rules defining R (i.e., with R in the head) are in the same stratum and for all atoms in the bodies of those rules, the definitions of those relations are in a smaller or the same stratum, if the atom is positive, or strictly in a smaller stratum, for negative atoms.

For stratified Datalog¬ programs, one can evaluate within each stratum considering atoms of relations defined in smaller strata as edbs. Then, a negated atom is satisfied in the body of the rule if the corresponding tuple does not appear in that relation (as it appears in the base data or computed for the subprograms of smaller strata).

Key Applications

Current and Potential Users and the Motivation of Studying This Area

Although Datalog was originally proposed as the foundation of deductive databases, which never succeeded in becoming part of commercial systems, it has recently seen a revival in the areas of data integration and exchange. This is due to the similarity of Datalog rules with popular *schema mapping* formalisms (*GLAV* or *tuple generating dependencies* [5]) used to describe relationships between heterogeneous schemas. In particular, [3] proposed the *inverse rules* algorithm for reformulating queries over a target schema to queries over source schemas in data integration. Other work has used Datalog rules to compute *data exchange* [7] or *update exchange* [4] solutions. In these cases, the authors employed an extension of Datalog with *Skolem* functions in the head of rules, to deal with existentially quantified variables in the target of mappings. Another extension of Datalog, *Network Datalog (NDlog)* [6] allows the declarative specification of a large variety of network protocols with a handful of lines of program code, resulting to orders of magnitude of reduction in program size.

Cross-references

► Conjunctive Query
► Relational Calculus

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, MA, USA, 1995.
2. Bidoit N. Bases de Données Déductives: Présentation de Datalog. Armand Colin, 1992.
3. Duschka O., Genesereth M., and Levy A. Recursive query plans for data integration. J. Logic Program., special issue on Logic Based Heterogeneous Information Systems, 43(1), 2000.
4. Green T.J., Karvounarakis G., Ives Z.G., and Tannen V. Update exchange with mappings and provenance. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
5. Lenzerini M. Tutorial – data integration: a theoretical perspective. In PODS, 2002.
6. Loo B.T., Condie T., Garofalakis M.N., Gay D.E., Hellerstein J.M., Maniatis P., Ramakrishnan R., Roscoe T., and Stoica I. Declarative networking: language, execution and optimization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 97–108.
7. Popa L., Velegrakis Y., Miller R.J., Hernández M.A., and Fagin R. Translating web data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
8. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edn. McGraw-Hill, New York, 2003.
9. Ullman J.D. Principles of Database and Knowledge-Base Systems, Volume II. Addison-Wesley, MA, USA, 1989.

Datalog Query Processing and Optimization

► Query Processing in Deductive Databases

Datastream Distance

► Stream Similarity Mining

Datawarehouses Confidentiality

► Data Warehouse Security

DBC

► Database Clusters

DBMS

► Database Management System

DBMS Component

Johannes Gehrke
Cornell University, Ithaca, NY, USA

Synonyms

DBMS Module

Definition

A component is a self-contained piece of software in a database system. A component can be defined at different levels of coarseness. At the coarsest level, the components of a relational database management system consist of the client communications manager, the process manager, a relational query processor, a transactional storage manager, and utilities [1].

Key Points

The components of a relational database management system can be further refined into subcomponents [1]. The client communications manager consists of local client protocols and remote client protocols. The process manager consists of admission control and dispatch and scheduling. The relational query processor consists of query parsing and authorization, query rewrite, query optimization, plan execution, and DDL and utility processing. The transactional storage manager consists of access methods, a buffer manager, a lock manager, and a log manager. Sub-components that comprise the utilities component include the catalog manager, the memory manager, and replication and loading services.

This high-level division varies among commercial and open-source database products, both by level of granularity and by functionality of individual (sub-) components. Database management systems optimized for certain types of workloads (for example, decision support workloads) or database management systems with specialized architectures (for example, main-memory database management systems) may lack some of these components [2].

Cross-references

► Client Communications Manager
► Process Manager
► Relational Query Processor
► Transactional Storage Manager.

Recommended Reading

1. Joseph M. Hellerstein, Michael Stonebraker and James Hamilton. Architecture of a Database System. In Foundations and Trends in Databases: Vol. 1: No. 2, pp. 141–259.
2. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edition. McGraw-Hill Science/Engineering/Math, 2002.

DBMS Interface

Johannes Gehrke
Cornell University, Ithaca, NY, USA

Synonyms

Communication boundary of a DBMS

Definition

A DBMS interface is the abstraction of a piece of functionality of a DBMS. It usually refers to the communication boundary between the DBMS and clients or to the abstraction provided by a component within a DBMS. A DBMS interface hides the implementation of the functionality of the component it encapsulates.

Key Points

DBMS interfaces can be external or internal [3]. An external DBMS interface is the communication boundary between the DBMS and clients. The external DBMS interface enables clients to access internal DBMS functionality without exposing the mechanisms of how this functionality is implemented. Well-known external DBMS interfaces are SQL, XPath, and XQuery. There are many different types of external DBMS interfaces, for example, stand-alone languages (such as SQL), extensions to existing languages with features from SQL (such as JDBC), and integration into middle-tier programming languages (such as PHP). The DBMS interface can also be graphical; for example, the DBMS interface to the data data definition language is often a graphical editor enabling database designers to visualize and manipulate logical and physical database schemas in the Entity-Relationship Model or in UML.

Internal DBMS interfaces exist between different DBMS components, for example, the query processor and the storage manager [2]. Although standards for these interfaces do not exist, their design is as important as the design of the external interfaces. For example, by exposing a queue interface for internal DBMS components and arranging the DBMS components in stages, the DBMS can be designed such that queries interact with a single component at a given time, permitting enhanced processor utilization during query processing [1].

Cross-references

► DBMS Component
► Query Language
► SQL
► XPath
► XQuery

Recommended Reading

1. Harizopoulos S. and Ailamaki A. StagedDB: Designing Database Servers for Modern Hardware. IEEE Data Engineering Bulletin, 28(2):11–16, June 2005.
2. Hellerstein J.M., Stonebraker M. and Hamilton J. Architecture of a Database System. In Foundations and Trends in Databases: Vol. 1: No. 2, pp. 141–259. http://dx.doi.org/10.1561/1900000002.
3. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edition. McGraw-Hill Science/Engineering/Math, 2002.

DBTG Data Model

► Network Data Model

DCE

Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA

Synonyms

Distributed computing environment

Definition

The Distributed Computing Environment (DCE) [1–3] is a technology standardized by the Open Group for client/server computing. A primary goal of DCE is interoperability using remote procedure call semantics.

Key Points

The Distributed Computing Environment (DCE) technology was developed through the efforts of the Open Software Foundation (OSF) in the late 1980s and early 1990s as an interoperable solution for client-server distributed computing. A primary objective was to overcome the heterogeneity in operating systems and network technologies. The DCE technology uses procedural programming semantics provided by languages, such as C. OSF is now part of the Open Group, which releases DCE code under the LGPL license via its DCE portal [2].

A primary feature provided by DCE is the remote procedure call [1]. DCE is, however, not only about remote procedure calls. In addition, DCE provides a number of fundamental building blocks to make applications secure, as well as provides a number of services, such as a cell directory service, a distributed file system service, a time service and a threading service.

DCE supports the notion of a cell, which is a collection of nodes that are managed by a single authority. Intra-cell communication is highly optimized and secure. Inter-cell communication requires more advanced configurations. The distributed file system service provides a high performance network file system. A POSIX-like API available locally hides the distributed aspects of the file system.

The DCE concepts and technology laid the foundation for the next generation of object-oriented client-server paradigms, such as CORBA in the mid 1990s. DCE RPC is used as the building block technology for Microsoft COM/DCOM technologies.

Cross-references

► Client-Server Architecture
► CORBA
► DCOM
► J2EE Middleware
► Java RMI
► .NET Remoting
► Request Brokers
► SOAP

Recommended Reading

1. The Open Group, "DCE 1.1: Remote Procedure Call," CAE Specification, Document no. C706, 1997, Published online at http://www.opengroup.org/onlinepubs/9629399.
2. The Open Group, "DCE Portal," http://www.opengroup.org/dce/.
3. Software Engineering Institute, "Distributed Computing Environment," http://www.sei.cmu.edu/str/descriptions/dce.html.

DCOM

Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA

Synonyms

Distributed component object model

Definition

Distributed Component Object Model (DCOM) [1,2] is a Microsoft technology for component-based distributed computing.

Key Points

Distributed Component Object Model (DCOM) is an extension of Microsoft's Component Object Model (COM) to enable distribution across address spaces and networks. By leveraging COM, which is the fundamental technology used to build many Microsoft applications, DCOM applications can derive all the power of COM applications. Distribution capabilities in DCOM are realized by leveraging DCE Remote Procedure Call mechanisms and extending them to support the notion of remote objects.

DCOM provides most of the capabilities of a Request Broker. For example, it supports location transparency, connection management, resource management, concurrency control, versioning, language-neutrality and QoS management. Since DCOM is a component model, additional capabilities for composing and deploying component-based applications also exist.

Despite its numerous benefits, DCOM has its own limitations. For example, DCOM uses its own binary protocol for communicating between applications, which restricts interoperability to communication between the same object models. Additionally, although some efforts exist at porting DCOM to other platforms, such as Linux, DCOM remains predominantly a Microsoft platform-specific technology, which limits its applicability to a wider range of applications.

These limitations have necessitated the move towards more advanced technologies, such as .NET Remoting, which provide a wider range of interoperable solutions.

Cross-references

► Client-Server Architecture
► CORBA
► DCE
► DCOM
► J2EE Middleware
► Java RMI
► .NET Remoting
► Request Brokers
► SOAP

Recommended Reading

1. Brown N. and Kindel C. Distributed Component Object Model Protocol – DCOM/1.0, Internet Draft, Network Working Group, November 1996.
2. Horstmann M. and Kirtland M. DCOM Architecture, DCOM Technical Articles, Microsoft Developer Network Library, July 23, 1997.

Deadlocks in Distributed Database Systems

► Distributed Deadlock Management

Decay Models

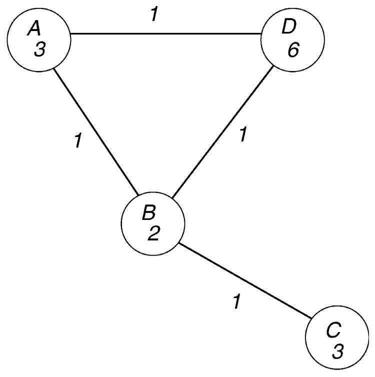
EDITH COHEN
AT&T Labs-Research, Florham Park, NJ, USA

Definition

Decay models are applicable on data sets where data items are associated with points in a metric space (*locations*) and there is a notion of "significance" of a data item to a location, which decays (decreases) with the distance between the item and the location. This decrease is modeled by a *decay function*.

Each location has a "view" of the data set through a different weight distribution over the items: the weight associated with each item is its *decayed weight* which is a product of its *original weight* and a decay function applied to its distance from the observing location.

While global aggregates or statistics are computed over the set of items and their original weights, *decaying aggregates* or *decaying statistics* depend on the location with respect to which the aggregation is performed and on the decay function. Figure 1 illustrates a small network and the decaying sum with respect to all nodes.



Node	Decaying sum
A	8
B	8
C	7
D	9.5

Decay Models. Figure 1. Decaying sum over a network with respect to decay function $g(d) = \frac{1}{1+d}$. Items of distance 1 have decayed weight that is 1⁄2 of their original weight. Similarly, items of distance 2 have decayed weight that is 1⁄3 of their original weight. The global (non-decaying) sum is 14. The decaying sums (listed for all nodes) are location-dependent.

Historical Background

Kernel Estimation

The earliest use of decay models that the author is aware of is *Kernel estimation.* Kernel estimation is a non-parametric density function estimation method [19]. The metric space is the Euclidean space and the method is applied to a set of points with weights. The decay function captures a spherically symmetric probability density (typically a Gaussian which corresponds to Exponential decay); the decaying sum at a point is the density estimate.

Recently, Hua et al. [16] proposed the use of decayed sum to assign *typicality score* to items, based on Kernel estimation. They also propose methods to efficiently compute approximate decaying sum in a high dimensional Euclidean space.

Time-Decay

Time-decay, where significance of a data point decreases with elapsed time (or with number of observed items), is a natural model. Decaying aggregation on data streams predates the formal analysis of stream algorithms but was mostly limited to exponential decay. Several applications are listed next.

The random early detection (RED) protocol: RED is a popular protocol deployed in Internet routers for congestion avoidance and control. RED uses the weighted average of previous queue lengths to estimate the impending congestion at the router; the estimate is then used to determine what fraction of packets to discard [13,17]. *Holding-time policies for ATM virtual circuits:* Circuit-switched data connections have an associated cost of being kept open; data transfers are bursty and intermittent and, when a data burst arrives, it incurs smaller delay when the circuit is open. Thus, when there are multiple connections, it is important to assess the anticipated idle times (time to the next data burst) of each circuit and to close first those circuits that have longer anticipated idle times. This can be done using a time-decaying weighted average of previous idle times [18]. A similar model applies to maintaining open TCP connections at a busy Web server [7]. *Internet gateway selection products:* Multiple internet paths are available to each destination host and the product needs to assess the reliability of each path in order to facilitate a better selection of a path. A time-decaying average of previous measurements can be used as a measure of path quality [2].

Datar et al. [12] introduced the *sliding-window* model for data streams. *Sliding-window* are a threshold decay function defined over the sequence number of data items. They introduced a synopsis data structure called *Exponential Histograms* that supports approximate sum queries on sliding windows with respect to the "current" time and developed algorithms for approximately maintaining sum and average aggregates and variance [1]. Gibbons and Tirthapura [14] developed an alternative data structure that can handle multiple streams. The sliding window model was extensively studied.

Cohen and Strauss [8,9] considered decaying aggregation on data streams under *general* decay functions. They showed that general decay can be reduced to sliding windows decay. However, sliding window decay is in a sense the "hardest" decay function and other functions, such as polynomial and exponential decay can be computed more efficiently.

Network

Decaying aggregation over a network is a generalization of decaying aggregation over data streams (data streams correspond to path networks). It was first considered by Cohen in [3] as Threshold decay (aggregation over neighborhoods). Cohen proposed efficient algorithms for decaying sum. General decay functions, other aggregates, and efficient algorithms and analysis to maintain these sketches in distributed settings were considered by Cohen and Kaplan [5,6]. On networks, just like on data streams, threshold decay is the "hardest" decay function, as all-distances sketches that support sums over neighborhoods, also support decaying sums under general decay functions. It is not known, however, whether other decay functions such as exponential or polynomial decay can be computed or approximated more efficiently by a tailored approach. Further work on decayed aggregation over a networks includes [10].

Euclidean Plane

Cohen and Kaplan [5] also considered decaying sums in the Euclidean plane and proposed an efficient construction of a data structure (based on incremental construction of Voronoi diagrams [15]) that can support point queries. That is, for a query location and a query decay function, the data structure returns a respective approximate decaying sum.

Foundations

Metric Space

Decaying aggregates were considered on different metric spaces. The simplest one is one-dimensional Euclidean space such as the time dimension. Items have time stamps or sequence numbers and the relevance of an item at a certain time decays with elapsed time or with the number of items with a later time stamp. Decayed aggregates are also used on a higher dimensional Euclidean space and on a networks (directed or undirected graphs) where nonnegative lengths are associated with edges and distances correspond to shortest paths lengths.

Decay Functions

The decayed weight of an item at a location depends linearly on its original weight and decreases with distance. It is defined as a product of its original weight and the value of the decay function on the distance of the item from the location. A decay function $g(x)$ is a non-increasing function defined over the nonnegative reals. It is often convenient to define $g(0) = 1$. Some natural decay functions are *threshold functions*, where items within a certain distance from the location have their original weights and other items have 0 weight, *Exponential decay*, where the weight decreases exponentially with the distance, and *Polynomial decay*, where the weight decreases polynomially with the distance.

Threshold decay assigns equal importance to all data values within a distance of r and disregards all other data values. Exponential decay is defined with respect to a parameter $\lambda > 0$, $g(x) = \exp(-\lambda x)$. Exponential decay is convenient to compute when used for time decay as it can be maintained easily using a single register (It is not known, however, if it is simpler to compute than other decay functions on other metric spaces). Exponential-decay over time captures some natural phenomena such as radioactive decay.

Polynomial decay has the form $g(x) = 1/(1 + ax^{\alpha})$ for parameters $a > 0$ and $\alpha \geq 1$. Many natural effects (for example, electro-magnetic radiation) have polynomial decrease with distance. Polynomial decay is often a natural choice when a smooth decay is desired and when Exponential decay is too drastic [9]. In many natural graphs (like d-dimensional grids), the neighborhood size increases polynomially with the distance and exponential decay suppresses longer horizons.

Aggregate Functions

Any aggregate function over a set of weighted items can be applied to the decayed weights of the items to obtain a corresponding decaying aggregate.

The *decaying sum* is the sum of the decaying weights of the items. In the special case where the weights are binary, this aggregate is referred to as the *decaying count*. A related aggregate is the *decaying average*, defined as the ratio of the decaying sum and the decaying count of the items. Important aggregates that can be reduced to (approximate) decaying sums are (approximate) decaying variance and moments [4]. These aggregates can also be defined with respect to a subpopulation of items that is specified by a predicate.

Other aggregates are *decaying weighted random sample of a certain size* with or without replacement and derived aggregates such as quantiles (using a folklore technique, an approximate quantile with confidence $1 - \delta$ can be obtained by taking the p quantile of $O(\in^{-2} \ln\delta^{-1})$ independent random samples) and heavy hitters (all distinct identifiers with total decaying weight above some threshold).

Computational Challenges

A decaying aggregate of a location with respect to a decay function can be computed by processing all items, calculating the distances and computing the decayed weights, and finally computing the desired aggregate on this set. This approach provides exact answers (up to numerical computation errors) but is highly inefficient and often infeasible.

This naive approach requires a linear pass over all items for each location and decay function pair of interest in aggregates with respect to multiple locations or multiple decay functions.

On massive data sets, this approach is infeasible: on massive data stream, one can not store the full history of observed items and in distributed data sets, one can not replicate the full information at every node.

The common approach for massive data sets is to use summaries that enable *more efficient* computation of *approximate* aggregate values. Algorithms maintain a concise data structures that “summarizes” the full information. The size of these data structure determines the amount of book keeping and/or communication required. A desirable feature of these summaries is that they support aggregations over *selected subpopulations* of the items.

Decaying aggregation, where summaries must be able to support multiple locations and decay functions, imposes greater challenges.

All Distances Sketches

Useful data structures for decayed aggregation are *all-distances sketches*. The all-distances sketch of a location is a concise encoding of sketches of the weighted sets of items that are within some distance from the location, for all possible distances. If the sketches support approximate sum, the all-distances sketch supports decaying sum with respect to any decay function.

In many applications, it is desired to efficiently obtain these sketches for multiple locations. Fortunately, all-distances sketches can be computed and maintained efficiently in many settings.

Exponential histograms [1] and the wave summaries [14] are applicable on data streams. For the more general network setting, *all-distances k-mins sketches* [3], which can be computed efficiently in a distributed setting [5]. *All-distances bottom-k sketches*, which provide tighter estimates of the decaying sum than all-distances *k*-mins sketches were proposed in [6].

The data structure used in [5] for the Euclidean plane essentially encodes all-distances *k*-mins sketch for any query point in the plane.

Key Applications

Applications of decaying aggregation can be classified into two main categories.

The first category is *prediction or estimation* of a value at a location. In this context, the data items are viewed as samples or measurements from some underlying smooth distribution and the decayed aggregate is a way to estimate or predict the value or some statistics at a query location. For example, the items are measurements of some environmental parameter (humidity, temperature, air pollution) collected by a sensor network and the decaying aggregate is an estimate of the value at other location. Application in this category are [7,13,16–18].

The second category is some measure of *influence*: the items constitute the complete data set and the decayed aggregate is a measure of influence or closeness of a property to a location. For example, items are undirected sources of electro magnetic radiation and the decaying aggregate is a measure of the radiation level at a query location. Other applications in this category are content-based routing in p2p networks [11].

Cross-references

► Approximate Query Processing
► Data Sketch/Synopsis
► Data Streams

Recommended Reading

1. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data stream systems. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002.
2. Bremler-Barr A., Cohen E., Kaplan H., and Mansour Y. Predicting and bypassing internet end-to-end service degradations. In Proc. 2nd ACM SIGCOMM Workshop on Internet Measurement, 2002.
3. Cohen E. Size-estimation framework with applications to transitive closure and reachability. J. Comput. Syst. Sci., 55:441–453, 1997.
4. Cohen E. and Kaplan H. Efficient estimation algorithms for neighborhood variance and other moments. In Proc. 15th Annual ACM-SIAM Symp. on Discrete Algorithms, 2004.
5. Cohen E. and Kaplan H. Spatially-decaying aggregation over a network: model and algorithms. J. Comput. Syst. Sci., 73:265–288, 2007.
6. Cohen E. and Kaplan H. Summarizing data using bottom-k sketches. In Proc. ACM PODC'07 Conf., 2007.
7. Cohen E., Kaplan H., and Oldham J.D. Managing TCP connections under persistent HTTP. Comput. Netw., 31:1709–1723, 1999.
8. Cohen E. and Strauss M. Maintaining time-decaying stream aggregates. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003.
9. Cohen E. and Strauss M. Maintaining time-decaying stream aggregates. J. Algorithms, 59:19–36, 2006.
10. Cormode G., Tirthapura S., and Xu B. Time-decaying sketches for sensor data aggregation. In Proc. ACM PODC'07 Conf., 2007.
11. Crespo A. and Garcia-Molina H. Routing indices for peer-to-peer systems. In Proc. 18th Int. Conf. on Data Engineering, 2002.
12. Datar M., Gionis A., Indyk P., and Motwani R. Maintaining stream statistics over sliding windows. SIAM J. Comput., 31 (6):1794–1813, 2002.
13. Floyd S. and Jacobson V. Random early detection gateways for congestion avoidance. IEEE/ACM Trans. Netw., 1(4), 1993.
14. Gibbons P.B. and Tirthapura S. Distributed streams algorithms for sliding windows. In Proc. 14th Annual ACM Symp. on Parallel Algorithms and Architectures, 2002, pp. 63–72.
15. Guibas L.J., Knuth D.E., and Sharir M. Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica, 7:381–413, 1992.
16. Hua M., Pei J., Fu A.W.C., Lin X., and Leung H.-F. Efficiently answering top-k typicality queries on large databases. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
17. Jacobson V. Congestion avoidance and control. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988.
18. Keshav S., Lund C., Phillips S., Reingold N., and Saran H. An empirical evaluation of virtual circuit holding time policies in IP-over-ATM networks. J. Select. Areas Commun., 13 (8):1371–1382, 1995.
19. Scott D.W. Multivariate Density Estimation: Theory, Practice and Visualization. Wiley, New York, 1992.

Decentralized Data Integration System

► Peer Data Management System

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:34

Decision Rule Mining in Rough Set Theory

Tsau Young (T.Y.) Lin
San Jose State University, San Jose, CA, USA

Synonyms

Decision rules; Classification rules; Rough Set theory (RST); Extensional relational databases (ERDB)

Definition

Rough set theory (RST) has two formats, abstract and table formats. In this entry, abstract format is hardly touched. The table format, by definition, is a theory of extensional relational databases (ERDB). However, their fundamental goals are very different. RST is on data analysis and mining, while databases are on data processing.

In RST, a relation, which is also known as information table, is called a decision table (DT), if the attributes are divided into two disjoint families, called conditional and decision attributes. *A tuple in such a DT, is interpreted as a decision rule, namely, the conditional attributes functionally determine decision attributes.* A sub-relation is called a Value Reduct, if it consists of a minimal subset of minimal length decision rules that has the same decision power as the original decision table. RST has the following distinguished theorem.

Every decision table can be reduced to a value reduct [5].

This theorem has been regarded as a data mining theorem. However, in its true natural, it is a data reduction theorem. There are some trivial points in this theorem: If there is a conditional attribute that is a candidate key, then the column is an attribute reduct, and each attribute value is a value reduct. There are more than one value reducts.

One fundamental weak point of this theorem is the assumption that every tuple is a rule. So a new theory, based on the assumption that only the high frequency tuples can be regarded as rules, has started, but not very far [3,4].

Historical Background

Rough set theory (RST) has two formats, abstract and table formats. In the abstract format, a rough set is represented by the upper and lower approximations of equivalence classes. In terms of pure math, they are the closure and interior of special topological spaces, called clopen spaces. However, this entry does not cover this aspect.

The table format of RST is a formal theory derived from the logical studies of tables (relations). So theoretically speaking, RST is a sub-theory of ERDB, however, historically, they were developed under different philosophies. RDB assumes the semantics of data is known and focuses on organizing data through its semantics. On the other hand, RST assumes data is completely defined by the given table, and focuses on data analysis, rule discovery and etc.

Foundations

Though RST has been regarded as a methodology of data mining, the core results are actually data reduction or rule reduction. Namely, RST provides a methodology to reduce a decision table to a minimal but an equivalent table. They are equivalent in the sense that both tables provide the same decision power. This minimal decision table is called value reduct. This reduction is not unique. In other words, given a decision table, there is a set of value reducts, each is equivalent to the original table.

In this section some fundamental concepts will be introduced, In addition, the procedure will be illustrated by an example. There are two major steps; one is the attribute reducts that are generalization of candidate keys. The second step is tuple reduction.

Attribute(Column) Reducts and Candidate Keys

In a relation, an attribute, or a set of attributes K is called a candidate key if all attributes of the relation is functionally depended on K (or K functionally determines every attribute), and K is a minimal set [1]. In other words, candidate key is a set of "independent variables" that all other attributes can be expressed as functions of the candidate keys. In decision table, the corresponding concept is equivalent to find a set "independent conditions" for "if-then" rules.

Let S = $(U, \mathcal{A} = \mathcal{C} \bigcup \mathcal{D}, Dom, \rho)$ be a decision table, where

$$\mathcal{C} = A_1, A_2, \ldots, A_i, \ldots, A_n,$$

$$\mathcal{D} = B_1, B_2, \ldots, B_i, \ldots, B_m.$$

Then S is said to be a consistent decision table, if $\mathcal{C}$ functionally determine $\mathcal{D}$.

Definition 1 *$\mathcal{B}$ is called a attribute reduct of* S, *if $\mathcal{B}$ is a minimal subset of $\mathcal{C}$ such that $\mathcal{B}$ functionally determine $\mathcal{C}$.*

It is clear that such a $\mathcal{B}$ is not necessarily unique. If $\mathcal{D}$ is chosen to be the whole table $\mathcal{A}$, then the reduct is the extensional candidate key. The algorithm of finding the attribute reduct is quite straightforward; it is nearly the same as that of finding a candidate key, which can be found in almost any database text.

Value Reducts – Simplest Decision Rules

In the step of attribute reduct, the reduction algorithm find the sets of minimal columns. In this section, a similar procedure is considered, but one tuple at a time. The minimal set is called value reduct.

Illustration

Without losing the generality, the idea can be explained by the following table:

U	Location	Test	New	Case	Result
ID-1	Houston	10	92	03	10
ID-2	San Jose	10	92	03	10
ID-3	Palo Alto	10	92	04	10
ID-4	Berkeley	11	92	04	50
ID-5	New York	11	92	04	50
ID-6	Atlanta	11	92	04	50
ID-7	Chicago	11	93	70	99
ID-8	Baltimore	11	93	70	99
ID-9	Seattle	11	93	70	99
ID-10	Chicago	51	95	70	94
ID-11	Chicago	51	95	70	95

Select a Decision Table From the table given above, the following decision table will be considered: Let U be the universe, RESULT be the decision attribute, and C = TEST, NEW, CASE be the set of conditional attributes.

Each tuple in the DT is considered as an if-then rule.

Split the Decision Table Two tuples ID-10 and ID-11 form a table of inconsistent rules. Nine tuples ID-1 to ID-9 form a consistent table. From now on the term in this section "decision table" is referred to this consistent table.

Decision Classes The equivalence relation IND(RESULT) classifies entities into three equivalence classes, called decision classes

$$\text{DECISION1} = \{\text{ID-1}, \text{ID-2}, \text{ID-3}\} = [10]_{RESULT}$$
$$\text{DECISION2} = \{\text{ID-4}, \text{ID-5}, \text{ID-6}\} = [50]_{RESULT}$$
$$\text{DECISION3} = \{\text{ID-7}, \text{ID-8}, \text{ID-9}\} = [99]_{RESULT}$$

Condition Classes Let C = {TEST, NEW, CASE} be the conditional attributes. The equivalence relation IND (C) classifies entities into four equivalence classes, called condition classes:

$$\text{IND(C)-1} = \{\text{ID-1}, \text{ID-2}\},$$
$$\text{IND(C)-3} = \{\text{ID-4}, \text{ID-5}, \text{ID-6}\},$$
$$\text{IND(C)-2} = \{\text{ID-3}\},$$
$$\text{IND(C)-4} = \{\text{ID-7}, \text{ID-8}, \text{ID-9}\}$$

Knowledge Dependencies Pawlak regards a partition (classification) as a knowledge, and observe that an attribute defines a partition on the entities. So relationships between attributes are regarded as relationships between partitions, and hence between knowledges.

Observe that the entities that are indiscernible by conditional attributes are also indiscernible by decision attributes, namely the following inclusions are obtained

$$\text{IND(C)-1} \subseteq \text{DECISION1};$$
$$\text{IND(C)-2} \subseteq \text{DECISION1};$$
$$\text{IND(C)-3} \subseteq \text{DECISION2};$$
$$\text{IND(C)-4} \subseteq \text{DECISION3}.$$

These inclusions imply that the equivalence relation IND(C) is a refinement of IND(RESULT) in mathematics. In RST, they imply that the knowledge IND (RESULT) is knowledge depended on (coarser than) the knowledge IND(C). Or equivalently, RESULT are Knowledge Depended on C.

If-then Rules Knowledge dependences can be expressed by if-then rules:

1. If TEST = 10, NEW = 92, CASE = 03, then RESULT = 10
2. If TEST = 10, NEW = 92, CASE = 04, then RESULT = 10
3. If TEST = 11, NEW = 92, CASE = 04, then RESULT = 50
4. If TEST = 11, NEW = 93, CASE = 70, then RESULT = 99

Attribute (Column) Reducts It is easy to verify that {TEST, NEW} and {TEST CASE} are two attribute reducts. Note that row ID-10 and ID-11 do not contribute to the consistent decision; so they should be ignored (For clarity, these have been removed.)

The two tables can be expressed as two sets of uniformly shortened rules:

U	Test	New	Result
ID-1	10	92	10
ID-2	10	92	10
ID-3	10	92	10
ID-4	11	92	50
ID-5	11	92	50
ID-6	11	92	50
ID-7	11	93	99
ID-8	11	93	99
ID-9	11	93	99

U	Test	Case	Result
ID-1	10	03	10
ID-2	10	03	10
ID-3	10	04	10
ID-4	11	04	50
ID-5	11	04	50
ID-6	11	04	50
ID-7	11	70	99
ID-8	11	70	99
ID-9	11	70	99

Set One:

1. If TEST = 10 and NEW = 92, then RESULT = 10
2. If TEST = 11 and NEW = 92, then RESULT = 50
3. If TEST = 11 and NEW = 93, then RESULT = 99

Set Two:

1. If TEST = 10, and CASE = 3, then RESULT =10
2. If TEST = 10, and CASE = 4, then RESULT = 10
3. If TEST = 11, and CASE = 4, then RESULT = 50
4. If TEST = 11, and CASE = 70, then RESULT = 99

More plainly, the *consistent* decisions made from the original decision table can be accomplished equivalently by either one of the two sets.

U	Test	New	Result
ID-1	10	X	10
ID-2	10	X	10
ID-3	10	X	10
ID-4	11	92	50
ID-5	11	92	50
ID-6	11	92	50
ID-7	11	X	99
ID-8	11	X	99
ID-9	11	X	99
ID-10	51	95	94
ID-11	51	95	95

U	Test	Case	Result
ID-1	10	X	10
ID-1	X	03	10
ID-2	10	X	10
ID-2	X	03	10
ID-3	10	04	10
ID-4	11	04	50
ID-5	11	04	50
ID-6	11	04	50
ID-7	11/X	70	99
ID-8	11/X	70	99
ID-9	11/X	70	99
ID-10	51	70	94
ID-11	51	70	95

Value Reducts First, observe that ID-10 and ID-11 have been moved back. The reasons are that the two inconsistent rules will have impact to the final form of value reduct; see below. The first table gives

1. FIRST SET of Shortest Rules
   Rule1 If TEST = 10, then RESULT = 10
   Rule2 If TEST = 11 and NEW = 92, then RESULT = 50
   Rule3 If NEW = 93, then RESULT = 99

A casual user will not realize that the rules are derived from the consistent sub-table. So a more natural view is to look at the whole table. Fortunately these three rules are not disturbed by the inclusion of ID-10 and ID-11 tuples. In other words, The FIRST SET is a value reduct of the consistent rules.

The second table provides two sets of Shortest Rules

2. SECOND SET of Shortest Rules are: [Rule4a], [Rule6], [Rule7] in the list below.
3. THIRD SET of Shortest Rules are: [Rule4b], [Rule5], [Rule6], [Rule7] where each bracket [•] is referring to the following rules:
   Rule4a If TEST = 10, then RESULT = 10
   Rule4b If CASE = 3, then RESULT = 10
   Rule5 If TEST = 10 then RESULT = 10
   Rule6 If TEST = 11 and CASE = 4, then RESULT = 50
   Rule7 If TEST = 11 and CASE = 70, then RESULT = 99
   C-Rule8 If CASE = 70, then RESULT = 99 (This rule is valid only on consistence table.)

Note that the choice of [C-Rule8] is valid, if only the consistence table is considered. Pawlak had adopted this view [5]. However, as remarked earlier, a more natural view is: the value reduct is derived from the original whole table. In this case, [C-Rule8] is not a rule, as it is "in conflict" with ID-10 and ID-11. So Rule7 is used.

The illustrations on how to find the minimal sets of shortest rules is completed. There are three solutions (a modified view of Lin): (1) FIRST SET (Rule 1-3) (2) SECOND SET (Rule4a, 6, 7), and (3) THIRD SET (Rule4b, 6, 7). Taking the approach of Pawlak's book, the FIRST and SECOND SETS are the same as those of the author, but Third SET is (Rule4b, 6, C-Rule8).

Several comments are in order. In rough set theory, Pawlak had assumed that every tuple in a decision table is a rule; this assumption is debatable. So the view – only high frequency tuples (as in frequent itemsets) can be regarded as rules – should be explored. There are such efforts, but not very far [3,4]. Other variations also exist and should be explored.

Cross-references

► Data Mining
► Data Reduction
► Decision Tree Classification
► Decision Trees
► Frequent Itemsets and Association Rules
► Granular Computing
► Rough Set Approach

Recommended Reading

1. Gracia-Molina H., Ullman J., and Windin J. Database Systems The Complete Book, Prentice-Hall, Englewood, Cliffs, NJ, 2002.
2. Lee T.T. Algebraic theory of relational databases. Bell Syst. Tech. J., 62(10):3159–3204, December 1983.
3. Lin T.Y. Rough set theory in very large databases. In Symp. in Modelling Analysis and Simulation, 1996, pp. 936–941.
4. Lin T.Y. and Han J. High frequent value reduct in very large databases. RSFDGrC, 2007, pp. 346–354.
5. Pawlak Z. Rough Sets. Theoretical Aspects of Reasoning about Data. Kluwer, Dordecht, 1991.

Decision Rules

► Decision Rule Mining in Rough Set Theory

Decision Rules, Classification

► Deductive Data Mining Using Granular Computing

Decision Support

► Clinical Decision Support

Decision Tree

► Decision Tree Classification

Decision Tree Classification

Alin Dobra
University of Florida, Gainesville, FL, USA

Synonyms

Decision tree; Classification tree

Definition

Decision tree classifiers are decision trees used for classification. As any other classifier, the decision tree classifiers use values of attributes/features of the data to make a class label (discrete) prediction. Structurally, decision tree classifiers are organized like a *decision tree* in which simple conditions on (usually single) attributes label the edge between an intermediate node and its children. Leaves are labeled by class label predictions. A large number of learning methods have been proposed for decision tree classifiers. Most methods have a tree growing and a pruning phase. The tree growing is recursive and consists in selecting an attribute to split on and actual splitting conditions then recurring on the children until the data corresponding to that path is pure or too small in size. The pruning phase eliminates part of the bottom of the tree that learned noise from the data in order to improve the generalization of the classifier.

Historical Background

Decision tree classifiers were first introduced by Breiman and his collaborators [2] in 1984 in the statistics community. While the impact in statistics was not very significant, with their introduction in 1986 by Quinlan in machine learning literature[11], the decision tree classifiers become of the premier classification method. A large amount of research was published on the subject since in the machine learning literature. There was a renewed interest in decision tree classifiers in the 1990 in the data-mining literature due to the fact that scalable learning is nontrivial and, more importantly, the business community prefers decision tree classifiers to other classification methods due to their simplicity. A comprehensive survey of the work on decision tree classifiers can be found in [10].

Foundations

Decision tree classifiers are especially attractive in a data mining environment for several reasons. First, due to their intuitive representation, the resulting model is easy to assimilate by humans [2]. Second, decision tree classifiers are non-parametric and thus especially suited for exploratory knowledge discovery. Third, decision tree classifiers can be constructed relatively fast compared to other methods [8]. And last, the accuracy of decision tree classifiers is comparable or superior to other classification models [8].

As it is the case for most classification tasks, the kind of data that can be represented by decision tree classifiers is of tabular form, as depicted in Table 1. Each data point occupies a row in the table. The names of columns are characteristics of the data and are called *attributes*. Attributes whose domain is numerical are called *numerical attributes*, whereas attributes whose domain is not numerical are called *categorical attributes*. One of the categorical attributes is designated as the *predictive attribute*. The predictive attribute needs to be predicted from values of the other attributes. For the example in Table 1, “Car Type” is a categorical attribute, “Age” is a numerical attribute and “Lives in Suburb?” is the predictor attribute.

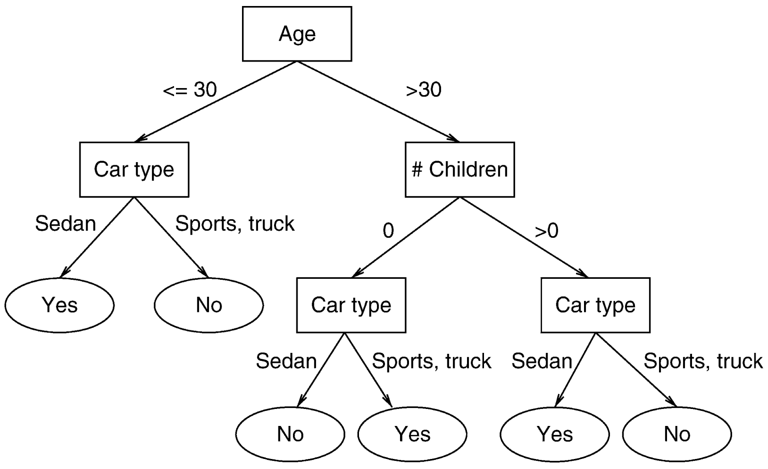
Figure 1 depicts a classification tree, which was built based on data in Table 1. It predicts if a person lives in a suburb based on other information about the person. The predicates, that label the edges (e.g., Age $\leq$ 30), are called *split predicates* and the attributes involved in such predicates, *split attributes*. In traditional classification and regression trees only deterministic split predicates are used (i.e., given the split predicate and the value of the the attributes, it can be determined if the attribute is true or false). Prediction with classification trees is done by navigating the tree on true predicates until a leaf is reached, when the prediction in the leaf (YES or NO in the example) is returned.

Formal Definition

A classification tree is a directed, acyclic graph $\mathcal{T}$ with tree shape. The root of the tree – denoted by Root($\mathcal{T}$) – does not have any incoming edges. Every other node has exactly one incoming edge and may have 0, 2 or more outgoing edges. A node T without outgoing edges is called *leaf node*, otherwise T is called an *internal node*. Each leaf node is labeled with one class label;

Decision Tree Classification. Table 1. Example training database

Car Type	Driver Age	Children	Lives in Suburb?
sedan	23	0	yes
sports	31	1	no
sedan	36	1	no
truck	25	2	no
sports	30	0	no
sedan	36	0	no
sedan	25	0	yes
truck	36	1	no
sedan	30	2	yes
sedan	31	1	yes
sports	25	0	no
sedan	45	1	yes
sports	23	2	no
truck	45	0	yes



Decision Tree Classification. Figure 1. Example of decision tree classifier for training data in Table 1.

each internal node T is labeled with one attribute variable X_T, called the *split attribute*. The class label associated with a leaf node T is denoted by Label(T).

Each edge (T, T') from an internal node T to one of its children T' has a predicate $q_{(T,T')}$ associated with it where $q_{(T,T')}$ involves only the splitting attribute X_T of node T. The set of predicates Q_T on the outgoing edges of an internal node T must contain disjoint predicates involving the split attribute whose conjunction is true – for any value of the split attribute exactly one of the predicates in Q_T is true. The set of predicates in Q_T will be reffered to as *splitting predicates of T.*

Given a classification tree $\mathcal{T}$, the associated classifier is defined as $C_{\mathcal{T}}(x_1, \ldots, x_m)$ in the following recursive manner:

$$C(x_1, \ldots, x_m, T) = \begin{cases} \text{Label}(T) & \text{if } T \text{ is a leaf node} \\ C(x_1, \ldots, x_m, T_j) & \text{if } T \text{ is an internal node, } X_i \text{ is label of } T\text{, and } q(T, T_j)\ (xi) = \text{true} \end{cases}$$

$$C_{\mathcal{T}}(x_1, \ldots, x_m) = C(x_1, \ldots, x_m, \text{Root}(\mathcal{T}))$$

thus, to make a prediction, start at the root node and navigate the tree on true predicates until a leaf is reached, when the class label associated with it is returned as the result of the prediction.

Building Decision Tree Classifiers

Several aspects of decision tree construction have been shown to be NP-hard. Some of these are: building optimal trees from decision tables [6], constructing minimum cost classification tree to represent a simple function [4], and building optimal classification trees in terms of size to store information in a dataset [14].

In order to deal with the complexity of choosing the split attributes and split sets and points, most of the classification tree construction algorithms use the greedy induction schema in Fig. 2. It consists in deciding, at each step, upon a split attribute and split set or point, if necessary, partitioning the data according with the newly determined split predicates and recursively repeating the process on these partitions, one for each child. The construction process at a node is terminated when a termination condition is satisfied. The only difference between the two types of classification trees is the fact that for k-ary trees no split set needs to be determined for discrete attributes. By specifying the split attribute selection criteria and the split point selection criteria various decision tree classifier construction algorithms are obtained.

Once a decision is made to make a node a leaf, the majority class is used to label the leaf and will be the prediction made by the tree should the leaf be reached.

Split Attribute and Split Point Selection

At each step in the recursive construction algorithm, a decision on what attribute variable to split is made. The purpose of the split is to separate, as much as possible, the class labels from each others. To make this intuition useful, a metric that estimates how much the separation of the classes is improved when a particular split is performed is needed. Such a metric is called a *split criteria* or a *split selection method.*

```
Input: node $T$, data-partition $D$, splits election method $\mathcal{V}$
Output: classification tree $\mathcal{T}$ for $D$ rooted at $T$

Top-Down Classification Tree Induction Schema:
BuildTree(Node $T$, data-partition $D$, split attribute selection method $\mathcal{V}$)
(1)   Apply $\mathcal{V}$ to $D$ to find the split attribute $X$ for node $T$.
(2)   Let $n$ be the number of children of $T$.
(2)   if ($T$ splits)
(3)      Partition $D$ into $D_1$,..., $D_n$ and label note $T$ with split attribute $X$
(4)      Create children nodes $T_1$,...,$T_n$ of $T$ and label the edge ($T$, $T_i$)
               with predicate $q(T, T_i)$
(5)      foreach $i \in \{1,.., n\}$
(6)         BuildTree($T_i$, $D_i$, $\mathcal{V}$)
(7)      endforeach
(8)   else
(9)      Label $T$ with the majority class label of $D$
(10)  endif
```

Decision Tree Classification. Figure 2. Classification tree induction schema.

There is extensive research in the machine learning and statistics literature on devising split selection criteria that produce classification trees with high predictive accuracy [10].

The most popular class of split selection methods are *impurity-based* [2,11]. The popularity is well deserved since studies have shown that this class of split selection methods have high predictive accuracy [8], and at the same time they are simple and intuitive. Each impurity-based split selection criteria is based on an impurity function $\Phi(p_1,\ldots,p_k)$, with p_j interpreted as the probability of seeing the class label c_j. Intuitively, the impurity function measures how impure the data is. It is required to have the following: (i) to be concave, (ii) to have a unique maximum at $(1/k,\ldots,1/k)$ (the most impure situation), and (iii) to achieve the minimum value for $(1, 0,\ldots,0), (0, 1, 0,\ldots,0),\ldots, (0,\ldots,0, 1)$ (i.e., when all data has the same class label). Given such an impurity measure, the impurity of a node T is $i(T) = \Phi(P[C = c_1|T], \ldots, P[C = c_k|T])$, where $P[C = c_j|T]$ is the probability that the class label is c_j given that the data reaches node T. These statistics are computed from the training data in the process of building the decision tree classifier.

Given a set Q of split predicates on attribute variable X that split a node T into nodes $T_1,\ldots,T_n$, define the *reduction in impurity* as:

$$\begin{aligned}\Delta i(T, X, Q) &= i(T) - \sum_{i=1}^{n} P[T_i|T] \, . \, i(T_i) \\ &= i(T) - \sum_{i=1}^{n} P[q_{(T,T_i)}(X)|T] \, . \, i(T_i)\end{aligned} \quad (1)$$

Intuitively, the reduction in impurity is the amount of purity gained by splitting, where the impurity after split is the weighted sum of impurities of each child node.

Two popular impurity measures are:

$$\texttt{Gini index:}\ \mathrm{gini}(p_1, \ldots, p_n) = 1 - \sum_{i=1}^{n} p_i^2$$

$$\textbf{Entropy:}\ \mathrm{entropy}(p_1, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log(p_i)$$

When used in conjunction with the (1), they produce the *Gini Gain* and *Information Gain* split point selection criteria. A split attribute criteria can simply be defined as the largest value of the split point selection criteria for any predicate involving the attribute (i.e., the best split). In this way, the best split point is determined simultaneously with the evaluation of an attribute thus no other criteria is necessary.

The selection of the attribute can be dissociated from the selection of the split point. A measures that test the usefulness of a split on an attribute without considering split points is χ^2-test.

Some comments on efficiently finding the split point for the two types of attributes, categorical and numerical, are in order. For numerical attributes, only splits of the form $X > 10$ are considered thus only as many split as there are data-points are possible. For categorical attributes in the case when k-ary splits are allowed (Quinlan decision tree classifier), there is only one possible split on an attribute. The situation is more complicated for binary split trees (Breiman et al.) since there are an exponential number of possible split points (sets in this case) to consider. Fortunately, in this last situation a powerful result due to Breiman [2] leads to a linear algorithm after a sort in a specific order is performed.

Tree Pruning

An important question with respect to building decision tree classifiers is when to stop the growing of the tree. Since the estimation of probabilities that make up the formulas for the split criteria becomes less and less statistically reliable as the tree is build (data is fragmented at an exponential speed due to splits) the tree will eventually learn noise. In machine learning terminology this is called *overfitting* and should be avoided to the greatest extent possible.

Finding the point where the actual learning stops and overfitting starts is problematic. For decision tree classification there are two distinct approaches to addressing this problem: (i) detect overfitting during tree growth and stop learning, and (ii) grow a large tree and, using extra independent data, determine what part of the tree is reliable and what part should be discarded.

A statistical test like χ^2-test can be used to detect the point where overfitting occurs. This turns out to work in some circumstances but not others. This method of stopping the tree growth is preferred when the χ^2-test is used for attribute selection as well.

The most popular method to producing good quality trees is to *prune* an overly large tree. The most popular method to pruning is *Quinlan's Re-substitution Error Pruning*. Re-substitution error pruning consists of eliminating subtrees in order to obtain a tree with the smallest error on the *pruning set*, a separate part of the data used only for pruning. To achieve this, every node estimates its contribution to the error on pruning data when the majority class is used as an estimate. Then, starting from the leaves and going upward, every node compares the contribution to the error by using the local prediction with the smallest possible contribution to the error of its children (if a node is not a leaf in the final tree, it has no contribution to the error, only leaves contribute), and prunes the tree if the local error contribution is smaller – this results in the node becoming a leaf. Since, after visiting any of the nodes the tree is optimally pruned – this is the invariant maintained – when the overall process finishes, the whole tree is optimally pruned.

Other pruning techniques can be found in [9]. They tend to be significantly more complicated than re-substitution error pruning.

Key Applications

Scientific classification, medical diagnosis, fraud detection, credit approval, targeted marketing, etc.

URL to Code

http://www.dtreg.com/
http://eric.univ-lyon2.fr/∼ricco/sipina_overview.html
http://www.statistics.com/resources/glossary/c/cart.php
http://www.statsoft.com/textbook/stcart.html
http://www.salfordsystems.com/

Cross-references

► Classification
► Decision Trees

Recommended Reading

1. Agresti A. Categorical data analysis. John Wiley and Sons. (1990).
2. Breiman L., Friedman J.H., Olshen R.A., and Stone C.J. (1984). Classification and regression trees. Belmont: Wadsworth.
3. Buntine W. Learning classification trees. Artificial Intelligence frontiers in statistics Chapman & Hall, London. (pp. 182–201).
4. Cox L.A., Qiu Y., and Kuehner W. Heuristic least-cost computation of discrete classification functions with uncertain argument values. Annals of Operations Research, 21, 1–30. (1989).
5. Frank E. Pruning decision trees and lists. Doctoral dissertation, Department of Computer Science, University of Waikato, Hamilton, New Zealand. (2000).
6. Hyafil L., and Rivest R.L. Constructing optimal binary decision trees is np-complete. Information Processing Letters, 5, 15–17. (1976).
7. James M. Classification algorithms.Wiley. (1985).
8. Lim T.-S., Loh W.-Y., and Shih Y.-S. An empirical comparison of decision trees and other classification methods (Technical Report 979). Department of Statistics, University of Wisconsin, Madison. (1997).
9. Loh W.-Y. and Shih Y.-S. Split selection methods for classification trees. Statistica Sinica, 7. (1997).
10. Murthy S.K. Automatic construction of decision trees from data: A multi-disciplinary survey. Data Mining and Knowledge Discovery. (1997).
11. Quinlan J.R. Induction of decision trees. Machine Learning, 1, 81–106. (1986).
12. Quinlan J.R. Learning with Continuous Classes. In: Proc. 5th Australian Joint Conference on Artificial Intelligence (pp. 343–348). (1992).
13. Quinlan J.R. C4.5: Programs for machine learning. Morgan Kaufman. (1993b).

14. Murphy O.J. and Mccraw R.L. Designing storage efficient decision trees. IEEE Transactions on Computers, 40, 315–319. (1991).

Decision Trees

Alin Dobra
University of Florida, Gainesville, FL, USA

Synonyms

Classification trees; DT

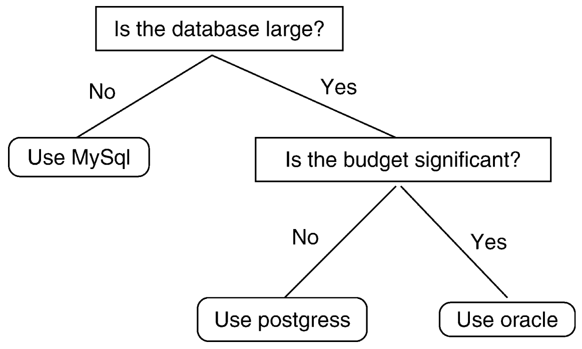
Definition

Decision trees are compact tree like representations of conditions that specify when a decision should be applied together with the actions/decision. Decision trees consist into intermediate nodes and leaf nodes. The outgoing edges from intermediate nodes are labeled by conditions. The leaf nodes are labeled by decisions or actions. The way decision trees are used is by starting at the root then navigating down on true conditions until a leaf is reached. The action or decision in the leaf is then taken. Decision trees are just a compact representation of decision rules: the condition under which an action is taken is the conjunction of conditions starting at the root of the decision tree and leading to the leaf labeled by the action. An example of a decision tree is given in Fig. 1.

Key Points

Decision trees are an important type of representation for any kind of complex set of decisions that are conditioned on multiple factors. They are more compact than decision rules but less flexible. The main appeal of decision trees is their intuitiveness; it is very easy to understand how the decision is taken.

While they have uses in any areas that need decision support of some kind, in databases their main use is in specifying decision tree classifiers and decision tree regressors. Decision trees are mostly used in one of these two forms in databases.



Decision Trees. Figure 1. Example of a decision tree for selecting a database.

Cross-references

► Decision Tree Classification
► Scalable Decision Tree Construction

Recommended Reading

1. Lindley D.V. Making Decisions. Wiley, Hoboken, NJ, USA, 1991.

Declarative Networking

Timothy Roscoe
ETH Zurich, Switzerland

Synonyms

Declarative overlay networks

Definition

Declarative Networking refers to the technique of specifying aspects of networked systems, such as routing algorithms, in terms of declarative queries over distributed network state. These queries are then executed by a distributed query processor to obtain the same effect as executing an implementation of the algorithm in an imperative language such as C or Java. Executable descriptions of distributed algorithms as queries are typically much more concise than imperative implementations, and more amenable to automated analysis.

Historical Background

Declarative Networking emerged in about 2004 as an application of results in data management and logic programming to problems of network overlay maintenance. Its roots can be traced in several areas: attempts to describe real-world network configurations formally, e.g., [11], network management systems built over a declarative framework, such as IBM Tivoli Console and various research systems, e.g., [12,15], and perhaps most significantly the field of distributed query processing systems (e.g., [5,6,10]).

The observation that distributed query processors need to construct overlay networks of some form to route data led to the idea that the routing protocols required might be specified declaratively. The idea of describing Internet routing protocols (and new variants on them) as queries over distributed data was proposed in [9], and shortly afterwards real systems for building overlay networks from declarative specifications began to appear. P2 [8] uses a variant of the Datalog language and pushes most aspects of the distributed algorithm into the declarative realm, while Node Views [3] uses a SQL-like language [2] and builds overlay networks as views over an underlying node database.

Further work has expanded the applicability of the approach from routing algorithms networks to more general distributed algorithms such as Chandy-Lamport consistent snapshots [14], specifying security properties of networks [1], and building complete sensor network applications [4].

Foundations

The basic principle of declarative networking can be illustrated by reference to the example of routing protocols. The purpose of a routing protocol is to continously maintain a *routing table* at each node in the network. This table provides a mapping from destination addresses to “next hop” nodes – those which are directly connected to the node where the table resides. The collection of routing tables in a network can be viewed as the result of a distributed calculation whose inputs are external data such as node liveness, link load levels, user-specified policy, etc. If one represents such external data as relations, routing tables can be regarded as a distributed view over such relations, and consequently the routing – the process of maintaining the routing tables – can be regarded as an instance of distributed view maintenance.

This can be illustrated further with the example of link-state routing (as used by the widespread Internet routing protocols OSPF and IS-IS), where connectivity information is flooded globally through the network and each router performs shortest-path computations on the complete connectivity graph. Using the simplified notation of [9], based on Datalog, one can write:

```
path(S,D,P,C) :- link(S,D,C),
                 P = f concatPath(link
                 (S,D,C), nil).
path(S,D,P,C) :- link(S,Z,C1), path
                 (Z,D,P2,C2),
                 C = f_sum(C1, C2),
                 P = f concatPath
                 (link(S,Z,C1), P2).
```

These first two query rules give the standard inductive defintion of reachability in a network: there is a path `P` from source node `S` to destination `D` with cost `C` if there is a (direct) link of cost `C` from `S` to `D`, or if there is a path `P2` to `D` from a node `Z` adjacent to `S`, and `C` is the sum of the link cost `C1` to `Z` and the path cost `C2` from `Z` to `D`.

Two further rules can compute the best path from `S` to `D`, using the standard Datalog function for aggregates:

```
bestPathCost(S,D,AGG<C>) :- path
                          (S,D,P,C).
bestPath(S,D,P,C) :- bestPathCost
                     (S,D,C),
                     path(S,D,P,C).
```

Note that the definition of “best” in this example is deliberately unbound: by changing the sum function `f_sum` and the aggregate `AGG`, a variety of network metrics can be used. This illustrates some of the key claimed benefits of declarative networking: conciseness of specification, and ease of modification.

While the four rules above specify a link-state routing algorithm in some sense, they of course say nothing about how such a specification is to executed, *where* the rules are evaluated, or what messages need to traverse the network. However, as [9] shows, each term in a rule head and body can be annotated with a “location specifier”, which identifies which node the tuple resides on when the rule is executed. With such identifiers, the specification above becomes:

```
path(@S,D,P,C) :- link(@S,D,C),
                  P = f concatPath(link
                  (@S,D,C), nil).
path(@S,D,P,C) :- link(@S,Z,C1), path
                  (@Z,D,P2,C2),
                  C = f_sum(C1, C2),
                  P = f concatPath(link
                  (@S,Z,C1), P2).
bestPathCost(@S,D,AGG<C>) :- path
                  (@S,D,P,C).
bestPath(@S,D,P,C) :- bestPathCost
                      (@S,D,C),
                      path(@S,D,P,C).
```

It can be seen that all rules execute entirely locally (on whichever node corresponds to the value `S` in the operand tuples) except the second one. For this, a message must be sent from `Z` to `S`, which is conveniently directly connected to `S`. Such a specification can be trivially planned and executed independently on each node in a network, and will result in best-path routing table on each node. Furthermore, the messages that will be sent during execution correspond to those that an imperative link-state implementation would need to transmit.

Loo et al. [7] detail various evaluation strategies, and also discuss some automated checks that may be performed on such algorithm specifications. In particular, if a set of relations (such as `link` in the above example) are asserted to mean direct connectivity, the system can determine at query plan time whether or not the routing protocol is well-formed, in that it will only cause a node to send messages to its direct neighbours. It is this "global" specification of rules which sets declarative networking systems apart from purely local event-condition-action (ECA) systems.

A declarative networking system like P2 [8] can directly execute such a specification, allowing very concise programs for distributed algorithms. [8] presents an implementation of the Chord overlay in 47 lines of Datalog, considerably shorter than the several thousand lines of C++ in the original imlemenentation.

A further key benefit of declarative networking is the flexibility afforded by the query framework. Very few assumptions are made about the network in question – even the point-to-point `link` predicate in the example above could be replaced a more complex relation conveying direct connectivity, for example one giving radio strength in a wireless network. The declarativity of the specification allows external data about the network to be cleanly integrated into the algorithmic framework using familiar techniques in ways that would be tedious, cumbersome, and brittle in an imperative implementation.

Key Applications

Declarative networking is currently the domain of research rather than industrial adoption. In addition to the specification of protocols for extensible network routing and the overlay component of distributed applications, the approach has been applied to building sensor network applications over wireless networks and the implementation of fault-tolerant replication algorithms for the purposes of performance analysis [13].

Future Directions

There are a number of open questions in declarative networking. Evaluating rules concurrently on a node so as to preserve intelligible semantics to the programmer is an open area, as is the field of optimizations of such distributed queries over the kinds of sparsely-connected graphs found computer networks. The best way of integrating the declarative networking functionality of a distributed application with imperative functionality local to a node is an area of ongoing study.

URL to Code

http://p2.berkeley.intel-research.net/, mirrored at http://p2.cs.berkeley.edu/
http://developer.berlios.de/projects/slow/
http://www.dvs.tu-darmstadt.de/research/OverML/index.html

Cross-references

▶ Deductive Databases
▶ Distributed Query Processing
▶ Event-Condition-Action Systems

Recommended Reading

1. Abadi M. and Loo B.T. Towards a declarative language and system for secure networking. In Proc. Third Int. Workshop on Networking meets Databases (NetDB), USA, April 2007.
2. Behnel S. SLOSL – a modelling language for topologies and routing in overlay networks. In Proc. First Int. Workshop on Modeling, Simulation and Optimization of Peer-to-Peer Environments (MSOP2P), Naples, Italy, 2008. to appear.
3. Behnel S. and Buchmann A. Overlay networks – implementation by specification. In Proc. ACM/IFIP/USENIX 6th Int. Middleware Conf., 2005.
4. Chu D., Tavakoli A., Popa L., and Hellerstein J. Entirely declarative sensor network systems. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 1203–1206.
5. Chun B., Hellerstein J.M., Huebsch R., Jeffery S.R., Loo B.T., Mardanbeigi S., Roscoe T., Rhea S., Shenker S., and Stoica I. Querying at Internet Scale (Demo). In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
6. Huebsch R., Chun B., Hellerstein J.M., Loo B.T., Maniatis P., Roscoe T., Shenker S., Stoica I., and Yumerefendi A.R. The architecture of PIER: an Internet-scale query processor. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005.
7. Loo B.T., Condie T., Garofalakis M., Gay D.E., Hellerstein J.M., Maniatis P., Ramakrishnan R., Roscoe T., and Stoica I. Declarative networking: language, execution and optimization.

In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 97–108.
8. Loo B.T., Condie T., Hellerstein J.M., Maniatis P., Roscoe T., and Stoica I. Implementing declarative overlays. In Proc. 20th ACM Symp. on Operating System Principles, 2005, pp. 75–90. ACM Press, New York.
9. Loo B.T., Hellerstein J.M., Stoica I., and Ramakrishnan R. Declarative routing: extensible routing with declarative queries. In Proc. Int. Conf. of the on Data Communication, 2005.
10. Loo B.T., Huebsch R., Hellerstein J.M., Roscoe T., and Stoica I. Analyzing P2P overlays with recursive queries. Technical Report IRB-TR-003-045, Intel Research, November 2003.
11. Roscoe T., Hand S., Isaacs R., Mortier R., and Jardetzky P. Predicate routing: enabling controlled networking. In Proc, First Workshop on Hot Topics in Networks (HotNets-I), 2002.
12. Roscoe T., Mortier R., Jardetzky P., and Hand S. InfoSpect: using a logic language for system health monitoring in distributed systems. In Proc. 2002 ACM SIGOPS European Workshop, 2002.
13. Singh A., Maniatis P., Druschel P., and Roscoe T. BFT protocols under fire. In Proc. 5th USENIX Symp. on Networked Systems Design and Implementation, 2008.
14. Singh A., Maniatis P., Roscoe T., and Druschel P. Using queries for distributed monitoring and forensics. In Proc. First European Systems Conference (Eurosys), 2006.
15. Wawrzoniak M., Peterson L., and Roscoe T. Sophia: an information plane for networked systems. In Proc. 5th USENIX Symp. on Networked Systems Design & Implementation, 2003.

Declarative Overlay Networks

► Declarative Networking

Deductive Data Mining using Granular Computing

Tsau Young (T. Y.) Lin
San Jose State University, San Jose, CA, USA

Synonyms

Decision rules, classification; Deductive data mining, model for automated data mining; Rough set theory, granular computing on partition

Definition

What is Deductive Data Mining (DDM)? It is a methodology that derives patterns from the mathematical structure of a given *stored* data. Among three core techniques of data mining [1], *classifications and association rule mining* are deductive data mining, while clustering is not, because its algorithms often use some properties of the ambient space.

What is Granular Computing (GrC)? In general, it is a problem solving methodology deeply rooted in human thinking. For example, human body is granulated into head, neck, and etc. However, the main concerns here are on the data mining aspect of GrC. Two views are presented. One is based on current technology, and the other is on the incremental approach to the ultimate goal.

A) *Mining Relational Databases (RDB) using GrC*: In GrC, a relation K (also known as information table in rough set theory) is a knowledge representation that maps a set U of entities into a set of attribute values. The collection of entities that is mapped to the same attribute value is called a granule. By replacing each attribute value with its granule, a relation of granules is formed. Observe that the collection of granules in each column is a partition (hence an equivalence relation) on U. Hence, the relation of granules is equivalent to the pair $(U, \mathcal{R})$, called Granular Data Model (GDM), where R is the set of equivalence relations that are defined by those columns. Observe that GDM determines a relation (up to isomorphism) uniquely. See Table 1. Note that GDM has a very explicit mathematical structure of data, hence

1. Mining RDB using GrC is DDM on GDM (or relation of granules). See “Key Applications – Deductive Data Mining on GDM”. Two striking results will be explained.

   B) *Mining RDB over “real world sets” using GrC*: This is a goal statement. What is a “real world set”? Strictly speaking, this can not be answered mathematically (there is no semantics in mathematics). In GrC, the “real world set” has been modeled by the Fifth GrC model (relational GrC model), (U, β), where β is a collection of n-ary relations (n is not fixed). In this approach, the focus has been on capturing the interactions among elements of the universe. For example, in human society, n person may form a committee (hence they interact). In this formulation, the committee is a tuple (as each person may play a disticnt role) in one of the relation in β [18,20]. Note that Fifth GrC Model (in finite U) is the relational structure of the First Order Logic (without function symbols). RDB over a model of “real world set” can be viewed as a Semantic Oriented RDB. For now, the “real world set" is Fifth GrC Model, so the corresponding GDM is *R*elational GrC Model (Fifth GrC Model) based *G*DM, denoted by *RGDM*.

2. Mining RDB over "real world sets" using GrC is DDM on RGDM. See "Future Directions".

Historical Background

How was the term GrC coined? In the academic year 1996–1997, when T. Y. Lin took his sabbatical leave at UC-Berkeley, Zadeh suggested granular mathematics to be his research area. To limit the scope, Lin proposed the term granular computing [26] to label the research area. Though the label is new, the notion is quite ancient, e.g., the infinitesimal granules led to the invention of calculus. More recent examples are Heisenberg uncertainty principle (A precise position can only determine a granule of momentum) and fuzzy set theory [25]. For database and data analysis areas, the first work may be due to Hsiao in 1970 [7]; incidentally he is also well known for database machines. His model is essentially equivalent to relational database model and was called Attribute Based File Organization (evolved into attribute based data model (ABDM)) by Eugene Wong in 1971 [2]; Wong and Stonebraker co-founded database system INGRESS.

In ABDM, Hsiao imposed equivalence relations on attribute domains to increase the precision of data access. Around 1981, Ginsburg and Hull imposed another binary relation, "partial ordering" on attribute domains [4,5]. The goal is to capture the additional information provided by the order.

About the same time, Pawlak (1982) and Lee (1983) observed that an attribute can be regarded as a partition of the universe [8,24], and developed rough set theory and algebraic theory of relational databases. Their approaches are quite different from previous theories.

In 1988–1989, for approximate retrievals, Lin generalized topological neighborhood system to Neighborhood Systems (NS) by simply dropping the axioms of topology. He imposed NS structure on attribute domains and studied the Topological Data Model [9,11] (see cross references). NS is equivalent to a set of binary relations, so it is mathematically a generalization equivalence relation (Hsiaos works) and partial ordering (Ginsburg and Hulls works). However, semantically, they are different; in NS, each neighborhood (a granule) is regarded as a unit of uncertainty. Also in 1989, Lin applied the idea of computer security [10]. This is related to the information flow on access control model (see cross references) In 1992, Lin developed a NS-version of RS theory [12]. These are pre-GrC stories.

Next, are early works. By mapping NS onto Zadehs intuitive statements, Lin used NS as his first mathematical GrC model, developed Binary Relation based Data Model [13,22], and observed "complete Pawlak theories" in symmetric binary relations and some partial covering [19,20]. Binary relation based Data Model has many names, such as binary knowledge based, granular table, clustered table, semantic oriented table and, etc.; now it is called a Binary Granular Data Models (BGDM). Intrinsically, BGDM is a semantic oriented data model, and has been used to mine semantic oriented decisions (classifications) and association rules; it is still an ongoing theory.

Foundations

What is Data Mining?

What is data mining? There is no universally accepted formal definition of data mining, however the following informal description from [3] is rather universal: "data mining is a non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns from data."

As having been pointed out previously in several occasions, the terms, "novel," "useful," and "understandable" represent subjective views, and hence can not be used for scientific purpose, it is paraphrased as follows:

1. Deriving useful patterns from data.

This "definition" points out four key ingredients: data, patterns, derivation methodology and the real world meaning of patterns (implied by usefulness).

Deductive Data Mining

Convention. A *symbol* is a string of "bit and bytes" that has no *formal* real world meaning, or whose real world interpretation (if there is one) does *not* participate in formal processing. A symbol is termed a *word*, if the intended real world meaning *does participate* in the formal processing.

What is the *nature* of the data in DDM? It is best to examine how the data are created: In traditional data processing: (i) first a set of attributes, called relational schema, is selected. Then (ii) tuples of words are entered the system. They represent a set of real world entities. These words are called attribute values. Each such a word is created to represent a real world fact, however, only the symbol, but not the real world semantic, is stored.

How are the data processed? In traditional data processing environment, DBMS processes these data, under *human commands*, carries out the human perceived-semantics. However, from the system point of view, each word is a pure symbol (semantics are not stored). So in an automated data mining algorithm (without the human in the loop),

1. Words are treated as symbols.

For example, association mining algorithm merely *counts* the symbols without consulting their real world meanings. So association mining algorithm transforms a table of symbols into a set of expressions (frequent itemsets) of symbols. In summary,

1. A relation is a table of stored symbols.
2. Patterns are the mathematical expressions of the stored symbols. For examples, frequency itemsets.
3. Principle of derivations is the mathematical deduction.

Definition 1 *Deductive data mining is a data mining methodology that derives (using mathematical deductions only) patterns from the mathematical structure of stored symbols.*

Among three core topics of data mining [1],

Theorem 1 *Classifications and association rule mining are deductive data mining, while clustering may be not, because its algorithms use some properties of the ambient space.*

Isomorphism – A Critical Concept

As having pointed out that a relation is a table of symbols. So the following definition of isomorphism is legitimate: Let $K = (V, \mathcal{A})$ and $H = (V, \mathcal{B})$ be two relations, where $A = \{A^1, A^2, \ldots A^n\}$ and $B = \{B^1, B^2, \ldots B^m\}$.

Attributes A^i and A^j are isomorphic iff there is a one-to-one and onto map, $s : Dom(A^i) \to Dom(A^j)$ such that $A^j(v) = s(A^i(v)) \forall v \in V$. The map s is called an isomorphism. Intuitively, two attributes (columns) are isomorphic iff one column turns into another one by properly renaming its attribute values.

Definition 2 *Two relations K and H are said to be isomorphic if there is a one-to-one and onto correspondence between two families of attributes so that every A^i is isomorphic to the corresponding B^j, and vice versa.*

Two relational Schema K and H are said to be isomorphic if all their instances are isomorphic.

Isomorphism is an equivalence relation defined on the family of all relational tables, so it classifies the tables into isomorphic classes. Moreover, in [15], the following observation was made: *Isomorphic relations have isomorphic patterns.* Its implications are rather far reaching. It essentially declares that patterns are syntactic in nature. They are patterns of the whole isomorphic class, even though many isomorphic relations may have very different semantics. Here is the important theorem:

Theorem 2 *A pattern is a property of an isomorphic class.*

In next section, a canonical model, called Granular Data Model (GDM), for each isomorphic class will be constructed.

Granular/Relational Data Models

In RDB, a relation (called an information table in rough set theory) can be viewed as a knowledge representation of a given universe U(a set of entities) in terms of a given set of attributes $\mathcal{A} = \{A_1, A_2, \ldots A_n\}$. A tuple(row) is a representation of an entity. An attribute (column) A_j is a mapping that maps U to its attribute domain dom(A_j) (another classical set). Let those entities that are mapped to the same value be called granules. By replacing each attribute value with the corresponding granule, a new *relation of granules* is formed. Observe that the collection of granules in a column are mutually disjoint So it defines a partition and hence an equivalent relation. Let $\mathcal{R}$ be the collection of such equivalence relations. So the pair $(U, \mathcal{R})$, called a granular data model (GDM), is equivalent to the relation of granules.

The details are further illustrated in Table 1. Note that the first two columns, A-partition and B-partition, is a relation of granules. It is obviously equivalence to the GDM (U, $\{A - partition, B - partition\}$).

The last two columns, A-NAME and B-NAME, is the original given relation. It can be viewed as a relation of "meaningful" names of these granules. So a relation also will be called Data Model of Names (DMN), to emphasize this contrast to the relation of granules.

Deductive Data Mining using Granular Computing.
Table 1. The granular data model of a relation.

		Relation of granules		Original relation	
U	→	A-partition	B-partition	A-NAME	B-NAME
*e*1	→	{*e*1, *e*2, *e*6, *e*7}	{*e*1, *e*4}	diaper	sss
*e*2	→	{*e*1, *e*2, *e*6, *e*7}	{*e*2, *e*6, *e*7}	diaper	beer
*e*3	→	{*e*3, *e*5, *e*8}	{*e*3, *e*8}	coo	ttt
*e*4	→	{*e*4}	{*e*1, *e*4}	paat	sss
*e*5	→	{*e*3, *e*5, *e*8}	{*e*5}	coo	bar
*e*6	→	{*e*1, *e*2, *e*6, *e*7}	{*e*3, *e*8}	diaper	beer
*e*7	→	{*e*1, *e*2, *e*6, *e*7}	{*e*2, *e*6, *e*7}	diaper	beer
*e*8	→	{*e*3, *e*5, *e*8}	{*e*3, *e*8}	coo	ttt

Table 1 also illustrates the following isomorphism theorem, which was observed by Pawlak ([24], p. 56).

Theorem 3 *A relation (up to isomorphism) determines a GDM and vice versa. In short, DMN, up to isomorphism, is equivalent to GDM.*

So "data" processing is equivalent to "granule" processing – Granular Computing. This introduced the data mining aspect of GrC in RDB.

Key Applications

In this section, two key applications are presented. Conceptually, the results are rather striking and counterintuitive; see Theorem 4 and Theorem 5.

Feature Constructions

What is an attribute (feature)? Surprisingly, this is a thought provoking question. In database theory, an attribute (in a relation schema) is a property, a view, or a reference (e.g., a coordinate). These are the concepts in human level. Based on these concepts, database experts represent entities into relations. Unfortunately, such knowledge representations are not thorough, namely, the semantics of attributes and attribute values are not stored in computer systems. So, automated data mining algorithms cannot utilize these semantics, simply because they are not in the computer systems. So the key question is:

1. How could the concept of features be described by data?

The first two columns of the Table 1 illustrates the idea: an attribute is a partition of the universe. Since the universe is a finite set, so there are the following surprise:

Theorem 4 *The number of non-isomorphic features that can be constructed from the given relation is finite; see [15].*

This is rather counterintuitive results. The reason is largely due to the fact that most people view features from data processing point of view, not data mining point of view.

The following example may illuminate the critical concepts: Consider a numerical relation of two columns. It can be visualized as a finite set of points in Euclidean plane. In this view, attributes are coordinates. So by simply rotating the X–Y-coordinate system (from 0^o to 360^o), infinitely many relations of P_θ can be generated, where θ is the angle rotated. Now we will examine these P_θ and four points whose polar coordinates are $(1, 0^o)$,$(1, 30^o)$,$(1, 45^o)$,$(1, 90^o)$; see Table 2. Note that the X_θ-partition and Y_θ-partition do not change most of the time, when θ moves. It only changes at $\theta = 45^o$ and its multiples, namely, $\theta = 135^o$, 315^o. Note that though X_θ-coordinate and Y_θ-coordinate do change, but most of them are isomorphic to each other. Non-isomorphic ones occurs only at $\theta = 45^o$ and its multiples.

The key point is:

1. As *U* is finite, both A-Partition and B-partition columns only have finitely many possible distinct choices.

High Frequency Patterns in RDB

GDM is a powerful data mining representation: Observe that the frequency of the pair, (diaper, beer), is equal to the cardinality of the intersection of two granules, the diaper granule $\{e1, e2, e6, e7\}$ and the beer granule $\{e2, e6, e7\}$.

This illustration immediately gives us the following generalization. Let Δ be the Boolean algebra generated by granules. Then a Boolean algebraic expression in Δ is a set theoretical expression in *U*, hence it has the cardinality.

Definition 3 *A Boolean algebraic expression in Δ is a high frequency pattern (generalized frequent itemsets), if the cardinality of the expression is greater than the given threshold* [14].

This idea can also be expressed by Pawlak's decision logic. A formula is a high frequency pattern, if its meaning set has adequate cardinality. Here is a striking result:

Theorem 5 *High frequency patterns can be enumerated by solving a set of linear inequalities; see [17].*

This theorem has not been useful in applications; this is due to the high volume of the patterns. Theoretically, this theorem together with the facts in "Paradoxical Phenomena in Data Mining" led to the conclusion: Semantic oriented patterns are needed.

Future Directions

There are three subsections. In the first subsection, the goal statement and incremental approaches are outlined. In the second subsection, an example is given to illustrate the inadequacy of current results. The last subsection shows the feasibility of the outline in the first subsection will work.

Relational Database Theory Over "Real World Sets"

RDB and RST are based on classical set theory, in which every set is discrete. In other words, there are no interactions among elements of any set. However, a "real world set" contains interactions. For example in human society, n persons may form a committee (hence they have interactions). In GrC modeling, Fifth GrC Model (U, β) have been proposed as a model of "real world sets", where β is a collection of n-ary relations (n is not a fixed integer) that represent the interactions among elements in U [18,20]. Note that the roles of each member in the committee may be all distinct, so members are not exchangeable. So *granules are tuples, not necessarily subsets*. Many authors, including (Lin), have used subsets as granules; such notions may not be general enough.

A relational database over "real world sets," as in the classical relational database, is equivalent to "real world" GDM $((U,\beta),\mathcal{R})$, where (U, β) is a model of "real world sets" and $\mathcal{R}$ is a family of "equivalence relations" on the model of a "real world sets," (U, β). In this paper, only the idea of simplest case will be illustrated, namely, β is a single n-nary relation. By combining β and $\mathcal{R}$, the pair $((U,\beta),\mathcal{R})$ can be transformed to the pair $(U,\mathcal{S})$, where each member S_i of $\mathcal{S}$ is a new relation that combined the n-nary relation β and the "equivalence relations" R_i. Namely, each attribute value in a relation β is replaced by an equivalence class of R_i. The resulting new β is S_i.

Deductive Data Mining using Granular Computing. Table 2. The granular data model of a relation

		Relation of granules of P_0		Original relation P_0	
U	$\rightarrow$	X_0-partition	Y_0-partition	X_0-coordinate	Y_0-coordinate
p_1	$\rightarrow$	$\{p_1\}$	$\{p_1\}$	1	0
p_2	$\rightarrow$	$\{p_2\}$	$\{p_2\}$	$\sqrt{3}/2$	1/2
p_3	$\rightarrow$	$\{p_3\}$	$\{p_3\}$	$\sqrt{2}/2$	$\sqrt{2}/2$
p_4	$\rightarrow$	$\{p_4\}$	$\{p_4\}$	1/2	$\sqrt{3}/2$
p_5	$\rightarrow$	$\{p_5\}$	$\{p_5\}$	0	1
		Relation of granules of P_{30^o}		Original relation P_{30^o}	
U	$\rightarrow$	X_{30^o}-partition	Y_{30^o}-partition	X_{30^o}-coordinate	Y_{30^o}-coordinate
p_1	$\rightarrow$	$\{p_1\}$	$\{p_1\}$	$\sqrt{3}/2$	$-1/2$
p_2	$\rightarrow$	$\{p_2\}$	$\{p_2\}$	1	0
p_3	$\rightarrow$	$\{p_3\}$	$\{p_3\}$	$\sqrt{3}/2$	1/2
p_4	$\rightarrow$	$\{p_4\}$	$\{p_4\}$	$\sqrt{2}/2$	$\sqrt{2}/2$
p_5	$\rightarrow$	$\{p_5\}$	$\{p_5\}$	1/2	$\sqrt{3}/2$
		Relation of granules of P_{45^o}		Original relation P_{45^o}	
U	$\rightarrow$	X_{45^o}-partition	Y_{45^o}-partition	X_{45^o}-coordinate	Y_{45^o}-coordinate
p_1	$\rightarrow$	$\{p_1, p_4\}$	$\{p_1\}$	$\sqrt{2}/2$	$-\sqrt{2}/2$
p_2	$\rightarrow$	$\{p_2\}$	$\{p_2\}$	$\sqrt{3}/2$	$-1/2$
p_3	$\rightarrow$	$\{p_3\}$	$\{p_3\}$	1	0
p_4	$\rightarrow$	$\{p_1, p_4\}$	$\{p_4\}$	$\sqrt{2}/2$	$\sqrt{2}/2$

In this entry, only the case $n = 2$ will be discussed. In other words, only the following type of GDM over "real world sets" $(U,\mathcal{B})$, called Binary GrC Data Model (BGDM), will be considered, where $\mathcal{B}$ is a collection of binary relations.

Paradoxical Phenomena in Data Mining

In this subsection, some paradoxical phenomena in data mining will be discussed. First, two semantically very different, but syntactically equivalent relations K and K are presented; see Table 3. Namely, two isomorphic relations K and K are considered (see "Isomorphism – A Critical Concept", where K is a relation about human beings and the other K is about hardware. The central theme is their paradoxical phenomena of high frequency patterns. Let the threshold be 3, namely, a sub-tuple in Table 3 is regarded as a pattern (frequent itemset or undirected association

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:39

Deductive Data Mining using Granular Computing. Table 3. Relational table K and K'

Relation K				Relation K'			
($S\#$	Amount	Birth month	CITY)	($P\#$	Weight	Part name	Material)
(S_1	TWENTY	MAR	SC)	(P_1	20	SCREW	STEEL)
(S_2	TEN	MAR	SJ)	(P_2	10	SCREW	BRASS)
(S_3	TEN	FEB	MV)	(P_3	10	NAIL	ALUMINUM)
(S_4	TEN	FEB	MV)	(P_4	10	NAIL	ALUMINUM)
(S_5	TWENTY	MAR	SJ)	(P_5	20	SCREW	BRASS)
(S_6	TWENTY	MAR	SJ)	(P_6	20	SCREW	BRASS)
(S_7	TWENTY	APR	SJ)	(P_7	20	PIN	BRASS)
(S_8	FIFTY	NOV	LA)	(P_8	300	HAMMER	ALLOY)
(S_9	FIFTY	NOV	LA)	(P_9	300	HAMMER	ALLOY)

rule), if its number of occurrences is greater then or equal to 3:

1. Relation K:(TWENTY, SJ) is an interesting rule; it means the amount of business in San Jose is likely to be 20 million.

1′ Relation K' (20, BRASS) is an isomorphic pattern. However, this rule is not meaningful at all. Material, such as BRASS, has no specific weight. The schema indicates that weight 20 is referred to PIN, not BRASS.

2′ Relation K':(SCREW, BRASS) is interesting rule; it says screws are most likely made from BRASS.

2. Relation K: (MAR, SJ) is an isomorphic pattern; it is not interesting, because MAR refers to the birth month of a supplier, not to a city. The schema indicates that totally unrelated two columns (attributes) contain the pair.

3′ Relation K': (20, SCREW) is interesting rule; it says screws are most likely weighing 20.

3. Relation K: (TWENTY, MARCH) is an isomorphic pattern; TWENTY refers to an ability of a supplier, not to birth month.

This analysis clearly indicates that the "real word" meanings of the universe and attribute domains have to be considered. In other words, a relational database theory over "real word set" is needed; in GrC, the real world model is Fifth GrC Model (relational GrC Model).

BGDM – Mining with Constraints

In this section, the solutions for the case $n = 2$ is illustrated: The following "real word" BGDM structures are added to K and K' ; K' is unchanged. But, binary relations B_{Amount} and B_{City} are defined on the two attribute domains of Relation K:

1. B_{Amount} is the smallest reflexive and symmetric binary relation that has the pair (TWENTY, TEN) as its member (Recall that a binary relation is a subset of the Cartesian Product of Domains.
2. B_{City} is the smallest reflexive and symmetric binary relation, in which the three cites, SC, MV, SJ are considered to be very near to each other and very far away from LA.

Note that B_{Amount} and B_{City} induce two binary relations B_A and B_C on U (they form the $\mathcal{S}$ in previous discussions). Again K' has no changes. Relation K with two additional binary relations and Relation K define the following BGDM

1. BGDM of K is (U, $R_{BirthMonth}$, B_A, B_C).
2. BGDM of K' has no changes, namely, (U, R_{Weight}, $R_{PartName}$, $R_{Material}$),

where R_{A_j} denotes the equivalence relation induced by the attribute A_j. So from *BGDM point of view, the two relation K and K' are not isomorphic*. and hence the paradoxical phenomena disappear. This is a formal model for data mining with *constraints*.

Various GrC models provided various degree of "real world set" structures on attribute domains. GrC may provide a right direction to "real world" deductive data mining.

Cross-references

► Access Control
► Association Rule

► Data Mining
► Data Reduction
► Decision Tree Classification
► Decision Trees
► Frequent Itemsets and Association Rules
► Frequent Itemsets Mining with Constraints
► Granular Computing
► Information Integration
► Rough Set Approach
► Search Engine Metrics
► Topological Data Models

Recommended Reading

1. Dunham M. Data Mining Introduction and Advanced Topics. Prentice Hall, Englewood Cliffs, NJ, 2003, ISBN 0-13-088892-3.
2. Eugene Wong T.C. Chiang: canonical structure in attribute based file organization. Commun. ACM 14(9)563–597, 1971.
3. Fayad U.M., Piatetsky-Sjapiro G., and Smyth P. From data mining to knowledge discovery: an overview. In Knowledge Discovery in Databases, Fayard, Piatetsky-Sjapiro, Smyth, and Uthurusamy (eds.). AAAI/MIT Press, Cambridge, MA, 1996.
4. Ginsburg S. and Hull R. Ordered attribute domains in the relational model. XP2 Workshop on Relational Database Theory, 1981.
5. Ginsburg S. and Hull R. Order dependency in the relational model. Theor. Comput. Sci., 26:149–195, 1983.
6. Gracia-Molina H., Ullman J., and Windin J. Database Systems – The Complete Book. Prentice HallEnglewood Cliffs, NJ, 2002.
7. Hsiao D.K., and Harary F. A formal system for information retrieval from files. Commun. ACM 13(2), 1970; Corrigenda, 13(4), April 1970.
8. Lee T.T. Algebraic theory of relational databases. The Bell System Tech. J., 62:(10)3159–3204, 1983.
9. Lin T.Y. Neighborhood systems and relational database. In Proc. CSC'88, 1988, p. 725.
10. Lin T.Y. (1989) Chinese wall security policy – an aggressive model. In Proc. Fifth Aerospace Computer Security Application Conf., 1989, pp. 286–293.
11. Lin T.Y. Neighborhood systems and approximation in database and knowledge base systems. In Proc. Fourth Int. Symp. on Methodologies of Intelligent Systems (Poster Session), 1989, pp. 75–86.
12. Lin T.Y. (1992) Topological and fuzzy rough sets. In Decision Support by Experience – Application of the Rough Sets Theory, R. Slowinski (ed.). Kluwer, Dordecht, 1992, pp. 287–304.
13. Lin T.Y. Granular computing on binary relations: I. Da mining and neighborhood systems. I, II: Rough set representations and belief functions. In Rough Sets in Knowledge Discovery, A. Skoworn and L. Polkowski (eds.). Physica-Verlag, Wurzburg, 1998, pp. 107–140.
14. Lin T.Y. Data mining and machine oriented modeling: a granular computing approach. Appl. Intell. 13(2)113–124, 2000.
15. Lin T.Y. Attribute (feature) completion – the theory of attributes from data mining prospect. Proc. 2002 IEEE Int. Conf. on Data Mining, 2002:282–289.
16. Lin T.Y. Chinese wall security policy models: information flows and confining Trojan horses. DBSec 2003:282–289.
17. Lin T.Y. Mining associations by linear inequalities. Proc. 2004 IEEE Int. Conf. on Data Mining, 2004:154–161.
18. Lin T.Y. Granular computing: examples, intuitions and modeling. GrC pp. 40–44.
19. Lin T.Y. A roadmap from rough set theory to granular computing. RSKT 2006:33–41, 2006.
20. Lin T.Y. Granular computing: ancient practices, modern formalization and future directions. In: EDBS Granular Computing Section.
21. Lin T.Y. and Chiang I.J. A simplicial complex, a hypergraph, structure in the latent semantic space of document clustering. Int. J. Approx. Reason. 40(1–2):55–80, 2005.
22. Lin T.Y. and Liau C.J. Granular computing and rough sets. Data Min. Knowl. Discov. Handbook 2005:535–561, 2004.
23. Louie E. and Lin T.Y. Finding association rules using fast bit computation: machine-oriented modeling. ISMIS 2000:486–494, 2000.
24. Pawlak Z. Rough sets. Theoretical aspects of reasoning about data. Kluwer, Dordecht, 1991.
25. Zadeh L.A. Fuzzy Sets Inf. Control 8(3):338–353, 1965.
26. Zadeh L.A. Some reflections on soft computing, granular computing and their roles in the conception, design and utilization of information/ intelligent systems. Soft Comput., 2:23–25, 1998.

Deductive Data Mining, Model for Automated Data Mining

► Deductive Data Mining using Granular Computing

Deductive Databases

► Datalog

Dedup

► Deduplication

Deduplication

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

Dedup; Single Instancing

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:39

Definition

The term Deduplication refers to the task of eliminating redundant data in data storage so as to reduce the required capacity. The benefits of Deduplication include saving rack space and power consumption of the data storage. Deduplication is often implemented in archival storage systems such as content-addressable storage (CAS) systems and virtual tape libraries (VTLs). The term Deduplication is sometimes shortened to Dedup.

Key Points

An address mapping table and a hash index are often used for implementing Deduplication. The address mapping table converts a logical address to a physical location for each block, and the hash index converts a hash value to a physical location for each block. When a block X is to be written to the data storage, a hash value is calculated from the content of X and then the hash index is searched. If the same hash value is not found in the hash index, a new block is allocated in the storage space, X is written to the allocated block, and then the mapping table and the hash index are updated: a pair of the logical address and the physical location of X is registered to the mapping table, and a pair of the hash value and the physical location is also registered to the hash index. In contrast, if the same hash value is found, an identical block is supposed to exist in the storage. A pair of the logical address of X and the physical location retrieved from the hash index is merely registered to the mapping table, such that the block redundancy can be avoided. Note, above, that the possibility of hash collision is ignored. In reality there is the potential that the hash function produces the same hash value for different block contents, and in such rare cases, Deduplication would lose data in the data storage. To mitigate the issue, some vendors introduce hash algorithms which have a very low possibility of hash collisions. However, this solution cannot eliminate the potential of data loss completely. Others have chosen to check block contents every time the same hash value is found in the hash index. Such careful block checking may degrade archiving throughput, but can completely avoid data loss.

Deduplication techniques may generally operate at different granularities and at different implementation levels. For example, file systems may do Deduplication at the file level, and disk array controllers may do Deduplication at the sector level.

In enterprise systems, Deduplication is often implemented at the controller of archiving storage systems such as content-addressable storage (CAS) systems and virtual tape libraries (VTLs). Intrinsically, enterprise systems often store redundant data in the data storage. Suppose that an email message which attaches a document file is forwarded to many persons in the office. A separate mailbox of each recipient thus stores the same document file. Eliminating such duplication at archival storage systems sounds a natural and effective idea.

Deduplication is recognized as a technique of energy saving in data centers. Deduplication has the benefit of reducing the number of disk drives that are required to accommodate the explosively expanding data. Consequent effects of saving electric power are expected to help the improvement of system cost efficiency and the reduction of environmental impact.

Cross-references

► Information Lifecycle Management
► Storage Management
► Storage Power Management
► Storage Virtualization

Recommended Reading

1. Diligent Technologies. A hyper factor: a breakthrough in data reduction technology. White Paper.
2. Patterson H. Dedupe-centric storage for general applications. White Paper, Data Domain.
3. Quinlan S. and Dorward S. Venti: a new approach to archival storage. In Proc. Fast 2002 Conf. on file and storage Technologies, (USENIX FAST), 2002, pp. 89–102.

Deduplication in Data Cleaning

Raghav Kaushik
Microsoft Research, Redmond, WA, USA

Synonyms

Reference reconciliation; Record matching; Merge-purge; Clustering

Definition

Many times, the same logical real world entity has multiple representations in a relation, due to data entry errors, varying conventions, and a variety of other reasons. For example, when Lisa purchases products from SuperMart twice, she might be entered as two different customers, e.g., `[Lisa Simpson, Seattle, WA, USA, 98025]` and `[Simson Lisa, Seattle, WA, United States, 98025]`. Such duplicated information can cause significant problems for the users of the data. For example, it can lead to increased direct mailing costs because several customers may be sent multiple catalogs. Or, such duplicates could cause incorrect results in analytic queries (say, the number of SuperMart customers in Seattle), and lead to erroneous data mining models. Hence, a significant amount of time and effort are spent on the task of detecting and eliminating duplicates.

This problem of detecting and eliminating duplicated data is referred to as deduplication. It is an important problem in the broader area of data cleaning. Note that this is markedly more challenging than the standard duplicate elimination problem for answering `select distinct` queries in relational database systems, which consider two tuples to be duplicates if they match exactly.

Historical Background

The problem of deduplication has been studied for a long time under application-specific settings such as library cataloging [17] and the Census Bureau [19]. Several techniques for measuring the similarity between records for these domains have been developed. For instance, the BigMatch technique [19] is used for deduplication by the US Census Bureau.

With the advent of data warehouses, data cleaning and deduplication in particular arose as a critical step in the Extract-Transform-Load (ETL) cycle of loading a data warehouse. Ensuring good quality data was essential for the data analysis and data mining steps that followed. Since the most common field of interest was the customer address field used for applications such as direct mailing, several tools such as Trillium [16] emerged that performed deduplication of addresses.

For the past several years, the database research community has addressed the deduplication problem [1,4,10,11,13]. This has been largely in a domain-independent context. The idea is to seek fundamental operations that can be customized to any specific application domain. This domain-independent approach has had a commercial presence. For instance, Microsoft SQL Server 2005 Integration Services supports data cleaning operators called *Fuzzy Lookup* and *Fuzzy Grouping* that respectively perform approximate matching and deduplication.

A new area of application for data cleaning technology and deduplication in particular has been with the data on the internet. There are web sites such as Citeseer, Google Scholar, Live Search Academic, Froogle and Live Products that integrate structured data such as citations and product catalogs from various sources of data and need to perform deduplication prior to hosting the web service.

Foundations

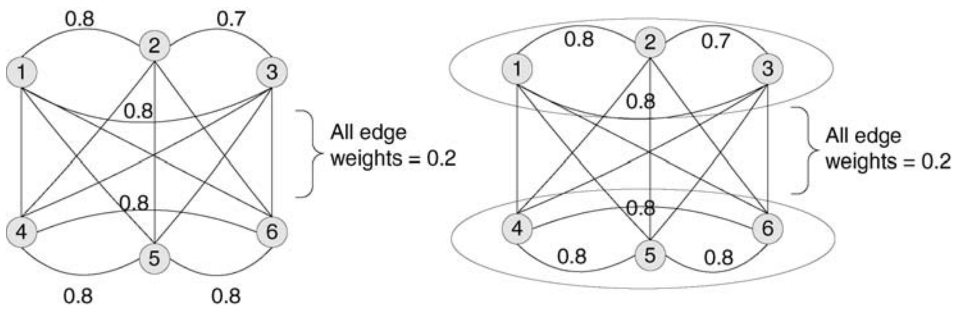
Deduplication can be loosely thought of as a fuzzy or approximate variant of the relational `select distinct` operation. It has as its input a table and a set of columns; the output is a partition of this table where each individual group denotes a set of records that are approximately equal on the specified columns. A large amount of information can be brought to bear in order to perform deduplication, namely the textual similarity between records, constraints that are expected to hold over clean data such as functional dependencies and attribute correlations that are known to exist. Techniques that use this wide variety of information have been explored in prior work. Figure 1 shows an example table containing citations and a partition defined over all the textual columns, illustrating the output of deduplication.

At the core of the deduplication operation is a *similarity function* that measures the similarity or distance between a pair of records. It returns a similarity score which is typically a value between 0 and 1, a higher value indicating a larger similarity with 1 denoting equality. Example similarity functions are edit distance, jaccard similarity, Jaro-Winkler distance, hamming similarity and cosine similarity [12]. Given a table, the similarity function can be applied to all pairs of records to obtain a weighted similarity graph where the nodes are the tuples in the table and there is a weighted edge connecting each pair of nodes, the weight representing the similarity. Figure 2 shows the similarity graph for the data in Fig. 1 (the numbers inside the nodes correspond to the IDs). In practice, the complete graph is rarely computed since this involves a cross-product. Rather, only those edges whose weight is above a given threshold are materialized. There are well-known similarity join algorithms that perform this step efficiently as the data scales.

The deduplication operation is defined as a partition of the nodes in the similarity graph. Intuitively, one desires a partition where nodes that are connected

ID	Authors	Title	Conference
1	Nick Koudas, Sunita Sarawagi, Divesh Srivastava	Record linkage: similarity measures and algorithms	SIGMOD 2006
2	N Koudas, S Sarawagi, D Srivastava	Record linkage: similarity measures and algorithms	ACM SIGMOD 2006
3	N Koudas et al.	Record linkage: similarity measures and algorithms	ACM SIGMOD Intl. Conf. on Mgmt. of data 2006
4	A. Elmagarmid and P. G. Ipeirotis and V. Verykios	Duplicate record detection: a Survey	Information systems 2006
5	A. Elmagarmid, P. G. Ipeirotis, V. Verykios	Duplicate record detection: a Survey	Information systems 2006
6	A. Elmagarmid et al.	Duplicate record detection: a Survey	Information systems 2006

Deduplication in Data Cleaning. Figure 1. Example citation data and its deduplication.

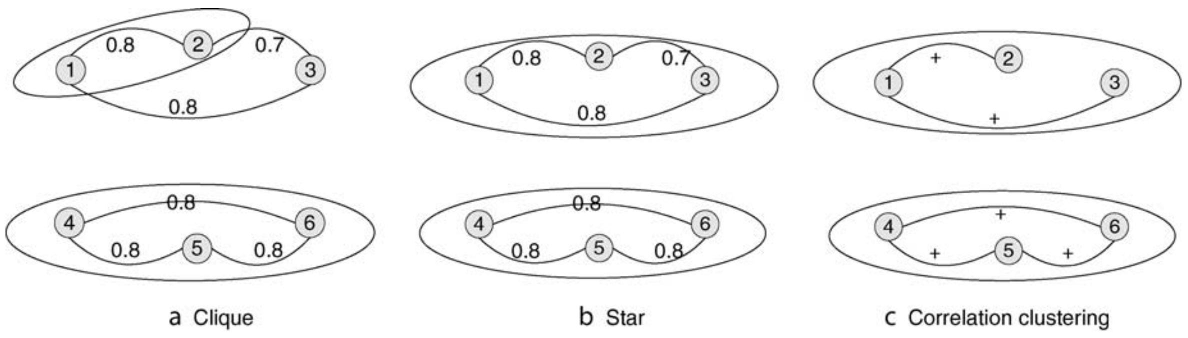


Deduplication in Data Cleaning. Figure 2. Weighted similarity graph.

with larger edge weights have a greater likelihood of being in the same group. Since similarity functions often do not satisfy properties such as triangle inequality, there are multiple ways of partitioning the similarity graph. Accordingly, several intuitive ways of partitioning the graph have been proposed in the literature. Note that this is closely related to the problem of clustering in data mining. Not surprisingly, a lot of the clustering algorithms such as K-means clustering [15] are also applicable for deduplication. While a comprehensive survey of these methods is beyond the scope of this entry, some of the key graph partitioning algorithms are reviewed.

In the *Single-Linkage Clustering* algorithm [14], at any point there is a partitioning of the nodes. At each step, the algorithm chooses a pair of groups to merge by measuring the similarity between all pairs of groups and picking the pair that has the highest similarity. The similarity between two groups of nodes is defined as the maximum similarity among all pairs of nodes, one from each group. The algorithm can be terminated in various ways, for example if there are a given number of clusters or if there are no more pairs of groups to merge which happens when all the connected components in the similarity graph have been collapsed. Figure 2 also shows the output of single-linkage when thresholds the similarity graph and retains only edges whose similarity is larger than or equal to 0.8. Notice that this partition corresponds to the one shown in Fig. 1.

One of the limitations of the single-linkage approach is that nodes two groups may be merged even if only one pair of nodes has a high similarity even if all other pairs do not. In order to address this limitation, an alternative is to decompose the graph into *cliques*. This ensures that whenever two tuples are collapsed, their similarity is large. Figure 3a shows the output of the clique decomposition for the similarity graph in Fig. 2, which only considers edges whose threshold is greater than or equal to 0.8. Since the similarity between nodes 2 and 3 is smaller than 0.8, the three nodes 1,2,3 cannot be collapsed as with single-linkage. Clique partitioning can often be overly restrictive and leads to many



Deduplication in Data Cleaning. Figure 3. Example partitions.

pairs that have a large similarity getting split into multiple groups, as is the case with nodes 1 and 3 in Fig. 3a. Further, clique partitioning is known to be NP-hard.

To address this limitation, a relaxed version of the clique approach is proposed, namely *star* clustering [2]. A *star* is a pattern consisting of a single *center* node and a set of *satellite* nodes such that there is an edge with weight above a given threshold that connects the center to each of the satellite nodes. The intuition behind this choice is that for similarity functions that do satisfy the triangle inequality, the similarity between any pair of satellite nodes cannot be arbitrarily low. Aslam et al. [2] propose a greedy algorithm to compute a star partitioning of the input similarity graph. The graph is thresholded in that only edges with weight above a given threshold are retained. At any point in the algorithm, one operates with a current partitioning of the nodes. Nodes that are not assigned to any partition so far are deemed *uncovered*. At each step, one picks the uncovered node that such that the sum of the weights of its uncovered neighbors is the largest. This node and is uncovered neighbors define the next star and are grouped together. This process is repeated until no node is left uncovered. This algorithm has been empirically proven to be effective on several real-world datasets and is deployed commercially in the Fuzzy Groupby transform in Microsoft SQL Server 2005 Integration Services. Figure 3b shows the output of the star clustering algorithm over the data in Fig. 2.

One of the more recently proposed partitioning algorithms is *correlation clustering* [3]. Here, each edge in the similarity graph is viewed as a positive(+) edge when the similarity is high, say above a fixed threshold and as a negative(−) edge if it is not a positive edge. Figure 3c shows the similarity graph in which one views each edge with weight greater than or equal to 0.8 as a positive edge. Negative edges are not shown explicitly. The goal of correlation clustering is then to find a partition of the nodes that has the *least* disagreement with the pairwise edges. For a given partition of the nodes, the disagreement cost is the number of positive edges where the nodes at either end are split into different groups plus the number of negative edges where the nodes at either end are grouped together. While finding the optimal partition is NP-hard, efficient approximation algorithms have been proposed [3,7]. In fact, a simple one-pass randomized algorithm similar in spirit to the star-clustering algorithm can be shown to be a 3-approximation with high probability. The algorithm maintains a current partition of the nodes. At each step, it picks a random uncovered node. This node with all its uncovered positive neighbors define the next group to be merged. This process is repeated until no node is uncovered. Figure 3c shows the output of the correlation clustering algorithm on the data in Fig. 2.

Multi-Attribute Deduplication

Thus far, the entire discussion has focused on the computation of a single partition of the input table. However, one might often wish to partition the same input table or even multiple tables in a database in different ways. For example, the data in Fig. 1 could be partitioned by `author`, by `title` and by `conference`. These partitions of the data do not necessarily have to agree. One approach to computing these partitions is to separately run the algorithms described above for each of the chosen attributes. But clearly these attributes are not independent of one another. For example, the fact that the paper titles of papers 2 and 3 are grouped together provides additional evidence that the respective conference values must also be grouped together.

Ananthakrishna et al. [1] exploit this intuition in a multi-table setting where there is a hierarchical relationship among the different tables. For example, consider an address database with separate tables for nations, states, cities and street addresses. The idea is to proceed bottom-up in this hierarchy. The idea is to begin by deduplicating streets and then follow it up with cities, states and finally nations. At each step, the output of the previous step is used as contextual information for the current step. For example, if two states share many common cities, this provides further evidence that the two states must be collapsed.

The same intuition has also been explored in a more general setting where the data objects are arranged in a graph [9].

Constraints

A large body of work has focused on incorporating various constraints over the data into the deduplication process [3–5,8,18]. Five classes of work are identified in this overall space of incorporating constraints.

The first consists of constraints on individual tuples. These are constraints that express the condition that only some tuples (for instance "authors that have published in SIGMOD") may participate in the deduplication. Such constraints are easily enforced by pushing these filter conditions before deduplication is invoked.

The second consists of constraints that are in effect parameters input to the deduplication algorithm. For instance, the *K*-means clustering algorithm takes as input the number *K* that indicates the number of output groups desired. If the user of deduplication has some expectation for the number of groups returned after the data is partitioned, this may be used as a constraint during the deduplication itself.

Another class of constraints consists of positive and negative examples [5]. For example, for the data in Fig. 1, one might know that records 1 and 2 are indeed the same. Alternatively, one might also provide negative examples such as the fact that records 1 and 4 are *not* identical. These constraints may be used either as hard constraints to only consider partitions where the constraints are met, or to *adapt* the similarity function to accommodate the input examples.

The role of functional dependencies and inclusion dependencies in deduplication has also been explored [6]. The idea is to model the deduplication problem as a renaming problem where one is allowed to rename the attribute values of any record. At the end of the renaming, the partitions are defined by exact matching. Thus if the titles of records 1 and 2 in Fig. 1 are renamed to be the same, this signifies that these two titles have been collapsed. Every renaming has a *cost* defined by a distance function. The cost of a renaming is the distance between the new and old values. The input also consists of a set of functional and inclusion dependencies that are supposed to hold on the clean data. For example, one might require that `title` functionally determine `conference` for the data in Fig. 1. The goal is to compute the lowest cost renaming such that the output of the renaming satisfies all the constraints. Since this problem is NP-hard to solve optimally, efficient heuristics are proposed in [6].

Finally, the use of *groupwise* constraints for deduplication has also been investigated [18,8]. Here every *group* of tuples is expected to satisfy a given aggregate constraint. Such constraints arise when multiple sources of data are integrated. For instance, suppose one is integrating data from ACM and DBLP and that one is performing deduplication on the set of authors in the union of these sources. One constraint one might wish to assert on any subset of authors being considered for a collapse is that the set of papers authored in ACM and DBLP have a substantial overlap.

With an appropriate use of similarity function thresholds, constraints, and correlation attributes a user can guide the deduplication process to achieve the desired result.

Key Applications

See applications of data cleaning.

Cross-references

► Data Cleaning
► Data Integration
► Inconsistent Databases
► Probabilistic Databases
► Record Matching

Recommended Reading

1. Ananthakrishna R., Chaudhuri S., and Ganti V. Eliminating fuzzy duplicates in data warehouses. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
2. Aslam J.A., Pelehov K., and Rus D. A practical clustering algorithm for static and dynamic information organization. In Proc. 10th Annual ACM -SIAM Symp. on Discrete Algorithms, 1999.
3. Bansal N., Blum A., and Chawla S. Correlation clustering. Mach. Learn., 56(1–3):89–113, 2002.

4. Bhattacharya I. and Getoor L. Collective entity resolution in relational data. In Data Engineering Bulletin, 2006.
5. Bilenko M., Basu S., and Mooney R.J. Integrating constraints and metric learning in semi-supervised clustering. In Proc. 21st Int. Conf. on Machine Learning, 2004.
6. Bohannon P., Fan W., Flaster M., and Rastogi R. A cost based model and effective heuristic for repairing constraints by value modification. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
7. Charikar M., Guruswami V., and Wirth A. Clustering with qualitative information. In J. Comp. and System Sciences, 2005.
8. Chaudhuri S., Sarma A., Ganti V., and Kaushik R. Leveraging aggregate constraints for deduplication. In Proc. ACM SIGMOD Int. Conf. on Management of Data. 2007.
9. Dong X., Halevy A.Y., and Madhavan J. Reference reconciliation in complex information spaces. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
10. Fuxman A., Fazli E., and Miller R.J. ConQuer: efficient management of inconsistent databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
11. Galhardas H., Florescu D., Shasha D., Simon E., and Saita C. Declarative data cleaning: Language, model, and algorithms. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
12. Koudas N., Sarawagi S., and Srivastava D. Record linkage: similarity measures and algorithms. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.
13. Sarawagi S. and Bhamidipaty A. Interactive deduplication using active learning. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002.
14. Single Linkage Clustering. http://en.wikipedia.org/wiki/Single_linkage_clustering.
15. The K-Means Clustering Algorithm. http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html.
16. Trillium Software. http://www.trilliumsoft.com/trilliumsoft.nsf.
17. Toney S. Cleanup and deduplication of an international bibliographic database. Inform. Tech. Lib., 11(1), 1992.
18. Tung A.K.H., Ng R.T., Lakshmanan L.V.S., and Han J. Constraint-based clustering in large databases. In Proc. 8th Int. Conf. on Database Theory, 2001.
19. Yancey W.E. Bigmatch: a program for extracting probable matches from a large file for record linkage. Statistical Research Report Series RRC2002/01, US Bureau of the Census, 2002.

Deep-Web Search

Kevin C. Chang
University of Illinois at Urbana-Champaign, Urbana, IL, USA

Synonyms

Hidden-Web search

Definition

With the proliferation of dynamic Web sites, whose contents are provided by online databases in response to querying, *deep-Web search* aims at finding information from this "hidden" or "deep" Web. Current search engines crawl and index pages from statically linked Web pages, or the "surface" Web. As such crawlers cannot effectively query online databases, much of the deep Web is not generally covered by current search engines, and thus remains invisible to users. A deep-Web search system queries over online databases to help users search these databases uniformly.

Historical Background

In the last few years, the Web has been rapidly "deepened" by the massive networked databases on the Internet. While the *surface Web* has linked billions of static HTML pages, a far more significant amount of information is hidden in the *deep Web*, behind the query forms of *searchable* databases. A July 2000 survey [1] claims that there were 500 billion hidden pages in 10^5 online sources, and an April 2004 study [3] reports 1.2 million query interfaces on the Web. Such information cannot be accessed directly through static URL links; they are only available as responses to dynamic queries submitted through the query interface of a database. Because current crawlers cannot effectively query databases, such data is invisible to traditional search engines, and thus remain largely hidden from users.

With the proliferation of such dynamic sources, deep-Web search has become crucial for bridging users to the vast contents behind query forms. Unlike traditional search on the surface Web, searching over the deep Web requires integrating structured data sources at a large scale. While information integration has been investigated actively for decades, since 2000, *large scale information integration* has become the key issue for enabling deep-Web search: How to tackle the challenge of large scale? How to take advantage of large scale?

Foundations

Search Architecture

With the dynamic, structured nature of deep-Web data sources, depending on the application domains of focus, a deep-Web search system can take two forms of architecture:

1. *Crawl-and-Index*: A search system can crawl deep-Web sources regularly to build a local index for search, similar to what the current "surface-oriented" Web search engines do. This approach, by building a local index, will give fast online search performance but may risk returning stale data, if target sources are updated frequently. For comprehensive and up-to-date crawling from data sources offline to retrieve their structured data objects, it will require special "structure-aware" crawling techniques that can deal with query-driven sources.
2. *Discovery-and-Forward*: A search engine can automatically discover databases from the Web, by crawling and indexing their query interfaces (and not their data pages), and forward user queries to the right sources for the actual search of data, upon user querying. This approach, with its on-the-fly querying, gives up-to-date results, while there may be increased query latency online. As a key challenge, it needs to be able to interact with data sources to execute user queries.

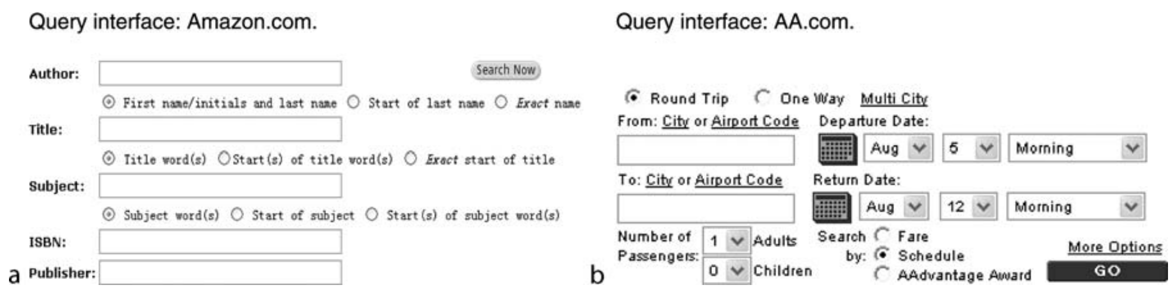
Source Modeling

How does one to find sources and model what they are about? In building a deep-Web search system, one first needs to determine the set of sources to integrate, either by crawling offline or querying online. With the large scale of the Web, where numerous sources are scattered everywhere, it is necessary to crawl the Web to discover and model their query interfaces. After crawling finds a query interface, its query attributes and capabilities must be extracted. The objective of this *form extraction*, which extracts the structure of query forms, is to provide a "model" of how a source is queried, so that the search system can interact with the source.

Guarding data behind them, such query interfaces are the "entrance" to the deep Web. These interfaces, usually in HTML *query forms*, express *query conditions* for accessing objects from databases behind. Each condition, in general, specifies an *attribute*, one or more supported *operators* (or modifiers), and a *domain* of allowed values. A condition is thus a three-tuple [`attribute`; operators; domain], e.g., [`author`; {`"first name..."`, `"start..."`, `"exact name"`}; `text`] in interface shown in Fig. 1a. Users can then use the condition to formulate a specific constraint (e.g., [`author = "tom clancy"`] by selecting an operator (e.g., `"exact name"`) and filling in a value (e.g., `"tom clancy"`). For modeling and integrating Web databases, the very first step is to "understand" what a query interface says – i.e., what *query capabilities* a source supports through its interface, in terms of specifiable conditions. For instance, Amazon.com (Fig. 1a) supports a set of five conditions (on `author`, `title`, ..., `publisher`). Such query conditions establish the *schema model* underlying the Web query interface. These extracted schema model, each of which representing the query capabilities of a source, can then be used for determining *whether* a source is relevant to the desired applications or user queries and *how* to query the source.

There are two main approaches for form extraction: The first approach, *grammar-based parsing* [9], observes that there are common visual patterns in the layout of query forms, abstracts these patterns as grammar production rules, and executes a "best-effort parsing" to extract the form structure. That is, given the set of grammar rules, it assembles the 'best-possible" parse of the query form. In this approach, it is shown that a small set of (20–30) patterns can already capture common layouts across different subject domains (books, airlines, *etc.*) of query forms.

The second approach, *classifier-based recognition*, applies learning-based or rule-based classifiers to determine the role of each form element, to formulate an overall interpretation of the form (e.g., [4]). It can first identify, for each input element (e.g., a text



Deep-Web Search. Figure 1. Query interfaces examples.

input box), a set of text elements as the candidates of its labels. Based on the candidates for each input element, the process then goes on to form a global assignment, which will reconcile conflicts between the candidates. The recognition of candidates can be based on features such as the proximity between elements (e.g., what are the closest text elements to this input element?) or the content of text elements (e.g., can one classify this text element as likely a label, e.g., "author"?).

Schema Matching

Schema matching, for corresponding attributes at different sources, is crucial for translating queries (as will be discussed later) to search these sources. It has been a central problem in information integration, and thus also a key challenge for deep-Web search. Because one is dealing with querying Web sources, such matching takes query forms (as extracted by source modeling) as input schemas (where attributes are specified for querying) and attempts to unify them by finding their correspondence (or by matching them to a common mediated schema). While a classic problem, however, it takes a new form in this context, with both new challenges and new insights.

As the new challenge, for searching the deep Web, in many application scenarios, such matching must deal with *large scale*, to match numerous sources (e.g., many airfare sources; book sources). Traditional schema matching targets at finding attribute correspondence between a pair of sources. Such pairwise matching, by tackling two sources at a time, does not directly handle a large number of data sources.

On the other hand, however, the new setting also hints on new approaches that can exploit the scale. Given there are many sources on the Web, instead of focusing on each pair of sources, one can observe the schemas of all the sources together, to take advantage of the "holistic" information observable only across many sources. Such holistic methods can, in particular, explore the "regularity" existing among Web sources. Note that – since one is dealing with query forms for schema matching – these query forms are "external" schemas, which are designed for users to easily understand. Unlike "internal" schemas (say, for database tables), which may use arbitrary variable names (e.g., "au_name"), these external schemas tend to use common terms (e.g., "author") as the names of attributes. As conventions naturally emerge, it has been observed [2] that certain regularities exist, in particular, for what attributes are likely to appear and what names they may be labeled with.

There are several approaches that leverage this new insight of *holistic* schema matching, to exploit the regularities across multiple sources. First, the approach of *model discovery* [2]: Schema matching may be considered as discovering a hidden generative model that dictates the occurrences of attribute names (such as when to use "author" versus "first name" and "last name"). Such a model effectively captures how attribute terms would occur – some attributes are *synonyms* that will not be used together (e.g., "author" and "name"), and some attributes are usually *grouped* with each other (e.g., "first name" and "last name"). With this conceptual view, the schema matching problem is transformed into finding an underlying model that is statistically consistent with the input schemas (to be matched). Various statistical methods can then be used for this model discovery, such as hypothesis testing or correlation mining.

Second, the approach of *attribute clustering*: In this view, one considers finding synonym attributes as clustering of those attributes into equivalent classes [8]. Given a set of query forms, where attributes are extracted with a hierarchy that indicates how they are related to each other in a form, this approach clusters attributes from different sources into a hierarchy of clusters that represent their synonym and grouping relationships. The clustering process will exploit the hierarchical relationship extracted from each individual query form as well as the similarities between attribute labels.

Furthermore, another new insight can be observed that, in the context of the deep Web, since one is dealing with matching between query interfaces, it is possible to actually try out querying, or *probing*, each source to derive their attribute correspondence. Exploiting the dynamic response of query forms has been explored for schema matching [6].

Offline Crawling

In the Crawl-and-Index architecture, like a typical search engine, data objects can be collected from various sources, index them in a local database, and provide search from the database. To realize this search framework, one needs to crawl data offline (before user searches) from various sources. The objective of this *deep Web crawling* is to collect data, as exhaustively as possible, from each source to provide comprehensive coverage. As such sources are accessed by querying through their query forms, a deep Web "crawler" [5] thus, for each source, repeatedly submitting queries and collect data from query results. There are two issues for realizing this crawling:

First, *comprehensiveness*: How to formulate queries so that the results cover as much as possible the data at the source? Each query is a way to fill in the query form of the source. For those query attributes that have a clearly defined set of input values (e.g., a selection-box of two choices {paperback, hardcover} for book format), each value can be enumerated in querying. For attributes with an open set of input values, such as a text input that takes any keywords, the crawler will need to be driven by certain "domain knowledge" that can generate a set of keyword terms relevant to the application, e.g., {computer, internet, web, databases, . . .}.

Second, *efficiency*: How to minimize the number of queries to submit to a source for crawling its content? Since different queries may retrieve overlapping results, it is possible to choose different sets of queries for crawling. For efficiency, one wants to minimize the cost of querying, such as the number of queries submitted. The problem is thus, as each data object can be reached by multiple queries, to select the smallest set of queries that together *reach* all the data objects [7].

Online Querying

In the Discover-and-Forward architecture, user queries are directed to search relevant sources, at the time of querying. Unlike offline crawling, which aims at collecting *all* the data from each source by *any* query (subject to cost minimization), this online querying must execute a specific user query at run time. Given a query (in some unified front-end language), for each source, one needs to translate the query for the source, to fill in its query form. This *query translation* will rewrite the original query into a new format that satisfies two criteria: First, it is executable at the source: The new query must contain only query conditions and operators that the source can accept in its query form. Second, it is a good approximation of the original query: The translated query, if not "equivalent" to the original query, should be as close to it as possible. In terms of techniques, query translation (e.g., [10]) can be driven by either generic type hierarchies (e.g., how to transform commonly-used data types such as date and number) or specialized domain knowledge.

Key Applications

- Enhancing General Web Search: Current search engines have started to combine specialized search responses for certain categories of queries (e.g., returning stock information for search of company names).
- Enabling Vertical Web Search: Searching data on the Web in specific domains, such as travel or real estate.

Cross-references

▶ Information Integration
▶ Query Translation
▶ Schema Matching

Recommended Reading

1. BrightPlanet.com. The Deep Web: Surfacing Hidden Value. http://www.brightplanet.com/resources/details/deepweb.html, 2000.
2. He B. and Chang K.C.C. Statistical Schema Matching across Web Query Interfaces. In Proc. ACM SIGMOD Int. Conf. on Management of Data,, 2003, pp. 217–228.
3. He H., Meng W., Lu Y., Yu C.T., and Wu Z. Towards deeper understanding of the search interfaces of the deep Web. In Proc. 16th Int. World Wide Web Conference, 2007, pp. 133–155.
4. He B., Patel M., Zhang Z., and Chang K.C.C. Accessing the deep Web: a survey. Commun ACM, 50(5): 94–101, 2007.
5. Raghavan S. and Garcia-Molina H. Crawling the hidden Web. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 129–138.
6. Wang J., Wen J.R., Lochovsky F.H., and Ma W.Y. Instance-based schema matching for web databases by domain-specific query probing. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 408–419.
7. Wu P., Wen J.R., Liu H., and Ma W.Y. Query selection techniques for efficient crawling of structured Web sources. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 47.
8. Wu W., Yu C.T., Doan A., and Meng W. An interactive clustering-based approach to integrating sourc query interfaces on the deep Web. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 95–106.
9. Zhang Z., He B., and Chang K.C.C. Understanding Web query interfaces: best-effort parsing with hidden syntax. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 117–118.
10. Zhang Z., He B., and Chang K.C.C. Light-weight domain-based form assistant: querying web databases on the fly. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 97–108.

Degrees of Consistency

▶ SQL Isolation Levels

DEMs

► Digital Elevation Models

Degrees of Cosistency

► SQL Isolation Levels

Dendrogram

► Visualizing Clustering Results

Dense Index

Mirella M. Moro[1], Vassilis J. Tsotras[2]
[1]Federal University of Rio Grande do Sul, Porte Alegre, Brazil
[2]University of California-Riverside, Riverside, CA, USA

Definition

Consider a tree-based index on some numeric attribute *A* of a relation *R*. This index is called *dense* if every search-key value of attribute *A* in relation *R* also appears in the index. Hence for every search-key value *x* in *A*, there is an index record of the form <*x, pointer*>, where *pointer* points to the first record (if many such exist) in relation *R* that has *R.A* = *x*.

Key Points

Tree-based indices are built on numeric attributes and maintain an order among the indexed search-key values. Hence, they provide efficient access to the records of a relation by attribute value. Consider for example an index built on attribute *A* of relation *R*. The leaf pages of the index contain *index-records* of the form <*search-key, pointer*>, where search-key corresponds to a value from the indexed attribute *A* and *pointer* points to the respective record in the indexed relation *R* with that attribute value. If all distinct values that appear in *R.A* also appear in index records, this index is *dense*, otherwise it is called *sparse*. If there are many records in relation *R* that have *R.A* = *x*, then the index record <*x, pointer*> points to the first of these records. If relation *R* is ordered on attribute *A*, the rest of these records are immediately following after the first accessed record (in the same page of the relation file). If relation *R* is not ordered according to attribute *A*, the remaining records can be accessed from a list of pointers after this first record.

Tree-indices are further categorized by whether their search-key ordering is the same with the relation file's physical order (if any). Note that a relation file may or may not be ordered. For example, if a relation *R* is ordered according to the values on the *A* attribute, the values in the other attributes will not be in order. If the search-key of a tree-based index is the same as the ordering attribute of a (ordered) file then the index is called *primary*. An index built on any non-ordering attribute of a file is called *secondary*. If an index is secondary then it should also be dense. Since the index is secondary, the relation is not ordered according to the search-key value of this index. As a result, finding a given record in the file by this index requires that all search-key values of the indexed attribute are present in the index, i.e. it is a dense index.

A dense index is typically larger than a sparse index (since all search-key values are indexed) and thus requires more space. It also needs to be updated for every relation update that involves the attribute value being indexed.

Cross-references

► Access Methods
► B+-Tree
► Indexing
► ISAM

Recommended Reading

1. Elmasri R. and Navathe S.B. Fundamentals of Database Systems, 5th edn. Addisson-Wesley, Boston, MA, 2007.
2. Manolopoulos Y., Theodoridis Y., and Tsotras V.J. Advanced Database Indexing. Kluwer, Dordecht, 1999.
3. Silberschatz A., Korth H.F., and Sudarshan S. Database System Concepts, 5th edn. McGraw-Hill, New York, 2006.

Dense Pixel Displays

Daniel A. Keim, Peter Bak, Matthias Schäfer
University of Konstanz, Konstanz, Germany

Synonyms

Data visualization; Information displays; Pixel oriented visualization techniques; Visual data exploration;

Information visualization; Visualizing large data sets; Visualizing multidimensional and multivariate data; Visual data mining

Definition

Dense Pixel Displays are a visual data exploration technique. Data exploration aims at analyzing large amounts of multidimensional data for detecting patterns and extracting hidden information. Human involvement is indispensable to carry out such a task, since human's powerful perceptual abilities and domain knowledge are essential for defining interesting patterns and interpreting findings. Dense pixel displays support this task by an adequate visual representation of as much information as possible while avoiding aggregation of data-values. Data is shown using every pixel of the display for representing one data point. Attributes of the data are mapped in separate sub-windows of the display, leaving one attribute for one sub-window. The data point's value for an attribute is mapped to the pixel's color. The arrangement of pixels in sub-windows is determined by the mapping of the data-sequence into a two dimensional space preserving the pixel's neighborhood relations.

A number of different pixel-oriented visualization techniques have been proposed in recent years and shown to be useful for visually exploring data in many applications. These techniques differ mainly in their approach to arrange individual pixels in relation to each other, and in their choice of shaping the sub-windows to make maximal use of space.

Historical Background

Scientific, engineering and environmental databases containing automatically collected data have grown to an overwhelming size, which need to be analyzed and explored. The desire to augment human's perceptual abilities and understand multidimensional data inspired many scientists to develop visualization methodologies. The progress in information technology offers more and more options to visualize data, in particular multidimensional data. Detecting information in low dimensions (1D, 2D, 3D) by taking advantage of human's powerful perceptual abilities is crucial to understand the higher dimensional data set. This assumption is fundamental in all visualization techniques.

One important thing in analyzing and exploring large amounts of data is to visualize the data in a way that supports the human. Visual representations of the data are especially useful for supporting a quick analysis of large amounts of multi-modal information, providing the possibility of focusing on minor effects while ignoring known regular features of the data. Visualization of data which have some inherent two- or three-dimensional semantics has been done even before computers were used to create visualizations. In the well-known books [12,13], Edward R. Tufte provides many examples of visualization techniques that have been used for many years. Since computers are used to create visualizations, many novel visualization techniques have been developed by researchers working in the graphics field. Visualization of large amounts of arbitrary multidimensional data, however, is a fairly new research area. Early approaches are scatterplot matrices and coplots [5], Chernoff faces [4], parallel coordinates [6], and others [1,3]. Researchers in the graphics/visualization area are extending these techniques to be useful for large data sets, as well as developing new techniques and testing them in different application domains. Newer approaches for dense pixel displays are developed for several applications for visualizing data, i.e. geospatial data [9], financial data [14], text [10], and neuroscience data [11].

Foundations

Dense Pixel Displays as Optimization Problem – Pixel Arrangement

One of the first questions in creating a dense pixel display is how the pixels are arranged within each of the sub-windows. Therefore, only a good arrangement will allow a discovery of clusters and correlations among the attributes. It is reasonable to distinguish between data sets where a natural ordering of the items is given, such as in time-series data or a production-line, and between data sets without inherent ordering, such as results of search-queries. In all cases, the arrangement has to consider the preservation of the distance of the one-dimensional ordering in the two-dimensional arrangement.

Mappings of ordered one-dimensional data sets to two dimensions have been attracted the attention of mathematicians already long before computer came into existence.

So called space-filling curves attempt to solve this problem. The Peano-Hilbert curve provides one possible solution. However, with this method it is often difficult to follow the flow of items. The Morton curve is an alternative proposed to overcome this problem, which has a simpler regularity.

The general idea, as first proposed in [8] is based on the idea of a line and column-wise arrangement of the pixels. If simple left-right or top-down arrangements are used on pixel level, the resulting visualizations do not provide useful results, since it creates random neighborhoods that bias user's perception of correlations and clusters.

One possibility to improve the visualizations is to organize the pixels in small groups and arrange the groups to form some global pattern, as suggested by recursive pattern visualizations. The basic idea of recursive pattern visualization technique is based on a scheme which allows lower-level patterns to be used as building blocks for higher-level patterns. In the simplest case the patterns for all recursion levels are identical. In many cases, however, the data has some inherent structure which should be reflected by the pattern of the visualization. In order to create an appropriate visualization of patterns in a recursive manner, the algorithm must allow users to provide parameters for defining the structure for the low and high recursion levels.

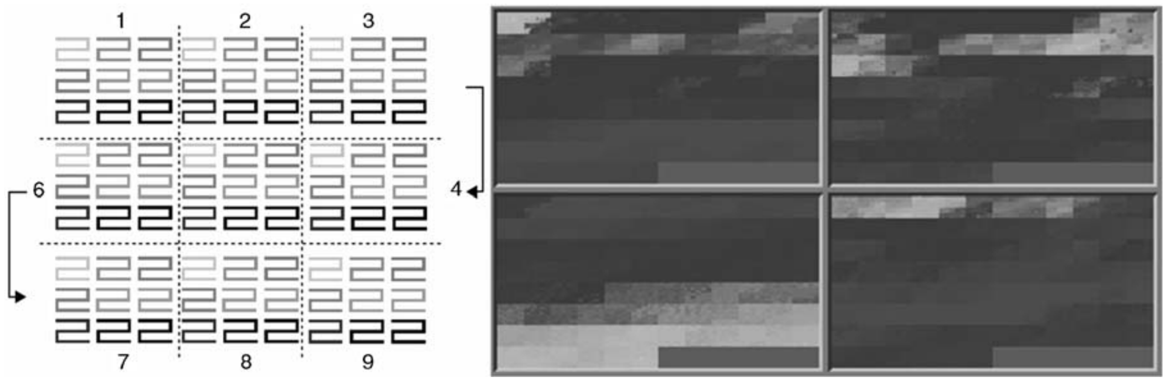
Therefore, one major difference between the "recursive pattern" technique and other visualization techniques is that it is based on a generic algorithm allowing the users to control the arrangement of pixels. By specifying the height and width for each of the recursion levels, users may adapt the generated visualizations to their specific needs. This allows the "recursive pattern" technique to be used for a wide range of tasks and applications.

A schematic example for a highly structured arrangement is provided in the Fig. 1 The visualization shows the financial data of four companies for eight years. The subspaces are subdivided into different time intervals. A sub-window represent the whole time period, which is further divided into single years, months, weeks and days. Since days represent the lowest recursion level, they correspond to one single pixel (as schematically shown in Fig. 1 Left). This structure is preserved in the visual representation of the dense pixel display allowing fast extraction detailed information (Fig. 1 Right). One of the tasks that would benefit from such a representation is detecting a strong correlation for example between IBM stock-prices and the dollar's exchange rate in the first 2.5 years.

Dense Pixel Displays as Optimization Problem – Shape of Sub-windows

The second important question concerns how the shapes of the sub-windows should be designed. Sub-windows are used to represent single attributes of the dataset. In the optimization functions used in the past, the distance between the pixel-attributes belonging to the same data object was not taken into account. This, however, is necessary in order to find appropriate shapes for the sub-windows. In many cases a rectangular partitioning of the display was suggested. The rectangular shape of the sub-windows allows a good screen usage, but at the same time it leads to a dispersal of pixels belonging to one data object over the whole screen, especially for data sets with many attributes. Because the sub-windows for the attributes are rather far apart, it might be difficult to detect correlating patterns.

An idea for an alternative shape of the sub-windows is the circle segments technique. The technique



Dense Pixel Displays. Figure 1. Left: Schematic representation of a highly structured arrangement. Right: Recursive arrangement of financial data (September 1987–February 1995: above left IBM, above right DOLLAR, below left: DOW JONES, below right GOLD) (adopted from [8]).

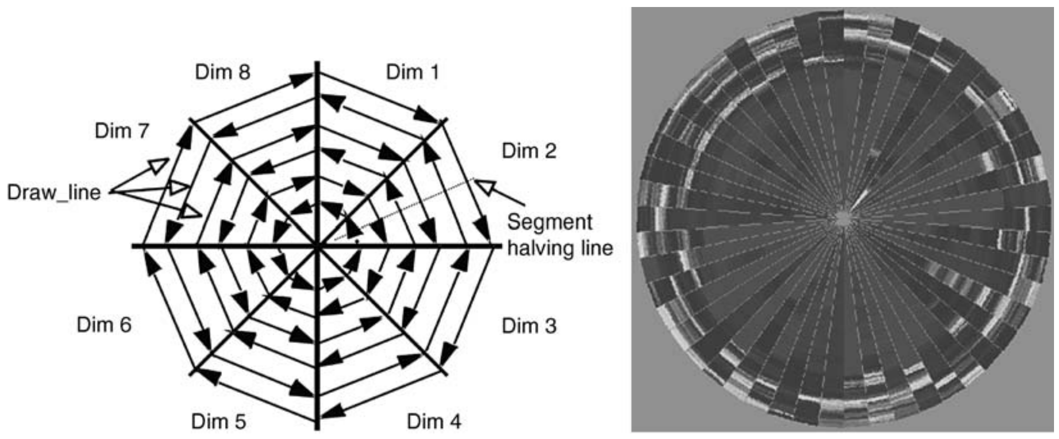
tries to minimize the average distance between the pixels that belong to the dimensions of one data object.

The fundamental idea of the "circle segments" visualization technique is to display the data dimensions as segments of a circle (as shown in Fig. 2). If the data consists of k attribute-dimensions, the circle is partitioned into k segments, each representing one dimension. The data items within one segment are arranged in a back and forth manner orthogonal to the line that halves the two border lines of the segment (cf. Fig. 2). The drawing starts in the centre of the circle and draws the pixels from one border line of the segment to the other. Whenever the drawing hits one of the border lines, it is moved towards the outside of the circle by changing the direction. When the segment is filled out for one attribute, the next segment will start, until all dimensions are visualized. A schematic representation is shown in Fig. 2 (left). The results for such a representation is shown for 50 stock prices from the Frankfurt stock index, representing about 265,000 data values (Fig. 2 Right). The main advantage of this technique is that the overall representation of the whole data set is better perceivable – including potential dependencies, analogies, and correlations between the dimensions.

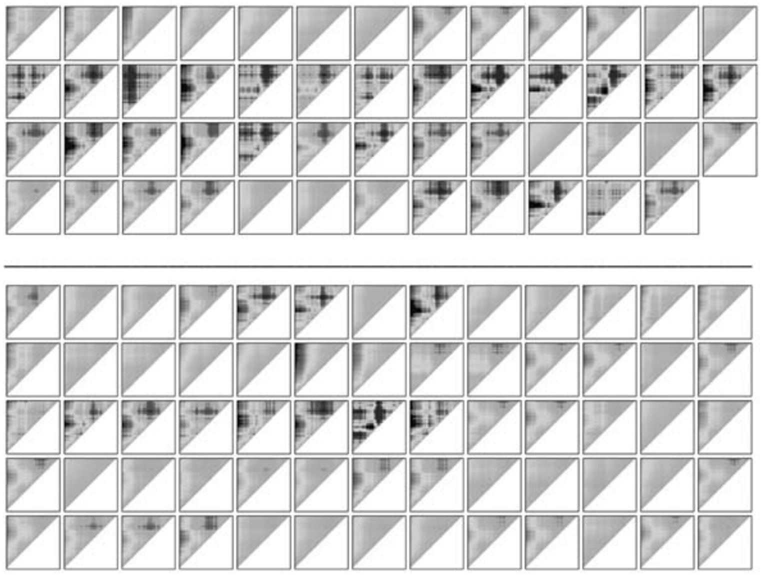
Another arrangement of sub-windows is the arrangement as bar chart, and resulting technique is called Pixelbarcharts (see Fig. 4). The basic idea of Pixelbarcharts is to use the intuitive and widely used presentation paradigm of bar charts, but also use the available screen space to present more detailed information. By coloring the pixels within the bars according to the values of the data records, very large amounts of data can be presented to the user. To make the display more meaningful, two parameters of the data records are used to impose an ordering on the pixels in the x- and y-directions. Pixel bar charts can be seen as a generalization of bar charts. They combine the general idea of xy-plots and bar charts to allow an overlap-free, non-aggregated display of multi-attribute data.

Dense Pixel Displays as Optimization Problem – Ordering of Dimensions

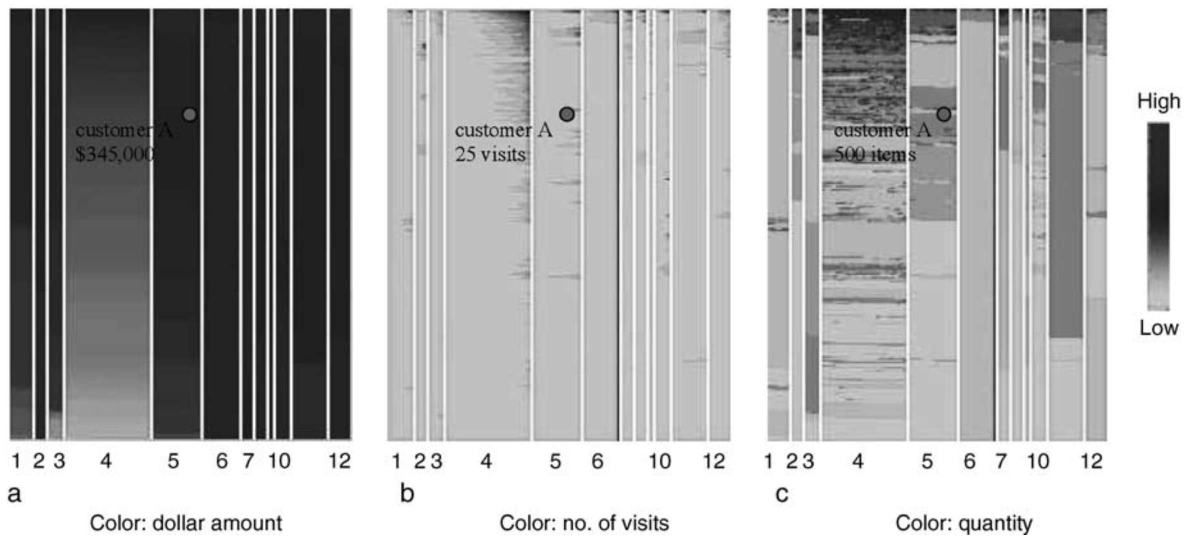
The final question to consider is the ordering of attributes. This problem is actually not just a problem of pixel-oriented techniques, but a more general problem which arises for a number of other techniques such as the parallel coordinates technique. The idea is that the data attributes have to be positioned in some one- or two-dimensional ordering on the screen, and this is usually done more or less by chance – namely in the order in which the dimensions happen to appear in the data set. The ordering of dimensions, however, has a major impact on the expressiveness of the visualization. One solution for finding an effective order of dimensions is to arrange the dimensions according to their similarity. For this purpose, the technique has to define measures which determine the similarity of two dimensions. Data mining algorithms, such as regression models or clustering tools, but also statistical approaches, such as inter-correlation matrixes, can provide solutions for such a problem.



Dense Pixel Displays. Figure 2. Left: Circle segment technique schema for eight attribute-dimensions. Right: Twenty years of daily data of the FAZ Index (January 1974–April 1995) (adopted from [1]).



Dense Pixel Displays. Figure 3. Top: Shows 51 various funds of the same financial institution and their performance over time, green means plus, red minus. Good and bad performing funds are visible easily. Bottom: Shows 65 various funds of another financial institution and their performance over time, green means plus, red minus. Good and bad performing funds are visible easily. Summed up, it is also visible that funds of the below financial institution are more continuous and have less volatility.



Dense Pixel Displays. Figure 4. Illustrates an example of a multi-pixel bar chart of 405,000 multi-attribute web sales transactions. The dividing attribute is product type; the ordering attributes are number of visits and dollar amount. The colors in the different bar charts represent the attributes dollar amount, number of visits, and quantity (adopted from[7]).

Key Applications

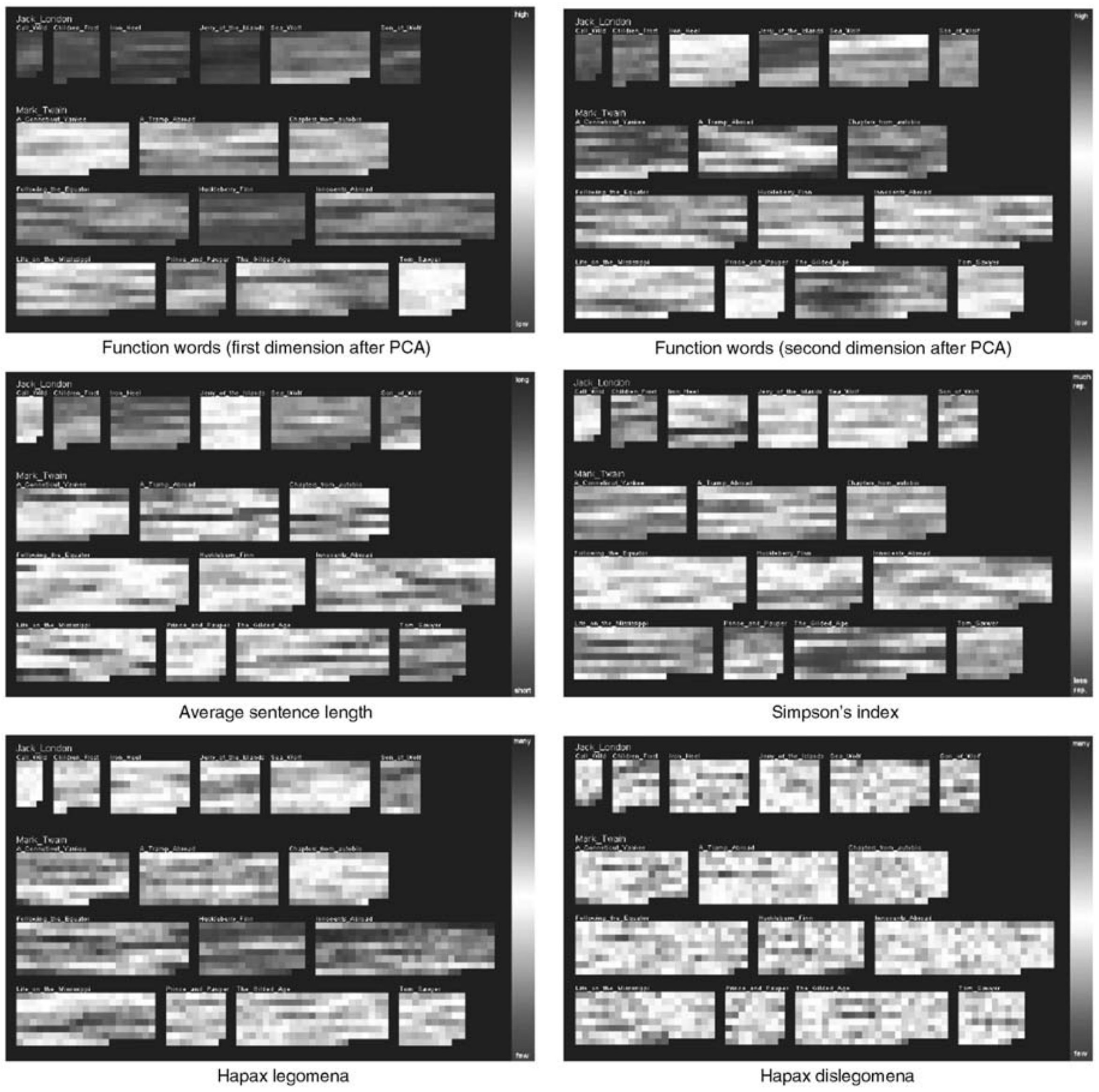
Dense Pixel Displays have a wide range of applications. They are especially important for very large high dimensional data sets, which occur in a number of applications. Example applications include financial data analysis like the "financial matrix" [14], business data analysis like "Pixelbarcharts" [7], text analysis like "literature fingerprinting" [10] and geospatial analysis like

"PixelMaps" [9]. The following sections show a few examples of the techniques.

Financial Matrix

"Financial matrix" visualization is a relevance driven visualization technique of financial performance measures. Standard statistical measures for technical financial data analysis often produce insufficient and misleading results that do not reflect the real performance of an asset.

The technique for visualizing financial time series data eliminates these inadequacies, offering a complete



Dense Pixel Displays. Figure 5. Analysis of the discrimination power of several text measures for authorship attribution. Each pixel represents a text block and the pixels are grouped into books. Color is mapped to the feature value, e.g. in c) to the average sentence length. If a measure is able to discriminate between the two authors, the books in the first line (that have been written by J. London) are visually set apart from the remaining books (written by M. Twain). This is true for example for the measure that was used in c) and d) but not for the measure that was used to generate f). Outliers (such as the book Huckleberry Finn in the middle of the third line in c)) stick out immediately. The technique allows a detailed analysis of the development of the values across the text (adopted from [10].

view on the real performance of an asset. The technique is enhanced by relevance and weighting functions according to the user's preferences in order to emphasize specific regions of interest. Figure 3 shows an example of the visualization technique, the performance of funds is highlighted green or red over various buying and selling points over the time.

Pixelbarcharts

Pixelbarcharts retain the intuitiveness of traditional bar charts while allowing very large data sets to be visualized in an effective way. For an effective pixel placement, it solves the complex optimization problem. In Fig. 4 the following facts can be observed:

- Product type 10 and product type 7 have the top dollar amount customers (dark colors of bar 7 and 10 in Fig. 4a).
- The dollar amount spent and the number of visits are clearly correlated, especially for product type 4 (linear increase of dark colors at the top of bar 4 in Fig. 4b).
- Product types 4 and 11 have the highest quantities sold (dark colors of bar 4 and 11 in Fig. 4 c).
- By clicking on a specific pixel (A), one may find out that customer A visited 25 times, bought 500 items, and spent $345,000 on product type 5.

Literature Fingerprinting

In computer-based literary analysis different types of features are used to characterize a text. Usually, only a single feature value or vector is calculated for the whole text. "Literature fingerprinting" combines automatic literature analysis methods with an effective visualization technique to analyze the behavior of the feature values across the text. For an interactive visual analysis, a sequence of feature values per text is calculated and presented to the user as a characteristic fingerprint. The feature values may be calculated on different hierarchy levels, allowing the analysis to be done on different resolution levels. Figure 5 gives an impression of the technique.

Cross-references

► Data Visualization
► Text Visualization
► Visual Analytics
► Visual Data Mining
► Visualizing Categorical Data
► Visualizing Clustering Results
► Visualizing Hierarchical Data
► Visualizing Network Data
► Visualizing Quantitative Data

Recommended Reading

1. Anderson E. A semigraphical method for the analysis of complex problems. Proc. Nat. Acad. Sci. USA, 13:923–927, 1957.
2. Ankerst M., Keim D.A., and Kriegel H.-P. Circle segments: a technique for visually exploring large multidimensional data sets. In Proc. IEEE Symp. on Visualization, 1996.
3. Brissom D. Hypergraphics: Visualizing Complex Relationships in Art, Science and Technology (AAAS Selected Symposium; 24). Westview Press, 1979.
4. Chernoff H. The use of faces to represent points in k-dimensional space graphically. J. Am. Stat. Assoc., 68(342):361–368, 1973.
5. Cleveland W.S. Visualizing Data. Hobart Press, Summit, NJ, 1993.
6. Inselberg A. N-Dimensional Graphics Part I: Lines and Hyperplanes, IBM LA Science Center Report, # G320–2711, 1981.
7. Keim D.A., Hao M.C., Dayal U., and Hsu M. Pixel bar charts: a visualization technique for very large multi-attribute data sets. Inf. Visualization, 1(1):20–34, 2001.
8. Keim D.A., Kriegel H.-P., and Ankerst M. Recursive pattern: a technique for visualizing very large amounts of data. In Proc. IEEE Symp. on Visualization, 1995, pp. 279–286.
9. Keim D.A., North S.C., Panse C., and Sips M. PixelMaps: a new visual data mining approach for analyzing large spatial data sets. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 565–568.
10. Keim D.A. and Oelke D. Literature fingerprinting: a new method for visual literary analysis. IEEE Symp. on Visual Analytics and Technology (VAST 2007), 2007.
11. Langton J.T., Prinz A.A., Wittenberg D.K., and Hickey T.J. Leveraging layout with dimensional stacking and pixelization to facilitate feature discovery and directed queries, Visual Information Expert Workshop (VIEW2006), 2006.
12. Tufte E.R. The Visual Display of Quantitative Information. Graphics Press, Cheshire, CT, 1983.
13. Tufte E.R. Envisioning Information. Graphics Press, Cheshire, CT, 1990.
14. Ziegler H., Nietzschmann T., Keim D.A. Relevance driven visualization of financial performance measures. In EuroVis 2007: Eurographics/IEEE-VGTC Symposium on Visualization, 2007, pp. 19–26.

Density-based Clustering

Martin Ester
Simon Fraser University, Burnaby, BC, Canada

Definition

Density-based clusters are dense areas in the data space separated from each other by sparser areas.

Furthermore, the density within the areas of noise is lower than the density in any of the clusters. Formalizing this intuition, for each *core point* the neighborhood of radius *Eps* has to contain at least *MinPts* points, i.e., the density in the neighborhood has to exceed some threshold. A point *q* is *directly-density-reachable* from a core point *p* if *q* is within the Eps-neighborhood of *p*, and *density-reachability* is given by the transitive closure of direct density-reachability. Two points *p* and *q* are called *density-connected* if there is a third point *o* from which both *p* and *q* are density-reachable. A *cluster* is then a set of density-connected points which is maximal with respect to density-reachability. *Noise* is defined as the set of points in the database not belonging to any of its clusters. The task of density-based clustering is to find all clusters with respect to parameters *Eps* and *MinPts* in a given database.

Historical Background

In the 1990s, increasingly large spatial databases became available in many applications. This drove not only the development of techniques for efficiently managing and querying these databases, it also provided great potential for data mining. The application of clustering algorithms to spatial databases raised the following requirements:

1. Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.
2. Good efficiency on large databases, i.e., on databases of significantly more than just a few thousand objects.
3. Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases.

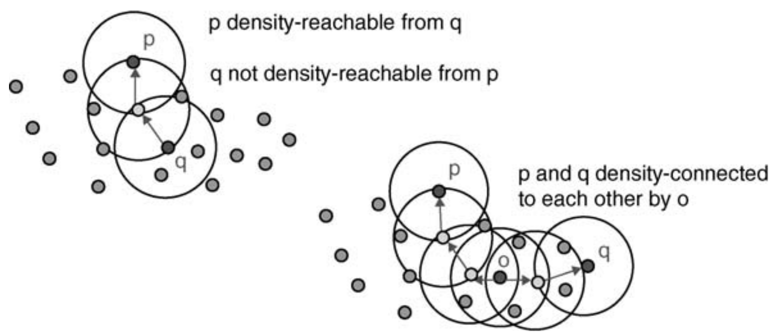
Due to the local nature of density-based clusters, this new clustering paradigm promised to address all of these requirements. As required, dense connected areas in the data space can have arbitrary shape. Given a spatial index structure that supports region queries, density-based clusters can be efficiently computed by performing at most one region query per database object. Different from clustering algorithms that optimize a certain objective function, the number of clusters does not need to be specified by the user.

Foundations

The paradigm of density-based clustering was introduced in [4]. Let *D* be a database of points. The definition of density-based clusters assumes a distance function *dist* (*p*, *q*) for pairs of points. The *Eps-neighborhood* of a point *p*, denoted by NEps(*p*), is defined by NEps (*p*) = {*q* ∈ *D* | *dist*(*p*, *q*)≤ *Eps*}. A point *p* is *directly density-reachable* from a point *q* w.r.t. Eps, MinPts if (1) p ∈ NEps(q) and (2) |NEps(q)|≥MinPts. A point p is *density-reachable* from a point q w.r.t. Eps and MinPts if there is a chain of points $p_1, \ldots, p_n$, $p_1 = q$, $p_n = p$ such that $p_i + 1$ is directly density-reachable from p_i. Density-reachability is a canonical extension of direct density-reachability. Since this relation is not transitive, another relation is introduced. A point *p* is *density-connected* to a point *q* w.r.t. *Eps* and *MinPts* if there is a point *o* such that both, *p* and *q* are density-reachable from *o* w.r.t. *Eps* and *MinPts*. Figure 1 illustrates these concepts.

Intuitively, a density-based cluster is a maximal set of density-connected points. Formally, a *cluster C* wrt. *Eps* and *MinPts* is a non-empty subset of *D* satisfying the following two conditions:

1. $\forall p, q$: if $p \in C$ and q is density-reachable from p w.r.t. *Eps* and *MinPts*, then $q \in C$. (maximality)


p density-reachable from q
q not density-reachable from p
p
q
p and q density-connected to each other by o
p
o
q

Density-based Clustering. Figure 1. Density-reachability and connectivity.

2. $\forall p, q \in C:$ p is density-connected to q w.r.t. *Eps* and *MinPts*. (connectivity)

Let C_1, ..., C_k be the clusters of the database *D* w.r.t. Eps and MinPts. The *noise* is defined as the set of points in *D* not belonging to any cluster C_i, i.e., $noise = \{p \in D | \forall i: \ p \notin C_i\}$.

Density-based clustering distinguishes three different types of points (see Fig. 2):

1. *Core points*, i.e., points with a dense neighborhood ($|N_{Eps}(p)| \geq MinPts$)
2. *Border points*, i.e., points that belong to a cluster, but whose neighborhood is not dense, and
3. *Noise points*, i.e., points which do not belong to any cluster

In the right part of Fig. 1, e.g., *o* is a core point, *p* and *q* are border points, and *n* is a noise point.

Density-based clusters have two important properties that allow their efficient computation. Let p be a core point in *D*. Then the set O = {o | o∈ *D* and o is density-reachable from p wrt. *Eps* and *MinPts*} is a cluster with regard to *Eps* and *MinPts*. Let *C* be a cluster in *D*. Each point in *C* is density-reachable from any of the core points of *C* and, therefore, a cluster *C* contains exactly the points which are density-reachable from an arbitrary core point of *C*. Thus, a cluster *C* with regard to *Eps* and *MinPts* is uniquely determined by *any* of its core points. This is the foundation of the DBSCAN algorithm for density-based clustering [5].

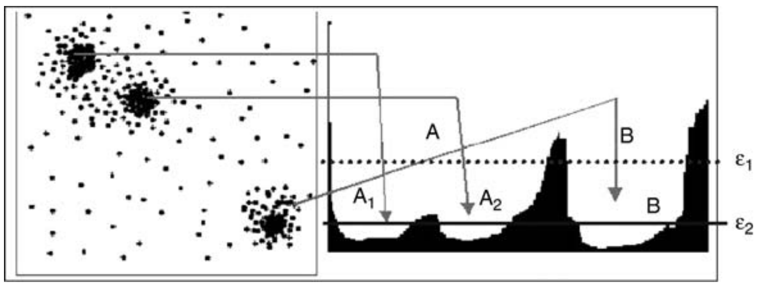
DBSCAN

To find a cluster, DBSCAN starts with an arbitrary database point *p* and retrieves all points density-reachable from p wrt. *Eps* and *MinPts*, performing region queries first for p and if necessary for *p*'s direct and indirect neighbors. If *p* is a core point, this procedure yields a cluster wrt. *Eps* and *MinPts*. If *p* is not a core point, no points are density-reachable from *p*. DBSCAN assigns *p* to the noise and applies the same procedure to the next database point. If *p* is actually a border point of some cluster *C*, it will later be reached when collecting all the points density-reachable from some core point of *C* and will then be (re-)assigned to *C*. The algorithm terminates when all points have been assigned to a cluster or to the noise.

In the worst case, DBSCAN performs one region query (retrieving the Eps-neighborhood) per database point. Assuming efficient index support for region queries, the runtime complexity of DBSCAN is $O(n \log n)$, where n denotes the number of database points. In the absence of efficient index structures, e.g., for high-dimensional databases, the runtime complexity is $O(n^2)$. Ester et al. [4] show that a density-based clustering can be updated incrementally without having to re-run the DBSCAN algorithm on the updated database. It examines which part of an existing clustering is affected by an update of the database and presents algorithms for incremental updates of a clustering after insertions and deletions. Due to the local nature of density-based clusters, the portion of affected database objects tends to be small which makes the incremental algorithm very efficient.

GDBSCAN

The basic idea of density-based clusters can be generalized in several important ways [8]. First, any notion of a neighborhood can be used instead of a distance-based *Eps*-neighborhood as long as the definition of the neighborhood is based on a predicate *NPred*(*p*, *q*) which is symmetric and reflexive. The neighborhood *N* of *p* is then defined as the set of all points *q* satisfying *NPred*(*p*, *q*). Second, instead of simply counting



Density-based Clustering. Figure 2. Reachability-plot for data set with hierarchical clusters of different sizes, densities and shapes.

the elements in a neighborhood, a more general predicate *MinWeight*(*N*) can be used to determine whether the neighborhood *N* is "dense," if *MinWeight* is monotone in *N*, i.e., if *MinWeight* is satisfied for all supersets of sets that satisfy *N*. Finally, not only point-like objects but also spatially extended objects such as polygons can be clustered. When clustering polygons, for example, the following predicates are more natural than the *Eps*-neighborhood and the *MinPts* cardinality constraint: "$NPred\ (X, Y)$ iff $intersect\ (X, Y)$" and "$MinWeight(N)$ iff $\sum_{P \in N} population(P) \geq MinPop$." The GDBSCAN algorithm [8] for finding generalized density-based clusters is a straightforward extension of the DBSCAN algorithm.

Denclue

Denclue [6] takes another approach to generalize the notion of density-based clusters, based on the concept of *influence functions* that mathematically model the influence of a data point in its neighborhood. Typical examples of influence functions are square wave functions or Gaussian functions. The overall *density* of the data space is computed as the sum of the influence functions of all data points. Clusters can then be determined by identifying *density-attractors*, i.e., local maxima of the overall density function. Most data points do not contribute to the density function at any given point of the data space. Therefore, the Denclue algorithm can be implemented efficiently by computing only a local density function, guaranteeing tight error bounds. A cell-based index structure allows the algorithm to scale to large and high-dimensional datasets.

OPTICS

In many real-life databases the intrinsic cluster structure cannot be characterized by *global* density parameters, and very different local densities may be needed to reveal clusters in different regions of the data space. In principle, one could apply a density-based clustering algorithm with different parameter settings, but there are an infinite number of possible parameter values. The basic idea of the OPTICS algorithm [2] to address this challenge is to produce a novel "cluster-ordering" of the database objects with respect to its density-based clustering structure containing the information about *every* clustering level of the data set (up to a "generating distance" *Eps*). This ordering is visualized graphically to support interactive analysis of the cluster structure.

For a constant *MinPts*-value, density-based clusters with respect to a higher density (i.e., a lower value for *Eps*) are completely contained in clusters with respect to a lower density (i.e., a higher value for *Eps*). Consequently, the DBSCAN algorithm could be extended to simultaneously cluster a database for several *Eps* values. However, objects which are density-reachable with respect to the lowest *Eps* value would always have to be processed first to guarantee that clusters with respect to higher density are finished first. OPTICS works in principle like such an extended DBSCAN algorithm for an infinite number of distance parameters Eps_i which are smaller than a "generating distance" *Eps*. The only difference is that it does not assign cluster memberships, but stores the order in which the objects are processed (the *clustering order*) and the following two pieces of information which would be used by an extended DBSCAN algorithm to assign cluster memberships. The *core-distance* of an object *p* is the smallest distance *Eps*' between *p* and an object in its *Eps*-neighborhood such that *p* would be a core object with respect to *Eps*' if this neighbor is contained in $N_{Eps}\ (p)$. The *reachability-distance* of an object *p* with respect to another object *o* is the smallest distance such that *p* is directly density-reachable from *o* if *o* is a core object. The clustering structure of a data set can be visualized by a *reachability plot* (see Fig. 2) that shows the reachability-distance values *r* for all objects sorted according to the clustering order. "Valleys" in the reachability plot correspond to clusters, which can be hierarchically nested. In Fig. 2, e.g., cluster A can be decomposed into subclusters A_1 and A_2.

Grid-Based Methods

Sheikholeslami et al. [9] present a grid-based approach to density-based clustering, viewing the dataset as a multi-dimensional signal and applying signal processing techniques. The input data is first discretized, and then a wavelet transform is applied to convert the discretized data into the frequency space, in which the natural clusters become more distinguishable. Clusters are finally identified as dense areas in the transformed space. A strength of the WaveCluster algorithm is that, due to the multi-resolution property of wavelet transforms, clusters can be discovered at multiple resolutions. The runtime complexity of WaveCluster is $O(n \cdot m^d)$, where *m* denotes the number of discrete intervals per dimension and *d* denotes the number of dimensions, i.e., it is linear in the dataset size and exponential in the number of dimensions.

In high-dimensional datasets, clusters tend to reside in lower-dimensional subspaces rather than in the full-dimensional space, which has motivated the task of subspace clustering. As a prominent algorithm for this task, the CLIQUE algorithm [1] discretizes the data space and defines a subspace cluster as a set of neighboring dense cells in an arbitrary subspace. Such subspace clusters can be efficiently discovered in a level-wise manner, starting with 1-dimensional clusters, and extending clusters by one dimension at every level. In CLIQUE, as in all grid-based approaches, the quality of the results crucially depends on the appropriate choice of the number and width of the grid cells. To address this problem, Hinneburg and Keim [7] suggest a method of contracting projections of the data space to determine the optimal cutting hyperplanes for partitioning the data. Dimensions are only partitioned if they have a good partitioning plane. It can be shown analytically that the OptiGrid algorithm finds all center-defined clusters, which roughly correspond to clusters generated by a Gaussian distribution.

Key Applications

Geographic Information Systems (GIS)

GIS manage spatial data including points, lines and polygons and support a broad range of applications. In geo-marketing, one may want to find clusters of homes with a given characteristic, e.g., high-income homes, while in crime analysis one of the goals is to detect crime hotspots, i.e., clusters of certain types of crimes. Density-based clustering is a natural choice in these applications.

Image Data

Clustering is an important tool to discover objects in image data. Clusters in such data can have non-spherical shapes and varying density, and they can be hierarchically nested. Density-based clustering has been successfully applied to generate landuse maps from satellite data and to detect celestial sources from astronomical images.

Future Directions

While the driving applications for density-based clustering were geographic information systems, recently other applications of density-based clustering have emerged, in particular in data streams and graph data, which create interesting challenges for future research. For density-based clustering of data streams, Cao et al. [3] introduces the concepts of core-micro-clusters, potential core-micro-clusters and outlier micro-clusters to maintain the relevant statistical information. A novel pruning strategy is designed based on these concepts, which allows accurate clustering in the context of limited memory. One of the major tasks in social network analysis is the discovery of communities, which can be understood as dense subnetworks. As the first algorithm adopting the paradigm of density-based clustering to networks, SCAN [10] efficiently detects not only communities, but also hubs and outliers, which play prominent roles in social networks.

URL to Code

The open source data mining software Weka includes Java implementations of DBSCAN and OPTICS, see http://www.cs.waikato.ac.nz/~ml/weka/index.html.

Cross-references

► Clustering Overview and Applications
► Indexing and Similarity Search

Recommended Reading

1. Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 94–105.
2. Ankerst M., Breunig M.M., Kriegel H-P., and Sander J. OPTICS: Ordering Points To Identify the Clustering Structure. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 49–60.
3. Cao F., Ester M., Qian W., and Zhou A. Density-based clustering over an evolving data stream with noise. In Proc. SIAM Conf. on Data Mining (SDM), 2006.
4. Ester M., Kriegel H-P., Sander J., Wimmer M., and Xu X. Incremental Clustering for Mining in a Data Warehousing Environment. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 323–333.
5. Ester M., Kriegel H-P., Sander J., and Xu X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, 1996, pp. 226–231.
6. Hinneburg A. and Keim D.A. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 58–65.
7. Hinneburg A. and Keim D.A. Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 506–517.

8. Sander J., Ester M., Kriegel H-P., and Xu X. Density-based clustering in spatial databases: the algorithm GDBSCAN and its applications. Data Min. Knowl. Discov., 2(2):169–194, 1998.
9. Sheikholeslami G., Chatterjee S., and Zhang A. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 428–439.
10. Xu X., Yuruk N., Feng Z., Thomas A., and Schweiger J. SCAN: a structural clustering algorithm for networks. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007, pp. 824–833.

Dependencies

► FOL Modeling of Integrity Constraints (Dependencies)

Derived Event

► Complex Event

Description Logics

Alexander Borgida
Rutgers University New Brunswick, NJ, USA

Synonyms

Terminologic languages; KL-ONE style languages; Concept languages

Definition

Description Logics (DLs) are a family of knowledge representation languages providing features for defining and describing concepts. The associated formal logics answer such questions as "*Is concept C or knowledge base T consistent?*" and "*Is concept A more specific (subsumed by) concept B* ?."

DLs view the world as being populated by individuals, grouped into classes ("concepts"), and related by binary relationships ("roles"). DLs define concepts recursively starting from atomic identifiers by using concept and role *constructors*. A key characteristic of every DL's expressiveness is therefore the set of constructors it supports. The collection of constructors considered has been determined empirically, by experience with a variety of tasks in Natural Language processing and other subfields of Artificial Intelligence. Considerable research has been devoted to finding the complexity of reasoning with various DLs. The result is a family of logics that is intermediate in expressive power between propositional logic and first order logic, with emphasis on decidable fragments.

DLs are particularly useful in the field of databases for conceptual modeling and the specification of ontologies used for information integration or in the Semantic Web.

Historical Background

Semantic Networks and Frame Systems, as originally used for representing knowledge in Artificial Intelligence, were faulted for lack of precise semantics, and Brachman [4] was the first to suggest a (graphical) notation, called KL-ONE, for structured networks based on so-called "epistemologic primitives," with a precise intended meaning. Brachman and Levesque [5] introduced the more logical compositional syntax and semantics for a restricted DL, and considered for the first time the famous expressiveness vs. complexity trade-off: more expressive concept constructors often require provably harder reasoning.

DLs have recently attracted considerable attention since they form the basis of the OWL web ontology language, and there are several widely available implementations.

The *Description Logic Handbook* [1] is the definitive reference book on this topic, covering the theory, implementation and application of Description Logics.

Foundations

Concept and Role Constructors

Consider the concept *"objects that are related by locatedIn to some City, an d by ownedBy to some Person"*. Much like one can build a complex type in a programming language by using type constructors, or a complex boolean formula using connectives that can be viewed as term constructors (e.g., $p \wedge \neg q$ written as the term **and**(p,**not**(q))), so in a DL one could specify this concept using a term-like syntax

$$\mathbf{and}(\mathbf{some}(locatedIn, City), \mathbf{some}\,(ownedBy, Person))$$

The conventional notation used for this in the research literature is actually

$$(\exists locatedIn.City \sqcap \exists ownedBy.Person)$$

where $\exists r.\, A$ denotes all object with at least one *r*-role filler being in concept *A*, and $\sqcap$ intersects the set of instances of its argument concepts. A variety of other syntactic notations for complex concepts exist, including one for the OWL web ontology language, based on XML.

Other frequently used concept constructors include: so-called "boolean connectives" for unioning $\sqcup$, or complementing $\neg$ concepts; value restrictions $\forall p.C$, as in $\forall locatedIn.SmallTown$ denoting individuals related by *locatedIn* only to instances of *SmallTown*; cardinality constraints such as $\geq n\, p.C$, as in $\geq 2\, ownedBy.RichPerson$ denoting objects that are owned by at least two instances of concept *RichPerson*; enumerations $\{J_1, J_2, \ldots\}$, as in $\{ROMULUS, REMUS\}$ denoting the set of these two individuals. Other constructors can often be defined in terms of these e.g., objects with RED hair color value $(hair\_color : RED)$ is just $\exists hair\_color.\{RED\}$

Useful role constructors include: role inversion r^-, as in $ownedBy^-$ denoting what might be called the "ownerOf" relationship; role restriction $r|_C$, as in $childOf|_{Male}$ denoting what might be called "sonOf"; and role composition , as in $(childOf \circ childOf)$, denoting what might be called "grandchildOf."

Of particular utility for database applications are concept constructors that concern role fillers that are "concrete types" such as integers and strings (e.g., $\forall age.min(15)$ denotes objects whose age value is at least 15), and those that select individuals for whom combinations of roles act as unique identifiers ("keys").

Summarizing, the following properties are notable and distinguishing features of DLs as far as concept constructors:

- Although concepts resemble programming language record types, roles are multi-valued, in contrast to record fields; e.g., a house may be *ownedBy* by more than person.
- Although symbols like $\forall$ and $\exists$ are used in the mathematical notation of DLs, there are in fact *no variables* (free or bound) appearing in concept specifications.
- Complex concepts can be constructed by nesting, as in $\exists ownedBy.(Person \sqcap (\geq 3\, childOf^-.Teenager))$, with intended meaning "objects which are owned by a person that is the parent of at least three teenagers"; and they need not be given names, acting as complex noun phrases with relative clauses.

Formal Semantics

The precise meaning/denotation of concepts is usually specified by an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, which assigns to every concept name a subset of the domain $\Delta^{\mathcal{I}}$, to every role name a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to every individual name some object in $\Delta^{\mathcal{I}}$. Table 1 shows how this is extended from atomic identifiers to complex terms for some of the concept and role constructors. (The others can be defined from these using complement.) It is often possible to achieve the same effect indirectly by providing a translation from concepts/roles to Predicate Calculus formulas with one/two free variables.

Using Concepts in Knowledge Bases

Concepts and roles do not actually make any assertions – they are like formulas with free variables. The following are some of the possible uses for them.

First, one can make assertions about *necessary properties* of atomic concepts, e.g., "every house is located in some city and is owned by some person," by using a syntax like

Description Logics. Table 1. Semantics of composite concepts and roles

Term	Interpretation	Translation to FOPC
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$C(x) \wedge D(x)$
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	$\neg C(x)$
$\exists p.C$	$\{\delta \in \Delta^{\mathcal{I}} \mid p^{\mathcal{I}}(\delta) \cap C^{\mathcal{I}} \neq \emptyset\}$	$\exists y\ p(x,y) \wedge C(y)$
$\geq np.C$	$\{\delta \in \Delta^{\mathcal{I}} \mid \lvert p^{\mathcal{I}}(\delta) \cap C^{\mathcal{I}} \rvert \geq n\}$	$\exists z_1, \ldots, \exists z_n\ p(x,z_1) \wedge \ldots \wedge p(x,z_n) \wedge z_i \neq z_j \wedge C(z_i)$
$\{b_1, \ldots, b_m\}$	$\{b_1^{\mathcal{I}}, \ldots, b_m^{\mathcal{I}}\}$	$x = b_1 \vee \ldots \vee x = b_m$
p^-	$\{(\delta, \delta') \mid (\delta', \delta) \in R^{\mathcal{I}}\}$	$p(y,x)$
$p\vert_C$	$\{(\delta, \delta') \in p^{\mathcal{I}} \mid \delta' \in C^{\mathcal{I}}\}$	$p(x,y) \wedge C(y)$
$p \circ q$	$p^{\mathcal{I}} \circ q^{\mathcal{I}}$	$\exists z\ p(x,z) \wedge q(z,y)$

$$House \sqsubseteq (\exists\ locatedIn.City \sqcap \exists\ ownedBy.Person)$$

where $\sqsubseteq$ is read as "is subsumed by." An assertion such as $A \sqsubseteq B$ is satisfied only in interpretations where $A^{\mathcal{I}}$ is a subset of $B^{\mathcal{I}}$.

Second, one can *give definitions* to atomic names; e.g., "a French house" can be defined to be a house located in a French city, using the syntax

$$FrenchHouse \equiv (House \sqcap \forall locatedIn.FrenchCity)$$

For database *cognoscenti*, definitions will be familiar as views.

If one allows *general inclusion axioms* of the form $\alpha \sqsubseteq \beta$, the definition above is equivalent to two statements

$$FrenchHouse \sqsubseteq (House \sqcap \forall locatedIn.FrenchCity)$$
$$(House \sqcap \forall locatedIn.FrenchCity) \sqsubseteq FrenchHouse$$

Such assertions about concepts form part of the *terminology* of a domain, and hence are gathered in a so-called *TBox* $\mathcal{T}$. Other assertions in a TBox may specify properties of roles, such as role inclusion (e.g., *ownedBy* $\sqsubseteq$ *canSell*) or whether a role is transitive.

Given a terminology $\mathcal{T}$, one can then ask whether some subsumption $\alpha \sqsubseteq \beta$ follows from it, written as $\mathcal{T} \models \alpha \sqsubseteq \beta$, i.e., whether the subsumption holds in all interpretations satisfying $\mathcal{T}$. This is the basis of a number of standard services that are provided by a DL reasoner, such as automatically classifying all identifiers into a subsumption taxonomy, or checking whether concepts or the entire knowledge base are (in)coherent. Other, non-standard but useful reasoning services include computing the least common subsumer of a pair of concepts (a non-trivial task if one excludes concept union $\sqcup$ as a constructor), rewriting concepts using definitions in a TBox for goals such as abbreviation, and pinpointing errors/reasoning in inconsistent terminologies.

In order to describe specific states of a world, one uses formulas of two forms: (i) *LYON* : *City*, expressing that the individual *LYON* is an instance of concept *City*; and (ii) *locatedIn*(*LYON*, *EUROPE*), indicating that *LYON* and *EUROPE* are related by the *locatedIn* relationship. A collection $\mathcal{A}$ of such assertions about the state of the world is called an *ABox*. A reasoner can then determine whether a particular concept membership (resp. role relationship) follows from an (ABox, TBox)-pair: $(\mathcal{A}, \mathcal{T}) \models d : \beta$ (resp. $(\mathcal{A}, \mathcal{T}) \models r(d, e)$). For example, from the above facts about *LYON*, and the definition *EuropeanCity* $\equiv$ (*City* $\sqcap$ *locatedIn* : EUROPE), one can conclude that *LYON* : *EuropeanCity*. Based on the above judgment, DL reasoners can determine the (in)consistency of an ABox $\mathcal{A}$ with respect to a TBox $\mathcal{T}$, and can classify individuals in $\mathcal{A}$ under the most specific concept in the taxonomy. It is important to note that, unlike standard databases, DLs do *not* adopt the so-called "closed-world assumption," so that from the above facts one cannot conclude that LYON is located in only one place; i.e., to obtain this conclusion, (*LYON*:≤ 1*locatedIn*) needs to be explicitly added to the previous collection of assertions.

Mathematical Properties

The formal complexity of reasoning with a DL-based knowledge base is determined by the concept and role constructors allowed, and the form of the axioms (e.g., Are axioms permitted to create cyclic dependencies among concept names? Are there role subsumption axioms?). Also, interesting connections have been found between DLs and epistemic propositional logic, propositional dynamic logic, and various restricted subsets of First Order Logic. Many of these results have been summarized in [1].

Implementations

Unlike the earliest implemented systems (KL-ONE and LOOM), the second generation of systems (BACK, KANDOR, and CLASSIC), took seriously the notion of computing *all* inferences with respect to some semantic specification of the DL, and in fact tried to stay close to polynomial time algorithms for judgments such as subsumption, often by finding a normal form for concepts. In contrast, the DLs implemented in the past decade have been considerably more expressive (theoretically, the worst-case complexity of deciding subsumption ranges from PSPACE to non-deterministic exponential space complete), and are implemented using so-called "tableaux techniques," which have been highly optimized so that performance on practical KBs is acceptable.

Key Applications

Conceptual Modeling

As knowledge representation languages, with concept constructors that have been found to be empirically useful for capturing real-world domains of discourse, DLs are relevant to databases as conceptual modeling languages, used in the first step of database design; and for database integration, as the mediated conceptual schema seen by external users. The advantages of DLs over other notations such as Extended ER diagrams and UML is their greater expressive power and the existence of implemented reasoners that are provably correct, and that can, for example, detect inconsistencies in domain models. In fact, it has been shown that EER and UML class diagrams can be translated into the $\mathcal{SHIQ}$ DL [2,6], for which multiple implementations exist currently.

Ontology Specification

DLs have been adopted as the core (OWL-DL) of the OWL web ontology language proposed by W3C [8]. Ontologies, and reasoning with them, are predicted to play a key role in the Semantic Web, which should support not just data integration but also web service composition, orchestration, etc.

Management and Querying of Incomplete Information

Because of open-world reasoning, and the ability to make assertions about objects not explicitly present in the knowledge base (e.g., $VLDB09 : \forall takesPlaceIn.(City \sqcap locatedIn : ASIA \sqcap \forall hasPopulation.min(1000000))$ says that wherever *VLDB*09 will be held, the city will be in Asia, and it will have population over 1 million), DLs are particularly well suited to represent and reason with partial knowledge [3]. Among others, DLs face and resolve aspects of the view update problem. Industrial applications of this have been found in configuration management. Research has also addressed issues related to more complex queries, with a flurry of recent results concerning conjunctive query answering over ABoxes, starting with [7].

Query Organization and Optimization

Since a complex concept C can be used to return all individuals b classified under it (i.e., $b : C$), C can be thought of as a query, though one of somewhat limited power since it cannot express all conditions that might be stated in SQL, nor can it return sets of tuples. In exchange, query containment (which is just subsumption) is decidable, and sometimes efficiently computable. This can be used to organize queries, and help answer queries using results of previous queries.

In addition, DL-based conceptual models of database schemata can be used to enable advanced semantic query optimization, in particular when combined with DL-based description of the physical layout of data. In this setting, identification constraints, such as keys and functional dependencies, are often added to DLs [9] and subsequently used to enable rewriting of queries for purposes such as removing the need for duplicate elimination in query answers.

Future Directions

In addition to continued work on extending the expressive power of DLs, and various forms of implementations, important topics of current concern include (i) representation and reasoning about actions, (ii) extracting modules from and otherwise combining DL ontologies, (iii) restricted expressive power DLs that are tractable or of low complexity yet can handle important practical ontologies in the life sciences and conceptual models, (iv) combining rules with DLs, and, (v) conjunctive query processing over DL ABoxes.

URL to Code

http://www.dl.kr.org/ The Description Logics web site. Among others, look for links to the Description Logic Workshops (containing the most up-to-dateresearch in the field), the "navigator" for the complexity of reasoning with variousDLs, and for DL implementations.

Cross-references

► Conceptual Modeling
► Data Integration
► Semantic Web

Recommended Reading

1. Baader F., Calvanese D., McGuinness D.L., Nardi D., and Patel-Schneider P.F. The Description Logic Handbook: Theory, Implementation, and Applications, 2nd edn. Cambridge University Press, 2003.
2. Berardi D., Calvanese D., and De Giacomo G. Reasoning on UML class diagrams. Artif. Intell. J., 168(1-2): 70–118, 2005.
3. Borgida A. Description logics in data management. IEEE Trans. Knowl. Data Eng., 7(5): 671–682, 1995.
4. Brachman R.J. What's in a concept: structural foundations for semantic networks. Int. J. Man-Machine Studies, 9(2): 127–152, 1997.
5. Brachman R.J. and Levesque H.J. The tractability of subsumption in frame-based description languages. In Proc. AAAI. 1984, pp. 34–37.
6. Calvanese D., Lenzerini M., and Nardi D. Unifying class-based representation formalisms. J. Artif. Intell. Res. (JAIR), 11199–240, 1999.
7. Horrocks I. and Tessaris S. A conjunctive query language for description logic aboxes. In Proc. 12th National Conf. on AI, 2000, pp. 399–404.

8. OWL Web Ontology Language Reference, http://www.w3.org/TR/owl-ref/
9. Toman D. and Weddell G. On keys and functional dependencies as first-class citizens in description logics. J. Auto. Reason., 40(2–3): 117–132, 2008.

Design for Data Quality

Carlo Batini, Andrea Maurino
University of Milan Bicocca, Milan, Italy

Synonyms

Schema normalization; Design for quality

Definition

The design for data quality (DQ) is the process of designing data artifacts, such as information systems, databases, and data warehouses where data quality issues are considered relevant.

In information systems different types of data are managed; these may be structured such as relational tables in databases, semi-structured data such as XML documents, and unstructured data such as textual documents. Information manufacturing can be seen as the processing system acting on raw data of different types, whose aim is to produce information products. According to this approach, the design for data quality aims to design information-related processes (e.g., creation, updating, and delivering of information) taking into account data quality dimensions.

In the database field, the design for data quality has the objective of producing good (with respect to a given set of quality dimensions) conceptual and relational schemas and corresponding good data values. Concerning schema design, the most important quality dimensions are:

- C*orrectness with respect to the model* concerns the correct use of the categories of the model in representing requirements
- *Correctness with respect to requirements* concerns the correct representation of the requirements in terms of the model categories
- *Minimality* requires that every part of the requirements is represented only once in the schema
- *Completeness* measures the extent to which a conceptual schema includes all the conceptual elements necessary to meet some specified requirement
- *Pertinence* measures how many unneeded conceptual elements are included in the conceptual schema
- *Readability* imposes that the schema is expressed in a clear way for the intended usage

It is worth noting that the quality of data is strongly influenced by the quality of the schema that dictates the structure of these data. Consequently, the design for data quality can be seen as the first activity in a DQ management program. In fact, let consider a schema composed of the two relations: Movie(Title, Director, Year, DirectorDateOfBirth) and Director(Name, Surname, DateOfBirth). The schema does not satisfy the minimality dimension due to the duplication of the DateOfBirth attribute. This design error will reflect on the data values that could suffer from consistency and accuracy problems.

The design for data quality is a relevant problem also for data warehouse systems. The data warehouse design process has to consider several dimensions. The *completeness* dimension is concerned with the preservation of all the entities in data sources in the data warehouse schema. The *minimality* dimension describes the degree up to which undesired redundancy is avoided during the source integration process. The *traceability* dimension is concerned with the fact that all kinds of data processing of users, designers, administrators and managers should be traceable in the data warehouse system. The *interpretability* dimension ensures that all components of the data warehouse schema are well described.

Historical Background

The problem of designing for data quality was considered relevant to the area of information systems since the late 1980s. The total data quality management methodology (TDQM) [11] introduces the information product (IP) approach, which considers information as one of the products that an organization produces. As a consequence, traditional manufacturing design for quality strategies can be applied to IPs also. Within the TDQM methodology, the IP conceptual definition phase is in charge of defining an Entity-Relationship (ER) schema enhanced with quality features, which defines the IP, its information quality (IQ) requirements, an information manufacturing system that describes how the IP is produced, and the interactions among information suppliers (vendors), manufacturers,

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:43

consumers, and IP managers. In this way, many IQ requirements can be enforced into the new information manufacturing system, resulting in design for information quality procedures analogous to that of design for quality in product manufacturing.

From a historical viewpoint, early methodologies for database design did not consider data quality issues as a relevant problem. Subsequently, design methodologies incorporated DQ issues by extending conceptual and logical models with quality features. Several solutions have been proposed for extending the Entity-Relationship model with quality characteristics. In [8], a methodology for modeling the quality of attribute values as another attribute of the same entity is described. A different approach is proposed in the Quality Entity-Relationship (QER) model [9], which extends the ER model by providing a mechanism to embed quality indicators into conceptual schemas. QER introduces two generic entities, DQ_Dimension and DQ_Measure. DQ_Dimension, with attributes Name and Rating, models all possible quality dimensions (e.g., accuracy) for an attribute (e.g., Address) and its values (e.g., accuracy = "0.8"). The entity DQ_Measure is used to represent metrics for corresponding data quality measurement values (e.g., in a [0,1] scale, "0" for very inaccurate, "1" for very accurate). For what concerns logical database models, [13] presents an attribute based model that extends the relational model by adding an arbitrary number of underlying quality indicator levels to attributes. Attributes of a relational schema are expanded into ordered pairs, called quality attributes, consisting of the attribute and a quality key. The quality key is a reference to the underlying quality indicator(s). Other extensions of the relational model are presented in [14,15]. Studies on normal forms, and more in general on the normalization process provide techniques for avoiding certain anomalies such as duplication of values or updates. These techniques can be considered as evaluation techniques for measuring the quality of logical schemas.

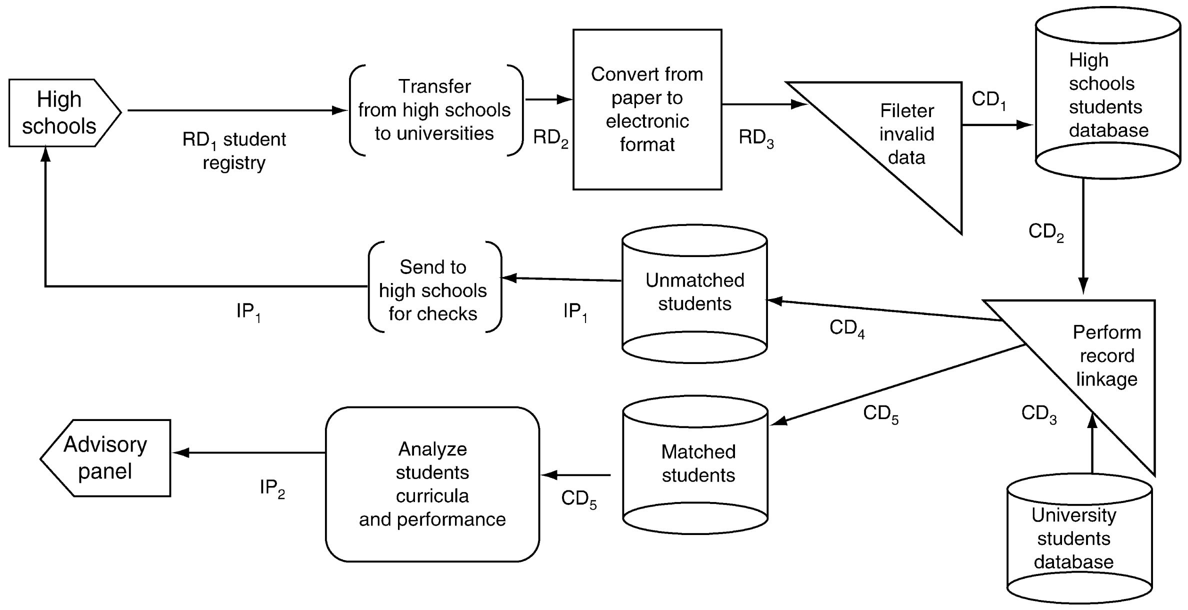
More recently, [5] presents a new design for data quality methodology for incorporating data quality requirements into database schemas by means of goal-oriented requirement analysis techniques. The proposal extends existing approaches for addressing data quality issues during database design in two ways. First, authors consider data quality requirements (including descriptive, supportive and reflective requirements). Second, a systematic way to move from high-level, abstract quality goals into quality assurance requirements is presented.

For what concerns data warehouse design, research on designing schemas with quality properties is limited. One of the most important results in this field is the DWQ framework proposed in [3], where a general-purpose model of quality has been defined to capture the set of quality factors associated to the various data warehouse objects, and to perform their measurement for specific quality goals. In particular, it is shown that the refreshment process and the selection of the materialized views are demonstrative examples where quality can drive the design process [10].

Foundations

In the design of information systems the term *information quality* refers to processes involving the information life-cycle (creation, updating, and delivering of information). In particular, two important quality dimensions can be defined: *efficiency*, which measures process by comparing the production with costs and resources, and *effectiveness*, which measures the process outcome; namely the real result of the process for which the process has been conceived. The most popular model for designing quality information systems is IP-MAP [7].

IP-MAP is a graphical model designed to help people to comprehend, evaluate, and describe how an information product such as an invoice, a customer order, or a prescription is assembled in a business process. The IP-MAP is aimed at creating a systematic representation for capturing the details associated with the manufacturing of an information product. IP-MAP models (hereafter IP-MAPs) are designed to help analysts visualize the information production process, identify ownership of the process phases, understand information and organizational boundaries, and estimate time and quality metrics associated with the production process to be considered. Figure 1 shows an example of IP-MAP representing high schools and universities of a district that have agreed to cooperate in order to improve their course offering to students, avoiding overlapping and becoming more effective in the education value chain. To this end, high schools and universities have to share historical data on students and their curricula. Therefore, they perform a record linkage activity that matches students in their education life cycle. To reach this objective, high


High schools
RD1 student registry
Transfer from high schools to universities
RD2
Convert from paper to electronic format
RD3
Fileter invalid data
CD1
High schools students database
CD2
Perform record linkage
CD3
University students database
CD4
Unmatched students
IP1
Send to high schools for checks
IP1
CD5
Matched students
CD5
Analyze students curricula and performance
IP2
Advisory panel

Design for Data Quality. Figure 1. Example of IP-MAP.

schools periodically supply relevant information on students; in case it is in paper format the information has to be converted into electronic format. At this point, invalid data are filtered and matched with the database of university students. Unmatched student records are sent back to high schools for clerical checks, and matched student records are analyzed; the results of the analysis on curricula and course topics are sent to the advisory panel of universities.

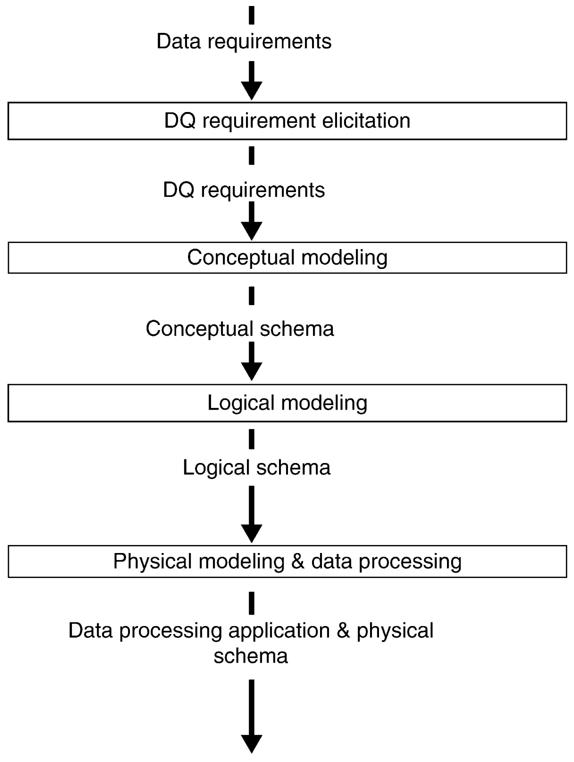
Typical activities for database design that are influenced by data quality include (see Fig. 2):

1. *DQ requirements elicitation*
2. *Conceptual modeling*
3. *Logical modeling*
4. *Data processing and Physical modeling*

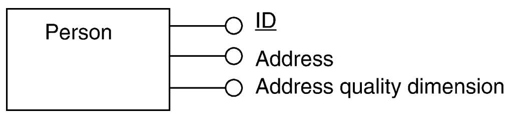
The process starts with the data quality requirements elicitation. Here there is a similarity between software development processes and the design for data quality in databases. In fact, in software development processes, functional requirements describe what the software does, while the non-functional properties describe qualities that functionalities need to support. In the design for data quality process, *data requirements* describe the Universe of Discourse that the design has to represent in the database, while *data quality requirements* model quality dimensions by which a user evaluates DQ and quality processes related to data acquisition, and update. Furthermore, quality attributes are considered at two levels: quality parameters, model quality dimensions by which a user evaluates DQ (e.g., accuracy and currency); quality indicators capture aspects of the data manufacturing process (e.g., when, where, how data is produced) and provide information about the expected quality of the data produced. The data quality requirements elicitation phase produces quality requirements that are inputs to the conceptual modeling phase.

During the conceptual modeling phase, concepts and their attributes are elicited and organized into a conceptual schema. Moreover, quality parameters are identified and associated to attributes in the schema. Finally, each parameter is refined into one or more quality indicators. There are two possible design choices. A first possibility is to model the quality of attribute values as another attribute of the same entity for each attribute [8,9]. As an example, in order to add a quality dimension (e.g., accuracy or completeness) for the attribute Address of an entity Person, it is possible to add (see Fig. 3) a new attribute Address-QualityDimension to the entity.

The drawback of this solution is that now the entity is no longer normalized. Another drawback is that if there is the need to define several dimensions, a new attribute for each dimension has to be added, resulting in a proliferation of attributes. The second possibility


Data requirements
DQ requirement elicitation
DQ requirements
Conceptual modeling
Conceptual schema
Logical modeling
Logical schema
Physical modeling & data processing
Data processing application & physical schema

Design for Data Quality. Figure 2. Design for data quality process.



Design for Data Quality. Figure 3. Representing quality dimensions in the entity-relationship model.

is to add specified data quality entities, and to create a many-to-many relationship among these entities and application entities.

The conceptual schema is the input of the next phase, concerned with logical modeling. In the process of translation from the conceptual schema to the logical schema, normalization has to be achieved. Normal forms guarantee the absence of data processing anomalies; as such normal forms are a relevant concept in design for data quality. The same holds true for intra-relational and inter-relational integrity constraints defined in the logical modeling phase; integrity constraints are the fundamental means to guarantee the consistency dimension for data.

The logical schema is an input for the physical modeling and data processing phase. Data processing activities are one of the most critical tasks from a data quality perspective. A careful design of the data processing procedures can significantly reduce data quality problems. For example, if the domain of an attribute is composed of a fixed set of values (e.g., the educational degrees), the use of predefined lists to insert values in attributes reduce possible typos and increases the syntactic accuracy.

The design of distributed databases especially when independently developed databases are considered, raises more challenges in the design for data quality. In this case, in fact, existing schemas cannot be modified and issues related to the quality of schema cannot be dealt with. The only possibility to partially work out the quality issues is the definition of effective schema integration techniques. Schema level conflicts include: i) *heterogeneity conflicts*, occurring when different data models are used; ii) *semantic conflicts*, regarding the relationship between model element extensions, iii) *description conflicts*, concerning concepts with different attributes, and iv) *structural conflicts*, regarding different design choices within the same data model. Instance level conflicts are another typical problem of distributed databases and occur when conflicting data values are provided by distinct sources for representing the same objects. At design time, it is possible to plan conflict resolution selecting suitable aggregation functions [2]; such functions take two (or more) conflicting values of an attribute as input and produce a value as output that must be returned as the result of the posed query. Common resolution functions are MIN, MAX, AVG. Other techniques are discussed and compared in [1]. Techniques for instance level conflict resolution at design time have a major optimization problem. Consider two relations EmployeeS1 and EmployeeS2 defined in two different databases, representing information about employees of a company. Also assume that no schema conflict is detected. Suppose that the two relations have instance level conflicts for the Salary attribute. Moreover, suppose that at design time it is specified that in case of conflicts the minimum salary must be chosen. Given the global schema, Employee (EmployeeID, Name, Surname, Salary, Email), let consider the following query:

```
SELECT EmployeeID, Email
FROM Employee
WHERE Salary < 2000
```

Since the Salary attribute is involved in the query, all employees must be retrieved in order to compute the minimum salary, not only employees with Salary < 2000, even if no conflicts on salary occur. Therefore conflict resolution at design time may be very inefficient.

Future Directions

The problem of measuring and improving the quality of data has been dealt with in the literature as an activity to be performed a posteriori, instead of during the design of the data. As a consequence, the design for data quality is still a largely unexplored area. In particular, while the data quality elicitation and the translation of DQ requirement into conceptual schemas have been investigated; there is a lack of comprehensive methodologies covering all the phases of design of information systems/databases/data warehouse systems. Moreover, easy-to-use and effective design tools could help the database designer.

Open areas of research concern the definition of quality information dimensions for semi-structured and unstructured data. In particular, while the use of XML technologies is growing within organizations, the research on design of the XML quality schemas is at an early stage.

Cross-references

► Data Profiling
► Data Provenance
► Data Quality Assessment
► Information Quality
► Quality Data Management
► Quality in Data Warehouses

Recommended Reading

1. Batini C. and Scannapieco M. Data Quality: Concepts, Methodologies and Techniques, Springer, New York, 2006.
2. Dayal U. Query processing in a multidatabase system. In Query Processing in Database Systems, W. Kim, D.S. Reiner, D.S. Batory (eds.). Springer, 1985, pp. 81–108.
3. Jarke M., Jeusfeld M.A., Quix C., and Vassiliadis P. Architecture and quality in data warehouses: an extended repository approach. Inf. Syst. 24(3):229–253, 1999.
4. Jeusfeld M.A., Quix C., and Jarke M. Design and analysis of quality information for data warehouses. In Proc. 17th Int. Conf. on Conceptual Modeling, 1998, pp. 349–362.
5. Jiang L., Borgida A., Topaloglou T., and Mylopoulos J. Data quality by design: a goal-oriented approach. In Proc. 12th Conf. on Information Quality, 2007.
6. Navathe S.B., Evolution of data modeling for databases. Commn. ACM, 35(9):112–123, 1992.
7. Shankaranarayanan G., Wang R.Y., and Ziad M. IP-MAP: representing the manufacture of an information product. In Proc. 2000 Conf. on Information Quality, 2000.
8. Storey V. and Wang R.Y. Extending the ER model to represent data quality requirements. In Data Quality, R. Wang, M. Ziad, W. Lee (eds.). Kluwer Academic, Boston, MA, 2001.
9. Storey V.C. and Wang R.Y. Modeling quality requirements in conceptual database design. In Proc. Third Conf. on Information Quality, 1998, pp. 64–87.
10. Vassiliadis P., Bouzeghoub M., and Quix C. Towards quality-oriented data warehouse usage and evolution. In Proc. 11th Conf. on Advanced Information Systems Engineering (CAiSE'99), LNCS, vol. 1626, 1999, pp. 164–179.
11. Wang R.Y. A product perspective on total data quality management. Commn. ACM, 41(2):58–65, 1998.
12. Wang R.Y., Kon H.B., and Madnick S.E. Data quality requirements analysis and modeling. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 670–677.
13. Wang R.Y., Reddy M.P., and Kon H.B. Toward quality data: an attribute-based approach. Decision Support Syst., 13 (3–4):349–372, 1995.
14. Wang R.Y., Storey V.C., and Firth C.P. A framework for analysis of data quality research. IEEE Trans. Knowl. Data Eng., 7 (4):623–640, 1995.
15. Wang R.Y., Ziad M., and Lee Y.W. Data Quality. Kluwer Academic, Boston, MA, 2001.

Design for Quality

► Design for Data Quality

Desktop Metaphor

► Direct Manipulation

Detail-in-Context

► Distortion Techniques

Deviation from Randomness

► Divergence from Randomness Models

Dewey Decimal Classification

► Dewey Decimal System

Dewey Decimal System

Prasenjit Mitra
Pennsylvannia State University, University Park, PA, USA

Synonyms

Dewey decimal classification

Definition

The Dewey Decimal Classification (DDC) System is a system primarily used in libraries to classify books. In general, the system claims to provide a set of categories for all human knowledge. The system consists of a hierarchy of classes. At the top level, there are ten main classes, that are divided into 100 divisions which are sub-divided into 1,000 sections. The system was conceived by Melvil Dewey in 1873 and published in 1876. DDC uses Arabic numerals to number the classes and explicates the semantics of a class and its relation to other classes.

Key Points

Since its first publication in 1876 by Melvil Dewey [2], the Dewey Decimal Classification (DDC) has been updated to accommodate changes to the body of human knowledge. The current version, DDC 22, was published in mid-2003 [3] (http://www.oclc.org/dewey/versions/ddc22print/). Currently, the Online Computer Library Center (OCLC) (http://www.oclc.org) of Dublin Ohio owns the copyrights associated with the Dewey Decimal system. Each new book is assigned a DDC number by the Library of Congress. As of the date this article was written, the OCLC has accepted all assignments of DDC numbers made by the Library of Congress. The OCLC claims that libraries in more than 135 countries use the DDC and it has been translated to over 30 languages (http://www.oclc.org/dewey/versions/ddc22print/intro.pdf).

The DDC has ten main classes: *Computer Science, information and general works; Philosophy and psychology; Religion; Social Sciences; Language; Science; Technology; Arts and recreation; Literature; and History and geography* [1]. Consequently, all fiction books fall under the category Literature (800). To avoid a large number of rows in the 800 range, libraries separately stack non-fiction and fiction books in different sections. To allow for further subdivision of classes, the DDC number for a class can contain further subdivisions of ten after the three digit number. After the three digits for a class number, the classes can be further subdivided. While determining the subject of a work, the editor, at the Dewey editorial office at the Decimal Classification Division of the Library of Congress, tries to determine the author's intent. The entire content of the book is taken into account along with reviews, reference works, and opinions of subject experts.

The DDC forms the basis for the more expressive Universal Decimal Classification. Alternatives to the DDC are the Library of Congress Classification. The construction of the DDC was top down (designed by one person, Dewey) and is somewhat inflexible to accommodating changes in human knowledge. In contrast, the Library of Congress Classification has 21 classes at the top-level of the hierarchy and was developed by domain experts in a bottom-up fashion. The simple numbering system of the DDC makes it easy to use.

Cross-references

► Library of Congress Classification

Recommended Reading

1. Chan L.M. Dewey Decimal Classification: A Practical Guide. Forest, 1994.
2. Dewey M. Dewey; Decimal classification and relative index for libraries, clippings, notes. Library Bureau, 1891.
3. Mitchell J.S. Summaries DDC 22: Dewey Decimal Classification. OCLC, 2003.

DHT

► Distributed Hash Table

Diagram

Carlo Batini
University of Milan Bicocca, Milan, Italy

Synonyms

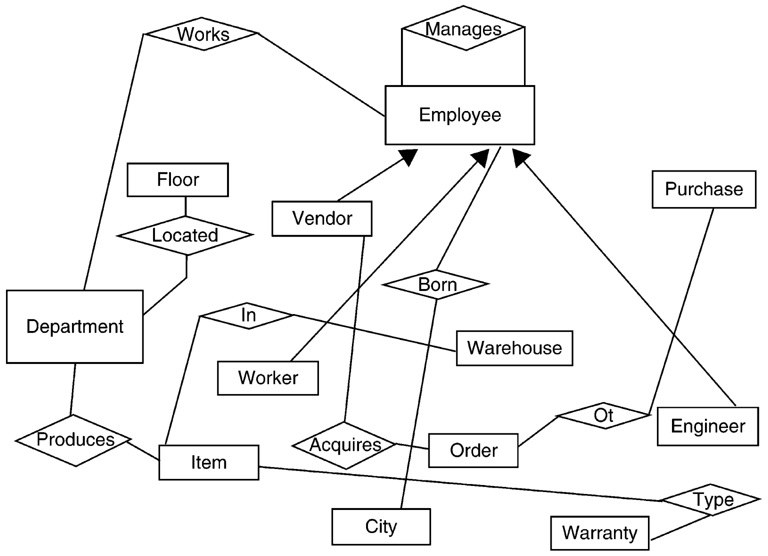
Diagram; Diagrammatic Representation; Graph; Graphic

Definition

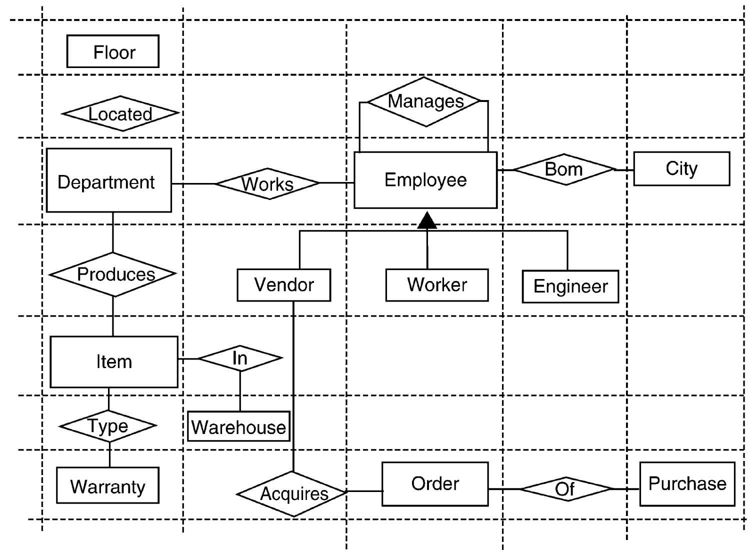
A *diagram* is (i) a set of *symbols* selected in an alphabet of *elementary shapes*, such as rectangles, circles, and *complex shapes*, built by composing elementary shapes, and (ii) a set of *connections* linking symbols, such as straight lines, curved lines, directed lines, etc. Diagrams are used to visualize in all aspects of data base design and usage a wide set of concepts, artifacts, schemas, values, such as conceptual schemas, logical schemas, UML diagrams, database instances, queries, results of queries.

The visual representation made possible by a diagram expresses a functional relationship between concepts represented and symbols/connections representing them. e.g. in the diagrammatic representation of Entity Relationship schemas, a rectangle is associated to an entity, a diamond is associated to a relationship.

When drawing a diagram, human beings and display devices adopt suitable *drawing conventions*, *aesthetic criteria*, and *drawing constraints*. Drawing conventions express general constraints on the geometric representation of symbols and connections.



a An unpleasant diagram



b An equivalent good diagram

Diagram. Figure 1. Examples of unpleasant and good diagrams.

The geometric representation is characterized by the number of spatial dimensions; the usual representation adopted is 2-dimensional (2D), other representations are 1D, 3D, 2–3D. Other drawing conventions refer to general rules on symbols and connections, for example, *polyline* drawing correspond to connections made of sequences of straight lines and *orthogonal* drawing use connections made of horizontal and vertical lines. Aesthetic criteria concern the shape of the diagram, independently from the meaning of symbols, and try to capture universal criteria for expressing the idea of beauty. Drawing constraints refer to the semantics of the underlying represented schema or instance.

Key Points

The above concepts are explained by means of Fig. 1. Concerning drawing conventions, both diagrams adopt a 2D representation. Diagram b adopts a further drawing convention, namely, symbols are inscribed in a rectangular grid. Drawing conventions may express some characteristic of the semantics of the diagram; for example, upward drawings visualize hierarchical relationships, such as generalization hierarchies in conceptual schemas.

Examples of aesthetic criteria are: (i) minimization of crossing between connections, (ii) minimization of the number of bends in connections, (iii) minimization of the area occupied by the diagram, (iv) maximization of the display of symmetries. The above comment to “try to capture universal criteria” is based upon recognition that an expression of beauty in Chinese culture, for example, is asymmetry, and not symmetry. Considering criterion (i), diagram a. has 6 crossings, while diagram b. has no crossing. In diagram b. Employee is symmetric with respect to Vendor, Worker, and Engineer. Note that one cannot simultaneously optimize aesthetic criteria, as intuitively understood considering, for example, previous criteria (i) and (iii).

Examples of drawing constraints are (i) place a given symbol in the centre of the drawing (e.g., Employee in Fig. 1b), keep a group of symbols close together.

Graph drawing is the research area that investigates algorithms that, having in input a nonvisual specification of a diagram, produce a corresponding diagram respecting a given set of drawing conventions, aesthetic criteria, and drawing constraints (see [1] and [2]). Diagrams can be extensively used for displaying visual information [3].

Cross-references

► Chart
► Data Visualization
► Graph
► Visual Formalism
► Visual Representation

Recommended Reading

1. Cruz I.F. and Tamassia R. Graph Drawing Tutorial, http://www.cs.brown.edu/~rt/
2. Di Battista G., Eades P., Tamassia R. and Tollis I.G. Graph Drawing. Prentice-Hall, Englewood Cliffs, NJ, 1999.
3. Tufte E.R. The Visual Display of Quantitative Information. Graphic Press, Cheshire, CT, 1998.

Diagrammatic Representation

► Diagram

Difference

Cristina Sirangelo
University of Edinburgh, Edinburgh, UK

Synonyms

Set-difference

Definition

The difference of two relation instances R_1 and R_2 over the same set of attributes U, denoted by $R_1 - R_2$, is another relation instance over U containing precisely the set of tuples occurring in R_1 and not occurring in R_2.

Key Points

The difference is one of the primitive operators of the relational algebra. It coincides with the notion of set difference, with the additional restriction that it can be applied only to relations over the same set of attributes. However the difference of two arbitrary relations having the same arity can be obtained by first renaming attributes in one of the two relations.

As an example, consider a relation *Students* over attributes (*number, name*), containing tuples {(1001, *Black*),(1002,*White*)}, and a relation *Employees* over attributes (*number, name*), containing tuples {(1001, *Black*),(1003,*Brown*)}. Then the difference *Students* –

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:45

Employees is a relation over attributes (*number, name*) containing the only tuple (1002,*White*).

In the absence of attribute names, the difference is defined on two relations with the same arity: the output is a relation with the same arity as the input, containing the difference of the the sets of tuples in the two input relations.

The number of tuples in the output relation is bounded by the the number of tuples in R_1.

Cross-references

► Relation
► Renaming
► Relational Algebra

Digital Archives and Preservation

Reagan W. Moore
University of California-San Diego, La Jolla, CA, USA

Synonyms

Persistent archives

Definition

Preservation is the set of processes that maintain the authenticity, integrity, and chain of custody of records for long periods of time. Authenticity is defined as the management of provenance information about the creation of the record. Integrity is defined as the ability to create an authentic copy. Chain of custody tracks all processing done to the record, including migration to new storage systems or to new data encoding formats. Digital preservation addresses the challenge of technology evolution by managing preservation properties independently of the choice of software and hardware systems. Preservation properties include the names used to identify archivists, records, and storage systems, and the preservation metadata which includes provenance information and administrative information about record management. The display and manipulation of digital data is maintained through the use of representation information that describes how the record can be parsed and interpreted. Digital preservation environments implement trustworthiness assessment criteria, enabling verification that the preservation properties are maintained over time.

Historical Background

The preservation community bases digital preservation upon the same concepts that are used to preserve paper records. The preservation environment manages the authenticity, integrity, and chain of custody of the digital records. At least four approaches have been implemented that define the digital preservation processes and policies that are needed to maintain authenticity and integrity. (i) Diplomatics defines the required provenance information that describes the institution, event, and ingestion information associated with the documentation of an event. Examples of events are treaties and government communiqués. The records are assumed to be held forever. (ii) The US National Archives and Records Administration associates records with a life-cycle data requirements guide. Each record is associated with a record group (institution), a record series (the set of records governed by a submission agreement), a file unit, and an entity. Each record series has a defined retention and disposition schedule. The arrangement of the records is governed by the submission order in the record series. Standard preservation processes include appraisal, accession, arrangement, description, preservation, and access. (iii) The digital library community manages preservation within the context of a collection, with the required preservation metadata and the arrangement governed by the purpose under which the collection was formed. (iv) The continuum model manages records within an active access environment. Records that are generated for one record series may be used as the basis for generating a new record series. The relationships between records within the multiple record series are tracked as part of their provenance. Each of these four communities imposes the preservation processes and preservation policies required to enforce their goals.

The OAIS standard defines a model for preservation that focuses on record submission, record preservation and record access. The information required for each process is aggregated respectively into a Submission Information Package (SIP), an Archival Information Package (AIP), and a Dissemination Information Package (DIP). An information package contains content (the digital record) and representation information (all information needed to understand the record). The OAIS representation information includes the structure and semantics of the record and links to a knowledge base of a designated community for interpreting the semantic term. The OAIS standard stores Preservation Description

Information within an AIP that includes Fixity information, Provenance information, Context information, and Reference information. This approach attempts to provide the context needed both to understand how to display and manipulate a record, and to interpret the meaning of the record.

The management of technology evolution is a major concern for digital data that is addressed outside of the OAIS model [3]. The storage technology may impose a proprietary referencing mechanism on the data that locks the record onto a single vendor's system. When that vendor goes out of business, access to the records may be lost. Display applications may no longer be available for reading a record. Even though the record can be accessed, it may not be possible to interpret the internal structure correctly. A preservation environment is the interface between the fixed records and the changing external world. A digital archive must ensure that the records can be viewed in the future, while maintaining authenticity, integrity, and chain of custody.

Foundations

Preservation is an active process, starting with the extraction of records from the environment in which they were created. Each record documents an event, or the materials upon which decisions have been made, or the material that is used to support a research or business or legislative objective. The extraction processes start with appraisal, the determination of which records are worthy of preservation. A formal accession process documents the ingestion of the records into the preservation environment, along with the provenance information needed to identify the source of the records, the representation information needed to interpret and understand the records, and the administrative information needed to document the ingestion process. Once the records are under the control of the archivist, a description process organizes the provenance information and an arrangement process organizes the records. A preservation process creates archival information packages that link the preservation information to the record and stores the records. An access process provides mechanisms to discover, display, and manipulate the records.

Preservation is a form of communication with the future. Since the technology that will be used in the future is expected to be more sophisticated and more cost effective than present technology, this appears to be an intractable situation. How can records that are archived today be accessible through arbitrary choices of technology in the future? By viewing preservation as an active process, this challenge can be addressed. At the point in time when new technology is being assimilated into the preservation infrastructure, both the old and new technologies are present. Infrastructure that supports interoperability across multiple versions of software and hardware systems make it possible to evolve preservation environments over time. The approach is called infrastructure independence, and is based on data virtualization, trust virtualization, and management virtualization. The preservation environment is the set of software that enables management of persistent archives independently of the choice of storage system, information syntax, access protocol, discovery mechanism, and display service. Preservation environments insulate records from changes in the environment, while holding fixed the set of standard preservation processes and the name spaces used to identify records, archivists, and storage systems.

Data virtualization consists of two components: (i) persistent name spaces for identifying records, preservation metadata, archivists, and storage systems, (ii) standard operations for interacting with storage repository protocols. Data grid technology provides both components, enabling the integration of new storage technology into a preservation environment, the use of new access protocols for interacting with records, and the migration of records to new encoding syntax.

Trust virtualization is the management of authorization independently of the choice of preservation technology. Data grids provide access controls that enforce relationships between any pair of persistent name spaces. For example, authorization may be a constraint imposed on the archivist identity and the record identity, or a constraint imposed on preservation metadata and the archivist identity, or a constraint based on record identity and storage identity. The constraints may be applied across multiple types of data management environments (file systems, tape archives, databases). Since the name spaces are managed by the data grid, the constraints remain invariant as the records are moved between storage systems within the preservation environment.

Preservation manages communication from the past in order to make assertions about preservation properties such as authenticity, integrity, and chain of custody. For an assertion to be verifiable, a preservation environment must document the preservation processes that have been applied to each record, and the preservation policies that controlled the application of the preservation processes. Unless both processes and policies can be tracked over time, an assertion cannot be verified. The outcome of the application of each preservation process on a record should be recorded as state information that is linked to the record. Assertions can then be validated as queries on the state information that verify that the desired property has been conserved. The processes, policies, and state information constitute representation information about the preservation environment.

To manage the evolution of data formats, three approaches are pursued: Emulation in which the original display application is ported to future operating systems; transformative migration in which the encoding format of the record is changed to a future encoding format; and persistent objects in which the structure and relationships present within the record are characterized and migrated to future information and knowledge representations. Operations that can be performed upon relationships defined between structures can be applied in the future by any application that knows how to manipulate that specific relationship. An example is a query on time track change relationships embedded in a Microsoft Word document.

Key Applications

Multiple technologies can be used to implement the preservation environment. However no single technology provides all of the capabilities that are required. A preservation environment is an integration of management systems, software systems, and hardware systems that isolate records from changes in technology, while preserving authenticity, integrity, and chain of custody. Technologies that implement subsets of the capabilities needed for infrastructure independence include:

- SRB – Storage Resource Broker data grid (http://www.sdsc.edu/srb). The SRB implements the name spaces needed for infrastructure independence, and manages descriptive metadata that can be associated with each collection and file. Collections stored in the SRB include observational data, simulation output, experimental data, educational material, office products, images, web crawls, and real-time sensor data streams. The SRB data grid enables the creation of shared collections from data distributed across multiple administrative domains, institutions, and nations. Data may be stored in file systems, archives, object-relational databases, and object ring buffers. International data grids have been based on the SRB technology through which hundreds of terabytes of data have been replicated.

  The system is designed to scale to petabytes of data and hundreds of millions of files. Applications of the technology include use as data grids for sharing data, digital libraries for publishing data, and persistent archives for preserving data.

  The set of standard operations that are performed upon remote storage systems include Posix file system I/O commands (such as open, close, read, write, seek, stat, ...) and operations needed to mask wide-area network latencies. The extended operations include support for aggregation of files into containers, bulk operations for moving and registering files, parallel I/O streams, and remote procedures that parse and subset files directly at the remote storage system.
- LOCKSS-Lots of Copies Keep Stuff Safe (http://www.lockss.org/lockss/Home). The LOCKSS system manages attributes on files that may be distributed across multiple storage systems. The original design of the system focused on management of data distributed by publishers. The original copy of the file was downloaded from a publisher through a security module that supported the publisher authentication requirements. Attributes were then associated with each file to track the publication source. LOCKSS systems that had retrieved data from the same publisher could then provide disaster recovery copies to each other. Types of access that are supported include file retrieval. The system is designed to scale to about 20 Tera bytes of archived data.
- IBP – Internet Backplane Protocol (http://loci.cs.utk.edu/ibp/). The IBP was designed to enable applications to treat the Internet as if it were a processor backplane. The IBP replicates or caches blocks of data across distributed storage systems at multiple sites. A file system such as LSTORE (http://www.lstore.org/pwiki/pmwiki.php) is implemented on top of the IBP protocol to support the required persistent file naming.
- DSpace (http://www.dspace.org/). This is a digital library that provides standard preservation services

for accessing, managing and preserving scholarly works. DSpace can store files on the local file system, or in the SRB data grid.

- FEDORA – Flexible Extensible Digital Object and Repository Architecture (http://www.fedora-commons.org/). This is digital library middleware that supports the characterization of relationships between records. The relationships may be queried to identify desired data subsets. Services support the creation, management, publication, sharing, annotation, and preservation of digital content.
- UVC-Universal Virtual Computer (http://en.wikipedia.org/wiki/Universal_Virtual_Computer). Thi s is a software system that provides standard operations that can be migrated onto future operating systems. The environment supports a Logical Data Schema for type description, a format decoder, and a Logical Data Viewer for displaying the parsed files.
- ADORE (http://african.lanl.gov/aDORe/projects/adoreArchive/). This is a write-once/read-many preservation system for digital objects. XML-based representations of digital objects are concatenated into a single, valid XML file called XMLtape. The associated data streams are aggregated into Internet Archive ARC files. Each XMLtape is accessed through the Open Archives Initiative – Protocol for Metadata Harvesting. The ARC files are accessed through OpenURL.

Most of the above systems support discovery and access to the record. A subset supports updates to the records, schema extension for provenance metadata, replicas, and bulk operations. The system managing the largest amount of material is the SRB. The SRB is used in the NARA Transcontinental Persistent Archive Prototype (http://www.sdsc.edu/NARA/) as a research tool for the investigation of properties that should be supported by a preservation environment.

Systems that manage representation information characterize records by file format type. The display and access of the record are accomplished by identifying an application that is capable of parsing the record format. Example systems range from stand alone environments to web services to highly integrated environments. Examples include:

- MVD – Multivalent Document (http://elib.cs.berkeley.edu/ib/about.html). This system presumes that a single document comprises multiple layers of related material. Media adaptors parse each layer. Behaviors that manipulate the parsed data are dynamically loaded program objects. It is possible to add new behaviors independently of the media adaptors to provide new operations for manipulating or viewing the layers.
- DFDL-Data Format Description Language (http://forge.ggf.org/sf/projects/dfdl-wg). This is an Open Grid Forum standards effort that characterizes the mapping of bit-steams to structures through creation of an associated XML file. This is the essential capability needed to interpret an arbitrary file. The structures can be named.
- EAST (http://nssdc.gsfc.nasa.gov/nssdc_news/mar02/EAST.html). EAST is a data description language that supplies information about the format of the described data. EAST is designed for building descriptions of data that are maintained separately from the data itself.
- CASPAR – Cultural, Artistic and Scientific knowledge for Preservation, Access and Retrieval (http://www.casparpreserves.eu/). This is a research project to identify the representation information required to understand digital objects. This includes not only the data format types, but also the designated community that will use the data, and the knowledge base that defines the required semantic terms.
- METS–Metadata Encoding and Transmission Standard (http://www.loc.gov/standards/mets/). This is a standard for encoding descriptive, administrative, and structural metadata regarding objects within a digital library.

As pointed out by the CASPAR project, the ability to interpret the representation information can require additional representation information. In order to close this recursion, a designated community is defined that understands how to interpret the semantics of the final set of representation information by accessing a community knowledge base.

Future Directions

Rule-based data management systems have the potential to virtualize management policies by providing infrastructure that guarantees the application of the policies directly at the remote storage systems independently of the remote administrative domain. The iRODS (integrated Rule Oriented Data Systems) data grid (http://irods.sdsc.edu) installs software middleware (servers) at each remote storage system. Each server includes a rule engine that controls the execution of micro-services at that storage system. The rules are expressed as event: condition: action-sets: recovery-sets, where the condition can include operations on any of the persistent state information that is managed by the data grid. The action-sets can include micro-services or rules, enabling the aggregation of micro-services into an execution hierarchy. For each action, a recovery procedure is specified to enable the tracking of transaction semantics. Additional name spaces are required that include:

- The names of the micro-services that aggregate the standard operations into well-defined functions.
- The names of the rules that control the execution of the micro-services.
- The persistent state information that tracks the results of applying the operations (think of the location of a replica as persistent state that must be saved).

Rule-based systems support asynchronous operations. Micro-services can be queued for deferred or periodic operations. Thus a recovery-set might include the scheduling of a deferred attempt to complete the operation, followed by an e-mail message if that attempt does not succeed, or it might roll-back any changed state information and report failure.

Rule-based systems make it possible to characterize explicitly the set of management policies that control the preservation environment. This includes the rules that govern data integrity (replication, data distribution, media migration), the rules that assert the presence of required descriptive or provenance metadata (including the extraction of the required metadata from an authoritative source), and the rules that govern chain of custody (assignment of access controls and parsing of audit trails to verify the enforcement). Rule-based systems also explicitly characterize the preservation processes as sets of named micro-services that can be ported to new operating systems. In effect, a rule-based data management system is able to build an emulation environment for the set of management policies and preservation processes that the archivist wants to apply. The environment guarantees that the desired policies and processes will continue to control the preservation environment even when new technology is incorporated into the system.

The identification of standard preservation management policies is being attempted through the RLG/NARA trustworthiness assessment criteria [4]. A mapping of these assessment criteria to iRODS rules is possible, and identifies some 105 rules that are required to enforce and verify preservation criteria. The NARA Electronic Records Archive has defined the set of capabilities that they require for long-term preservation [1]. These capabilities have also been mapped to iRODS rules and micro-services. The goal is to build the set of management principles and fundamental preservation processes required for long term preservation [2].

When representation information for preservation environments is available, it may be possible to design a theory of digital preservation. The components will include:

- ► Characterization of the representation information for the preservation environment
  - Definition of the properties that the preservation environment should conserve
  - Definition of the management policies that enforce the conservation of the desired properties
  - Definition of the capabilities (preservation processes) needed to apply the management policies
- ► Analysis that the system is complete
  - Demonstration that assessment criteria can be mapped to queries on persistent state information that are managed independently of the choice of technology
  - Demonstration that these management policies can be mapped to well-defined rules
  - Demonstration that the rules control the execution of well-defined micro-services that are independent of the choice of preservation technology
- ► Analysis that the system is closed
  - Demonstration that the state persistent information required to validate assessment criteria are generated by each micro-service
  - Demonstration that for every micro-service the associated persistent state information is updated on each successful operation

A theory of digital preservation defines the processes required to assert that the preservation environment has been implemented correctly and will successfully enable long-term preservation.

Cross-references

► Archiving Experimental Data
► Data Warehouse

► Disaster Recovery
► Information Lifecycle Management
► LOC METS
► Metadata Repository
► Provenance
► Replication

Recommended Reading

1. Electronic Records Archive capabilities list defines a comprehensive set of capabilities needed to implement a preservation environment, and can be examined at http://www.archives.gov/era/pdf/requirements-amend0001.pdf.
2. Moore R. Building preservation environments with data grid technology. Am Archivist, 69(1):139–158, 2006.
3. OAIS, Reference Model for an Open Archival Information System, ISO 14721:2003.
4. RLG/NARA TRAC – Trustworthy Repositories Audit and Certification: Criteria and Checklist. http://wiki.digitalrepositoryauditandcertification.org/pub/Main/ReferenceInputDocuments/trac.pdf.

Digital Curation

Greg Janée
University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Stewardship

Definition

Digital curation is the activity of maintaining and adding value to a trusted body of digital information for current and future use.

Key Points

Left unattended, digital information degrades over time. Even if the information's bits are correctly preserved (a difficult task in itself) the technological context surrounding the bits – the computing platforms, programming languages, applications, file formats, and so forth – will change sufficiently over time until the information is no longer usable. Changes in the information's social context are just as significant. The communities and organizations involved in the information's initial creation and use may attach different values and interpretation to the information over time, or cease to exist altogether. And the passage of time only exacerbates contemporary problems such as establishing the authenticity, quality, and provenance of the information.

Curation is the activity of maintaining a body of information so that it remains usable over time. Curation covers the entire lifecycle of the information, from creation to contemporary use, from archival to reuse. Specific curation activities include: selection and appraisal; capture of metadata and the information's larger technological, scientific, and social contexts; conversion to archival formats; establishment and maintenance of authenticity and provenance; annotation and linkage; provisioning for secure and redundant storage; transformation, migration, and emulation as needed over time; discoverability in contemporary search systems; creation of meaningful access mechanisms; and recontextualization.

Different types of information bring different curation requirements and present different challenges. Information intended for direct human consumption, such as many textual and multimedia documents, may only need to be migrated to new formats as older formats fall out of favor. But data, particularly scientific data, may require significant reprocessing and transformation. For example, climatalogical observations may need to be periodically recalibrated to support long-term longitudinal studies, a process requiring deep understanding and emulation of the original calibration.

Cross-references

► Preservation

Recommended Reading

1. Beagrie N. Digital curation for science, digital libraries, and individuals. Int. J. Digital Curat., 1(1), 2006.
2. Consultative Committee for Space Data Systems. Reference Model for an Open Archival Information System (OAIS). ISO 14721:2003, 2002.
3. Trustworthy Repositories Audit and Certification: Criteria and Checklist. Center for Research Libraries, 2007.

Digital Elevation Models

Leila De Floriani, Paola Magillo
University of Genova, Genova, Italy

Synonyms

Digital Terrain Model (DTM); Digital Surface Model DEMs

Definition

A Digital Elevation Model (DEM) represents the 3D shape of a terrain in a digital format. A terrain is mathematically modeled as a function $z = f(x, y)$ which maps each point (x, y) in a planar domain D into an elevation value $f(x, y)$. In this view, the terrain is the graph of function f over D.

In practice, a terrain is known at a finite set of points within D, which may (i) lie at the vertices of a *regular grid*, (ii) be scattered, or (iii) belong to *contour lines* (also known as *isolines*), i.e., the intersections of the terrain surface with a sequence of horizontal planes.

In case (i), the DEM consists of the grid structure plus elevation values at its vertices. This is called a *Regular Square Grid* (RSG). Within each grid cell, terrain elevation either is defined as constant, or it is modeled by a function, which can be linear (this involves cell decomposition in two triangles), or quadratic (usually, bilinear).

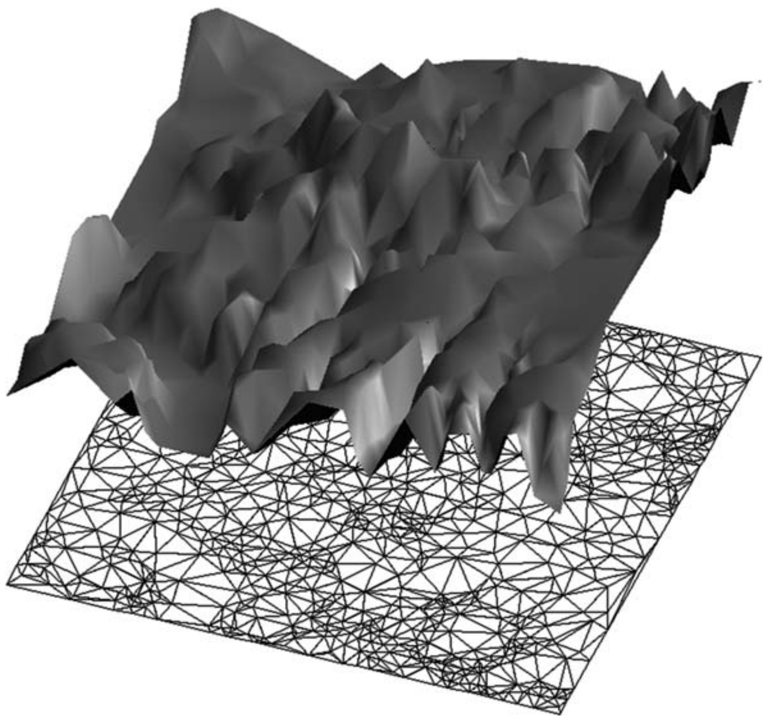
In case (ii), usually the DEM is defined based on a triangle mesh joining the data points in D and by a piecewise-linear function interpolating elevations at the three vertices of each triangle. This gives a *Triangulated Irregular Network* (TIN) (see Fig. 1).

In case (iii), the DEM consists of the polygonal lines forming each contour, plus the corresponding elevation, and the containment relation between contours at consecutive elevations. This provides a *contour map*.

Historical Background

Historically, terrain models were represented as three-dimensional relief maps, generally constructed for military or educational purposes from plaster, papier-maché, or vinyl plastic. For instance, such models were used extensively by military forces during World War II. Contour maps drawn on sheets of paper have probably been the most common form of terrain model. In the early ages of computer-based Geographic Information Systems, these maps were converted into digital format through scanning devices. DEMs based on contour lines are a way for representing a terrain, but not for performing computations, or simulations. For other applications, they are usually converted into triangulated models (TINs) by connecting two consecutive contour lines through a set of triangles.

The first DEMs of the computer age were *Regular Square Grids (RSGs)*. Very large and accurate gridded DEMs are usually acquired through remote sensing techniques, and are built from aerial or satellite raster



Digital Elevation Models. Figure 1. A triangle-based terrain representation.

images. Thanks to the regular structure of an RSG, both storing and processing an RSG is simple from the point of view of design, but the huge size of such models may cause serious inefficiency in both storage and manipulation. This problem can be addressed by applying techniques for terrain generalization, or multi-resolution models. *Generalization* means decimating data to achieve smaller memory size and faster processing time, at the expense of less accuracy. This can be achieved through grid subsampling or selection of meaningful vertices to build a TIN. *Multi-resolution* refers to the capability of maintaining multiple accuracy levels at the same time, and selecting the most appropriate one for the current working session. This latter aspect is covered in.

Triangulated Irregular Networks (TINs) are the most general model from the point of view of modeling power, since they do not assume any spatial distribution of the data. TINs can encompass morphological terrain features, *point features* (e.g., peaks, pits and passes) and *line features* (e.g., coast lines, rivers, contour lines). On the other hand, the internal representations for TINs and the algorithms for their construction are more complex than those for RSGs. These latter have been extensively studied within the field of *computational geometry*. Storing and manipulating TINs require more computational resources than RGSs, for the same number of vertices, but they may need much fewer vertices to achieve the same accuracy. This is especially true for terrains with an irregular morphology, since a TIN can adapt the density of the mesh to the local complexity of the terrain.

Foundations

There are two major categories of DEMs: Regular Square Grids (RSGs) and Triangulated Irregular Networks (TINs).

RSGs are based on a domain subdivision into a square grid. There are two major ways of approximating a terrain based on a grid. In a *stepped model*, each data point lies at the center of a grid cell, and its elevation is assigned to the whole cell. The resulting terrain model has the shape of a 2D histogram, and thus the surface presents jump discontinuities at cell edges. The second approach produces a continuous surface. The data points are the vertices of the grid cells. Within each cell, the elevation is approximated through a function that interpolates the known elevations of the four cell vertices. Let (x_0, y_0) be the coordinates of the lower left corner of the cell, (Δ_x, Δ_y) be the cell size and $z_{0,0}, z_{0,1}, z_{1,0}$ and $z_{1,1}$ be the elevations of its four corners (lower-left, upper-left, lower-right, upper-right, respectively; see Fig. 2).

A bilinear interpolant estimates the elevation at a point $P = (x, y)$ within the cell as:

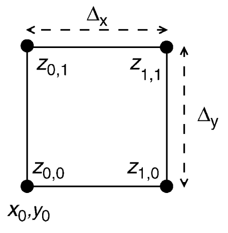
$$\begin{aligned} z = z_{0,0} + (z_{0,1} - z_{0,0})(y - y_0)/\Delta_y + \\ (z_{1,0} - z_{0,0})(x - x_0)/\Delta_x + \\ (z_{1,1} - z_{0,1} - z_{1,0} + z_{0,0})(x - x_0)(y - y_0)/\Delta_x\Delta_y \end{aligned}$$

and provides a continuous (but not differentiable) surface approximation.

RSGs are stored in a very simple data structure that encodes just a matrix of elevation values, the grid being implicitly represented. All other information (including interpolating functions and neighbor relations among cells) can be reconstructed in constant time. Moreover, the regular structure of RSGs makes them well suited to parallel processing algorithms.

The RSG is the main format for distributing elevation data. Many existing DEMs are provided in this format, including USGS (United States Geological Survey [11]) data as well as many proprietary GIS formats (and the interchange Arc/Info ASCII grid). Usually, the file contains a header followed by the elevations values listed in either row or column order. The header contains the information needed to decode the given values and to locate the grid on the Earth surface (geo-referencing) An RGS could also be encoded in a standard image format by mapping the elevation range to a grey level value, but this format does not support geo-referencing.

The main disadvantage of an RSG is its uniform resolution (i.e., the same cell size) over the whole domain. This may imply undersampling in raw areas and oversampling in flat areas for a terrain with irregular morphology. Uniformly increasing the resolution



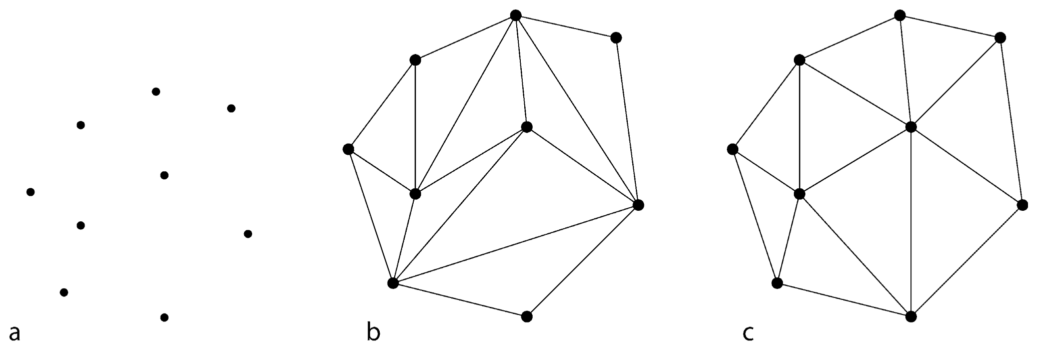
Digital Elevation Models. Figure 2. A cell in an RSG.

produces huge matrices and thus high storage and processing costs. Adaptive nested grids have been proposed to overcome this problem.

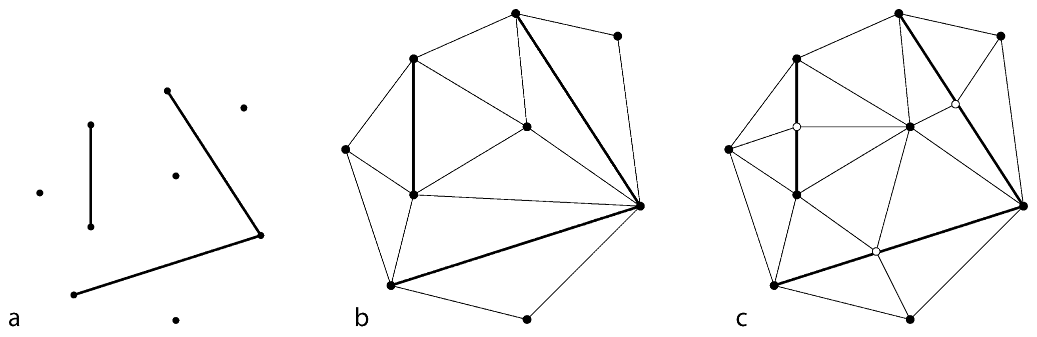
A TIN is based on an irregular domain subdivision, in which the cells are triangles, i.e., a triangle mesh with vertices at the set *S* of data points. Usually, a linear interpolating function is defined on the triangles of the mesh, thus providing a continuous terrain approximation. More precisely, a *triangle mesh T* consists of a set of triangles such that: (i) the set of vertices of *T* coincides with *S*, (ii) the interiors of any two triangles of *T* do not intersect, (iii) if the boundaries of two triangles intersect, then the intersection is either a common vertex, or a common edge (see Fig. 3). Each triangle of *T* is mapped in the three-dimensional space by considering the elevation values at its three vertices and the plane passing through the resulting three points in 3D space.

The quality of the terrain approximation provided by a TIN depends on the quality of the underlying triangle mesh, since the triangulation of a set of points is not unique. The most widely used triangle mesh is the *Delaunay* one. A Delaunay mesh is the one among all possible triangle meshes joining a given set of points in which the circumcircle of each triangle does not contain any data point in its interior. This means that the triangles of a Delaunay mesh are as much equiangular as possible, within the constraints imposed by the distribution of the data points [9]. It has also been proved that the use of a Delaunay mesh as the basis for a TIN improves the quality of the terrain approximation and enhances numerical stability in computations. Other triangulation criteria have been proposed which consider not only the 2D triangulation, but also the corresponding triangles in 3D space [4].

In many practical cases, a TIN must embed not only points, but also lines representing morphological terrain features (coast lines, rivers, ridges), man-made structures (roads, railways, gas lines), political or administrative boundaries, contour lines. The Delaunay criterion has been modified to deal with such lines in two different ways: (i) in the *constrained Delaunay triangulation*, the given lines appear as triangle edges [3]; (ii) in the *conforming Delaunay triangulation*, each line is discretized as a dense set of points [5] (see Fig. 4). The constrained Delaunay triangulation may present sliver triangles when segments are too long. Conforming



Digital Elevation Models. Figure 3. (a) A set of data points; (b) a triangle mesh; (c) Delaunay triangle mesh.



Digital Elevation Models. Figure 4. (a) A set of data points and lines; (b) Constrained Delaunay triangulation; (c) A conforming Delaunay triangulation.

triangulations may add a very large number of points in order to force the lines to be included in the resulting mesh.

The simplest storage format for a TIN is a *triangle soup*: each triangle is represented separately by listing the nine coordinates of its three vertices. The *indexed format* has been designed to avoid replicating the coordinates of vertices shared by several triangles incident in the same vertex. It stores each vertex of the TIN, as three coordinates, in a list of vertices, and each triangle as three vertex indices within such a list. This latter format can be enriched by adding the *adjacency relation* linking each triangle with the three triangles its shares an edge with. Triangle adjacency links support efficient navigation inside a TIN. To support an efficient traversal of the mesh passing through vertices, it is convenient to attach to each vertex the index of one of its incident triangles. The encoding of a TIN in an indexed format requires one half of the space of a triangle soup. The indexed format with adjacencies requires about 2/3 of the space of a triangle soup.

Data structures and algorithms for TINs [1,3,12] are more complex than those for RSGs. But, in many cases, a TIN can reach the same approximation error as an RGS in terrain representation with a much smaller number of vertices. The main advantage of a TIN is its flexibility in adapting the density of sampling in the case of a terrain with irregular morphology, to include relevant lines or points, and fit to irregularly shaped domains. Many multi-resolution terrain models are TIN-based.

Key Applications

Visualization of terrains is needed in many fields including environmental sciences, architecture, entertainment.

Morphology analysis, which is concerned with the extraction of ridges, rivers, water basins, is important in environmental monitoring and planning.

Visibility analysis of a terrain is concerned with the computation of visible, or invisible, areas from a set of viewpoints, with the computation of hidden or scenic paths, with the extraction of networks of mutually visible points. This is useful in many applications such as communication, surveillance, visual impact of infrastructures, etc.

Applications may require computing *paths or point networks* of minimum cost according to some criteria combining visibility, length, height variation, etc. A wide range of simulations (e.g. flood, erosion, pollution, etc.) are also possible on a terrain model.

For details, see [3,7,10].

Future Directions

Some geographical applications need to represent not only the surface of the earth, but also its internal structure. This requires modeling 3D volumes as well as 2D surfaces (e.g., boundaries between two rock layers), and their adjacency relations. 3D extensions of digital elevation models, such as RSGs and TINs, leads to regular volume models and irregular tetrahedral meshes, respectively. Regular volume models are grids of hexahedral cells connecting the data points, while irregular tetrahedral meshes are meshes formed by tetrahedra joining the data points. Thus, challenging issues arise here, such as the development of compact and effective data structures for encoding digital volumetric models and of efficient algorithms for building and manipulating such models.

Another field for 3D terrain modeling is *urban terrain modeling*, which provides the integration between elevation and urban data, i.e., laying buildings and other landscape or vegetation elements over a terrain. With urban terrain models, high-quality photorealistic rendering of 3D geovirtual environments can be achieved. These are used in common products like GoogleEarth [11] or Virtual Earth [6]. Urban terrain modeling has also important applications to 3D town maps for business and entertainment, and also for city administration and urban development planning.

Another challenging field is incorporating *time* in a DEM. This will allow modeling the evolution of a terrain over time and it is relevant for both historical record and simulation. From a mathematical point of view, the reference model is no longer a 2D surface embedded in 3D space, but a 3D volume embedded in 4D space (where the last dimension is time). Here, it is necessary to define and develop digital volumetric models, like regular models and irregular tetrahedral meshes, but embedded in 4D space.

In all such applications, multi-resolution models will play an important role, because of the even larger size of the data sets and of the corresponding volumetric models.

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:46

URL to Code

United States Geological Survey (USGS) home page, http://www.usgs.gov/
Google Earth home page, http://earth.google.com/
Microsoft Virtual Earth home page, http://www.microsoft.com/virtualearth/
Spatial Data Transfer Standard (SDTS) home page, http://mcmcweb.er.usgs.gov/sdts/

Cross-references

► GIS for Geological Applications
► Regular Entry on Multiresolution Terrain Modeling
► Three-Dimensional GIS
► Triangulated Irregular Networks (TIN)

Recommended Reading

1. de Berg M., van Kreveld M., Overmars M., and Schwarzkopf O. Computational Geometry – Algorithms and Applications. 2nd edn. Springer, Berlin, 2000.
2. De Floriani L., Magillo P., and Puppo E., Applications of computational geometry to Geographic Information Systems. In Handbook of Computational Geometry, Chap. 7, J.R. Sack, J. Urrutia (eds.). Elsevier Science, 1999, pp. 333–388.
3. De Floriani L. and Puppo E. An On-line Algorithm for Constrained Delaunay Triangulation, CVGIP: Graphical Models and Image Processing, 54(4): Academic, July 1992, pp. 290–300, Academic, Orlando, FL.
4. Dyn N., Levin D., and Rippa S. Data dependent triangulations for piecewise linear interpolation, IMA J. Numer. Analy., 10:137–154, 1990.
5. Edelsbrunner H. and Tan T.S. An upper bound for conforming Delaunay triangulation: Discrete Comput. Geom., 10:197–213, 1993.
6. Google Earth home page, http://earth.google.com/
7. Longley P.A., Goodchild M.F., Maguire D.J., and Rhind D.W. (eds.) Geographical Information Systems, 2nd edn. Whiley, New York, 1999.
8. Microsoft Virtual Earth home page, http://www.microsoft.com/virtualearth/
9. O'Rourke J., Computational Geometry in C, 2nd edn. Cambridge University Press, Cambridge, 1998.
10. Peckham R.J. and Jordan G. (eds.). Digital Terrain Modelling – Development and Applications in a Policy Support Environment, Lecture Notes in Geoinformation and Cartography, Springer, Berlin, 2007.
11. United States Geological Survey (USGS) home page, http://www.usgs.gov/
12. van Kreveld M. Digital elevation models and TIN algorithms. In Algorithmic Foundations of Geographic Information Systems, number 1340 in Lecture Notes in Computer Science (tutorials), M. van Kreveld, J . Nievergelt, T. Roos, and P. Widmayer (eds.). Springer, Berlin, 1997, pp. 37–78.

Digital Image

► Image

Digital Libraries

Venkat Srinivasan, Seungwon Yang,
Edward A. Fox
Virginia Tech, Blacksburg, VA, USA

Synonyms

Electronic libraries

Definition

Digital libraries (DLs) are complex information systems that have facilities for storage, retrieval, delivery, and presentation of digital information. They are complex in nature because of the broad range of activities they may need to perform, and because they may need to serve multiple types of audiences. Thus, the broadest definitions include the people and agents/actors involved, as well as the software, content, structure/organization(s), services, policies, procedures, etc.

Historical Background

One of the earliest detailed works about digital libraries (DLs) was prepared by Licklider [8], who envisioned a network of computers with digitized versions of all of the literature ever published. However, the term "digital library" became widely used only around 1991, in connection with a series of workshops funded by the US National Science Foundation, which later led to significant NSF support of R&D DL projects (http://www.dli2.nsf.gov/), e.g., Informedia, which focused on digital video [2].

In a Delphi study of digital libraries, Kochtanek et al. [7] defined a digital library as an "organized collection of resources, mechanisms for browsing and searching, distributed networked environments, and sets of services objectified to meet users' needs." The President's Information Technology Advisory Committee (PITAC) report [12] mentioned "These new libraries offer digital versions of traditional library, museum, and archive holdings, including text, documents, video, sound, and images. But they also provide powerful new technological capabilities that enable users to refine their inquiries, analyze the results, and change the form of the information to interact with it. . .".

Starting in 2005, the European Union, through the DELOS Network of Excellence on Digital Libraries (http://www.delos.info/) worked to develop a reference model for the digital library field. Their DL manifesto [1] defines a digital library as "a (potentially virtual) organization that comprehensively collects, manages, and preserves for the long term rich digital content and offers to its user communities specialized functionality on that content, of measurable quality, and according to prescribed policies."

Foundations

While the DELOS Reference Model [1] has aimed to identify the key constructs of a DL, in order to allow standardization and interoperability, there is one other effort that has tried to develop a formal foundation for the DL field. The 5S framework [4,5] was developed in the Digital Library Research Laboratory at Virginia Tech as a scientific base for digital libraries. There are five elements that underlie DL systems; these can be described (informally) as:

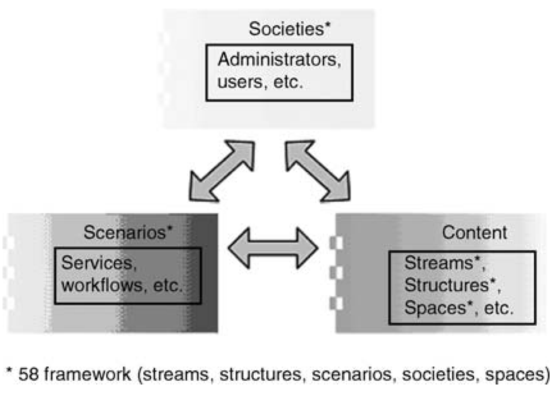
1. Streams: all types of content, as well as communications and flows over networks or into sensors, or sense perceptions. Examples include: text, video, audio, and image. These can be formalized as a sequence.
2. Structures: organizational schemes, including data structures, databases, and knowledge representations. Examples include: collection, catalog, hypertext, and document metadata. These can be formalized as a graph, with labels and a labeling function.
3. Spaces: 2D and 3D interfaces, GIS data, and representations of documents and queries. Examples include: storage spaces used in indexing, browsing, or searching services, as well as interfaces. These can be formalized as a set with operations (vector, topological, measurable, measure, and probability spaces).
4. Scenarios: system states and events, or situations of use by human users or machine processes, yielding services or transformations of data. Examples include: searching, browsing, and recommending. These can be formalized as a sequence of related transition events on a state set.
5. Societies: both software "service managers" and fairly generic "actors" who could be (collaborating) human users. Examples include: service managers (software), actors (learners, teachers, etc.) [1,4,5]. These can be formalized as a pair (i.e., a set of communities and a set of relationships).

A formal description can be found in [4].

DL systems encompass all the five Ss. The 5 Ss are also used in a formal definition of a minimal DL [5], which has the key constructs that most would agree must be found in any DL system. Of course, most real DLs are extended well beyond what constitutes a minimal DL, to better suit the needs of the users. Accordingly, the 5S framework has led to a growing set of meta-models for different types of DLs, each formally defined from a minimalist perspective: archaeological DL, image DL, personal DL, practical DL, and superimposed information DL (supporting annotations and knowledge management of the annotations and base information).

DLs also can be understood as a triad (see Fig. 1), which consists of content, societies, and scenarios. Digital libraries preserve and provide the "content" which is stored in various formats (Streams). The contents have certain "Structures" to help DLs efficiently serve their patrons and to help the administrators manage them. "Spaces" are required to store the content in a DL system. A DL system's user interface is the "Space" in which the patrons and the system interact, to submit, download, share, and discuss about the digital contents.

To perform a certain task, a series of steps is needed. The interaction between the system and its patrons involves "Scenarios," which often are described as workflows. Processes to be performed by the system also belong to this category. For the "Societies" in the


Societies*
Administrators, users, etc.
Scenarios*
Services, workflows, etc.
Content
Streams*, Structures*, Spaces*, etc.
* 5S framework (streams, structures, scenarios, societies, spaces)

Digital Libraries. Figure 1. 5S framework [4,5] represented as a triad.

triad, the people involved with any kind of DL activity would fit into this category.

Building Digital Libraries

Building a DL is not an exact science. The historical absence of formal models to guide the development of DLs has led to divergence and duplication of efforts. Also, interoperability has been a problem. There have been many distributed, heterogeneous, and federated DLs, e.g., NCSTRL (http://www.ncstrl.org) or the Alexandria Digital Library [14], which were built using ad hoc principles, wherein interoperability was achieved only on a case by case basis. Standards have been defined to help achieve interoperability, for example the Open Archives Initiative (OAI), (http://www.openarchives.org/) Protocol for Metadata Harvesting. Clearly there are tradeoffs between encouraging innovation and technological improvement, between functionality enhancement and autonomy in creating DLs – and compromises made so as to achieve interoperability.

Fortunately, a number of toolkits have been developed to aid those developing DLs. One of the first was the EPrints system (http://www.eprints.org), initially supporting electronic pre-prints, but later enhanced to assist with the growing movement toward institutional repositories.

Two other popular systems used to build DLs are DSpace (http://www.dspace.org/) Greenstone (http://www.greenstone.org) and Fedora (http://www.fedora.info/). DSpace can simply be used out of the box to build a DL. Figure 2 gives a high-level summary of the parts and architecture of DSpace.

Digital library systems support access through an application layer (see top of Fig. 2), which typically includes a User Interface (UI). Also important is support to load content in batch mode, to aid the sharing of metadata (e.g., through the OAI protocols), and to facilitate access to particular digital objects by way of their unique identifiers (e.g., their handles).

The main operations of a DL can be thought of as the business logic (see middle of Fig. 2), including key operations like searching (typically, with DSpace, using Lucene, see http://lucene.apache.org/) and browsing. For more flexibility, DLs can manage complex workflows. They can support authentication and authorization, protecting privacy, management of user groups, and broad suites of services to access and preserve content. Ultimately, the content of a DL must be supported through a storage subsystem (see bottom of Fig. 2), which typically makes use of database technology, as well as handling of multimedia files.

Key Applications

DLs generally are effective content management systems, offering a broad range of services such as archiving, digital preservation, browsing, searching, and presentation. Electronic libraries, virtual libraries, institutional repositories, digital repositories, courseware management systems, and personalized information systems are all considered to be different types of DLs. Any research in the area of DLs is thus inherently interdisciplinary in nature, encompassing especially computer science (CS) areas like database management (especially for the underlying storage layer), information retrieval, multimedia, hypertext, human-computer interaction, and library and information science (LIS).

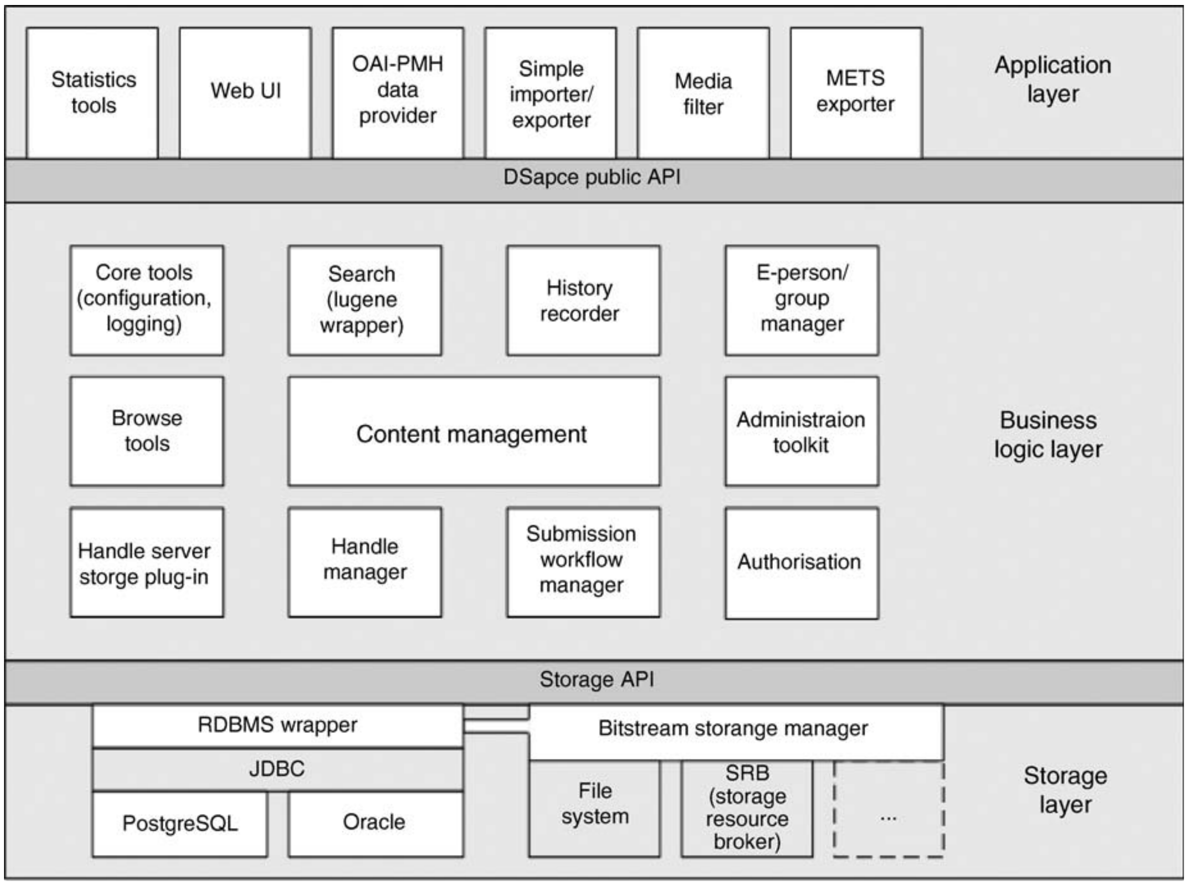
DLs can be applied to a variety of needs. Many DLs of today are suitable for personal content management systems, institutional repositories, or distributed global systems. DLs can be used to preserve documents (e.g., electronic theses and dissertations, as coordinated by NDLTD, at http://www.ndltd.org), to manage multimedia content (e.g., Informedia [2], which focuses on video), to support content-based retrieval of images (e.g., in connection with the needs of archaeologists in the ETANA DL, at http://www.etana.org), or to handle a combination of digital formats (e.g., ADEPT [6], which connects with GIS and mapping efforts).

One example of a highly visible and successful DL initiative is the Perseus project (http://www.perseus.tufts.edu/) which digitized ancient Greek literature and makes it available as an online repository, with many added services and additional information based on careful analysis and use of powerful tools. This project has completely changed the way classics are taught in universities across the world. Thus, DLs generally go well beyond archiving, providing value added services as well.

Another example is the Traditional Knowledge Digital Library (TKDL) [13], supported by the government of India, to digitize traditional Indian medical literature (relating to Ayurveda) in order to prevent bio-piracy and patents.

Future Directions

The DL community continues to spread and grow. Work on curricular resources [3,9,10,11] will help on



Digital Libraries. Figure 2. DSpace institutional repository architecture (from: http://www.dspace.org).

the education side. Publishing at conferences and online magazines (e.g., http://www.dlib.org/) will help with dissemination of findings related to research, development, practice, and policy. Work on standards and open access (http://www.openarchives.org/) will facilitate interoperability. Improvement of systems like DSpace will help with more widespread utilization of effective software solutions, including support for preservation. Additional help with archiving and preservation, beyond just preserving the bits (see, for example, the LOCKSS effort, http://www.lockss.org/), is required (see for example, http://home.pacbell.net/hgladney/ddq.htm).

It is hoped that further work on foundations, including the 5S framework and the DELOS Reference Model, will lead to a firm theoretical and practical base for the field. For example, work to apply 5S to the growing need for personal DLs seems particularly promising, supporting Personal Information Management.

Greater efficiency and effectiveness of DL systems can help address problems associated with the "information glut," and work on DL quality metrics can help those who select or maintain DL systems and installations. More tools are needed for digital librarians, to assist them as they address fundamental questions like what to store, how to preserve, how to protect intellectual property, how to display, etc. These, and many similar questions, arising from technical, economical, and sociological perspectives, also will need to be addressed, as DLs are more widely employed. Then, efforts in the research, development, deployment, and operational sectors will better support the growing community of digital librarians, who aim to provide interested societies with cyberinfrastructure, which incorporates suitable organizational structures and appropriate services.

Experimental Results

See http://www.dli2.nsf.gov/

Data Sets

See http://www.dli2.nsf.gov/

Url to Code

See http://www.eprints.org/, http://www.dspace.org/, and http://www.fedora.info/

Cross-references

► Browsing
► Searching

Recommended Reading

1. Candela L., Castelli D., Ioannidis Y., Koutrika G., Pagano P., Ross S., Schek H.-J., and Schuldt H. Setting the foundations of digital libraries: the DELOS manifesto. D-Lib Mag., 13(3/4), ISSN 1082–9873, 2007.
2. Christel M., Wactlar H., and Stevens S. Informedia. In Proceedings of the ACM Multimedia Conference, New York, pp. 480–481.
3. Digital Library Curriculum Development Project homepage (2007). http://curric.dlib.vt.edu/.
4. Gonçalves M.A. Streams, Structures, Spaces, Scenarios, and Societies (5S): A Formal Digital Library Framework and Its Applications. Computer Science Doctoral Dissertation. Blacksburg, VA: Virginia Tech, 161pp, 2004.
5. Gonçalves M., Fox E., Watson L., and Kipp N. Streams, structures, spaces, scenarios, societies (5S): a formal model for digital libraries. ACM Trans. Inf. Syst., 22:270–312, 2004.
6. Janée G. and Frew J. The ADEPT digital library architecture. In Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'02), 2002.
7. Kochtanek T. and Hein K.K. Delphi study of digital libraries. Inf. Process. Manage., 35(3):245–254, 1999.
8. Licklider J.C.R. Libraries of the Future. The MIT Press, Cambridge, MA, 1965.
9. Pomerantz J., Oh S., Wildemuth B., Yang S., and Fox E.A. Digital library education in computer science programs. In Proceedings of the Seventh ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'07), 2007.
10. Pomerantz J., Oh S., Yang S., Fox E.A., and Wildemuth B. The Core: Digital Library Education in Library and Information Science Programs. D-Lib Magazine, 12(11), 2006.
11. Pomerantz J., Wildemuth B., Oh S., Fox E.A., and Yang S. Curriculum development for digital libraries. In Proceedings of the Sixth ACM/IEEE-CS Joint Conference on Digital Libraries, 2006, pp. 175–184.
12. Reddy R. and Wladawsky-Berger I. Digital Libraries: Universal Access to Human Knowledge – A Report to the President. President's Information Technology Advisory Committee (PITAC), Panel on Digital Libraries 2001.
13. Sen N. TKDL - A safeguard for Indian traditional knowledge. Curr. Sci., 82(9):1070–71, 2002.
14. Smith T.R. and Frew J. Alexandria digital library. Commn. of the ACM, 38(4):61–62, 1995.

Digital Rights Management

Radu Sion
Stony Brook University, Stony Brook, NY, USA

Synonyms

DRM

Definition

Digital rights management (DRM) is a term that encompasses mechanisms and protocols deployed by content publishers and rights holders to enforce access licensing terms. This entry discusses mainly DRM for relational data, specifically such methods as database watermarking. General DRM techniques are discussed elsewhere [19].

Historical Background

Historically, DRM methods have found ample application in consumer entertainment and multimedia industries since the late 1980s. More recently, with the advent of massive relational data management and warehousing systems, increasingly, valuable data has been produced, packaged and delivered in relational form. In such frameworks, DRM assurances have become an essential requirement. As traditional multimedia DRM mechanisms are ill-suited for the new data domain, starting in 2001, researchers developed mechanisms for relational data rights protection [1,5,10,11,12,13,14,15] mainly centered around the concept of deploying steganography in hiding copyright "watermarks" in the underlying data. While initial efforts focused on basic numeric data [5,11,14,6], subsequent work handled categorical [9,10,14], and streaming [16,18] data types.

Foundations

Overview

As increasing amounts of data are produced, packaged and delivered in digital form, in a fast, networked environment, one of its main features threatens to become its worst enemy: zero-cost verbatim copies. The ability to produce duplicates of digital Works at almost no cost can now be misused for illicit profit. This mandates mechanisms for effective rights assessment and protection. Different avenues are available, each with its advantages and drawbacks. Enforcement by legal means is usually ineffective, unless augmented by a digital counterpart such as steganography (information hiding). *Digital Watermarking* as a method of rights assessment deploys information hiding to conceal an indelible "rights witness" ("rights signature", watermark) within the digital Work to be protected – thus enabling ulterior court-time proofs associating particular works with their respective rights holders. The soundness of such a method relies on the assumption that altering the Work in the process of hiding the mark does not destroy the value of the Work, while it is difficult for a malicious adversary ("Mallory") to remove or alter the mark beyond detection without doing so. The ability to resist attacks from such an adversary, mostly aimed at removing the watermark, is one of the major challenges.

Watermarking for Rights Protection

But how does the ability to prove rights in court relates to the final desiderata, namely to *protect* those rights? The ability to prove/assess rights convincingly in court constitutes a deterrent to malicious Mallory. It thus becomes a tool for rights protection if counter-incentives and legal consequences are set high enough. Such a method only works however if the rightful rights-holder (Alice) actually knows about Mallory's misbehavior *and* is able to prove to the court that: (i) Mallory possesses a certain Work *X* and (ii) *X* contains a "convincing" (e.g., very rare with respect to the space of all considered similar Works) and "relevant" watermark (e.g., the string "(c) by Alice"). This illustrates the game theoretic nature at the heart of the watermarking proposition and of information security in general. Watermarking is a game with two adversaries, Mallory and Alice. At stake lies the value inherent in a certain Work *X*, over which Alice owns certain rights. When Alice releases *X*, to the public or to a licensed but potentially untrusted party, she deploys watermarking for the purpose of ensuring that one of the following holds: (i) she can always prove rights in court over any copy or valuable derivative of *X* (e.g., segment thereof), (ii) any existing deviate *Y* of *X*, for which she cannot prove rights, does not preserve any signiicant value (derived from the value in *X*), (iii) the cost to produce such an un-watermarked derived *Y* of *X* that is still valuable (with respect to *X*) is higher than its value.

Once outsourced, i.e., out of the control of the watermarker, data might be subjected to a set of attacks or transformations; these may be malicious – e.g., with the explicit intent of removing the watermark – or simply the result of normal use of the data. An effective watermarking technique must be able to survive such use. In a relational data framework some of the important attacks and transformations are:

1. *Sampling.* The attacker (Mallory) can randomly select and use a subset of the watermarked data set that might still provide value for its intended purpose ("subset selection"). More specifically, here the concern is with both (1a) horizontal and (1b) vertical data partitioning – in which a valuable subset of the *attributes* are selected by Mallory.
2. *Data addition.* Mallory adds a set of tuples to the watermarked set. This addition is not to significantly alter the useful properties of interest to Mallory.
3. *Alteration.* Altering a subset of the items in the watermarked data set such that there is still value associated with the result. In the case of numeric data types, a special case needs to be outlined here, namely (3a) a linear transformation performed uniformly to all of the items. This is of particular interest as it can preserve significant valuable data-mining related properties of the data.
4. *Ulterior claims of rights.* Mallory encodes an additional watermark in the already watermarked data set and claims rights based upon this second watermark.
5. *Invertibility attack.* Mallory attempts to establish a plausible (watermark,key) pair that matches the data set and then claims rights based on this found watermark [2,3].

Consumer Driven Watermarking

An important point about watermarking should be noted. By its very nature, a watermark modifies the item being watermarked: it inserts an indelible mark in the work such that (i) the insertion of the mark does not destroy the value of the work, i.e., it is still useful for the *intended purpose*; and (ii) it is difficult for an adversary to remove or alter the mark beyond detection without destroying this value. If the work to be watermarked cannot be modified without losing its value then a watermark cannot be inserted. The critical issue is not to avoid alterations, but to limit them to acceptable levels with respect to the intended use of the work. Naturally, one can always identify some use that is affected by even a minor change to any portion of the data. It is therefore important that (i) the main intended purpose and semantics that should be preserved be identified during watermarking and that (ii) *the watermarking process not interfere with the final data consumer requirements*. This paradigm is called *consumer-driven watermarking*. In consumer-driven watermarking the rights holder and Mallory play against each other within subtle trade-off rules aimed at keeping the quality of the result within acceptable bounds. The data itself (its quality requirements) acts as an impartial referee moderating each and every "move".

In [4] Gross-Amblard introduce interesting theoretical results investigating alterations to relational data (or associated XML) in a consumer-driven framework in which a set of parametric queries are to be preserved up to an acceptable level of distortion. The author first shows that the main difficulty preserving such queries "is linked to the informational complexity of sets defined by queries, rather than their computational complexity" [4]. Roughly speaking, if the family of sets defined by the queries is not *learnable* [20], no query-preserving data alteration scheme can be designed. In a second result, the author shows that under certain assumptions (i.e., query sets defined by first-order logic and monadic second order logic on restricted classes of structures – with a bounded degree for the Gaifman graph or the tree-width of the structure) a query-preserving data alteration scheme exists.

Numerical Data Types

This section explores some of the watermarking solutions in the context of relational data in which one or more of the attributes are of a numeric type. Among existing solutions one distinguishes between *single-bit* (the watermark is composed of a single bit) and *multi-bit* (the watermark is a string of bits) types. Orthogonally, the encoding methods can be categorized into two: *direct-domain* and *distribution* encodings. In a direct-domain encoding, each individual bit alteration in the process of watermarking is directly correlated to (a part of) the encoded watermark. In distribution encodings, the encoding channel lies often in higher order moments of the data (e.g., running means, hierarchy of value averages). Each individual bit alteration impacts these moments for the purpose of watermark encoding, but in itself is not directly correlated to any one portion of the encoded watermark.

Single-bit encodings. In [1,5] Kiernan, Agrawal et al. propose a direct domain encoding of a single bit watermark in a numeric relational database. Its main algorithm proceeds as follows. A subset of tuples are selected based on a secret criteria; for each tuple, a secret attribute and corresponding least significant (ξ) bit position are chosen. This bit position is then altered according to yet another secret criteria that is directly correlated to the watermark bit to be encoded. The main assumption is, that changes can be made to any attribute in a tuple at any least significant ξ bit positions. At watermark detection time, the process will re-discover the watermarked tuples and, for each detected accurate encoding, become more "confident" of a true-positive detection.

The authors discuss additional extensions and properties of the solution including incremental updatability, blind properties, optimization of parameters, as well as handling relations without primary keys. To handle the lack of primary keys, the authors propose to designate another attribute, or a number of most significant bit-portions of the currently considered one, as a primary key. This however presents a significant vulnerability due to the very likely existence of duplicates in these values. Mallory could mount a statistical attack by correlating marked bit values among tuples with the same most significant bits. This issue has been also considered in [7] where a similar solution has been adopted.

Multi-bit encodings. While there likely exist applications whose requirements are satisfied by single-bit watermarks, often it is desirable to provide for "relevance", i.e., linking the encoding to the rights holder identity. This is especially important if the watermark aims to defeat against invertibility attacks (5). In a single-bit encoding this can not be easily achieved. Additionally, while the main proposition of watermarking is not covert communication but rather rights assessment, there could be scenarios where the actual message payload is of importance. One apparent direct extension from single-bit to multi-bit watermarks would be to simply deploy a different encoding, with a separate watermark key, for each bit of the watermark to be embedded. This however, might not be possible, as it will raise significant issues of inter-encoding interference: the encoding of later bits will likely distort previous ones. This will also make it harder to handle ulterior claim of rights attacks (4).

In [6] Li et al. extend the work by Kiernan, Agrawal et al. [1,5] to provide for multi-bit watermarks in a direct domain encoding. The scheme functions as follows. The database is parsed and, at each bit-encoding step, one of the watermark bits is randomly chosen for embedding; the solution in [1,5] is then deployed to encode the selected bit in the data at the "current" point. The "strength of the robustness" of the scheme is claimed to be increased with respect to [1,5] due to the fact that the watermark now possesses an additional dimension, namely length. This should guarantee a better upper bound for the probability that a valid watermark is detected from unmarked data, as well as for the probability that a fictitious secret key is discovered from pirated data (i.e., invertibility attacks (5)). This upper bound is said to be independent of the size of database relations thus yielding robustness against attacks that change the size of database relations. In [8] the same authors propose to use the multi-bit watermarking method [6] for "fingerprinting" relational data in order to track copyright violators.

Multi-bit distribution encoding. Encoding watermarking information in resilient numeric distribution properties of data presents a set of advantages over direct domain encoding, the most important one being its increased resilience to various types of numeric attacks. In [10,11,12,13,14,15], Sion et al. introduce a multi-bit distribution encoding watermarking scheme for numeric types. The scheme was designed with both an adversary and a data consumer in mind. More specifically the main desiderata were: (i) watermarking should be consumer driven – i.e., desired semantic constraints on the data should be preserved – this is enforced by a feedback-driven rollback mechanism, and (ii) the encoding should survive important numeric attacks, such as linear transformation of the data (3.a), sampling (1) and random alterations (3).

The solution starts by receiving as user input a reference to the relational data to be protected, a watermark to be encoded as a copyright proof, a secret key used to protect the encoding and a set of data quality constraints to be preserved in the result. It then proceeds to watermark the data while continuously assessing data quality, potentially backtracking and rolling back undesirable alterations that do not preserve data quality.

Watermark *encoding* is composed of two main parts: in the first stage, the input data set is securely partitioned into (secret) subsets of items; the second stage then encodes one bit of the watermark into each subset. If more subsets (than watermark bits) are available, error correction is deployed to result in an increasingly resilient encoding. Each single bit is encoded/represented by introducing a slight skew bias in the tails of the numeric distribution of the corresponding subset. The encoding is proved to be resilient to important classes of attacks, including subset selection, linear data changes and random item(s) alterations.

In [10,14,15] the authors discuss a proof of concept implementation. It is worth mentioning here due to its consumer-driven design. In addition to a watermark to be embedded, a secret key to be used for embedding, and a set of relations/attributes to watermark, the software receives as input also a set of external *usability plugin modules.* The role of these plugins is to allow user defined query metrics to be deployed and queried at run-time without re-compilation and/or software restart. The software uses those metrics to re-evaluate data usability after each atomic watermarking step.

To validate this consumer driven design the authors perform a set of experiments showing how, for example, watermarking with classification preservation can be enforced through the usability metric plugin mechanisms. Moreover, the solution is proved experimentally on real data to be extremely resilient to random alterations and uninformed alteration attacks. This is due to its distribution-based encoding which can naturally survive such alterations. For example, altering the *entire* watermarked data set within 1% of its original values only yields a distortion of less than 5% in the detected watermark.

Categorical Data Types

Categorical data is data drawn from a discrete distribution, often with a finite domain. By definition, it is either non-ordered (nominal) such as gender or city, or ordered (ordinal) such as high, medium, or low temperatures. There are a multitude of applications that would benefit from a method of rights protection for such data.

Additional challenges in this domain derive from the fact that one cannot rely on arbitrary small (e.g., numeric) alterations to the data in the embedding process. Any alteration has the potential to be significant, e.g., changing DEPARTURE_CITY from

"Chicago" to "Bucharest" is likely to affect the data quality of the result more than a simple change in a numeric domain. There are no "epsilon" changes in this domain. This completely discrete characteristic of the data requires discovery of fundamentally new bandwidth channels and associated encoding algorithms. Moreover, the ability of the adversary to simply re-map attributes values to a new domain needs to be considered.

In [9,17] Sion et al. introduce a method of watermarking relational data with categorical types, based on a set of new encoding channels and algorithms. More specifically, two domain-specific watermark embedding channels are used, namely (i) *inter-attribute associations* and (ii) *value occurrence frequency-transforms* of values. The mechanism starts with an initial user-level assessment step in which a set of attributes to be watermarked are selected. In its basic version, watermark encoding in the *inter-attribute association* channel is deployed for each attribute pair (*K*,*A*) in the considered attribute set. A subset of "fit" tuples is selected, as determined by the association between *A* and *K*. These tuples are then considered for mark encoding. Mark encoding alters the tuple's value according to secret criteria that induces a statistical bias in the distribution for that tuple's altered value. The detection process then relies on discovering this induced statistical bias. The authors validate the solution both theoretically and experimentally on real data (Wal–Mart sales). They demonstrate resilience to both alteration and data loss attacks, for example being able to recover over 75% of the watermark from under 20% of the data.

The authors further discuss additional extensions and properties of the solution, including a consumer-driven design, incremental updatability, its blind nature, optimizations for minimizing alteration distances, as well as the ability to survive extreme vertical partitioning, handle multiple data sources as well as attribute re-mapping.

Key Applications

Rights protection for relational data is important in scenarios where it is sensitive, valuable and about to be outsourced. A good example is a data mining application, where data is sold in pieces to parties specialized in mining it, e.g., sales patterns database, oil drilling data, financial data. Other scenarios involve for example online B2B interactions, e.g., airline reservation and scheduling portals, in which data is made available for direct, interactive use.

Watermarking in relational frameworks is a relatively young technology that has begun its maturity cycle towards full deployment in industry-level applications. Many of the solutions discussed above have been prototyped and validated on real data. Patents have been filed for several of them, including Agrawal et al. [1,5] and Sion et al. [9,10,11,12,13,14,15,17]. In the next few years one expects these solutions to become available commercially, tightly integrated within existing DBMS or as stand-alone packages that can be deployed simultaneously on top of multiple data types and sources. Ultimately, the process of resilient information hiding will become available as a secure mechanism for not only rights protection but also data tracing and authentication in a multitude of discrete data frameworks.

Future Directions

A multitude of associated future research avenues present themselves in a relational framework, including: the design of alternative primary or pseudo-primary key independent encoding methods, a deeper theoretical understanding of limits of watermarking for a broader class of algorithms, the ability to better defeat additive watermark attacks, an exploration of zero-knowledge watermarking, etc.

Moreover, while the concept of on-the-fly quality assessment for a consumer-driven design has the potential to function well, another interesting avenue for further research would be to augment the encoding method with direct awareness of semantic consistency (e.g., classification and association rules). This would likely result in an increase in available encoding bandwidth, thus in a higher encoding resilience. One idea would be to define a generic language (possibly subset of SQL) able to naturally express such constraints and their propagation at embedding time.

Additionally, of particular interest for future research exploration are cross-domain applications of information hiding in distributed environments such as sensor networks, with applications ranging from resilient content annotation to runtime authentication and data integrity proofs.

Cross-references

► Steganography

Recommended Reading

1. Agrawal R., Haas P.J., and Kiernan J. Watermarking relational data: framework, algorithms and analysis. VLDB J., 12 (2):157–169, 2003.
2. Craver S., Memon N., Yeo B.-L., and Yeung M.M. Resolving rightful ownerships with invisible watermarking techniques: Limitations, attacks, and implications. IEEE J. Select. Areas Commun., 16(4):573–586, 1998.
3. Cox I., Bloom J., and Miller M. Digital watermarking. In Digital Watermarking. Morgan Kaufmann, 2001.
4. Gross-Amblard D. Query-preserving watermarking of relational databases and xml documents. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. 2003, pp. 191–201.
5. Kiernan J. and Agrawal R. Watermarking relational databases. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
6. Li Y., Swarup V., and Jajodia S. A robust watermarking scheme for relational data. In Proc. Workshop on Information Technology and Systems (WITS), 2003, pp. 195–200.
7. Li Y., Swarup V., and Jajodia S. Constructing a virtual primary key for fingerprinting relational data. In Proc. 2003 ACM Workshop on Digital Rights Management, 2003, pp. 133–141.
8. Li Y., Swarup V., and Jajodia S. Fingerprinting relational databases: Schemes and specialties. IEEE Transactions on Dependable and Secure Computing, 2(1):34–45, 2005.
9. Sion R. Proving ownership over categorical data. In Proc. 20th Int. Conf. on Data Engineering, 2004.
10. Sion R. wmdb.*: A suite for database watermarking (demo). In Proc. 20th Int. Conf. on Data Engineering, 2004.
11. Sion R., Atallah M., and Prabhakar S. On watermarking numeric sets CERIAS TR 2001-60.
12. Sion R., Atallah M., and Prabhakar S. On watermarking numeric sets. In Proc. Int. Workshop on Digital Watermarking IWDW, Lecture Notes in Computer Science. Springer, 2002.
13. Sion R., Atallah M., and Prabhakar S. Watermarking Databases CERIAS TR 2002-28.
14. Sion R., Atallah M., and Prabhakar S. Rights protection for relational data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
15. Sion R., Atallah M., and Prabhakar S. Relational data rights protection through watermarking. IEEE Transactions on Knowledge and Data Engineering TKDE. 16(6), June 2004.
16. Sion R., Atallah M., and Prabhakar S. Resilient rights protection for sensor streams. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
17. Sion R., Atallah M., and Prabhakar S. Ownership proofs for categorical data. IEEE Transactions on Knowledge and Data Engineering TKDE, 2005.
18. Sion R., Atallah M., and Prabhakar S. Rights protection for discrete numeric streams. IEEE Transactions on Knowledge and Data Engineering TKDE. 18(5), May 2006.
19. Wikipedia. Digital Rights Management. http://en.wikipedia.org/wiki/Digital_rights_management.
20. Valiant L.G. A theory of the learnable. In Proc. Symp. on the Theory of Computing. 1984, pp. 436–445.

Digital Signatures

Barbara Carminati
University of Insubria, Varese, Italy

Synonyms

Signatures

Definition

Informally, given a message M, the digital signature of M generated by a signer S is a bit string univocally bound to M and some secret key known only by S. More precisely, since digital signature schemes are based on asymmetric cryptography, it is possible to define the digital signature of M generated by S as a bit string dependent on M and the private key of S. Digital signature schemes have the property that signatures generated with a private key can be validated only by the corresponding public key. This ensures the authenticity of the message. Moreover, any modification on the signed message will invalidate the signature itself. This means that if the signature is validated it provides an evidence that the message has not been altered after the digital signature has been applied on it. This ensures the integrity of the message.

Historical Background

The notion of digital signature appeared in 1976 in a paper by Diffie and Hellman [1]. In this paper, authors introduced for the first time the concept of asymmetric cryptography and discussed how it could be combined with one-way functions to ensure message authentication. However, the first digital signature scheme with a practical implementation appeared only in 1978, proposed by Rivest et al. [7], and still represents one of the most exploited signature technique. Later on, several other schemes have been proposed with an improved efficiency and/or offering further functionalities (see [4] for a survey). An example of these schemes are *one-time signatures*, that is, digital signature schemes where keys can be used to sign, at most, one message. These schemes have the benefit that are very efficient, making them particularly useful in applications requiring a low computational complexity. Other examples are, for instance, *arbitrated digital signatures*, where signatures are generated and verified with the cooperation of a trusted third party, or *blind signature* schemes devised in such a way that a signer is not able to observe the message it signs. This latter scheme are useful in digital cash protocols or electronic voting systems.

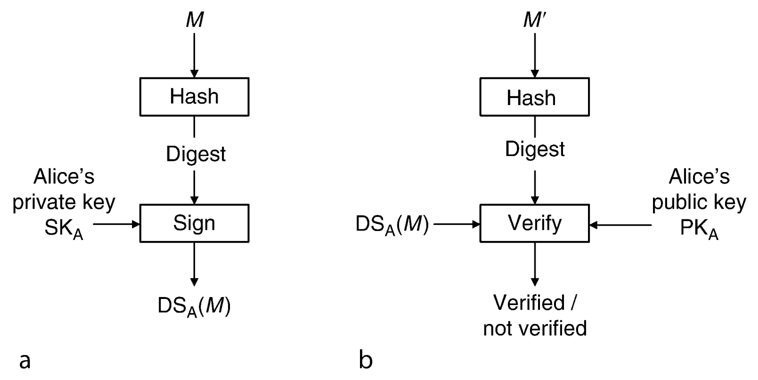
Foundations

Digital signatures have been introduced to obtain, in a digital form, the benefits of handwritten signatures. In general, by appending a handwritten signature to a document, the signer provides evidence that he or she has validated the document. This could mean that the document has been generated, modified, or at least simply read for approval by the person who claims to have done it. In a similar way, digital signatures aim to provide evidence that the signer has really elaborated the message. To achieve this result, digital signatures exploit asymmetric cryptography and collision-resistant hash functions. This implies that a user A willing to digitally sign a message has to be provided with a key pair consisting of a *private key*, SK_A, and a *public key*, PK_A. The security of the digital signature relies on the assumption that the private key is kept secret by the owner itself, whereas the public key is available to all users. Note that various digital signature schemes have been proposed, with differen signature and verification algorithms, which will be discussed later on. However, independently from the adopted algorithms, the overall process to generate an validate a digital signature is always the same. As an example, assume that Alice wishes to send Bob a messag $\mathcal{M}$ completed with its digital signature, denoted a $DS_A(\mathcal{M})$. The procedure to sign and verify the signature is described in what follows.

Digital signature generation. To sign a message $\mathcal{M}$, Alice first generates a condensed version of $\mathcal{M}$, called *digest* of the message, which is obtained by a collision resistant hash function $h()$. Then, the digest is signed using Alice's private key (cf. Fig. 1a). Notice that "signing" a digest implies to apply on it a set of transformations according to the digital signature scheme adopted.

Digital signature verification. Assuming that Bob receives the message $\mathcal{M}'$ from Alice. To validate its digital signature, i.e., to verify whether $\mathcal{M}'$ is equal to $\mathcal{M}$, he first computes the digest of $\mathcal{M}'$ by using the same hash function used for signature generation. Then, using Alice's public key and the new digest value he can verify whether the digital signature is valid or not (cf. Fig. 1b).

Digital signatures provide several benefits. The first is related to the property of asymmetric cryptography ensuring that it is computationally infeasible to validate a signature $DS_A(\mathcal{M})$ with a public key PK_x different from the one corresponding to the private key used for signature creation. Thus, if the signature is verified, it guarantees that the message has been really generated by the owner of the private key, that is, it ensures the *authenticity* of the message. This property entails as further benefits, the *non repudiation* of the message. Indeed, in case of a lying signer A claiming that the message was not generated by him or her, the digital signature $DS_A(\mathcal{M})$ acts like an evidence of the opposite, making thus A not able to repudiate the message. A further benefit is given by properties of hash functions, which ensures that it is computationally infeasible to find two different messages $\mathcal{M}$ and $\mathcal{M}'$such that $h(\mathcal{M}) = h(\mathcal{M})$. Thus, if the signature is validated, it means that the digest computed on the received message matches the digest extracted during the



Digital Signatures. Figure 1. (a) Digital signature generation. (b) Digital signature verification.

verification process. A validated signature provides, therefore, evidence that the message has not been altered after the digital signature has been applied on it. This ensures the *integrity* of the message.

In the following, two of the most used digital signature schemes, that is, the DSA and RSA algorithms, are illustrated (see [4] for a detailed description of them). However, for a more comprehensive overview of digital signatures, besides cryptographic details it is interesting to have also an example of how they can be used in real world scenarios. Given its relevance, it is considered the Web as reference scenario. For this reason, in the second part of this entry it is presented the standard proposed by W3C for digital signature encoding, called XML Signature [8].

Digital Signature Schemes

In general, a digital signature scheme is defined according to three algorithms:

1. *Key generation algorithm* that randomly computes a pair of keys (SK_A, PK_A) for the signer A.
2. *Signature generation algorithm* $S()$, which receives as input the messag $\mathcal{M}$ to be digitally signed (Hereafter, the discussion refers to a generic message M. However, it is important to recall that the signature process usually requires to digitally sign the digest of the original message.), the private key SK_A of the signer and generates the digital signature $DS_A(\mathcal{M})$ of the message $\mathcal{M}$.
3. *Signature verification algorithm* $V()$, that takes as input the digital signature $DS_A(\mathcal{M})$ of the message $\mathcal{M}$, the public key PK_A of the signer and, optionally, the message $\mathcal{M}$, and returns whether the verification succeeds or not.

It is possible to organize the many schemes proposed in the literature into two main classes, that is, *digital signature schemes with appendix* and *digital signature schemes with message recovery*, which are described in what follows.

Digital Signature Schemes with Appendix

The main characteristic that distinguishes these schemes with regard to those with message recovery is that they require the original message during the verification process. Thus, together with the signer's public key and the digital signature, the verification algorithm takes as input also the original message. This implies that the signer A has to send to the intended verifier B the original message complemented with its digital signature as "appendix."

In the literature, there exist several digital signature schemes with appendix. However, the most relevant and recognized is the one proposed in 1991 by the U.S. National Institute of Standards and Technology (NIST). The scheme, called Digital Signature Algorithm (DSA) [3], became an U.S. Federal Information Processing Standard (FIPS 186) in 1993, with the aim to be the first digital signature scheme recognized by any government (i.e., a Digital Signature Standard – DSS) (In 2000, NIST extended the standard to three FIPS-approved algorithms: Digital Signature Algorithm (DSA), RSA (as specified in ANSI X9.31), and Elliptic Curve DSA (ECDSA; as specified in ANSI X9.62).).

A summary of DSA algorithms is given in the following (the interested reader can refer to [6] for more details).

DSA – Key generation algorithm. Let A be the entity for which public and private keys are generated. PK_A and SK_A are calculated as follows:

1. Select a prime number p, where $2^{L-1} < p < 2^L$, L is multiple of 64, and $512 \leq L \leq 1024$;
2. Select q, a prime divisor of $p - 1$, where $2^{159} < q < 2^{160}$;
3. Select g, a number whose multiplicative order modulo p is q. (This can be calculated by $g = h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < p - 1$ such that $h^{(p-1)/q} \bmod p > 1$);
4. Generate a random or pseudorandom integer x, where $0 < x < q$;
5. Calculate $y = g^x \bmod p$;
6. Set $PK_A = (p, q, g, y)$ and $SK_A = (x)$;

DSA – Signature algorithm. Let M be the message to be signed with PK_A, and $h()$ be an hash function (FIPS 186–2 uses SHA-1 as hash function [5]. The forthcoming FIPS 186–3 uses SHA-224/256/384/512.), the digital signature $DS_A(\mathcal{M})$ is generated as follows:

1. Generate a random or pseudorandom integer k, where $0 < k < q$;
2. Calculate $r = g^k \bmod p$;
3. Calculate $s = (k^{-1}(h(m) + xr)) \bmod q$, where k^{-1} is the multiplicative inverse of k;
4. Set $DS_A(M) = (r,s)$;

DSA-Signature verification algorithm. Let M', r', and s' be the received versions of M, r, and s. To validate the digital signature, an entity B should do the following:

1. Obtain the public key of A, $PK_A = (p, q, g, y)$;
2. Verify that $0 < r' < q$ and $0 < s' < q$, if not the signature shall be rejected;
3. Compute $w = (s')^{-1} \bmod q$;
4. Compute $u_1 = (h(M')w) \bmod q$ and $u_2 = (r'w) \bmod q$;
5. Compute $v = (((g)^{u_1}(y)^{u_2}) \bmod p) \bmod q$;
6. If $v = r'$, then the signature is verified.

Digital Signature Schemes with Message Recovery

Differently to the previous signature schemes, these schemes do not require the original message during the verification process. In contrast, by taking as input only the digital signature and the public key of the signer, the verification algorithm V() recovers the original message M directly from the digital signature. The main advantage of these schemes is that they minimize the length of the message to be transmitted, in that only the digital signature has to be sent. This makes the digital signatures with message recovery particularly tailored to applications where the bandwidth is one of the main concern.

However, it is important to notice that adopting these schemes in the digital signature process depicted in Fig. 1 requires to send the original message even if $V()$ is able to recover the message. Indeed, if the message is hashed before signing it, these schemes are not able to recover the original message, rather they recover only its digest. Having the digest without the original message makes the receiver not able to verify the message integrity, in that it is not possible to match the two hash values. For this reason, when these schemes are adopted in the standard digital signature process (see Fig. 1) they are used like schemes with appendix, i.e., the digital signature is appended to the corresponding message.

One of the most known digital signature scheme with message recovery is the RSA algorithm [7], which is the first one in public-key cryptography to be suitable for signature as well as encryption. RSA algorithms are briefly illustrated in what follows, by focusing only on signatures. An interested reader should refer to [4] for a deeper discussion on RSA public key cryptography, and to PKCS#1 standard [2] for details on RSA implementation.

RSA – Key generation algorithm. Let A be the entity for which public and private keys are generated. PK_A and SK_A are calculated as follows:

1. Generate two large random or pseudorandom prime numbers p and q;
2. Compute $n = pq$ and $\phi = (p - 1)(q - 1)$;
3. Select a random integer e, $1 < e < \phi$, such that gcd $(e; \phi) = 1$;
4. Compute the unique integer d, $1 < d < \phi$, such that $ed \equiv 1(\bmod \phi)$;
5. Set $PK_A = (n, e)$ and $SK_A = (n, d)$;

RSA - Signature algorithm. Let M be the message to be signed with SK_A. The digital signature $DS_A(M)$ is generated as follows:

1. Compute $s = M^d \bmod n$;
2. Set $DS_A(M) = (s)$.

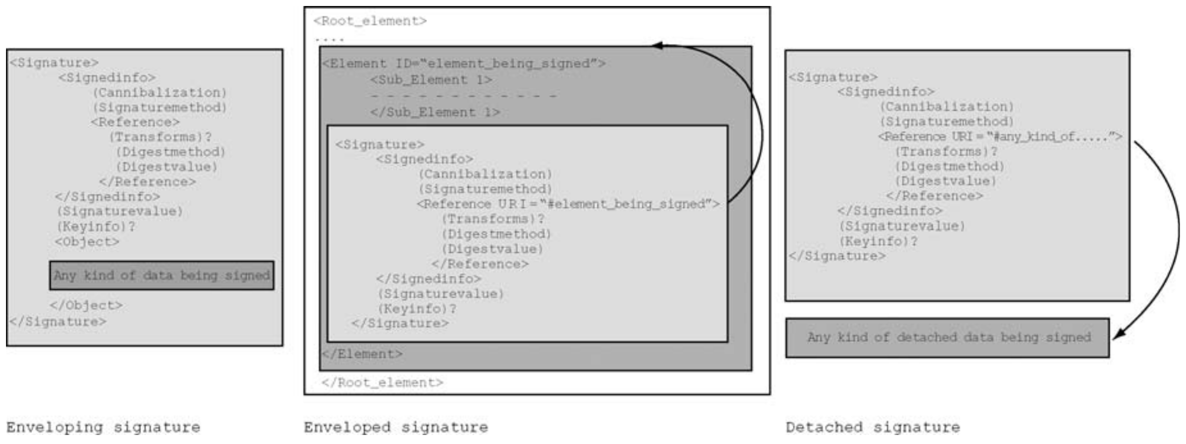
RSA – Signature verification algorithm. Let s' be the received versions of s. To validate the digital signature an entity B should do the following:

1. Compute $M' = s'^e \bmod n$;
2. If $M = M'$, then the signature is verified.

XML Signature

In conjunction with IETF, the W3C XML Signature Working Group has proposed a recommendation, called XML Signature [8], with the twofold goal of defining an XML representation for digital signatures of arbitrary data contents, as well as a description of the operations to be performed as part of signature generation and validation. The proposed XML syntax has been designed to be very flexible and extensible, with the result that a single XML Signature can sign more that one type of digital content. For instance, a single XML Signature can be used to sign an HTML file and all the JPEG files containing images linked to the HTML page. The overall idea is that data to be signed are digested, each digest value is placed into a distinct XML element with other additional information needed for the validation. Then, all the resulting XML elements are inserted into a parent element, which is digested and digitally signed.

Additionally, the standard supports a variety of strategies to locate the data being signed. These data can be either external or local data, that is, a portion of the XML document containing the signature itself. In particular, the XML Signature recommendation



```
<Signature>
      <Signedinfo>
           (Cannibalization)
           (Signaturemethod)
           <Reference>
             (Transforms)?
              (Digestmethod)
              (Digestvalue)
            </Reference>
      </Signedinfo>
      (Signaturevalue)
      (Keyinfo)?
      <Object>

      Any kind of data being signed

      </Object>
</Signature>
```

Enveloping signature

```
<Root_element>
....
<Element ID="element_being_signed">
      <Sub_Element 1>
      - - - - - - - - - - - - -
      </Sub_Element 1>

  <Signature>
      <Signedinfo>
           (Cannibalization)
           (Signaturemethod)
           <Reference URI="#element_being_signed">
              (Transforms)?
              (Digestmethod)
              (Digestvalue)
            </Reference>
      </Signedinfo>
      (Signaturevalue)
      (Keyinfo)?
  </Signature>
</Element>
</Root_element>
```

Enveloped signature

```
<Signature>
      <Signedinfo>
           (Cannibalization)
           (Signaturemethod)
           <Reference URI="#any_kind_of.....">
             (Transforms)?
             (Digestmethod)
             (Digestvalue)
            </Reference>
      </Signedinfo>
      (Signaturevalue)
      (Keyinfo)?
</Signature>

  Any kind of detached data being signed
```

Detached signature

Digital Signatures. Figure 2. Taxonomy of XML Signatures.

supports three different kinds of signatures, which differ for the localization of the signed data with regard to the XML element encoding its signature (see Fig. 2): the *Enveloping Signature*, where the signed data is embedded into the XML Signature; the *Enveloped Signature*, in which the signed data embeds its signature; the *Detached Signature*, where the signed data is either an external data, or a local data included as a sibling element of its signature.

In what follows, a brief overview of the process needed for XML Signature generation is given. In describing these steps the basic structure of an XML Signature, reported in Fig. 3, is referred, where symbol "?" denotes zero or one occurrences; "+" denotes one or more occurrences; and "*" denotes zero or more occurrences.

The first step in the generation of an XML signature requires to specify which are the data to be signed. To this purpose, an XML Signature contains a `Reference` element for each signed data, whose `URI` attribute stores the address of the signed data (In the case of enveloping signatures, URI attribute is omitted since the data is contained in the signature element itself, whereas for enveloped signature the URI attribute denotes the element being signed via a fragment identifier.). Then, the digest of the data is calculated and placed into the `DigestValue` supplement. Information on the algorithm used to generate the digest are stored into the `DigestMethod` element. The `Reference` element may contain an optional `Transforms` subelement specifying an ordered list of transformations (such as for instance canonicalization, compression, XSLT/XPath expressions) that have been applied to the data before it was digested. The next step is to collect all the `Reference` elements into a `SignedInfo` element, which contains the information that is actually signed. Before applying the digital signature, the `SignedInfo` element is transformed into a standard form, called canonical form. The aim of such transformation is that of eliminating from the element additional symbols eventually introduced during the processing (for instance, spaces introduced by an XML parser and so on), that may cause mistakes during the signature validation process. After the canonical form has been generated, the digest of the whole `SignedInfo` element is computed and signed. The resulting value is stored into the `SignatureValue` element, whereas information about the algorithm used for

```
<Signature>
<SignedInfo>
(CanonicalizationMethod)

  (SignatureMethod)
   (<Reference (URI=)? >
       (Transforms)?
       (DigestMethod)
       (DigestValue)
    </Reference>)+
 </SignedInfo>
  (SignatureValue)
  (KeyInfo)?

  (<Object>
         (SignatureProperties)?
         (Manifest)?
   </Object>)*
</Signature>
```

Digital Signatures. Figure 3. Basic structure of an XML Signature.

generating the digital signature is contained in the `SignatureMethod` element. The `Signature` element can also give the recipient additional information to obtain the keys to validate the signature. Such information is stored into the optional `KeyInfo` subelement. The last step is to wrap the `SignedInfo`, `SignatureValue`, and `KeyInfo` elements into a `Signature` element. In the case of enveloping signature the `Signature` element also contains the data being signed, wrapped into the `Object` subelement.

Key Applications

Digital signatures are widely adopted in scenarios where assurances of message authenticity and integrity are crucial. Examples of these scenarios are email applications. Due to the relevance of these applications, two different recommendation for applying digital signatures to email have been proposed (i.e., PGP and S/MIME). Other scenarios are those requiring the authenticity and integrity of messages exchanged during protocols execution, like, for instance, the SSL protocol which exploits digital signatures to create secure Web sessions. A further relevant scenario is given by Public Key Infrastructure (PKI). PKIs have been introduced to univocally bind public keys to the respective owners. PKI assumes the existence of one or more trusted Certificate Authorities (CAs) in charge of generating public key certificates containing information about the identity of the key owner. To provide evidence that a public key certificate has been generated by a CA, PKI requires that certificates are digitally signed by CA.

Cross-references

► Asymmetric Cryptography
► Blind Signatures
► Hash Functions
► XML

Recommended Reading

1. Diffie W. and Hellman M. New directions in cryptography. IEEE Trans. Inf. Theory, IT-22(6):644–654, 1976.
2. Jonsson J. and Kaliski B. Public-Key Cryptography Standards (PKCS) No. 1: RSA Cryptography. Request for Comments 3447, February 2003.
3. Kravitz D.W. (1993) Digital Signature Algorithm. U.S. Patent No. 5, 231, 668.
4. Menezes A.J., van Oorschot P.C., and Vanstone S.A. Handbook of Applied Cryptography. CRC, 1996.
5. National Institute of Standards and Technology. Secure Hash Standard. Federal Information Processing Standards Publication, FIPS 180–1, 1995.
6. National Institute of Standards and Technology. Digital Signature Standard (DSS). Federal Information Processing Standards Publication, FIPS 186–2, 2000.
7. Rivest R.L., Shamir A., and Adleman L.M. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21:120–126,1978.
8. World Wide Web Consortium. XML-Signature Syntax and Processing. W3C Recommendation, 2002.

Digital Surface Model

► Digital Elevation Models (DEMs)

Digital Terrain Model (DTM)

► Digital Elevation Models (DEMs)

Digital Video Retrieval

► Content-Based Video Retrieval

Digital Video Search

► Content-Based Video Retrieval

Dimension

Torben Bach Pedersen
Aalborg University, Aalborg, Denmark

Definition

A *dimension* is a hierarchically organized set of *dimension values*, providing categorical information for characterizing a particular aspect of the data stored in a multidimensional *cube*.

Key Points

As an example, a three-dimensional cube for capturing sales may have a Product dimension, a Time dimension, and a Store dimension. The Product dimension captures information about the product sold, such as textual description, color, weight, etc., as well as groupings of products (product groups, product families, etc.). The Time dimension captures information about the time of the sale, at the Date level or finer, as well as groupings of time such as Week, Month, Weekday, Quarter and Year. It may also contain application-specific time-related information, e.g., what the temperature was on the particular day (interesting for ice cream sellers) or whether there was a special event in town on that day, e.g., a big sports event. The Store dimension captures information about stores (Name, Size, Layout), as well as various groupings of Stores (City, State, Sales District).

The notion of a dimension is an essential and distinguishing concept for multidimensional cubes, where dimensions are a first-class object. Dimensions are used for two purposes: the *selection* of data and the *grouping* of data at a desired level of detail. A dimension is organized into a containment-like *hierarchy* composed of a number of *levels*, each of which represents a level of detail that is of interest to the analyses to be performed. The instances of the dimension are typically called *dimension values*. Each such value belongs to a particular level.

In some multidimensional models, a dimension level may have associated with it a number of *level properties* that are used to hold simple, non-hierarchical information. For example, the Weekday of a particular date can be a level property in the Date level of the Time dimension. This information could also be captured using an extra Weekday dimension. Using the level property has the effect of not increasing the dimensionality of the cube.

Unlike the linear spaces used in matrix algebra, there is typically no ordering and/or distance metric on the dimension values in multidimensional models. Rather, the only ordering is the containment of lower-level values in higher-level values. However, for some dimensions, e.g., the Time dimension, an ordering of the dimension values is available and is used for calculating cumulative information such as "total sales in year to date."

When implemented in a relational database, a dimension is stored in one or more *dimension tables* using either a so-called *star schema* (one table per dimension, with a surrogate key and one column per dimension level or level property) or a so-called *snowflake schema* (one table per dimension level, each with a surrogate key and an attribute for the textual name of the dimension value, as well as one attribute per level property).

Cross-references

► Cube
► Hierarchy
► Multidimensional Modeling
► Snowflake Schema
► Star Schema

Recommended Reading

1. Kimball R., Reeves L., Ross M., and Thornthwaite W. The Data Warehouse Lifecycle Toolkit. Wiley Computer, 1998.
2. Pedersen T.B., Jensen C.S., and Dyreson C.E. A foundation for capturing and querying complex multidimensional data. Inf. Syst., 26(5):383–423, 2001.
3. Thomsen E. OLAP Solutions: Building Multidimensional Information Systems. Wiley, New York, 1997.

Dimensional Modeling

► Multidimensional Modeling

Dimensionality Curse

► Curse of Dimensionality

Dimensionality Reduction

Heng Tao Shen
The University of Queensland, Brisbane, QLD, Australia

Synonyms

DR

Definition

From database perspective, dimensionality reduction (DR) is to map the original high-dimensional data into a lower dimensional representation that captures the content in the original data, according to some criterion. Formally, given a data point $P = \{p_1, p_2, \ldots, p_D\}$ in

D- dimensional space, DR is to find a *d*-dimensional subspace, where $d < D$, such that P is represented by a *d*-dimensional point by projecting P into the *d*-dimensional subspace.

Key Points

Advances in data collection and storage capabilities have led to an information overload in most sciences. Many new and emerging data types, such as multimedia, time series, biological sequence, have been studied extensively in the past and present new challenges in data analysis and management due to their high dimensionality of data space. One known phenomenon of "dimensionality curse" leads traditional data access methods to fail [3]. High-dimensional datasets present many mathematical challenges as well as some opportunities. In many cases, not all dimensions are equally "important" for understanding the underlying data. It is of interest in many applications to reduce the dimension of the original data prior to any modelling and indexing of the data, due to efficiency and effectiveness concerns. Generally, dimensionality reduction can be used for the following purposes:

Simplifying complex data: for many applications, particularly in database and information retrieval, high dimensionality of feature space leads to high complexity of data representation. The dimensionality has to be reduced to achieve satisfactory performance for an indexing structure. Typical methods include traditional Discrete Fourier transform (DFT) and Discrete Wavelet Transform (DWT), Adaptive Piecewise Constant Approximation (APCA), Principle Component Analysis (PCA) and its various improvements, Latent Semantic Indexing and its variants, and Locality Preservation Projection (LPP) etc. For these types of applications, the dimensionality reduction method must have an explicit mapping function to map the query points into the low-dimensional subspace for similarity search [2].

Modeling and analyzing data: for many applications, particularly in classification and pattern recognition, the underlying data structure is often embedded in a much lower-dimensional subspace. The task of recovering the meaningful low-dimensional structures hidden in high-dimensional data is also known as "manifold learning". Typical methods include Independent Component Analysis (ICA), Multidimensional Scaling (MDS), Isometric feature Mapping (Isomap) and its improvements, Locally Linear Embedding (LLE), Laplacian Eigenmaps, Semantic Subspace Projection (SSP), etc. For these types of applications, the dimensionality reduction is typically a very expensive process and performed on a small set of sample points for learning purpose. Since they are defined only on the sample/training data and have no explicit mapping function, they are not applicable to information retrieval and database applications [1].

Dimensionality reduction methods can be categorized to be linear or non-linear. Linear techniques are based on the linear combination of the original dimensions, while non-linear methods are mainly used to find an embedded non-linear manifold within the high dimensional space.

Cross-references

- ► Discrete Fourier Transform (Dft)
- ► Discrete Wavelet Transform (Dwt)
- ► Independent Component Analysis (Ica)
- ► Isometric Feature Mapping (Isomap)
- ► Latent Semantic Indexing (Lsi)
- ► Locality Preservation Projection (Lpp)
- ► Locally Linear Embedding (Lle) Laplacian Eigenmaps
- ► Multidimensional Scaling (Mds)
- ► Principle Component Analysis (Pca)
- ► Semantic Subspace Projection (Ssp)

Recommended Reading

1. Roweis S.T. and Saul L.K. Nonlinear dimensionality reduction by locally linear embedding. Science, 290(5500):2323–2326, 2000.
2. Shen H.T., Zhou X., and Zhou A. An adaptive and dynamic dimensionality reduction method for high-dimensional indexing. VLDB J, 16(2):219–234, 2007.
3. Weber R., Schek H.-J., Blott S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 194–205.

Dimensionality Reduction for Clustering, Attribute Selection for Clustering

► Feature Selection for Clustering

Dimension Reduction Techniques for Clustering

CHRIS DING
University of Texas at Arlington, Arlington, TX, USA

Synonyms

Subspace selection; Graph embedding

Definition

High dimensional datasets is frequently encountered in data mining and statistical learning. Dimension reduction eliminates noisy data dimensions and thus and improves accuracy in classification and clustering, in addition to reduced computational cost. Here the focus is on unsupervised dimension reduction. The wide used technique is *principal component analysis* which is closely related to *K*-means cluster. Another popular method is *Laplacian embedding* which is closely related to spectral clustering.

Historical Background

Principal component analysis (PCA) was introduced by Pearson in 1901 and formalized in 1933 by Hotelling. PCA is the foundation for modern dimension reduction. A large number of linear dimension reduction techniques were developed during 1950–1970s.

Laplacian graph embedding (also called *quadratic placement*) is developed by Hall [8] in 1971. *Spectral graph partitioning* [6], is initially studied in 1970s; it is fully developed and becomes popular in 1990s for circuit layout in VLSI community (see a review [1]), graph partitioning [10] and data clustering [3,7,11]. It is now standard technique [2,9].

Foundations

High dimensional datasets is frequently encountered in applications, such as information retrieval, image processing, computational biology, global climate research, For example, in text processing, the dimension of a word vector is the size of the vocabulary of a document collection, which is typically tens of thousands. In molecular biology, human DNA gene expression profiles typically involve thousands of genes, which is the problem dimension. In image processing, a typical 2D image has $128^2 = 16{,}384$ pixels or dimensions.

Clustering data in such high dimension is a challenging problem. Popular clustering methods such as *K*-means and EM methods suffer from the well-known local minima problem: as iterations proceed, the system is often trapped in the local minima in the configuration space, due to the greedy nature of these algorithms. In high dimensions, the iso-surface (where the clustering cost function remains constant) is very rugged, the system almost always gets trapped somewhere close to the initial starting configuration. In other words, it is difficult to sample through a large configuration space. This is sometimes called curse of dimension. Dimension reduction is widely used to relieve the problem. In this direction, the principal component analysis (PCA) is the most widely adopted. PCA is an example of linear dimension reduction or mapping.

A related problem is graph clustering. Given a graph with n nodes (objects) and a square matrix of pairwise similarities as the edge weights, the task is to cluster nodes into disjoint clusters. The state-of-the-art algorithm is spectral clustering using graph Laplacian eigenvectors. An effective implementation is to embed the graph in metric space and use the *K*-means algorithm to cluster the data. Here the graph embedding is a key step, reducing a problem of $n(n-1)/2$ data items (pairwise similarities) into a problem of nK data items where K is the dimension of the metric space. In this direction, the Laplacian embedding is the most widely adopted. Laplacian embedding is an example of nonlinear dimension reduction or mapping.

Dimension Reduction Versus Feature Selection

Dimension reduction often finds combinations of many variables which satisfy certain global optimal conditions. Feature selection (also called variable selection) considers individual variables separately, or combinations of small number variables. Although they share similar goals for clustering and classification, their approaches differ greatly. Only dimension reduction will be discussed here.

PCA and Other Linear Dimension Reduction

Linear dimension reduction seek linear combinations of variables that optimizes certain criteria. PCA seek linear combinations that maximizes the variances.

Consider a set of input data vectors $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ where $\mathbf{x}_i$ is a p-dimensional vector. Let $\bar{\mathbf{x}} = \sum_{i=1}^{n} \mathbf{x}_i/n$

be the center of the data. The covariance matrix is $C_X = (1/n \sum_{i=1}^{n} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$. Let $U = (\mathbf{u}_1, \ldots, \mathbf{u}_k)$ be the eigenvectors of C_X associated with the k largest eigenvalues of C_X. The PCA is the linear transformation of the original p-dimensional data $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ into k-dimensional data $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_n)$ with

$$\mathbf{y}_i = U^T \mathbf{x}_i. \tag{1}$$

The most important property of the transformed data Y is that they are uncorrelated:

$$C_Y = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{y}_i - \bar{\mathbf{y}})^T = U^T C_X U = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_k \end{pmatrix}. \tag{2}$$

Because of this, each dimension j of Y has a clean variance λ_j. The second important property of the transformed data is that dimensions of Y with small variances are ignored. Note that in the original data X, because different dimensions are correlated, there is no clear way to identify a dimension with small variance and eliminate it.

A third benefit is the relation to K-means clustering. The cluster centroids of the global solution of the K-means clustering form a subspace. This subspace is identical to the PCA subspace formed by transformed data $(\mathbf{u}_1, \ldots, \mathbf{u}_k)$ when k is the number of clusters, according to the theory of the equivalence [4] between PCA and K-means clustering. In other words, PCA automatically bring us to the narrow subspace containing the global solution of K-means clustering – a good starting place to find near global solutions. For this reason, PCA+K-means clustering is one of the most effective approach for clustering.

Linear Discriminant Analysis

PCA performs the unsupervised dimension reduction, without prior knowledge of class labels of each data object. If the class information is known, more appropriate dimension reduction can be devised. This is call linear discriminant analysis (LDA). In LDA, the optimal subspace $G = (\mathbf{g}_1, \ldots, \mathbf{g}_k)$ is obtained by optimizing

$$\max_U \operatorname{Tr} \frac{G^T S_b G}{G^T S_w G}, \tag{3}$$

where the between-class (S_b) and within-class (S_b) scatter matrices are defined as

$$S_b = \sum_k n_k(\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T,$$
$$S_w = \sum_k \sum_{i \in C_k} (\mathbf{x}_i - \mathbf{m}_k)(\mathbf{x}_i - \mathbf{m}_k)^T, \; C_X = S_b + S_w, \tag{4}$$

where $\mathbf{m}_k$ is the mean of class C_k and $\mathbf{m}$ is the global total mean. The central idea is to separate different classes as much as possible (maximize the between-class scatter S_b) while condense each class as much as possible (minimize the within-class scatter S_w). The solution of G is given the k eigenvectors of $S_w^{-1} S_b$ associated with the largest eigenvalues. The dimension k of the subspace is set to $k = C - 1$ where C is the number of classes.

Once G is computed, the transformed data is $\mathbf{y}_i = G^T \mathbf{x}_i$ or $Y = G^T X$. An important property of Y is that components of Y are uncorrelated.

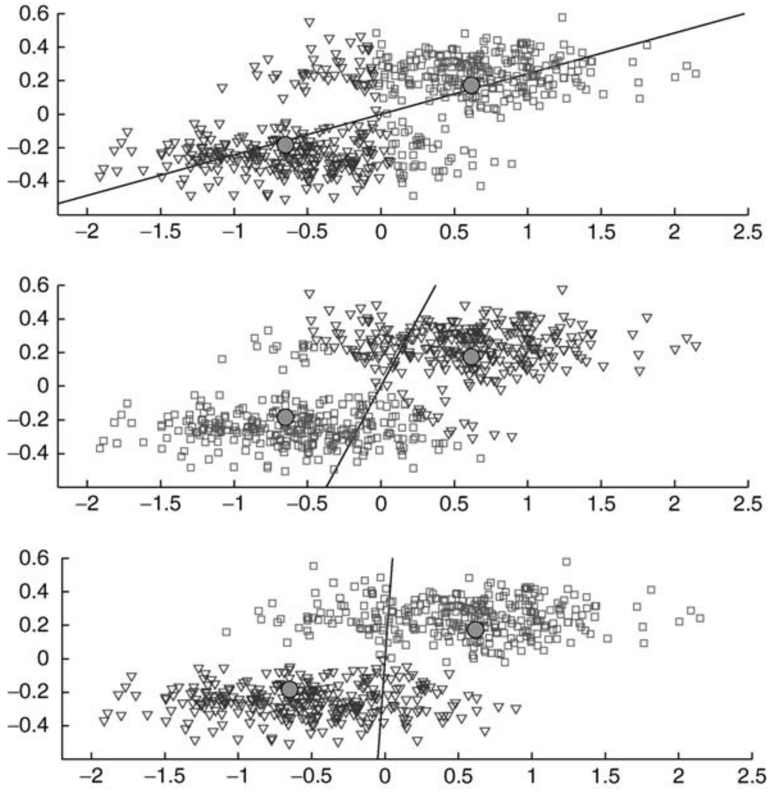
$$(C_Y)_{k\ell} = (G^T C_X G)_{k\ell} = \mathbf{g}_k^T C_X \mathbf{g}_\ell = \mathbf{g}_k^T S_w \mathbf{g}_\ell + \mathbf{g}_k^T S_b \mathbf{g}_\ell = 0, \quad k \neq \ell. \tag{5}$$

When the data dimension p is greater than n, the number of data points, S_w has zero eigenvalues and S_w^{-1} does not exist. This problem can resolved by first project data into PCA subspace with a dimension less than $p - C$ and then perform LDA on the projected data. LDA is very popularly in image processing where the data dimension is very high.

Adaptive Dimension Reduction – Combining Dimension Reduction and Clustering

The PCA subspace is not always the best subspace for data clustering; Figure 1 shows an example. LDA subspace is more suitable for data clustering; but LDA requires the knowledge of class labels. This dilemma is resolved by the adaptive dimension reduction (ADR).

ADR start with the PCA subspace and then adaptively compute the subspace by (A) clustering to generate class labels and (B) doing LDA to obtain the most discriminant subspace. (A) and (B) are repeated until convergence. Figure 2 shows the process: the computed 1D subspace (the direction) gradually moves towards the most discriminative direction. Theoretically, ADR optimizes the LDA objective function



Dimension Reduction Techniques for Clustering. Figure 1. A 2D dataset with 600 data points. *Green dots* are the centroids of each cluster. The *line* indicates the subspace *U*. *Top*: *K*-means clustering results PCA subspace. *Middle*: After two ADR iterations. *Bottom*: ADR converges after two more iterations. The subspace is the most discriminant.

$$\max_{G,H} \mathrm{Tr} \frac{G^T S_b G}{G^T S_w G} \qquad (6)$$

to obtain simultaneously the subspace G and the clusters represented by the cluster membership indicator matrix H, where $H_{ik} = 1/|C_k|^{1/2}$ if data point $\mathbf{x}_i$ belongs to cluster C_k; $H_{ik} = 0$ otherwise. $|C_k|$ is the size of C_k.

Metric Scaling

Metric scaling is the simplest form of *multidimensional scaling* and is also widely used in applications. Given a square matrix of pairwise distances (d_{ij}), the task is to embed the objects onto a lower dimensional embedding space where $||\mathbf{y}_i - \mathbf{y}_j|| \approx d_{ij}$.

Metric scaling is computed as the following. Let $A = (a_{ij}), a_{ij} = -\frac{1}{2} d_{ij}^2$. Compute B matrix: $B = (b_{ij})$, $b_{ij} = a_{ij} - \frac{1}{n}\sum_{i=1}^n a_{ij} - \frac{1}{n}\sum_{j=1}^n a_{ij} + \frac{1}{n^2}\sum_{i=1}^n \sum_{j=1}^n a_{ij}$. Compute the eigenvalues λ_k and eigenvectors $\mathbf{u}_k$, i.e., $B = \sum_{k=1}^n \lambda_k \mathbf{u}_k \mathbf{u}_k^T$. If $\lambda_k \geq 0, \forall k$, then the coordinates of object i in the embedding space is given by

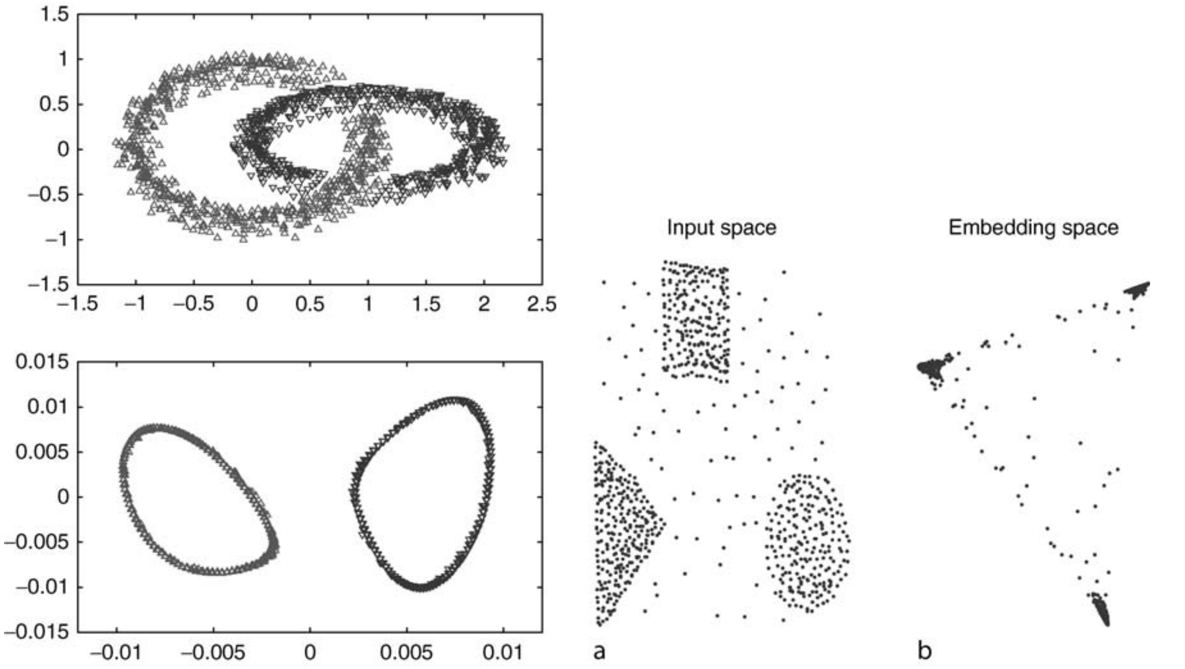
$$\mathbf{y}_i = [\lambda_1^{1/2} \mathbf{u}_1(i), \ldots, \lambda_m^{1/2} \mathbf{u}_m(i)]^T$$

m is the embedding dimension. It can be verified that $||\mathbf{y}_i - \mathbf{y}_j|| = d_{ij}$. In general, only a fraction of the largest $\lambda_k \geq 0$, and only positive eigenvalue subspace is embedded into these.

Most linear dimension reduction techniques are developed in early days from 1940s to 1970s. PCA, LDA, metric scaling are the most widely techniques. Many other techniques were also developed. *Canonical correlation analysis* extract linear dimensions that best capture the correlation between two set of variables. *Independent component analysis* extract linear dimensions one after another, using non-Gaussian criteria. *Partial least squares* constructs subspace similarly to PCA, but uses class label information.

Laplacian Embedding and Other Nonlinear Dimension Reduction

Laplacian Embedding The input is a square matrix W of pairwise similarities among n objects. We view W



Dimension Reduction Techniques for Clustering. Figure 2. Left (*upper panel*): Dataset A in 3D space. 700 data points distributed on two interlocking rings. Left (*lower panel*): Dataset A in eigenspace ($\mathbf{f}_2$, $\mathbf{f}_3$). Middle: Dataset B as shown in 2D space. Right: Dataset B in eigenspace ($\mathbf{f}_2$, $\mathbf{f}_3$).

as the edge weights on a graph with n nodes. The task is to embed the nodes of the graph in 1D space, with coordinates $(x_1,\ldots,x_n)$. The objective is that if i, j are similar (i.e., w_{ij} is large), they should be adjacent in embedded space, i.e., $(x_i - x_j)^2$ should be small. This is achieved by minimizing [8]

$$\min_{\mathbf{x}} J(\mathbf{x}) = \sum_{ij}(x_i - x_j)^2 w_{ij} = 2\sum_{ij} x_i (D-W)_{ij} x_j \\ = 2\mathbf{x}^T(D-W)\mathbf{x}, \tag{7}$$

where $D = \text{diag}(d_1,\ldots,d_n)$ and $d_i == \sum_j W_{ij}$. The minimization of $\sum_{ij}(x_i - x_j)^2 w_{ij}$ would get $x_i = 0$ if there is no constraint on the magnitude of the vector $\mathbf{x}$. Setting the normalization $\sum_i x_i^2 = 1$, and the constraint $\sum x_i = 0$ (x_i is centered around 0), the solution is given by the eigenvectors of

$$(D-W)\mathbf{f} = \lambda\mathbf{f}. \tag{8}$$

$L = D - W$ is called graph Laplacian. The solution is given by $\mathbf{f}_2$ associated with the second smallest eigenvalue (also called the Fiedler vector in recognition of his contribution to the graph connectivity).

This can be generalized to embedding in k-D space, with coordinates $\mathbf{r}_i \in \Re^k$. Let $||\mathbf{r}_i - \mathbf{r}_j||$ be the Euclidean distance between nodes i, j. The embedding is obtained by optimizing

$$\min_R J(R) = \sum_{i,j=1}^{n} ||\mathbf{r}_i - \mathbf{r}_j||^2 w_{ij} = 2\sum_{i,j=1}^{n} \mathbf{r}_i^T (D-W)_{ij} \\ \mathbf{r}_j = 2\,\text{Tr}\,R(D-W)R^T,\ \ R \equiv (\mathbf{r}_1,\ldots,\mathbf{r}_n). \tag{9}$$

With the normalization constraints $RR^T = I$, the solution is given by eigenvectors: $R = (\mathbf{f}_2,\ldots,\mathbf{f}_{k+1})^T$.

In solving (7) for 1D embedding, we may impose the normalization $\sum_i d_i x_i^2 = 1$, where $d_i = \sum_j w_{ij}$. With this condition, the solution for x of is given by the generalized eigenvalue problem

$$(D-W)\mathbf{u} = \lambda D\mathbf{u}. \tag{10}$$

For k-dimensional embedding of (9), the normalization is $RDR^T = I$. The solution is given by the eigenvectors: $R = (\mathbf{u}_2,\ldots,\mathbf{u}_{k+1})^T$. This approach is motivated by the normalized cut clustering (see (12) below).

Figure 2 gives two examples of the Laplacian embedding. The left panel [5] shows that the Laplacian embedding can effectively separate two interlocking data rings. It also shows that the embedding is not topology preserving. The right panel shows the self-aggregation property [5] of the embedding: data points of the same cluster self-aggregate and collapse onto a single centroid.

Relation to Spectral Clustering The most interesting aspect of Laplacian embedding is related to graph clustering, and its predecessor, the graph partitioning. In early 1990s, high performance computing is a very active research. One of the challenging task is to partition a mesh/graph into equal-size partition so that the problem could be solved on distributed processors. The popular technique of spectral graph partitioning [10] utilizes the eigenvectors of the Laplacian (the Fiedler vector $\mathbf{f}_2$). Specifically, a graph can be effectively partitioned into two equal parts depending on the sign $\mathbf{f}_2$: node i belongs to partition A if $\mathbf{f}_2(i) \geq 0$; it belongs to partition B if $\mathbf{f}_2(i) < 0$.

Soon it is recognized [7] that the 2-way Ratio Cut clustering objective function

$$J_{ratio\text{-}cut} = \frac{s(A,B)}{|A|} + \frac{s(A,B)}{|B|}, \quad s(A,B) \equiv \sum_{i\in A}\sum_{j\in B} w_{ij} \tag{11}$$

can be solved by the same Fiedler vector $\mathbf{f}_2$. This 2-way clustering is generalized to multi-way clustering [3] using $K-1$ eigenvectors $(\mathbf{f}_2,\ldots,\mathbf{f}_K)$. $\mathbf{f}_1 = (1,\ldots,1)/n^{1/2}$ is a constant vector. Later it was realized that using the sum of node degree to balance clusters has some advantages. This is the normalized cut [11]

$$J_{normalized-cut} = \frac{s(A,B)}{d_A} + \frac{s(A,B)}{d_B}, \quad d_A = \sum_{i\in A} d_i,$$
$$d_B = \sum_{i\in B} d_i, \quad d_i = \sum_j w_{ij}. \tag{12}$$

The solution to this problem is given by the second eigenvector $\mathbf{u}_2$ in (10). The K-way clustering uses eigenvectors $(\mathbf{u}_2,\ldots,\mathbf{u}_K)$. By 2001, the Laplacian based clustering methodology is fully developed.

Other Nonlinear Embedding Methods

Nonlinear embedding is a rather diverse and rapidly growing research area. They include (i) Multidimensional scaling, which embed a set of objects in in a lower-dimensional space while preserving the given pair-wise distance; (ii) Extension of PCA to nonlinear case, such as principal curves, kernel PCA, etc; (iii) Manifold learning, which uncovers a low-dimensional manifold embedded in the high-dimensional data space; isomap, local linear embedding, tangent space alignment, etc. (iv) Many other approaches, such as neuronal network based approach, called nonlinear PCA, etc.

A short description for some approaches. *Isomap* uses geodesic distances along the manifold by constructing the *k*NN subgraphs as the lower-dimensional manifold. It then use metric scaling to map objects into metric space. *Local Tangent Space Alignment* is a rigorous approach by building the local tangent spaces (which are local PCAs) and alignment them into a global system. *Kernel PCA* computes principal eigenvectors of the kernel matrix and embed in the eigenvector space. It should noted that even though manifold learning algorithm uncover nonlinear data structures, their primary goal is not necessarily data clustering.

Key Applications

PCA is used widely in a broad range of applications, from computer vision to text mining, gene expression profiles. Any data with high dimensions are often preprocessed with PCA. Laplacian embedding is used for network analysis, graph clustering.

Cross-references

► Dimensionality Reduction
► *K*-Means and *K*-Medoids
► Multi-Dimensional Scaling
► Principal Component Analysis
► Social Networks

Recommended Reading

1. Alpert C.J. and Kahng A.B. Recent directions in netlist partitioning: a survey. Integ. VLSI J., 19:1–81, 1995.
2. Belkin M. and Niyogi P. Laplacian eigenmaps and spectral techniques for embedding and clustering. NIPS, 2001.
3. Chan P.K., Schlag M., and Zien J.Y. Spectral k-way ratio-cut partitioning and clustering. IEEE Trans. CAD-Integ. Circuit. Syst., 13:1088–1096, 1994.
4. Ding C. and He X. K-means clustering and principal component analysis. Int'l Conf. Machine Learning (ICML), 2004.
5. Ding C., He X., Zha H., and Simon H. Unsupervised learning: self-aggregation in scaled principal component space. Proc. Sixth European Conf. on Principles of Data Mining and Knowledge Discovery (PKDD), 2002, pp. 112–124.
6. Fiedler M. Algebraic connectivity of graphs. Czech. Math. J., 23:298–305, 1973.

7. Hagen M. and Kahng A.B. New spectral methods for ratio cut partitioning and clustering. IEEE. Trans. Comput. Aided Desig., 11:1074–1085, 1992.
8. Hall K.M. R-dimensional quadratic placement algorithm. Manage. Sci., 17:219–229, 1971.
9. Ng A.Y., Jordan M.I., and Weiss Y. On spectral clustering: Analysis and an algorithm. In Proc. Neural Information Processing Systems (NIPS 2001), 2001.
10. Pothen A., Simon H.D., and Liou K.P. Partitioning sparse matrices with egenvectors of graph. SIAM J. Matrix Anal. Appl., 11:430–452, 1990.
11. Shi J. and Malik J. Normalized cuts and image segmentation. IEEE. Trans. Pattern Anal. Mach. Intell., 22:888–905, 2000.

Dimension-Extended Topological Relationships

Eliseo Clementini
University of L'Aquila, L'Aquila, Italy

Definition

This definition includes a group of models for topological relationships that have in common the use of two topological invariants – the set intersection empty/non empty content and the dimension – for distinguishing various relationships between spatial objects. These models had a strong impact in database technology and the standardization process.

Historical Background

Early descriptions of topological relationships (e.g., [10]) did not have enough formal basis to support a spatial query language, which needs formal definitions in order to specify exact algorithms to assess relationships. The importance of defining a sound and complete set of topological relationships was recognized in [13]. The first formal models were all based on point-set topology. In [12], the authors originally described the 4-intersection model (4IM) for classifying topological relationships between one-dimensional intervals. In [8], the authors adopted the same method for classifying topological relationships between regions. The 9-intersection model (9IM) is an extension of the 4IM based on considering the exterior of objects, besides interior and boundary [9]. In [6], the authors described the dimension-extended method (DEM), so called because they extended the 4IM with the dimension of the intersections. In the same paper, they introduced the Calculus-Based Method (CBM), made up of five relations and three boundary operators. A combination of the DEM and the 9IM was called the DE + 9IM in [1]. In this latter paper, the authors proved that the CBM was more expressive than the 4IM, 9IM, and DEM and was equivalent to the DE + 9IM. By expressive power they meant the number of topological relationships the models were able to distinguish. Later on, several extensions of all these models were developed: for example, the extension of the CBM to composite regions [5] and complex objects [2], and the extension of the 4IM to regions with holes [7].

Foundations

In the following, simple objects of the plane are considered, that is, regularly closed regions with connected interior and exterior, curved lines with only two endpoints and without self-intersections, and single points. The symbol λ will be used to denote any geometric object (simple region, simple line, or point), and $\partial\lambda$, λ°, $\overline{\lambda}$, λ^- will denote the boundary, the interior, the closure, and the exterior of λ, respectively. The function $\dim(\lambda)$returns the dimension of λ, with possible values in the two-dimensional space of 0, 1, 2 or nil ($-$) for the empty set. In case the object λ consists of multiple parts, the highest dimension is returned.

To assess the relationship between two geometric objects, various point-sets can be considered, which can be empty or non-empty. This is generally called the content invariant and can be calculated for several sets: intersections, set differences, symmetric differences [11]. The most convenient one is the intersection, since it gives a comprehensive categorization of topological relationships. Six groups of relationships can be distinguished: region/region (R/R), line/region (L/R), point/region (P/R), line/line (L/L), point/line (P/L), and point/point (P/P).

Definition 1. The 4IM is a 2 × 2 matrix of the intersections of the interiors and boundaries of the two objects λ_1 and λ_2:

$$\begin{pmatrix} \lambda_1^\circ \cap \lambda_2^\circ & \lambda_1^\circ \cap \partial\lambda_2 \\ \partial\lambda_1 \cap \lambda_2^\circ & \partial\lambda_1 \cap \partial\lambda_2 \end{pmatrix}$$

Each intersection may be empty (Ø) or non-empty (¬Ø), resulting in a total of $2^4 = 16$ combinations. Each case is represented by a matrix of values. It is possible to apply some simple geometric constraints to assess that not all combinations are possible. For example, for the R/R group the 4IM is able to recognize eight different relationships. All 16 combinations are instead possible for the L/L group. The geometric criteria to discover real cases are discussed in [1]. Overall, 43 real cases can be identified (see Table 1).

Definition 2. The 9IM is a 3 × 3 matrix containing the empty/non-empty values for interior, boundary, and exterior intersections:

$$\begin{pmatrix} \lambda_1^\circ \cap \lambda_2^\circ & \lambda_1^\circ \cap \partial\lambda_2 & \lambda_1^\circ \cap \lambda_2^- \\ \partial\lambda_1 \cap \lambda_2^\circ & \partial\lambda_1 \cap \partial\lambda_2 & \partial\lambda_1 \cap \lambda_2^- \\ \lambda_1^- \cap \lambda_2^\circ & \lambda_1^- \cap \partial\lambda_2 & \lambda_1^- \cap \lambda_2^- \end{pmatrix}$$

By considering the empty or nonempty content of such nine sets, the total is $2^9 = 512$ theoretical combinations. Excluding the impossible cases, 68 possible cases are remaining, as shown in Table 1.

The introduction of other invariants allows finer topological distinctions. A refinement of the content invariant is given by the dimension of each intersection set.

Definition 3. The DEM is a 2 × 2 matrix containing the dimension of the intersections of the interiors and boundaries of the two objects λ_1 and λ_2:

$$\begin{pmatrix} \dim(\lambda_1^\circ \cap \lambda_2^\circ) & \dim(\lambda_1^\circ \cap \partial\lambda_2) \\ \dim(\partial\lambda_1 \cap \lambda_2^\circ) & \dim(\partial\lambda_1 \cap \partial\lambda_2) \end{pmatrix}$$

Theoretically, with the four possible values for the dimension the DEM matrix might result into $4^4 = 256$ different cases. Geometric criteria can be adopted to reduce this number of cases by referring to specific groups of relationships, for a total of 61 real cases (see Table 1).

Definition 4. The DE + 9IM is a 3 × 3 matrix containing the dimension of interior, boundary, and exterior intersections:

$$\begin{pmatrix} \dim(\lambda_1^\circ \cap \lambda_2^\circ) & \dim(\lambda_1^\circ \cap \partial\lambda_2) & \dim(\lambda_1^\circ \cap \lambda_2^-) \\ \dim(\partial\lambda_1 \cap \lambda_2^\circ) & \dim(\partial\lambda_1 \cap \partial\lambda_2) & \dim(\partial\lambda_1 \cap \lambda_2^-) \\ \dim(\lambda_1^- \cap \lambda_2^\circ) & \dim(\lambda_1^- \cap \partial\lambda_2) & \dim(\lambda_1^- \cap \lambda_2^-) \end{pmatrix}$$

There are in general for this method $4^9 = 262144$ different cases. Reducing this number with geometric criteria, 87 real topological relationships are obtained (see Table 1).

The CBM is made up of five relations and three boundary operators. In [1], the authors proved that the CBM is equivalent to the DE + 9IM regarding the number of topological relationships these two models are able to express. The model was extended for complex objects in [2]. The definitions for simple objects are the following.

Definition 5. The *touch* relationship (it applies to the R/R, L/L, L/R, P/R, P/L groups of relationships, but not to the P/P group):

$$< \lambda_1, \mathit{touch}, \lambda_2 > \Leftrightarrow (\lambda_1^\circ \cap \lambda_2^\circ = \emptyset) \wedge (\lambda_1 \cap \lambda_2 \neq \emptyset).$$

Definition 6. The *in* relationship (it applies to every group):

$$< \lambda_1, \mathit{in}, \lambda_2 > \Leftrightarrow (\lambda_1 \cap \lambda_2 = \lambda_1) \wedge (\lambda_1^\circ \cap \lambda_2^\circ \neq \emptyset).$$

Definition 7. The *cross* relationship (it applies to the L/L and L/R groups):

$$\begin{aligned} < \lambda_1, \mathit{cross}, \lambda_2 > \Leftrightarrow & (\dim(\lambda_1^\circ \cap \lambda_2^\circ) \\ & < \max(\dim(\lambda_1^\circ), \dim(\lambda_2^\circ))) \\ & \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2). \end{aligned}$$

Definition 8. The *overlap* relationship (it applies to R/R and L/L groups):

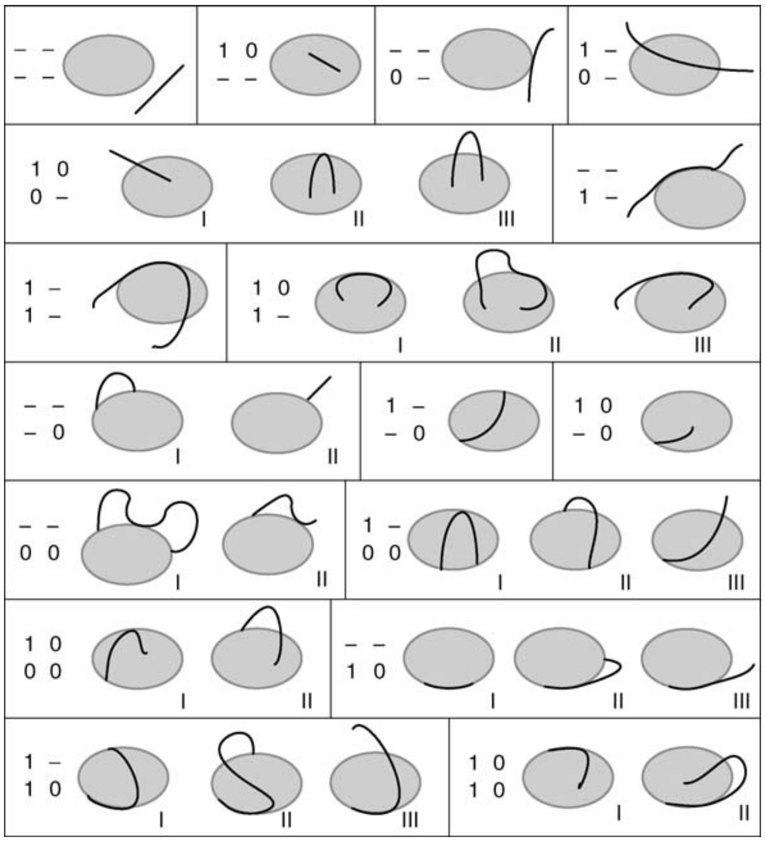
$$\begin{aligned} < \lambda_1, \mathit{overlap}, \lambda_2 > \Leftrightarrow & (\dim(\lambda_1^\circ) = \dim(\lambda_2^\circ) \\ & = \dim(\lambda_1^\circ \cap \lambda_2^\circ)) \\ & \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2). \end{aligned}$$

Definition 9. The *disjoint* relationship (it applies to every group):

$$< \lambda_1, \mathit{disjoint}, \lambda_2 > \Leftrightarrow \lambda_1 \cap \lambda_2 = \emptyset.$$

Dimension-Extended Topological Relationships. Table 1. A summary of topological relationships for all models and for all groups between simple objects

Model/group	R/R	L/R	P/R	L/L	P/L	P/P	Total
4IM	8	11	3	16	3	2	43
9IM	8	19	3	33	3	2	68
DEM	12	17	3	24	3	2	61
DE + 9IM ≡ CBM	12	31	3	36	3	2	87



Dimension-Extended Topological Relationships. Figure 1. The 31 different L/R relationships of the DE + 9IM model. Each box contains cases belonging to the same DEM case.

Definition 10. The boundary operator *b* for a region *A*: The pair (*A*, *b*) returns the circular line ∂A.

Definition 11. The boundary operators *from* and *to* for a line *L*: The pairs (*L*, *f*) and (*L*, *t*) return the two endpoints of the set ∂L.

To illustrate some of the relationships, Fig. 1 shows the list of the 31 L/R cases for the DE + 9IM.

Key Applications

The various models illustrated so far have been used by the Open Geospatial Consortium (OGC) for the definition of topological relationships in spatial databases. The same definitions have been adopted by the International Organization for Standardization (ISO). Various spatial database systems (e.g. Oracle, IBM DB2, PostgreSQL) have adopted the definitions suggested by OGC or slight variations of them to define the topological operators included in their spatial query language.

Future Directions

The process of adding granularity to topological relationships could be further extended by introducing more refined topological invariants [3]. Another direction for further research on the side of adding spatial operators to query languages is to consider other categories of spatial relationships besides topological, such as projective and metric [4].

Cross-references

► Topological Relationships
► Vague Spatial Data Types

Recommended Reading

1. Clementini E. and Di Felice P. A comparison of methods for representing topological relationships. Inf. Sci., 3(3):149–178, 1995.
2. Clementini E. and Di Felice P. A model for representing topological relationships between complex geometric features in spatial databases. Inf. Sci., 90(1–4):121–136, 1996.

3. Clementini E. and Di Felice P. Topological invariants for lines. IEEE Trans. Knowl. Data Eng., 10:38–54, 1998.
4. Clementini E. and Di Felice P. Spatial operators. ACM SIGMOD Rec., 29:31–38, 2000.
5. Clementini E., Di Felice P., and Califano G. Composite regions in topological queries. Inf. Syst., 20(7):579–594, 1995.
6. Clementini E., Di Felice P., and van Oosterom P. A small set of formal topological relationships suitable for end-user interaction. In Advances in Spatial Databases – Third Int. Symp., SSD '93. LNCS, vol. 692, 1993, pp. 277–295.
7. Egenhofer M.J., Clementini E., and Di Felice P. Topological relations between regions with holes. Int. J. Geogr. Inf. Syst., 8:129–142, 1994.
8. Egenhofer M.J. and Franzosa R.D. Point-set topological spatial relations. Int. J. Geogr. Inf. Syst., 5:161–174, 1991.
9. Egenhofer M.J. and Herring J.R. 1Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases. Department of Surveying Engineering, University of Maine, Orono, ME, 1991.
10. Freeman J. The modelling of spatial relations. Comput. Graph. Image Process, 4:156–171, 1975.
11. Herring J.R. The mathematical modeling of spatial and non-spatial information in geographic information systems. In Cognitive and Linguistic Aspects of Geographic Space, D. Mark and A. Frank (eds.). Kluwer Academic, Dordrecht, 1991, pp. 313–350.
12. Pullar D.V. and Egenhofer M.J. Toward the definition and use of topological relations among spatial objects. In Proc. Third Int. Symp. on Spatial Data Handling, 1988, pp. 225–242.
13. Smith T. and Park K. Algebraic approach to spatial reasoning. Int. J. Geogr. Inf. Syst., 6:177–192, 1992.

Direct Attached Storage

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms

DAS

Definition

Direct Attached Storage is a dedicated storage device which is directly connected to a server. The term Direct Attached Storage is often abbreviated to DAS. Derived from the original meaning, the term sometimes refers to a conventional server-centric storage system in which DAS devices are mainly connected; this definition is used in the context of explaining and comparing storage network architectures.

Key Points

A typical DAS device is a SCSI storage device such as an internal disk drive, a disk array or a tape library that is connected only to a single server by a SCSI bus cable. Such a DAS device is accessed only by the server to which the device is directly connected. SCSI storage devices sometimes have two or more SCSI interfaces which can connect to different servers. But such a storage device is also considered a DAS device rather than a SAN device, because only one server can actually access the storage device and the other servers are merely prepared for fault tolerance. Recently small servers sometimes accommodate Advanced Technology Attachment/Integrated Drive Electronics (ATA/IDE) disk drives and Universal Serial Bus (USB) disk drives, which can be also considered DAS devices.

Cross-references

► Network Attached Storage
► Storage Area Network
► Storage Network Architecture

Recommended Reading

1. Storage Network Industry Association. The Dictionary of Storage Networking Terminology. Available at: http://www.snia.org/.
2. Troppens U., Erkens R., and Müller W. Storage Networks Explained. Wiley, New York, 2004.

Direct Manipulation

Alan F. Blackwell[1], Maria Francesca Costabile[2]
[1]University of Cambridge, Cambridge, UK
[2]University of Bari, Bari, Italy

Synonyms

Graphical interaction; Desktop metaphor

Definition

The term *direct manipulation* was introduced by Shneiderman [2,3] to describe user interfaces that offer the following desirable properties:

1. Visibility of the objects and actions of interest
2. Physical actions on the object of interest instead of complex syntax
3. Effects of those actions that are rapid, incremental and reversible

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:51

Direct manipulation is generally associated with Graphical User Interfaces, although the above properties are also beneficial in interfaces where the user is working with text or numeric data. Direct manipulation has been most popularly applied in the "desktop metaphor" for managing disk files, a visual representation where the objects of interest are the user's files, and they are therefore continuously presented on the screen. This was an improvement over disk operating system consoles in which file listings were only transient results of a directory listing command.

The currently most common direct manipulation interfaces are called WIMP interfaces, referring to the interaction devices they use, namely windows, icons, menus and pointers. However, direct-manipulation ideas are at the core of many advanced non-desktop interfaces. Virtual reality, augmented reality, tangible user interfaces are newer concepts that extend direct manipulation.

Key Points

The main alternatives to direct manipulation are command line interfaces and programming languages. These are characterized by abstract syntax, representation of computational processes rather than data of interest, and effects that will take place in the future rather than immediately, perhaps suddenly and irreversibly. Some GUIs (e.g. modal dialog boxes) have these latter properties, in which case they will not provide the benefits of direct manipulation. For many database users, the object of interest is their data, rather than the computational features of the database. This offers a challenge to designers of interactive database systems, for whom the system itself might be an object of interest.

Direct manipulation interfaces represent a second generation, which evolved from the first generation of command line interfaces. DM interfaces often make interaction easier for users who are not computer science experts, by allowing them to point at and move objects rather than instructing the computer by typing commands in a special syntax.

Hutchins, Hollan and Norman [1] proposed a generalization of the direct manipulation principles, in terms of minimizing the cognitive effort that is required to bridge the gulf between the user's goals and the way those goals must be specified to the system. They drew attention to a *gulf of execution*, which must be bridged by making the commands and mechanisms of the system match the thoughts and goals of the user, and a *gulf of evaluation*, bridged by making the output present a good conceptual model of the system that is readily perceived, interpreted, and evaluated. Both gulfs can be bridged to some degree by use of an appropriate visual metaphor.

Cross-references

► Usability
► Visual Interaction
► Visual Interfaces
► Visual Metaphor

Recommended Reading

1. Hutchins E.L., Hollan J.D., and Norman D.A. Direct manipulation interfaces. In User Centered System Design, New Perspectives on Human-Computer Interaction, 1Norman, D.A. Draper (eds.). S.W. Lawrence Erlbaum, Hillsdale, NJ, 1986.
2. Shneiderman, B. The future of interactive systems and the emergence of direct manipulation. Behav. Inf. Technol., 1:237–256, 1982.
3. Shneiderman, B. Direct manipulation: a step beyond programming languages. IEEE Comput., 16(8):57–69, 1983.

Direct Manipulation Interfaces

► Visual Interfaces

Directional Relationships

► Cardinal Direction Relationships

Dirichlet Tessellation

► Voronoi Diagram

Disaster Recovery

KENICHI WADA
Hitachi Limited, Tokyo, Japan

Definition

The recovery of necessary data, access to that data, and associated processing through a comprehensive process in which a redundant site (including both equipment and work space) is set up, and operations are recovered to enable business operations to continue after a loss of all or part of a data center by disaster including fire, earthquake, hurricane, flood, terrorism and power grid failure. Such a recovery involves not only an essential set of data but also all the hardware and software needed to continue processing of the data.

Key Points

Disaster recovery should be a key focus of an organization's business continuity plan in case of disaster the following are key metrics:

RPO (Recovery Point Objective): The maximum acceptable time period prior to a failure or disaster during which changes to data may be lost as consequence of recovery. At a minimum, all data changes that occur before this period preceding the failure or disaster will be available after data recovery. RTO (Recovery Time Objective): The maximum acceptable time period required to bring one or more applications and associated data back from an outage to a correct operational state.

The most common disaster recovery plans include the following strategies:

- Backups are made to tape and sent off-site at regular intervals.
- Backups are made to disk on-site and automatically copied to an off-site disk, or made directly to an off-site disk.
- Data is replicated to an off-site location, using replication method including database replication, file system replication and replication by storage system.

Cross-references

► Backup and Restore
► Replication

Disclosure Risk

Josep Domingo-Ferrer
The Public University of Tarragona, Tarragona, Spain

Synonyms

Re-identification risk; Attribute disclosure; Identity disclosure

Definition

In the context of statistical disclosure control, disclosure risk can be defined as the risk that a user or an intruder can use the protected dataset $\mathbf{V}'$ to derive confidential information on an individual among those in the original dataset $\mathbf{V}$. This approach to disclosure risk was formulated in Dalenius [1].

Key Points

Disclosure risk can be regarded from two different perspectives, according to Paass [2]:

1. Attribute disclosure. Attribute disclosure takes place when an attribute of an individual can be determined more accurately with access to the released statistic than it is possible without access to that statistic.

2. Identity disclosure. Identity disclosure takes place when a record in the protected dataset can be linked with a respondent's identity. Two main approaches are usually employed for measuring identity disclosure risk: uniqueness and re-identification.

2.1. Uniqueness Roughly speaking, the risk of identity disclosure is measured as the probability that rare combinations of attribute values in the released protected data are indeed rare in the original population the data come from. This approach is typically used with nonperturbative statistical disclosure control methods and, more specifically, sampling. The reason that uniqueness is not used with perturbative methods is that, when protected attribute values are perturbed versions of original attribute values, it makes no sense to investigate the probability that a rare combination of protected values is rare in the original dataset, because *that* combination is most probably *not found* in the original dataset.

2.2. Re-identification This is an empirical approach to evaluate the risk of disclosure. In this case, software is constructed to estimate the number of re-identifications that might be obtained by a specialized intruder. The use of re-identification methods as a way to measure disclosure risk goes back, at least, to Spruill [3]. Re-identification methods provide a more unified approach than uniqueness methods because the former can be applied to any kind of masking and not just to non-perturbative masking. Moreover, re-identification can also be applied to synthetic data (see e.g., [4]).

Cross-references

► Inference Control in Statistical Databases
► Information Loss Measures
► Microdata
► Record Matching
► SDC Score

Recommended Reading

1. Dalenius T. Towards a methodology for statistical disclosure control. Statistisk Tidskrift, 5:429–444, 1977.
2. Paass G. Disclosure risk and disclosure avoidance for microdata. J. Bus. Econ. Stat., 6:487–500, 1985.
3. Spruill N.L. The confidentiality and analytic usefulness of masked business microdata. In Proc. Section on Survey Research Methods, Alexandria, VA. American Statistical Association, 1983, pp. 602–607.
4. Winkler W.E. Re-identification methods for masked microdata. In Privacy in Statistical Databases, vol. 3050, J. Domingo-Ferrer and V. Torra (eds.). LNCS, Springer, Berlin Heidelberg, 2004, pp. 216–230.

DISCO

► Discovery

Discounted Cumulated Gain

Kalervo Järvelin, Jaana Kekäläinen
University of Tampere, Tampere, Finland

Synonyms

Normalized discounted cumulated gain (nDCG); Discounted cumulated gain (DCG)

Definition

Discounted Cumulated Gain (DCG) is an evaluation metric for information retrieval (IR). It is based on non-binary relevance assessments of documents ranked in a retrieval result. It assumes that, for a searcher, highly relevant documents are more valuable than marginally relevant documents. It further assumes, that the greater the ranked position of a relevant document (of any relevance grade), the less valuable it is for the searcher, because the less likely it is that the searcher will ever examine the document – and at least has to pay more effort to find it. DCG formalizes these assumptions by crediting a retrieval system (or a query) for retrieving relevant documents by their (possibly weighted) degree of relevance which, however, is discounted by a factor dependent on the logarithm of the document's ranked position. The steepness of the discount is controlled by the base of the logarithm and models the searcher's patience in examining the retrieval result. A small base (say, 2) models an impatient searcher while a large base (say, 10) a patient searcher.

In its normalized form, as Normalized Discounted Cumulated Gain (nDCG), the actual DCG performance for a query is divided by the ideal DCG performance for the same topic, based on the recall base of the topic in a test collection.

Historical Background

Modern large retrieval environments tend to overwhelm their users by their large output. Since all documents are not equally relevant, highly relevant documents, or document components, should be identified and ranked first for presentation to the users. In order to develop IR techniques in this direction, it is necessary to develop evaluation approaches and methods that credit IR methods for their ability to retrieve highly relevant documents and rank them higher. For this goal, non-binary or graded relevance assessments and evaluation measures are needed. Most traditional IR evaluation measures, e.g. precision and recall, are based on binary relevance. The nDCG measure family presented in this entry is one solution for measuring IR effectiveness with graded relevance.

Foundations

Ranked Retrieval Result

In an IR experiment, using a test collection, a topic set and a recall base for each topic, the retrieval system gives for each query representing a topic a ranked output, which in the DCG based evaluation is examined from ranked position 1 to position *n* for each query.

Document Relevance Scores and Weights

Documents have non-binary relevance scores given in the recall base. Their range may be any continuous scale of real numbers, e.g. [−10.0, +10.0], [−1.0, +1.0], or [0.0, 1.0], or a discreet integer scale, say {−3,. . .,+3} or {0,. . .,10}. In the evaluation, the relevance scores may be reweighed if one wishes to emphasize the differences of relevance scores from the user point of view. In the following it is assumed that the relevance scores 0–3 are used (3 denoting high value, 0 no value).

Gain Vector

In the DCG evaluation, the relevance score of each document, or its reweighed value, is used as a gained value measure for its ranked position in the result. Assuming relevance scores 0–3 and result lists up to rank 200, one obtains corresponding gain vectors of 200 components each having the value 0, 1, 2 or 3. For example: G' = <3, 2, 3, 0, 0, 1, 2, 2, 3, 0,. . .>.

Cumulated Gain Vector

The cumulated gain at ranked position i is computed by summing from position 1 to i when i ranges from 1 to n, e.g., n = 200. Formally, the position i in the gain vector G is denoted by G[i]. Now the cumulated gain vector CG is defined recursively as the vector CG where:

$$\mathrm{CG}[i] = \begin{cases} \mathrm{G}[1], & \text{if } i = 1 \\ \mathrm{CG}[i-1] + \mathrm{G}[i], & \text{otherwise} \end{cases}$$

For example, from G' the vector CG' = <3, 5, 8, 8, 8, 9, 11, 13, 16, 16, . . . > is obtained. The cumulated gain at any rank may be read directly, e.g., at rank 7 it is 11.

Discounting Principle

The discounting principle states that the greater the rank of a document in the retrieval result, the smaller share of the document score is added to the cumulated gain. A discounting function is needed which progressively reduces the document score as its rank increases but not too steeply (e.g., as division by rank) to allow for searcher persistence in examining further documents. A simple way of discounting with this requirement is to divide the document score by the log of its rank. By selecting the base of the logarithm, sharper or smoother discounts can be computed to model varying searcher behavior.

Discounted Cumulated Gain Vector

Formally, if b denotes the base of the logarithm, the cumulated gain vector with discount is defined recursively as the vector DCG where:

$$\mathrm{DCG}[i] = \begin{cases} \mathrm{CG}[i], & \text{if } i < b \\ \mathrm{DCG}[i-1] + \mathrm{G}[i]/\log_b i, & \text{if } i \geq b \end{cases}$$

One must not apply the logarithm-based discount at rank 1 because $\log_b 1 = 0$. Moreover, the discount is not applied for ranks less than the logarithm base (that would give them a boost). This is also realistic, since the larger the base, the smaller the discount and the more likely the searcher is to examine the results at least up to the base rank (say 10).

For example, let $b = 2$. From G' given in the preceding section one obtains DCG' = <3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61, . . .>.

Average Vector

In an IR experiment, tests on IR methods are typically run with a set of test topics. To obtain an understanding of the average performance of an IR method, one needs average vectors across a query set. To compute the averaged vectors, one needs the vector sum operation and vector multiplication by a constant. Let V = <v_1, v_2,. . .,v_k> and W = <w_1, w_2,. . .,w_k> be two vectors. Their sum is the vector V + W = <v_1+ w_1, v_2+ w_2,. . .,v_k+ w_k>. For a set of vectors $\mathcal{V}$ = {V_1, V_2,. . .,V_n}, each of k components, the sum vector is generalised as $\Sigma_{V \in \mathcal{V}} V = V_1 + V_2 + \ldots + V_n$. The multiplication of a vector V = <v_1, v_2,. . .,v_k> by a constant r is the vector r*V = <r*v_1, r*v_2,. . .,r*v_k>. The average vector **AV** based on vectors $\mathcal{V}$ = {V_1, V_2,. . .,V_n}, is given by the function *avg-vect*(**V**):

$$\textit{avg-vect}(\mathrm{V}) = |\mathrm{V}|^{-1} * \Sigma_{V \in V} \mathrm{V}$$

Now the average CG and DCG vectors for vector sets **CG** and **DCG**, over a set of test queries, are computed by *avg-vect*(**CG**) and *avg-vect*(**DCG**).

Ideal Vector

In order to normalize the actual CG and DCG vectors one compares them to the theoretically best possible vectors for each topic. The latter vectors are constructed by arranging the recall base of each topic in descending order by relevance and then turning it into a gain vector, CG vector and DCG vector as described above. A sample ideal gain vector is: I' = <3, 3, 3, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0, . . .> and its CG vector is $\mathrm{CG_I}$' = <3, 6, 9, 11, 13, 15, 16, 17, 18, 19, 19, 19, 19, . . .>.

Normalized Vector

The (D)CG vectors for each IR technique can be normalized by dividing them by the corresponding ideal (D)CG vectors, component by component. In this way, for any vector position, the normalized value 1 represents ideal performance, and values in the range [0,1) the share of ideal performance cumulated by each technique. Given an (average) (D)CG vector

V = <v_1, v_2,. . . ,v_k> of an IR technique, and the (average) (D)CG vector I = <i_1, i_2,. . .,i_k> of ideal performance, the normalized performance vector n(D)CG is obtained by the function:

$$\textit{norm-vect}(\mathrm{V,I}) = < \mathrm{v}_1/\mathrm{i}_1, \mathrm{v}_2/\mathrm{i}_2, \ldots, \mathrm{v}_k/\mathrm{i}_k >$$

Normalized Discounted Cumulated Gain (nDCG) Vector

To assess whether two IR methods are significantly different in effectiveness from each other or not, when measured by DCG, one uses the normalized vectors because the (D)CG vectors are not relative to an ideal. Given an (average) DCG vector V of an IR method, and the (average) DCG vector I of ideal performance, the normalized performance vector nDCG is obtained by *norm-vect*(V, I).

For example, based on CG' and CG_I' from above, one obtains the normalized CG vector nCG' = *norm-vect*(CG', CG_I') = <1, 0.83, 0.89, 0.73, 0.62, 0.6, 0.69, 0.76, 0.89, 0.84, . . . >.

The Average Normalized Discounted Cumulated Gain Indicator

The average of a (n)(D)CG vector, up to a given ranked position, summarizes the vector (or performance) and is analogous to the non-interpolated average precision of a document cut-off value (DCV) curve up to the same given ranked position. The average of a (n)(D)CG vector V up to the position *k* is given by:

$$\textit{avg-pos}(\mathrm{V}, k) = k^{-1*} \sum_{i=1\ldots k} \mathrm{V[i]}$$

These vector averages can be used in statistical significance tests in the same way as average precision in IR evaluation.

Properties of (n)(D)CG

The strengths of the proposed CG, DCG, nCG, and nDCG measures can now be summarized as follows:

- They combine the degree of relevance of documents and their rank (affected by their probability of relevance) in a coherent way.
- At any number of retrieved documents examined (rank), CG and DCG give an estimate of the cumulated gain as a single measure no matter what is the recall base size.
- They are not heavily dependent on outliers (relevant documents found late in the ranked order) since they focus on the gain cumulated from the beginning of the result up to any point of interest.
- They are obvious to interpret, they are more direct than precision-recall curves by explicitly giving the number of documents for which each n(D)CG value holds. Precision-recall curves do not make the number of documents explicit for given performance and may therefore mask bad performance [7].

In addition, the DCG measure has the following further advantages:

- It realistically weights down the gain received through documents found later in the ranked results.
- It allows modeling searcher persistence in examining long ranked result lists by adjusting the discounting factor.

Further, the normalized nCG and nDCG measures support evaluation:

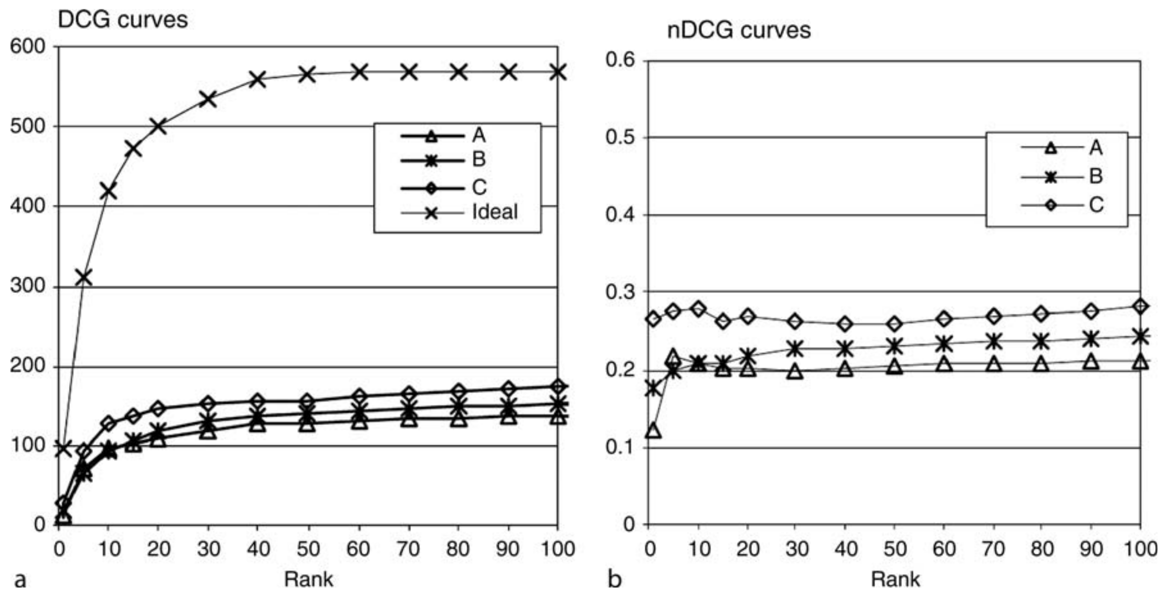
- They represent performance as relative to the ideal based on a known (possibly large) recall base of graded relevance assessments.
- The performance differences between IR techniques are also normalized in relation to the ideal thereby supporting the analysis of performance differences.

The following issues concerning the (n)(D)CG measure have been discussed in the literature:

1. Averaging over topics is not statistically reliable because the number of relevant documents (recall base size) varies by topics.
2. DCG does not discount the gain when the rank is smaller than the logarithm base.
3. Interpretation of non-normalized versions of the measure (CG, DCG) is difficult.
4. The measures should be robust when relevance judgements are incomplete.

These issues are discussed briefly below.

1. Averaging. This issue is discussed by, e.g., Hull [2] and Sakai [9]. Averaging over topics has been considered problematic when a fixed length for the result list (i.e. a fixed cut-off value, DCV) is used. Say, one evaluates effectiveness at DCV 5. One topic has five relevant documents with relevance score 3 retrieved at positions 1–5; another topic has 100 relevant documents with relevance score 3 and five of them



Discounted Cumulated Gain. Figure 1. (a) DCG curves. (b) nDCG curves.

are retrieved at positions 1–5. The (n)DCG values for the topics will be the same at DCV 5. From the point of view of searcher's effort this is correct; however, if the system-oriented effectiveness as an equal share of the relevant documents is aimed at, evaluation with fixed DCV should be avoided. With high DCVs, e.g. 500 or 1,000, the problem is less acute but the user-orientation is also lost.

A related problem is caused by the variation in the number of documents of different relevance level: As the number of most relevant documents tends to be low, weighting them strongly might lead to instability in evaluation (a loss of one highly relevant document in the top ranks hurts effectiveness badly) [11]. As a remedy large topic pools with varying recall bases could be used in evaluation.

One should also bear in mind the difference between (n)DCG curves and averaging the scores over topics at a given rank. The curves visualize the performance over a range of ranks and are useful to reveal crossovers concealed in averages. The single figure averages are needed to give concise information and for statistical testing.

2. Discounting. The original formulation of the (n)DCG, given above, does not perform discounting for ranks less than the base of the discount logarithm. When the base is relatively large, say 10, modeling a patient searcher, the 10 first ranks avoid discounting. Some scholars have been concerned about this feature [10]. It can be solved by modifying the discounting factor $\log_b i$ to the form $(1 + \log_b i)$ [8]. By doing so the first case (with the condition $i < b$) of the DCG formula can be omitted: all ranks are consistently discounted.
3. Interpretation of the scores. The CG and DCG measures are comparable only within one test setting and relevance weighting scheme. The relevance grades and their weights should be reported in order to make CG and DCG curves interpretable. Further, topics with large recall bases yield high CG and DCG figures and affect averages over all topics. The normalized versions are recommendable when averaging over topics or comparing over test settings is needed.
4. Robustness. An evaluation measure should be robust with relation to missing relevance judgements since, in practice, the whole test collection is seldom assessed for relevance. The pooling method for gathering documents for relevance judgement, and the later reuse of the test collection with new systems or methods leads unavoidably to a situation where all documents in the result lists are not assessed for relevance. The (n)DCG measure has been found robust in this respect, see [1,10].

Key Applications

The (n)(D)CG measures have been applied in traditional laboratory oriented evaluation with stable test collections when graded relevance assessment are available. They have also gained popularity in web IR evaluation, and interactive IR test settings.

Sample (n)DCG curves for three TREC 7 runs (A, B, C) are given in Fig. 1a and b. The measures use graded relevance with four relevance levels. The levels from non-relevant to the most relevant are weighted 0–1–10–100; logarithm base for discounting is 2.

Future Directions

Session-Based DCG

IR evaluation by (n)(D)CG assumes one query per topic/session. In real life however, interactive searchers often use multiple queries through reformulation and/or relevance feedback until they are satisfied or give up and move on to other means of information access. Evaluation metrics assuming one query per topic are insufficient in multiple query session evaluation, where the searcher's reformulation and feedback effort matters. Moreover, due to various reasons, the first queries often are unsuccessful. In real life, also stopping decisions are individual and variable and depend on many factors, including personal traits, task, context, and retrieval results. To overcome this limitation, it is possible to extend the (n)(D)CG into a session-based metric for multiple interactive queries. Such an extended, session-based DCG metric would incorporate query sequences as a further dimension in evaluation scenarios, allowing one to further discount any relevant documents found only after additional searcher effort, i.e., feedback or reformulation. The rationale here is that an IR system (or searcher-system combination) should be rewarded less for relevant results found by later queries.

Cross-references

► Information Retrieval
► Precision
► Recall
► Standard Effectiveness Measures

Recommended Reading

1. Bompada T., Chang C., Chen J., Kumar R., and Shenoy R. On the robustness of relevance measures with incomplete judgments. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 359–366.
2. Hull D. Using statistical testing in the evaluation of retrieval experiments. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 329–338.
3. Järvelin K. and Kekäläinen J. IR evaluation methods for retrieving highly relevant documents. In Proc. 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2000, pp. 41–48.
4. Järvelin K. and Kekäläinen J. Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst., 20(4):422–446, 2002.
5. Kekäläinen J. Binary and graded relevance in IR evaluations – comparison of the effects on ranking of IR systems. Information Process. Manag., 41(5):1019–1033, 2005.
6. Kekäläinen J. and Järvelin K. Using graded relevance assessments in IR Evaluation. J. Am. Soc. Inf. Sci. Technol., 53(13):1120–1129, 2002.
7. Losee R.M. Text retrieval and filtering: Analytic models of performance. Kluwer Academic, Boston, MA, 1998.
8. Sakai T. Average gain ratio: A simple retrieval performance measure for evaluation with multiple relevance levels. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 417–418.
9. Sakai T. On the reliability of information retrieval metrics based on graded relevance. Inf. Process. Manag. 43(2):531–548, 2007.
10. A suggestion by Susan Price (Portland State University, Portland, OR, USA), May 2007, (private communication).
11. Voorhees E. Evaluation by highly relevant documents. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 74–82.

Discovery

Schahram Dustdar[1], Christian Platzer[1]
Bernd J. Krämer[2]
[1]Technical University of Vienna, Vienna, Austria
[2]Distance University of Hagen, Hagen, Germany

Synonyms

WS-discovery; DISCO

Definition

The term discovery, as far as Web services (WS) are concerned, refers to the process of finding WS that match certain computational needs and quality requirements of service users or their software agents. More technically speaking, WS discovery mechanisms take a specification of certain functional or non-functional criteria characterizing a service and try to locate machine-readable descriptions of Web services that meet the search criteria. The services found may have been previously unknown to the requester.

Historical Background

Since Web services were introduced in the early 1990's, service-oriented architectures had to deal with the discovery problem, and it still persists. Initially, the possibilities to describe Web services properly were limited. XML remote procedure calls (XML-RPCs), which were used early on to connect WS, offered no proper way of describing a Web service's capabilities and therefore forced system designers to find other ways to publish this information. These early services were typically used to achieve a platform independent communication between remote peers, nothing more. This requirement was met with an XML-structured messaging protocol that evolved later to SOAP, a standard protocol of today's Web technology. For the act of discovering services, the actually used protocol made no difference. Services description files were propagated mostly by artificial means, by sending the file per e-mail for instance. In some cases, the developer of the Web service also worked on the client-side implementation.

A proper service description mechanism was only introduced when application developers realized that Web service technology had to be leveraged to a level that obviated the need of service consumer and provider to interact closely with each other prior to using a service. With the definition of the Web service description language (WSDL), it was finally possible to describe the interface of a WS in a standardized manner. The discovery problem, however, still persisted because no means existed to publish a service description in a widely known index or registry, once the implementation on the provider side was completed.

To overcome this barrier, a first draft of the Universal Description, Discovery and Integration, short *UDDI*, standard was released in 1999. UDDI was designed as a vendor-independent specification of an XML-based registry for service descriptions maintained in the form of WSDL documents. The standard was completed in 2002. Its purpose was to enable service providers and consumers to find each other if service request and service offer matched and to provide information about message formats and protocols accepted by a Web service. Apart from defining data models and registry structure, UDDI was also designed to offer simple search capabilities to help service consumers find Web services. Thus UDDI contributed to solve the discovery issue.

As a WSDL description of a Web service interface just lists the operations the service may perform and the messages it accepts and produces, the discovery mechanism of UDDI was constrained to match functionality only. If several candidate services could be found, the service consumer was unable to distinguish between them. Therefore people felt the need to be able to express semantic properties or quality aspects of requested services as well. But search mechanisms taking into account semantics and quality-of-service properties require richer knowledge about a registered Web service than WSDL can capture.

To complement the expressiveness of WSDL and facilitate service discovery, DAML-S, an ontology language for Web services, was proposed to associate computer-readable semantic information with service descriptions. Semantic service descriptions are seen as a potential enabler to enhance automated service discovery and matchmaking in various service oriented architectures. For the time being, however, services widely used in practice lack semantic information because no easy to use or even automated method to attach semantic information to service descriptions exists.

Foundations

The service discovery process encompasses several steps. Each step involves specific problems that have to be solved independently. The following list will discuss these steps in ascending order, beginning with the most generic step.

Enabling Discovery

At a first glance, the act of discovering a service description matching a set of terms characterizing a service, resembles a search processes for Web pages. Well-known search engines like Google or Live utilize a crawling mechanism to retrieve links to Web documents and create a index that can be searched effectively as users enter search terms. A crawler just analyzes a given Web page for hyperlinks and grinds through the tree structure generated by such hyperlinks to find other Web documents. For Web services, or more precisely Web service descriptions, the case is similar except for one major difference: WSDL files do not contain links to other services. Approaches to write crawlers that search web pages for possibly published service descriptions will produce very poor results, in general [4]. UDDI registries are just designed to eliminate the need for crawlers or alike. As a matter of fact, open UDDI registries for public Web services are increasingly loosing importance. Especially after the two largest UDDI registries from IBM and Microsoft were shut down in 2005, the vision of public services suffered immensely. Suddenly the starting point to find a public Web service was lost, leaving the possibility to query common Web search engines for Web services as the only alternative. There are, of course, some other registries but they usually do not implement the UDDi specification, which suggests that UDDI may not be an optimal solution for public service registries.

In a corporate environment, however, the initial discovery step is not a real issue. Web service descriptions can easily be published on an internal Web page or in a UDDI registry and are, therefore, easily accessible from within the institution.

Searching in Service Registries

Assuming that a comprehensive collection of service descriptions has already been established, the question is how to retrieve the closest match to a user query in an efficient manner.

As the title suggests, searching is usually performed by humans and not by software automatically. The challenge is how to create an index of services such that the addition and retrieval of service descriptions can be achieved accurately and fast. Common information retrieval methods are often used for this purpose, ranging from the vector space model for indexing and searching large repositories to graph theoretical approaches for fast processing of a rich data collection.

More or less every registry-based solution encompasses such a facility. Especially UDDI registries often come with a rudimentary interface to query the database for contained services. Unfortunately, the general structure of UDDI with its tModel component-layout, which serves as a container to store detailed service information, complicates data retrieval. Furthermore, most UDDI entries do not maintain a complete service description but include links to such descriptions kept elsewhere. But these links could be broken or inaccessible at search time. As a result, UDDI queries are usually processed on the business data related to a service and not the service description itself. This fact alone limits the usability of UDDI-based search mechanisms enormously or more precisely: it leaves most of the index quality in the hand of the users.

This area on the other hand is heavily investigated throughout the research community and several approaches have been presented that aim at improving search capabilities on service collections. Those approaches are mostly designed to handle natural language queries like "USA weather service" and are supposed to provide a user interface for various registry implementations.

Querying Repositories

A more detailed form of search in service descriptions is entitled as querying. Unlike direct search, in which a user simply provides a set of search terms, queries are formal expressions using some sort of query language. In the case of relational databases, SQL is typically used as a query language. Through a query expression it is possible to search for a specific service signature in a set of service descriptions. Assume, for example, that a user wants to find a weather service and can provide three bits of information: country, zip_code, and the desired scale for presenting the temperature. Assume further that the user wants to express certain quality requirements. Then, a query expression in a language alike SQL might read as follows:

SELECT description FROM services s WHERE

s.input.COMPOSEDOF(country AND zip_code AND useCelsiusScale)

AND s.response_time < 200ms AND s.downtime < 1%

This example also reveals a weakness of service descriptions as their signatures are usually not specified using exact type information such as *city* or *country* but rather basic types like string, integer etc. are used. Hence, it seems more appropriate to search for signatures in terms of basic data types only. But this would likely result in mismatches. Re-considering the query above, a corresponding signature using the basic types [string, integer, boolean] can easily be met by other services. There is no way to distinguish positive matches from unwanted ones without additional information or a richer index. These problems are addressed by introducing semantics and domain knowledge. For the values of response_time and downtime in this example, there already exist approaches that constantly monitor and invoke the services in a registry to create a statistical profile of the quality of such a service (*QoS*).

These approaches require a more sophisticated registry type than UDDI represents and are therefore mostly implemented in research prototypes.

Domain-Specific Knowledge in Service Descriptions
Another requirement that has to be met by more powerful discovery mechanisms is domain-specific knowledge about a service. To take on the sample above, a discovery mechanism able to match the terms city, zip_code and temperature with the semantic categories location and weather would select just the intersection of services dealing with location and temperature. Although domain information is semantic information in certain respects, it does not mean that the information has to be provided upon service registration. Certain approaches exist that are able to group service repositories according to their most probable domain. The grouping can, for instance, be achieved by using statistical cluster analysis and discover strongly related service descriptions.

On the other hand, domain-knowledge can also be gained by letting the service provider add this information. In practice however, it proved to be problematic to let users define semantic information for a service. Once, this is due to the fact that a certain amount of domain knowledge is needed by the programmer of the Web service but mostly because the categorization assigned by indexers cannot be validated and could therefore be incorrect. This field, just like the following, is still heavily investigated, e.g. under the heading "faceted search." It addresses a broad spectrum of issues but also bears a high potential for innovation.

Semantic Annotations
The next logical step towards enhancing service discovery is a complete semantic description of a Web service. A multitude of approaches exist, in which semantic annotations are used to define the concepts behind operations of Web services and their input and output messages. Those approaches use the widely known resource description framework (*RDF*) to add custom-designed semantic markup to service descriptions. A good example for such a semantic markup language is called DAML-S. It is a DAML-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. An example for such a markup is shown in Listing 1. This very short listing basically shows a Web service called `BookstoreBuy` and defines it as a process and therefore as a subclass of the class Process in the corresponding ontology. The second part shows an input to `BookstoreBuy` which is it is a subproperty of the property input of Process, from the process model. With this example, some of the limitations for semantic annotations become more obvious. First of all, the creator of the Web service is required to have additional knowledge about semantic annotations and ontologies. Furthermore, the appended information causes an additional amount of work. Secondly, the ontology used defines sharp boundaries for the level of detail that can be reached through the usage of such annotations. In addition, the problem of misused or erroneous annotations mentioned in the previous step persists. Put together, semantic Web services are seen as a technology with the potential to facilitate more powerful service discovery and machine-readable descriptions. Practical experience, however, showed that an exploitation of semantic information is difficult and still leaves room for further improvements.

Quality-of-Service Properties
The consideration of quality-of-service (QoS) properties in discovery attempts requires the definition of scales of measurements and metrics to qualify the properties per domain. The scales can be of different

```
<rdfs:Class rdf:ID="BookstoreBuy">
  <rdfs:subClassOf rdf:resource=
       "http://www.www.daml.org/services/daml-s/2001/05/Process#Process"/>
</rdfs:Class>

<rdf:Property rdf:ID="bookName">
  <rdfs:subPropertyOf rdf:resource=
       "http://www.www.daml.org/services/daml-s/2001/05/Process#input"/>
   <rdfs:domain rdf:resource="#BookstoreBuy"/>
   <rdfs:range rdf:resource="#rdfs:Literal"/>
</rdf:Property>
```

Discovery. Listing 1. DAML-S Sample.

kinds including nominal, ordinal, interval or ratio. They are used to assign appropriate QoS property values to a service. Here, a service provider has the choice to associate precise values or just value ranges with service property descriptions. The metrics are needed to rank services that match the functional and semantic requirement of a search according their degree of fulfillment of required QoS properties.

These issues and related modifications to service discovery schemes are still subject to a number of research projects. Wang et al. [6], for example, proposes a services discovery approach in which functional, semantic and QoS requirements are taken into account by iteratively applying related filters.

Key Applications

Some of the concepts presented above, especially the first, more general layers are already used in real world implementations. Service registries, especially those not strictly conforming to the UDDI specification are the main area of application for innovative discovery mechanisms. Search and matchmaking on the other hand is particularly required by IDEs. Especially service composition environments enormously benefit from fast and exact discovery mechanisms. Finding substitutes and alternatives for services in compositions is an important topic in service-oriented architectures.

Future Directions

Future directions show considerable tendencies towards the semantic web to enhance service discovery for Web services. This fact, however, creates a diversion among researchers on this particular area. Some argue that semantic descriptions are too complicated to be of any practical use, while others argue that they are the only way to leverage Web service discovery and search to a point where they can be processed automatically. Both views are valid and which direction proves to be the most promising will be decided by the work that is yet to come.

Some recent works have proposed a recommendation service and suggest to apply it to collaborative web service discovery. Experiments with such solutions are underway in research labs.

URL to Code

A Vector-space based search engine for Web services including a clustering algorithm: http://copenhagen.vitalab.tuwien.ac.at/VSMWeb/

A Web service registry with invocation capabilities: http://www.xmethods.net

Another Web service search engine: http://www.esynaps.com/search/default.aspx

WSBen, a Web service discovery and composition benchmark developed at Penn State: http://pike.psu.edu/sw/wsben/

Cross-references

► Business Process Management
► Publish/Subscribe
► Service-Oriented Architecture (SOA)
► Web Services

Recommended Reading

1. Benatallah B., Hacid M.-S., Leger A., Rey C., and Toumani F. On automating web services discovery. Int. J. VLDB, 14(1):84–96, 2005.
2. Bussler C., Fensel D., and Maedche A. A conceptual architecture for semantic web enabled web services. SIGMOD Rec., 2002.
3. Kokash N., Birukou A., and D'Andrea V. Web service discovery based on past user experience. In Proc. Int. Conf. on Business Information Systems (BIS). LNCS 4439. 2007, pp. 95–107.
4. Platzer C. and Dustdar S. A vector space search engine for Web services. In Proc. Third European IEEE Conf. on Web Services (ECOWS'05). 2005.
5. Rosenberg F., Platzer C., and Dustdar S. Bootstrapping performance and dependability attributes of web services. In Proc. IEEE Int. Conf. on Web Services (ICWS'06), 2006, pp. 205–212.
6. Wang X., Vitvar T., Kerrigan M., and Toma I. Synthetical evaluation of multiple qualities for service selection. In Proc. Fourth Int. Conf. on Service Oriented Computing. Springer, 2006, pp. 1–12.

Discrete Wavelet Transform and Wavelet Synopses

Minos Garofalakis
Technical University of Crete, Chania, Greece

Definition

Wavelets are a useful mathematical tool for hierarchically decomposing functions in ways that are both efficient and theoretically sound. Broadly speaking, the wavelet transform of a function consists of a coarse overall approximation together with detail coefficients that influence the function at various scales. The wavelet transform has a long history of successful applications in signal and image processing [11,12]. Several recent studies have also demonstrated the effectiveness of the wavelet transform (and *Haar wavelets*, in particular) as a tool for approximate query processing over massive relational tables [2,7,8] and continuous data streams [3,9]. Briefly, the idea is to apply wavelet transform to the input relation to obtain a compact data synopsis that comprises a select small collection of *wavelet coefficients*. The excellent energy compaction and de-correlation properties of the wavelet transform allow for concise and effective approximate representations that exploit the structure of the data. Furthermore, wavelet transforms can generally be computed in linear time, thus allowing for very efficient algorithms.

Historical Background

A growing number of database applications require on-line, interactive access to very large volumes of data to perform a variety of data-analysis tasks. As an example, large Internet Service Providers (ISPs) typically collect and store terabytes of detailed usage information (NetFlow/SNMP flow statistics, packet-header information, etc.) from the underlying network to satisfy the requirements of various network-management tasks, including billing, fraud/anomaly detection, and strategic planning. This data gives rise to massive, multi-dimensional *relational data tables* typically stored and queried/analyzed using commercial database engines (such as, Oracle, SQL Server, DB2). To handle the huge data volumes, high query complexities, and interactive response-time requirements characterizing these modern data-analysis applications, the idea of effective, easy-to-compute *approximate query answers* over precomputed, compact *data synopses* has recently emerged as a viable solution. Due to the exploratory nature of most target applications, there are a number of scenarios in which a (reasonably-accurate) fast approximate answer over a small-footprint summary of the database is actually preferable over an exact answer that takes hours or days to compute. For example, during a "drill-down" query sequence in ad-hoc data mining, initial queries in the sequence frequently have the sole purpose of determining the truly interesting queries and regions of the database. Providing fast approximate answers to these initial queries gives users the ability to focus their explorations quickly and effectively, without consuming inordinate amounts of valuable system resources.

The key behind such approximate techniques for dealing with massive data sets lies in the use of appropriate *data-reduction techniques* for constructing compact synopses that can accurately approximate the important features of the underlying data distribution. The *Haar wavelet decomposition* is one such technique with deep roots in the fields of signal and image processing, that has recently found its way into database applications as an important approximate query processing tool.

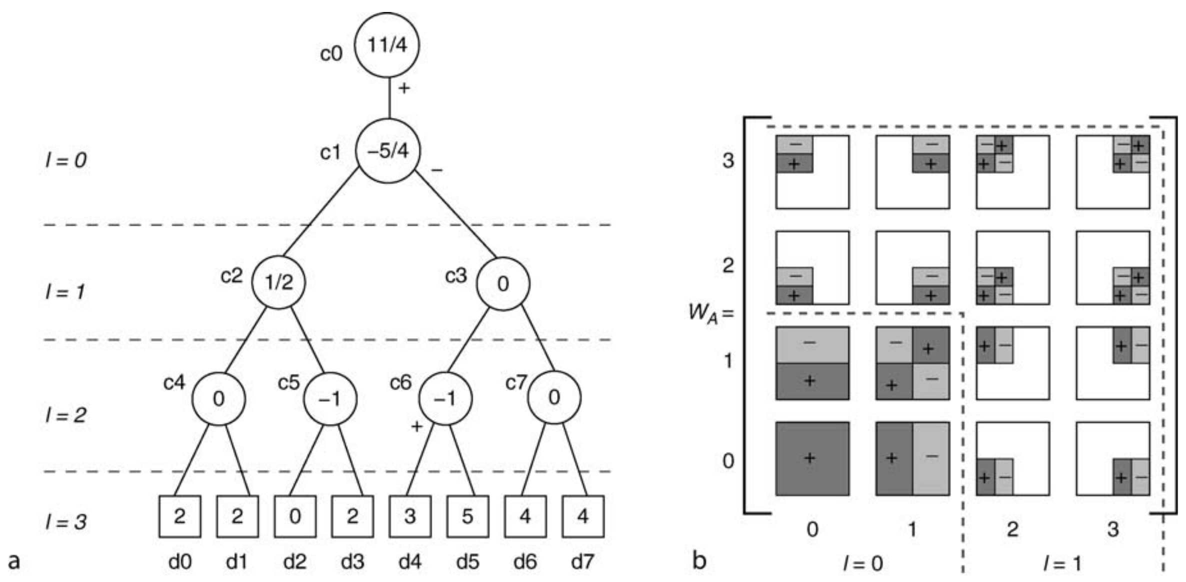
Foundations

Haar Wavelet Basics

Haar wavelets are conceptually simple, easy to compute, and have been found to perform well in practice for a variety of applications, ranging from image editing and querying to database selectivity estimation tasks. Consider a one-dimensional data vector A containing the $N=8$ data values $A=[2,2,0,2,3,5,4,4]$. The Haar wavelet transform of A can be computed as follows. The values are first averaged together pairwise to get a new "lower-resolution" representation of the data with the following average values $[2,1,4,4]$. To restore the original values of the data array, additional *detail coefficients* must be stored to capture the information lost due to this averaging. In Haar wavelets, these detail coefficients are simply the differences of the (second of the) averaged values from the computed pairwise average, that is, $[2-2,1-2,4-5,4-4]=[0,-1,-1,0]$. No information has been lost in this process – it is simple to reconstruct the eight values of the original data array from the lower-resolution array containing the four averages and the four detail coefficients. Recursively applying the above pairwise averaging and differencing process on the lower-resolution array containing the averages, gives the following full transform:

The *wavelet transform* W_A of A is the single coefficient representing the overall average of the data values followed by the detail coefficients in the order of increasing resolution, i.e., $W_A=[11/4,-5/4,1/2,0,0,-1,-1,0]$ (each entry is called a *wavelet coefficient*). For vectors containing similar values, most of the detail coefficients tend to be very small; thus, eliminating them from the wavelet transform (i.e., treating them as zeros) introduces only small errors when reconstructing the original data, resulting in a very effective form of lossy data compression [12].

A helpful tool for conceptualizing the recursive Haar wavelet transform process is the *error tree* structure (shown in Fig. 1a for the example array A). Each



Discrete Wavelet Transform and Wavelet Synopses. Figure 1. (a) Error-tree structure for the example data array A (N= 8). (b) Support regions and signs for the 16 nonstandard two-dimensional Haar basis functions.

internal node c_i (i=0,...,7) is associated with a wavelet coefficient value, and each leaf d_i (i=0,...,7) is associated with a value in the original data array; in both cases, the index i denotes the positions in the (data or wavelet transform) array. For instance, c_0 corresponds to the overall average of A. The resolution levels l for the coefficients (corresponding to levels in the tree) are also depicted.

Resolution	Averages	Detail coefficients
3	[2, 2, 0, 2, 3, 5, 4, 4]	–
2	[2, 1, 4, 4]	[0, −1, −1, 0]
1	[3/2, 4]	[1/2, 0]
0	[11/4]	[−5/4]

Given an error tree T and an internal node t of T, $t \neq c_0$, `leftleaves`(t) (`rightleaves`(t)) denotes the set of leaf (i.e., data) nodes in the subtree rooted at t's left (resp., right) child. Also, given any (internal or leaf) node u, $path(u)$ is the set of all (internal) nodes in T that are proper ancestors of u (i.e., the nodes on the path from u to the root of T, including the root but not u) with non-zero coefficients. Finally, for any two leaf nodes d_l and d_h, $d(l:h)$ denotes the range sum $\sum_{i=l}^{h} d_i$. Using the error tree representation T, the following important reconstruction properties of the Haar wavelet transform can be outlined.

- (P1) The reconstruction of any data value d_i depends only on the values of the nodes in $path(d_i)$. More specifically, $d_i = \sum_{c_j \in path(d_i)} \delta_{ij} \cdot c_j$, where $\delta_{ij} = +1$ if $d_i \in \texttt{leftleaves}(c_j)$ or j=0, and $\delta_{ij} = -1$ otherwise; for example, $d_4 = c_0 - c_1 + c_6 = \frac{11}{4} - (-\frac{5}{4}) + (-1) = 3$.
- (P2) An internal node c_j contributes to the range sum $d(l:h)$ *only if* $c_j \in path(d_l) \cup path(d_h)$. More specifically, $d(l:h) = \sum_{c_j \in \text{path}(d_l) \cup \text{path}(d_h)} x_j$, where

$$x_j = \begin{cases} (h-l+1) \cdot c_j, \text{ if } j = 0 \\ (|\texttt{leftleaves}(c_j, l:h)| - \texttt{rightleaves}(c_j, l:h)|) \cdot c_j, \text{otherwise.} \end{cases}$$

where `leftleaves`(c_j, l: h)=`leftleaves`(c_j) ∩{ d_l, d_{l+1},..., d_h} (i.e., the intersection of `leftleaves`(c_j) with the summation range) and `rightleaves`(c_j, l: h) is defined similarly. (Clearly, coefficients whose subtree is completely contained within the summation range have a net contribution of zero, and can be safely ignored.) For example, $d(2:6) = 5c_0 + (2-3) c_1 - 2c_2 = 5 \times \frac{11}{4} - (-\frac{5}{4}) - 1 = 14$.

Thus, reconstructing a single data value involves summing at most log N+1 coefficients and reconstructing a range sum involves summing at most 2 log N+1 coefficients, regardless of the width of the range. The *support region* for a coefficient c_i is defined as the set of (contiguous) data values that c_i is used to reconstruct.

The Haar wavelet transform can be naturally extended to *multi-dimensional* data arrays using two distinct methods, namely the *standard* and *nonstandard* Haar transform [12]. As in the one-dimensional case, the Haar transform of a d-dimensional data array A results in a d-dimensional wavelet-coefficient array W_A with the same dimension ranges and number of entries. Consider a d-dimensional wavelet coefficient W in the (standard or nonstandard) wavelet-coefficient array W_A. W contributes to the reconstruction of a d-dimensional rectangular region of cells in the original data array A (i.e., W's support region). Further, the sign of W's contribution ($+W$ or $-W$) can vary along the quadrants of W's support region in A.

As an example, Fig. 1b depicts the support regions and signs of the sixteen nonstandard, two-dimensional Haar coefficients in the corresponding locations of a 4×4 wavelet-coefficient array W_A. The blank areas for each coefficient correspond to regions of A whose reconstruction is independent of the coefficient, i.e., the coefficient's contribution is 0. Thus, $W_A[0,0]$ is the overall average that contributes positively (i.e.,"$+ W_A[0,0]$") to the reconstruction of all values in A, whereas $W_A[3,3]$ is a detail coefficient that contributes (with the signs shown) only to values in A's upper right quadrant. Each data cell in A can be accurately reconstructed by adding up the contributions (with the appropriate signs) of those coefficients whose support regions include the cell. Error-tree structures for d-dimensional Haar coefficients are essentially *d-dimensional quadtrees*, ϵwhere each internal node t corresponds to a *set* of (at most) $2^d - 1$ Haar coefficients, and has 2^d children corresponding to the quadrants of the (common) support region of all coefficients in t; furthermore, properties (P1) and (P2) can also be naturally extended to the multi-dimensional case [2,7,8].

Data Reduction and Approximate Query Processing

Consider a relational table R with d data attributes X_1, X_2,... X_d. The information in R can be represented as a d-dimensional array A_R, whose j^{th} dimension is indexed by the values of attribute X_j and whose cells contain the count of tuples in R having the corresponding combination of attribute values. A_R is essentially the *joint frequency distribution* of all the data attributes of R. Given a limited amount of storage for building a *wavelet synopsis* of an input relation R, a thresholding procedure retains a certain number $B << N$ of the coefficients in the wavelet transform of A_R as a highly-compressed approximate representation of the original data (the remaining coefficients are implicitly set to 0). (The full details as well as efficient transform algorithms can be found in [2,13].) The goal of *coefficient thresholding* is to determine the "best" subset of B coefficients to retain, so that some overall error measure in the approximation is minimized – the next subsection discusses different thresholding strategies proposed in the database literature.

The construction of wavelet synopses typically takes place during the statistics collection process, whose goal is to create concise statistical approximations for the value distributions of either individual attributes or combinations of attributes in the relations of a Database Management System (DBMS). Once created, a wavelet synopsis is typically stored (as a collection of B wavelet coefficients) as part of the *DBMS-catalog information*, and can be exploited for several different purposes. The primary (and, more conventional) use of such summaries is as a tool for enabling effective (compile-time) estimates of the result sizes of relational operators for the purpose of *cost-based query optimization*. (Accurate estimates of such result sizes play a critical role in choosing an effective physical execution plan for an input SQL query.) For instance, estimating the number of data tuples that satisfy a range-predicate selection like $l \leq X \leq h$ is equivalent to estimating the range summation $f(l:h) = \sum_{i=l}^{h} f_i$, where f is the frequency distribution array for attribute X. As mentioned earlier, given a B-coefficient synopsis of the f array, computing $f(l:h)$ only involves retained coefficients in $path(f_l) \cup path(f_h)$ and, thus, can be estimated by summing only $min\{B, 2 \log N + 1\}$ synopsis coefficients [13]. A B-coefficient wavelet synopsis can also be easily expanded (in $O(B)$ time) into an $O(B)$-bucket *histogram* (i.e., piecewise-constant) approximation of the underlying data distribution with several possible uses (e.g., as a data visualization/approximation tool).

More generally, wavelet synopses can enable very fast and accurate approximate query answers [6] during interactive data-exploration sessions. As demonstrated by Chakrabarti et al. [2], an approximate query processing algebra (which includes all conventional aggregate and non-aggregate SQL operators, such as `select`, `project`, `join`, `sum`, and `average`) can operate *directly over the wavelet synopses of relations*, while guaranteeing the correct relational operator semantics. Query processing algorithms for these operators work *entirely* in the wavelet-coefficient domain. This allows for extremely fast response times, since the approximate query execution engine can do the bulk of its processing over compact wavelet synopses, essentially postponing the (expensive) expansion step into relational tuples until the end-result of the query.

Conventional and Advanced Wavelet Thresholding Schemes

Recall that coefficient thresholding achieves data reduction by retaining $B << N$ of the coefficients in the wavelet transform of A_R as a highly-compressed, lossy representation of the original relational data. The goal, of course, is to minimize the amount of "loss" quantified through some overall approximation error metric. Conventional wavelet thresholding (the method of choice for most studies on wavelet-based data reduction) greedily retains the *B largest Haar-wavelet coefficients in absolute value* after a simple normalization step (that divides each coefficient value at resolution level l by $\sqrt{2^l}$). It is a well-known fact that this thresholding method is in fact *provably optimal* with respect to minimizing the overall root-mean-squared error (i.e., L_2*-norm error*) in the data compression [12]. More formally, letting $\hat{d}_i$ denote the (approximate) reconstructed data value for cell i, retaining the B largest normalized coefficients implies that the resulting synopsis minimizes $L_2(\hat{d}) = \sqrt{\sum_i (\hat{d}_i - d_i)}^2$ (for the given amount of space B).

Conventional wavelet synopses optimized for overall L_2 error may not always be the best choice for approximate query processing systems. The quality of the approximate answers such synopses provide can vary widely, and users have no way of knowing the accuracy of any particular answer. Even for the simplest case of approximating a value in the original data set, the absolute and relative errors can show wide variation. Consider the example depicted in Table 1. The first line shows the 16 original data values (the exact answer), whereas the second line shows the 16 approximate answers returned when using conventional wavelet synopses and storing eight coefficients. Although the first half of the values is basically a mirror image of the second half, all the approximate answers for the first half are 65, whereas all the approximate answers for the second half are exact! Similar data values have widely different approximations, e.g., 30 and 31 have approximations 30 and 65, respectively. The approximate answers make the first half appear as a uniform distribution, with widely different values, e.g., 3 and 127, having the same approximate answer 65. Moreover, the results do not improve when one considers the presumably easier problem of approximating the sum over a range of values: for *all possible* ranges within the first half involving $x = 2$ to 7 of the values, the approximate answer will be $65 \cdot x$, while the actual answers vary widely. For example, for both the range d_0 to d_2 and the range d_3 to d_5, the approximate answer is 195, while the actual answer is 285 and 93, respectively. On the other hand, *exact* answers are provided for all possible ranges within the second half.

The simple example above illustrates that conventional wavelet synopses suffer from several important problems, including the introduction of severe bias in the data reconstruction and wide variance in the quality of the data approximation, as well as the lack of nontrivial guarantees for individual approximate answers. To address these shortcomings, recent work has proposed novel thresholding schemes for building wavelet synopses that try to minimize different approximation-error metrics, such as the *maximum relative error* (with an appropriate *sanity bound* s) in the approximation of individual data values based on the synopsis; that is, minimize $\max_i \left\{\frac{|\hat{d}_i - d_i|}{\max\{|d_i|, s\}}\right\}$. Such relative-error metrics are arguably the most important quality measures for approximate query answers. (The role of the sanity bound is to ensure that relative-error numbers are not unduly dominated by small data values.)

More specifically, Garofalakis and Gibbons [7] introduce *probabilistic* thresholding schemes based on ideas from randomized rounding, that probabilistically

Discrete Wavelet Transform and Wavelet Synopses. Table 1. Errors with conventional wavelet synopses

Original data values	127	71	87	31	59	3	43	99	100	42	0	58	30	88	72	130
Wavelet answers	65	65	65	65	65	65	65	65	100	42	0	58	30	88	72	130

round coefficients either up to a larger rounding value (to be retained in the synopsis) or down to zero. Intuitively, their probabilistic schemes assign each non-zero coefficient *fractional storage* $y \in [0,1]$ equal to its retention probability, and then flip independent, appropriately-biased coins to construct the synopsis. Their thresholding algorithms are based on *Dynamic-Programming (DP)* formulations that explicitly minimize appropriate probabilistic metrics (such as the maximum normalized standard error or the maximum normalized bias) in the randomized synopsis construction; these formulations are then combined with a *quantization* of the potential fractional-storage allotments to give combinatorial techniques [7].

In more recent work, Garofalakis and Kumar [8] show that the pitfalls of randomization can be avoided by introducing efficient schemes for *deterministic* wavelet thresholding with the objective of optimizing a *general class of error metrics* (e.g., maximum or mean relative error). Their optimal and approximate thresholding algorithms are based on novel DP techniques that take advantage of the Haar transform error-tree structure. In a nutshell, their DP algorithms tabulate the optimal solution for the subtree rooted at each error-tree node c_j *given the error contribution that "enters" that subtree through the choices made at all ancestor nodes of* c_j *in the tree* (i.e., the choice of coefficients on $path(c_j)$). The key observation here is that, since the depth of the error tree is $O(\log N)$, all such possible selections can be tabulated while still keeping the running-time of the thresholding algorithm in the low-polynomial range. This turns out to be a fairly powerful idea for wavelet synopsis construction that can handle a broad, natural class of *distributive error metrics* (which includes several useful error measures for approximate query answers, such as maximum or mean weighted relative error and weighted L_p-norm error) [8].

The above wavelet thresholding algorithms for non-L_2 error metrics consider only the *restricted* version of the problem, where the algorithm is forced to select values for the synopsis from the standard Haar coefficient values. As observed by Guha and Harb [10], such a restriction makes little sense when optimizing for non-L_2 error, and can, in fact, lead to sub-optimal synopses. Their work considers *unrestricted* Haar wavelets, where the values retained in the synopsis are specifically chosen to optimize a general (weighted) L_p error metric. Their proposed thresholding schemes rely on a DP over the error tree (similar to that in [8]) that *also iterates over the range of possible coefficient values for each node*. To keep time and space complexities manageable, techniques for bounding these coefficient-value ranges are also discussed [10].

Extended and Streaming Wavelet Synopses

Complex tabular data sets *with multiple measures* (multiple numeric entries for each table cell) introduce interesting challenges for wavelet-based data reduction. Such massive, multi-measure tables arise naturally in several application domains, including OLAP (On-Line Analytical Processing) environments and time-series analysis/correlation systems. As an example, a corporate sales database may tabulate, for each available product (i) the number of items sold, (ii) revenue and profit numbers for the product, and (iii) costs associated with the product, such as shipping and storage costs. Similarly, real-life applications that monitor continuous time-series typically have to deal with several readings (measures) that evolve over time; for example, a network-traffic monitoring system takes readings on each time-tick from a number of distinct elements, such as routers and switches, in the underlying network and typically several measures of interest need to be monitored (e.g., input/output traffic numbers for each router or switch interface) even for a fixed network element. Deligiannakis et al. [4] show that obvious approaches for building wavelet synopses for such multi-measure data can lead to poor synopsis-storage utilization and suboptimal solutions even in very simple cases. Instead, their proposed solution is based on (i) *extended wavelet coefficients*, the first adaptive, efficient storage scheme for multi-measure wavelet coefficients; and, (ii) novel algorithms for selecting the optimal subset of extended coefficients to retain for minimizing the weighted sum of L_2 errors across all measures under a given storage constraint.

Traditional database systems and approximation techniques are typically based on the ability to make multiple passes over *persistent data sets*, that are stored reliably in stable storage. For several emerging application domains, however, data arrives at high rates and needs to be processed on a continuous (24×7) basis, without the benefit of several passes over a static, persistent data image. Such *continuous data streams* arise naturally, for example, in the network installations of large Telecom and Internet service providers where detailed usage information (Call-Detail-Records (CDRs), SNMP/RMON packet-flow data, etc.) from different parts of the underlying network needs to be continuously collected and monitored for interesting trends and phenomena (e.g., fraud or Denial-of-Service attacks). Efficiently tracking an accurate wavelet synopsis over such massive streaming data, using only small space and time (per streaming update), poses a host of new challenges. Recently-proposed solutions [3, 9] rely on maintaining small-space, *pseudo-random AMS sketches* (essentially, random linear projections) over the input data stream [1]. These sketches can then be queried to efficiently recover the topmost wavelet coefficients of the underlying data distribution within provable error guarantees [3].

Key Applications

Wavelet synopses are a general data-reduction tool with several important applications, including statistics for query optimization, lossy data compression, OLAP cube summarization, and interactive data exploration, mining, and query processing.

Data Sets

Several publicly-available real-life data collections have been used in the experimental study of wavelet synopses (and other data-reduction methods); examples include the US Census Bureau data sets (http://www.census.gov/), the UCI KDD Archive (http://kdd.ics.uci.edu/), and the UW Earth Climate and Weather Data Archive (http://www-k12.atmos.washington.edu/k12/grayskies/).

Future Directions

The area of wavelet-based data reduction is still rife with interesting algorithmic questions, including, for instance (i) designing efficient methods for building wavelet synopses that optimize different error metrics under general streaming models (e.g., allowing both item insertions and deletions), and (ii) developing a sound foundation and appropriate summarization tools for approximate *set-valued* (i.e., non-aggregate) queries. Dealing with the *curse of dimensionality* that invariably haunts space-partitioning techniques (such as wavelets and histograms) is another big open issue; some initial ideas based on combining these techniques with statistical-correlation models appear in [5]. And, of course, from a systems perspective, the problem of incorporating wavelets and other approximate query processing tools in an industrial-strength database engine (that can, e.g., select and optimize the appropriate tools for each scenario) remains wide open.

Cross-references

► Approximate Query Processing
► Data Compression
► Data Reduction
► Data Sketch/Synopsis
► Synopsis Structures
► Wavelets on Streams

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th Annual ACM Symp. on Theory of Computing, 1996, pp. 20–29.
2. Chakrabarti K., Garofalakis M.N., Rastogi R., and Shim K. Approximate query processing using wavelets. VLDB J., 10(2-3): 199–223, September 2001.
3. Cormode G., Garofalakis M., and Sacharidis D. Fast approximate wavelet tracking on streams. In Proc. 10th Int. Conf. on Extending Database Technology, 2006.
4. Deligiannakis A., Garofalakis M., and Roussopoulos N. Extended wavelets for multiple measures. ACM Trans. Database Syst., 32(2), June 2007.
5. Deshpande A., Garofalakis M., and Rastogi R. Independence is good: dependency-based histogram synopses for high-dimensional data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.
6. Garofalakis M. and Gibbons P.B. Approximate query processing: taming the terabytes. Tutorial in 27th International Conference on Very Large Data Bases, 2001.
7. Garofalakis M. and Gibbons P.B. Probabilistic wavelet synopses. ACM Trans. Database Syst., 29(1), March 2004.
8. Garofalakis M. and Kumar A. Wavelet synopses for general error metrics. ACM Trans. Database Syst., 30(4), December 2005.
9. Gilbert A.C., Kotidis Y., Muthukrishnan S., and Strauss M.J. One-pass wavelet decomposition of data streams. IEEE Trans. Knowl. Data Eng., 15(3): 541–554, May 2003.
10. Guha S. and Harb B. Wavelet synopsis for data streams: minimizing non-euclidean error. In Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2005, pp. 88–97.
11. Jawerth B. and Sweldens W. An overview of wavelet based multiresolution analyses. SIAM Rev., 36(3): 377–412, 1994.
12. Stollnitz E.J., DeRose T.D., and Salesin D.H. Wavelets for computer graphics – theory and applications. Morgan Kaufmann, San Francisco, CA, 1996.
13. Vitter J.S. and Wang M. Approximate computation of multidimensional aggregates of sparse data using wavelets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 193–204.

Discretionary Access Control

Gail-Joon Ahn
Arizona State University, Tempe, AZ, USA

Synonyms

DAC; Identity-based Access Control; etc.

Definition

Discretionary access control (DAC) provides for owner-controlled administration of access rights to objects. DAC, as the name implies, permits the granting and revocation of access permissions to be left to the discretion of the individual users. A DAC mechanism allows users to grant or revoke access to any of the objects under their control.

Historical Background

Trusted computer system evaluation criteria (TCSEC) published by the US Department of Defense, commonly known as the Orange Book, defined two important access control modes for information systems: discretionary access control (DAC) and mandatory access control (MAC). As the name implies, DAC allows the creators or owners of files to assign access rights. Also, a user (or subject) with discretionary access to information can pass that information on to another user (or subject). DAC has its genesis in the academic and research setting from which time-sharing systems emerged in the early 1970s.

Foundations

As defined in the TCSEC and commonly implemented, DAC policy permits system users (or subjects) to allow or disallow other users (or subjects) access to the objects under their control. The TCSEC DAC policy is defined as follows [1]:

A means of restricting access to objects based on the identity of subjects or groups, or both, to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject.

DAC is a means of restricting access to objects based on the identity of users or the groups to which they belong. The controls are discretionary in the sense that a user or subject given discretionary access to a resource is capable of passing that information on to another subject. To provide this discretionary control, DAC is based on the notion that individual users are "owners" of objects and therefore have complete discretion over who should be authorized to access the object and in which access mode [2]. Ownership is usually acquired as a consequence of creating the object [3]. In other words, DAC policies govern the access of users to the information on the basis of the user's identity and authorizations (or rules) that specify the access modes the user is allowed on the object. Each request of a user to access an object is checked against the specified authorizations. If there exists an authorization stating that the user can access the object in the specific mode, the access is granted, otherwise it is denied.

DAC mechanisms tend to be very flexible. The flexibility of discretionary policies makes them suitable for a variety of systems and applications. For these reasons, they have been widely used in a variety of implementations, especially in the commercial and industrial environments.

DAC policies based on explicitly specified authorization are said to be closed, in that the default decision of the reference monitor is denial. Similar policies, called open policies, could also be applied by specifying denials instead of permissions [4,5]. In this case, the access modes the user is forbidden on the object are specified. Each access request by a user is checked against the specified authorizations and granted only if no authorizations denying the access exist. The use of positive and negative authorizations can be combined, allowing the specification of both the accesses to be authorized as well as the accesses to be denied to the users. The combination of positive and negative authorizations can become enormously complicated. In addition, for many enterprises within industry and government, their users do not own the information to which they are allowed access as is claimed by DAC policies. For such organizations, the organization is the actual owner of system objects and it may not be appropriate to allow users to manipulate access rights on the objects.

However, even though DAC mechanisms are in wide commercial use today, they are known to be inherently weak since they do not provide real assurance on the flow of information in a system. Granting read access is transitive so nothing stops a user from copying the contents of other's file to an object that s/he controls. For example, a user who is able to read data can pass it to other users not authorized to read it without the cognizance of the owner. Therefore, it is easy to bypass the access restrictions stated through the authorizations.

The reason is that discretionary policies do not impose any restriction on the usage of information by a user once the user has got it. In other words, further dissemination of information is not governed.

In addition, access rights need to be assigned explicitly to users who need access. Programs executed by a user have the same access rights as the user who is invoking it. This means that the security of the database system depends on the applications that are being executed. That is, a security breach in an application can affect all the objects to which the user has access as well. This makes DAC mechanisms vulnerable to "Trojan Horse" attacks. Because programs inherit the identity of the users who are invoking them, a user may write a program for other user that performs some legitimate system activities, while at the same time reads the contents of other user's files and writes the contents of the files to a location that both users can access. The user may then move the contents of the files to a location not accessible to the other user [6].

In summary, DAC is very flexible and suitable for various applications but it has an inherent weakness that information can be copied from one object to another, so access to a copy is possible even if the owner of the original does not provide access to the original. Moreover, such copies can be propagated by Trojan Horse software without explicit cooperation of users who are allowed access to the original [7,8].

DAC in Relational Database

The SQL standard includes the access control facilities to support DAC features. The creator of a relation in an SQL database becomes an owner of the relation. The owner also has the fundamental ability to grant other users access to that relation. The access privileges recognized in SQL correspond explicitly to the CREATE, SELECT, INSERT, DELETE and UPDATE statements. There is also a REFERENCES privilege to control the foreign keys to a relation. SQL does not require direct privilege for a user to create a relation, unless the relation is defined to have a foreign key to another relation. For the latter case the user must have the REFERENCES privilege for the relation. To create a view a user should have the SELECT privilege on every relation mentioned in definition of the view. If a user has access privileges on these relations, corresponding privileges are obtained on the view as well.

In addition, the owner of a relation can grant one or more access privileges to another user. This can be done with or without the GRANT OPTION. If the owner grants, say, SELECT with the GRANT OPTION the user receiving this grant can further grant SELECT to other users. The general format of a grant operation in SQL is as follows.

GRANT	privileges
[ON	relation]
TO	users
[WITH	GRANT OPTION]

The GRANT command applies to base relations as well as views. The ON and WITH clauses denote that these are optional and may not be present in every GRANT command. INSERT, DELETE and SELECT privileges apply to the entire relation as a unit. INSERT and DELETE are operations on entire rows so this is appropriate. SELECT, however, allows users to select on all columns. Selection on a subset of the columns can be achieved by defining a suitable view, and granting SELECT on the view. The UPDATE privilege in general applies to a subset of the columns. For example, a user could be granted the privilege to update the ADDRESS but not the GRADE of an STUDENT.

Also, the REVOKE statement is necessary to take away a privilege that has been granted by a GRANT. It is often required that revocation should cascade. In a cascading revoke, not only is the revoked privilege taken away, but also all GRANTs based on the revoked privilege are accordingly revoked. For example say that user Alice grants Bob SELECT on relation R with the GRANT OPTION. Furthermore, Bob subsequently grants Chris SELECT on R. Now suppose Alice revokes SELECT on R from Bob. The SELECT on R privilege is taken away not only from Bob, but also from Chris. The precise methods of a cascading revoke is somewhat complicated. Suppose Bob had received the SELECT on R privilege not only from Alice, but also from David before Bob granted the SELECT to Chris. In this case Alice's revocation of the SELECT from R privilege from Bob will not cause either Bob or Chris to loose this privilege. This is because the GRANT from David remains legitimate.

Cascading revocation is not always desirable. A user's privileges to a given table are often revoked because the user's job functions and responsibilities have changed. Thus the Head of a Division may move on to a different assignment. His/her privileges to that Division's data need to be revoked. However, a cascading revoke may cause lots of employees of that Division to loose their privileges. These privileges should be reassigned to maintain the business continuity.

Several database products take the approach that a database is always created with a single user, usually called the Database Administrator (DBA). The DBA essentially has all privileges with respect to this database. The DBA is also responsible for enrolling users and creating relations. Some systems recognize a special privilege which can be granted to other users at the initial DBA's discretion and allows these users to successfully act as the DBA [9,10].

Key Applications

Database systems, operating systems, and owner-centered web applications.

Future Directions

There are many subtle issues in DAC such as multi-step grant, cascading revocation, and so on. All these subtleties of DAC are still being discussed, debated and refined in the literature. The driving principle of DAC is ownership and such an owner-based access control can be applied to preserve privacy attributes in database systems. Nevertheless DAC has the inherent weakness, DAC's flexibility and suitability are needed to be further studied to support emerging critical applications.

Cross-references

► Access Control
► Access Control Administration Policies
► Access Control Policy Languages
► Mandatory Access Control
► Role-Based Access Control

Recommended Reading

1. Bertino E., Samarati P., and Jajodia S. Authorizations in relational database management systems. In Proc. First ACM Conf. on Computer and Communications Security, 1993, pp. 130–139.
2. Bishop M. Computer Security: Art and Science. Addison-Wesley, Reading, MA, 2003.
3. Castano S., Fugini M.G., Martella G., and Samarati P. Database Security. Addison Wesley, Reading, MA, 1994.
4. Fagin R. On an authorization mechanism. ACM Trans. Database Syst., 3(3):310–319, 1978.
5. Ferraiolo D.F., Gilbert D.M., and Lynch N. An examination of federal and commercial access control policy needs. In Proc. NIST–NCSC National Computer Security Conference, 1993, pp. 107–116.
6. Graham G.S. and Denning P.J. Protection: principles and practice. In Proc. AFIPS Spring Joint Computer Conference. 40:417–429, 1972.
7. Griffiths P.P. and Wade B.W. An authorization mechanism for a relational database system. ACM Trans. Database Syst., 1(3):242–255, 1976.
8. Lampson B.W. Protection. In Proc. Fifth Princeton Symp. on Information Science and Systems, 1971, pp. 437–443. Reprinted in ACM Operat. Syst. Rev., 8(1):18–24, 1974.
9. Rabitti F., Bertino E., Kim W., and Woelk D. A model of authorization for next-generation database systems. ACM Trans. Database Syst., 16(1), 1991.
10. Sandhu R.S. and Samarati P. Access control: principles and practice. IEEE Commun., 32(9):40–48, 1994.

Disk

Peter Boncz
Database Architectures and Information Access, CWI, Amsterdam, The Netherlands

Synonyms

Hard disk; Magnetic disk; Disk drive

Definition

In disk storage, data is recorded on planar, round and rotating surfaces (disks, discs, or platters). A disk drive is a peripheral device of a computer system, connected by some communication medium to a disk controller. The disk controller is a chip, typically connected to the CPU of the computer by the internal communication bus. Main implementations are hard disks, floppy disks and optical discs, of which the first is the usual interpretation.

Recently, Solid State Disks have been introduced; though the term "disc" is a misnomer for these devices, as internally they consist of NAND Flash memory chips. Similarly, the term RAM Disk is used for a storage device consisting of volatile DRAM memory. Both offer the same data storage abstraction as a hard disk at the operating system level, though their price, size, performance and persistence characteristics are very different from a hard disk.

Key Points

The history of the hard disk started at IBM in San Jose, California, when Rey Johnson created a rotating drum that was coated in a magnetically polarizable film that

RPM	3,600	5,400	7,200	10,000	15,000	Solid state
Disk model	CDC wrenl 94145–3	Seagate ST41600	Seagate ST15150	Seagate ST39102	Seagate ST373453	Samsung MCBOE
Year	1983	1990	1994	1998	2003	2008
Capacity (GB)	0.03	1.4	4.3	9.1	73.4	32
Seq. bandwidth (MB/s)	0.6	4	9	24	86	80
Read latency (ms)	48.3	17.1	12.7	8.8	5.7	0.3
Write latency (ms)						40

could be used to store data by changing and sensing magnetic polarization.

Hard disks nowadays consist of a number of platters connected by a single axis, spinning at a fixed number of rotations per minute (rpm). Data is on a platter organized by track (distance from the center) and sector (angular region on a track). The disk head, one for each platter, mounted on a disk arm that moves in and out, therefore cover more distance per unit of time on an outer track than on an inner track. To read or write data, the head must be moved to the correct position above the track (seek time), and then wait until the region of interest spins past it (rotational delay). Therefore, the average random access latency of hard disks is seek time plus rotational delay divided by two. The bandwidth of a disk is determined by both communication infrastructure with the controller as well as rotation speed and data density, which determine the amount of bits that pass the head per second. Data density is closely related to disk capacity and has increased enormously, surpassing the historical development of any other performance factor in computer architecture (i.e., orders of magnitude faster than latency and quite a bit faster than CPU MHz as well as disk bandwidth). The historical development of disk performance parameters is shown in the below table. For comparison, the last column shows characteristics of a solid state drive.

The consequence of these developments is that relatively speaking, disk latency currently is much slower with respect to all other performance factors than it was a few decades ago, which means that fast and scalable algorithms involving I/O must focus more on sequential bulk access than fine-grained random access. Managing and optimizing I/O access is one of the primary tasks of a database system. In order to counter the trend of expensive random disk access latency, modern database systems (should) make use of asynchronous I/O to amortize disk arm movements over multiple requests, multi-disk or RAID systems to increase the I/O operation throughput, larger page sizes, as well as compression, clustered indices and efficient cooperative scan techniques to profit more from efficient sequential I/O.

Cross-references

- ► CPU
- ► Non-Volatile Memory (Flash)
- ► Primary Storage (RAM)
- ► RAID
- ► Tertiary Storage (tape)

Disk Array

► Redundant Array of Independent Disks (RAID)

Disk Drive

► Disk

Disk Power Saving

HAZUO GODA
The University of Tokyo, Tokyo, Japan

Definition

The term Disk Power Saving refers to the function of reducing the electric power that is consumed in a disk drive. Typically, Disk Power Saving cuts power supply to a particular component of the disk drive or slows down the component. In most cases, the disk drive

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:54

cannot respond to input/output requests while it is in such a power saving mode.

Key Points

Power modes of modern disk drives are often classified as normal and stand-by. A disk drive in the normal mode is either processing read/write requests or ready to immediately start processing read/write requests. In this mode, all the components are almost fully operating. The disk drive thus consumes higher electric power than in the other modes. In contrast, a disk drive in the other modes cannot start processing read/write requests immediately. In the stand-by mode, the spindle motor is spun down and the head assembly is unloaded to the ramp and powered off; only the controller or a partial circuit of the controller remains operational. The disk drive thus consumes much less power than in the normal mode. Yet, when a disk drive accepts a read/write request, the controller needs to spin up the spindle motor and power on the head assembly again so as to transition to the normal mode. This process is often time- and energy-consuming.

Many commercial disk drives support stand-by commands such that operating systems and applications running on host computers can control disk power modes. In addition, some of the disk drives have the capability of threshold-based hibernation; those disk drives can automatically change to the stand-by mode when a specified time has elapsed after the last read/write access.

The difficulty of power management of disk drives is due to the significant penalties of mode changing. Many commercial disk drives need more than 10 sec and over one hundred joules to change from the stand-by mode to the normal mode.

Some manufacturers are providing more flexibility by introducing new types of power saving modes such as unloaded and low-rpm. In the unloaded mode, the head assembly is merely unloaded to the ramp and the other components are normally operating. In the low-rpm mode, the head assembly is unloaded to the ramp and powered off, but the spindle motor is spinning at lower rotational speeds. These new modes can save less power but give also smaller penalties than the traditional stand-by mode.

In another approach, some research groups are studying the possibility of multi-speed disk drives, which have the capability of dynamically changing rotational speeds in the normal mode. The use of the multi-speed disk drives sounds effective, but only several prototypes have been reported so far.

Cross-references

► Massive Array of Idle Disks
► Storage Power Management

Recommended Reading

1. Gurumurthi S., Sivasubramaniam A., Kandemir M., and Franke H. Reducing disk power consumption in servers with DRPM. IEEE Comput., 36(12):59–66, 2003.
2. HGST Inc. Quietly cool. White Paper, HGST, 2004.
3. Yada H., Ishioka H., Yamakoshi T., Onuki Y., Shimano Y., Uchida M., Kanno H., and Hayashi N. Head positioning servo and data channel for HDDs with multiple spindle speeds. IEEE Trans. Magn., 36(5):2213–2215, 2000.

Disk Process

► Storage Manager

Disk-based Model

► I/O Model of Computation

Disks

► Storage Devices

Distance between Streams

► Stream Similarity Mining

Distance Indexing

► Indexing Metric Spaces

Distance Space

► Metric Space

Distance-preserving Mapping

► Space Filling Curves
► Space-Filling Curves for Query Processing

Distillation

► Summarization

Distortion Techniques

Carpendale Sheelagh
University of Calgary, Calgary, AB, Canada

Synonyms

Fisheye views; Nonlinear magnification; Multiscale views; Detail-in-context; or Focus-plus-context

Definition

While the word “distortion” often has unfavorable connotations in terms of data, a *distortion technique* in digital information viewing or data exploration is the use of deformation of some aspect of the information or data in order to provide a better view or better access to some other aspect of the data.

Historical Background

The uses of distortion in digital information exploration interfaces have two independent starting points: Spence and Apperley’s Bifocal Display [18] and Furnas’ Generalized Fisheye Views [5]. From these origins, research initially focused on developing algorithmic solutions for distortion techniques. Well-known examples include: Sarkar and Brown’s Graphical Fisheyes [16], which expand upon Furnas’ approach creating spatial reorganizations of visual representations; Hyperbolic Display [9], which uses mathematical function to create detail-in-context presentations; Perspective Wall [11], and Document Lens [15], which make use of 3D perspective projection to create detail-in-context presentations; and Elastic Presentation [3], which also uses 3D manipulations and perspective projection to offer a mathematical framework that encompassed distortion techniques to date. Other methods [1,5,8,12,16] create distortion presentations by using a 2D-to-2D transformation function to spatially adjust a given two-dimensional layout (for survey see [10]).

Though many varieties exist and there continues to be interest further developing the domain within the research community, distortion techniques have not yet received widespread acceptance. This may be due to a general discomfort with the use of distortion and fears that distortion might lead to misinterpretation of data. There is still room for significant research into the relative merit among differing distortion techniques and it is likely that this kind of evaluation will be dependent on the type of task, the nature of the information, and the preferences and skills of the person using it.

Foundations

With regard to data presentation, access, navigation, and exploration, the development of distortion techniques has not been the goal in itself. Instead, these techniques are used to achieve, aid, or ease a data- or task-related goal. As such, the discussion and concepts that motivate the development of distortion techniques are important and remain relevant.

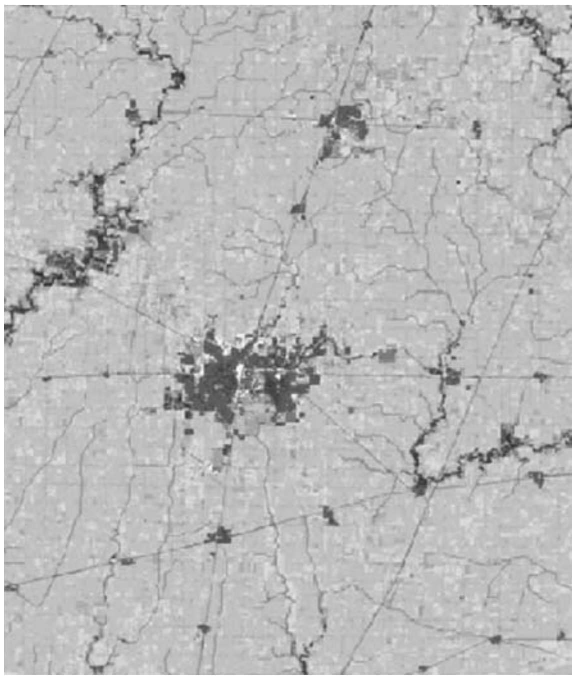
The primary motivation has always been a lack of display space. Whether it is because of the sheer size of the data to be displayed or because of the number of related windows needed to accomplish a task, it seems that there is never enough display space. While the amount of available display space is expanding – note current interest in high-resolution, large displays – it is not keeping up with either the computing power or the deluge of data. The response to this trend has been through interaction – zooming, scrolling, and panning – and various distortion techniques. Different types of distortion techniques were initially motivated independently.

Spence and Apperley [18] started by noting the frequency of both crowding and navigation problems in interfaces in general. Their response was based on an understanding of how human memory is often less than precise and can lead to fuzzy search practices. They discuss how searching in physical space is supported by several factors including spatial memory, memory of previous actions, and visual and verbal clues and can be reinforced by a reasonable degree of constancy. They suggest that interfaces might better support these factors by maintaining a full context in which spatial constancy is preserved. Spatial constancy requires ensuring that all items stay present, and that all parts or regions stay in the same positions relative to each other. This thinking led to integrated views with two distinct levels of magnification, called Bifocal Displays [18].

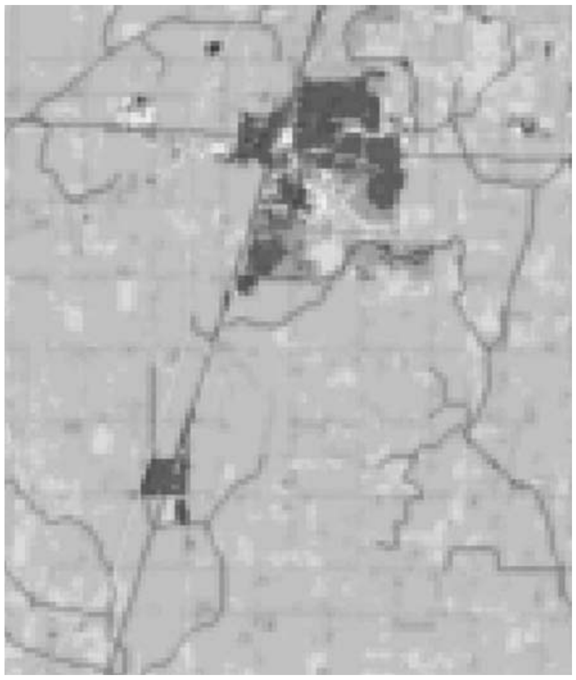
Furnas [5] studied how people retain and present information in various subject areas and workplaces such as geography, history, and newspapers and noted that people usually know the details about their own interests, set in enough domain knowledge to provide context. His widespread evidence led him to suggest that Generalized Fisheye Views may be a useful and intuitive way to present information. A Generalized Fisheye View has a focus or center of interest about which detail is displayed, gradually decreasing in detail as the distance from the focus increases, with the exception that some specific items of noted importance within the domain should be included regardless of the distance from the focus. To achieve this, Furnas proposed filtering the context by using a degree of interest (DOI) function. A DOI is based upon the distance from the current focus and a domain-specific a priori importance (API) function. These concepts led to a variety of filtered and variant magnification presentations.

Another much-discussed point is the importance of visual integration in reducing cognitive load. If a particular data presentation is too large to fit the available display space, it can be compressed uniformly to fit (Fig. 1). This can result in a presentation that is too dense to discern detail. For instance, in Fig. 1 one cannot see the two airports, which should show as white lines on a green background. To obtain a better view of these details one can zoom in, as in Fig. 2, and see one of the airports. The viewer must now either flip between views to know which airport is being displayed or view both the compressed and detailed view side by side and work out how the two views relate to each other. In this case, this is not too difficult, but does impose some additional attentional costs. Local magnification or an inset-in-place can be used, as in Fig. 3, but now the actual connections are occluded. Alternatively, an integrated detail-in-context view can be used (Fig. 4). Note that while visual integration is maintained, it makes use of a distortion technique [3].

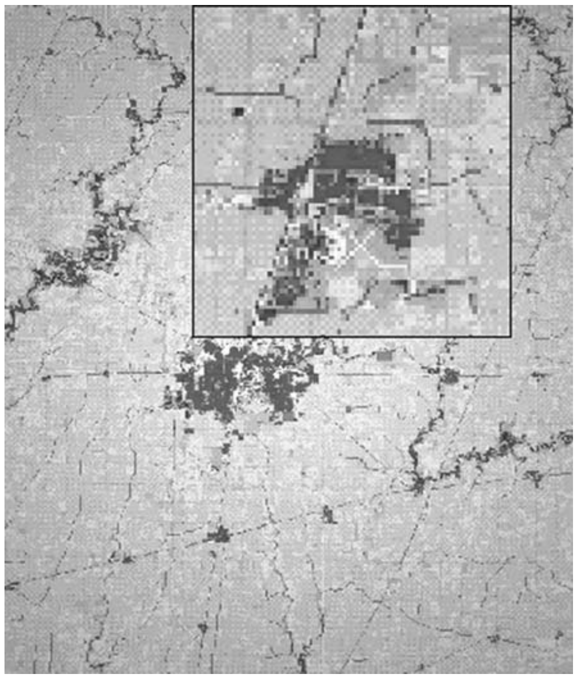
Several researchers noted that temporal continuity was also important [3,17]. Misue et al. [12] raised the issue of developing techniques that support a viewer's mental map. In particular, they note that the

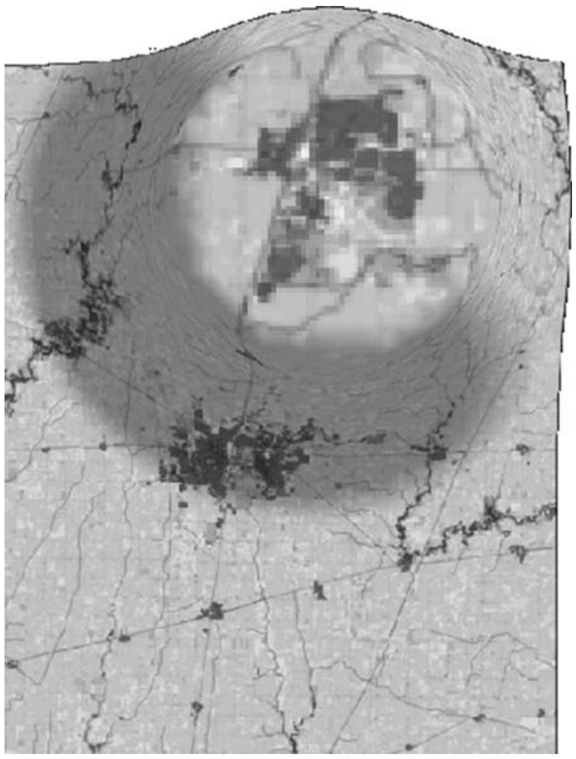


Distortion Techniques. Figure 1. This image shows a land usage map of Champaign, Illinois compressed uniformly to fit.



Distortion Techniques. Figure 2. Magnifying one region of the land use map of Champaign Illinois reveals details of airport runways not previously visible.


Distortion Techniques. Figure 3. Magnified inset occludes local context.


Distortion Techniques. Figure 4. Distortion technique provides integrated detail-in-context view.

mathematical concepts of proximity, orthogonality, and topology should be algorithmically ensured. Carpendale et al. [2] noted that viewing cues can aid in the general readability of data presentations that utilize distortion techniques.

In summary, the issues and considerations around the development of distortion techniques include:

- Maintenance of spatial constancy to support spatial memory;
- Maintenance of full context, that all items maintain some visual evidence of presence;
- Introduction of more than one magnification scale in a unified presentation;
- Reduction of cognitive effort needed for reintegration of information across separate views;
- Support for setting detail in its context, as is common practice in human memory patterns;
- Maintenance of sufficient context;
- Providing for varying needs for domain significant data within a reduced context;
- Varying the amount of detail needed around the focus;
- Exploring the possibility of more than one area of interest, or foci;
- Utilizing degree of interest (DOI) functions, as appropriate;
- Providing visual continuity;
- Providing temporal continuity; and
- Supporting the mental map by maintaining proximity, orthogonally, and topology.

Three frameworks emerged from the great variety of distortion techniques developed. Leung and Apperley [10] categorized existing methods and unified them by showing how to derive 2D-to-2D transformations for 3D-based techniques. However, these are complex formulations and have not been implemented.

There is also a family of 2D-to-2D distortion techniques. In these, a 2D data representation is transformed through use of a distortion technique to create an adjusted presentation that provides magnified detail in the foci, which are set in their context via distorted regions. Of these techniques, Magnification Fields [8] is probably the most general. It describes an approximate integration based approach that, given a pattern to strive for, can create a magnification pattern. A 2D distortion transformation function performs adjustments in x and/or y. The resulting pattern of magnification and compression is the derivative of the transformation function. Magnification Fields determines the transformation function from the magnification function [8]. This would allow a person to create a multiscale view by requesting the pattern magnification that suits their task. Their approach starts with a grid and a set of desired magnification amounts. The grid is adjusted iteratively, ensuring that no grid points overlap until the difference between the magnification provided by the adjusted grid and the desired magnification is sufficiently small [8].

Elastic Presentation Framework (EPF) [3] unifies many individual distortion techniques, allowing the seamless inclusion of more than one distortion technique in a single interface. By interpolating between the methods it describes, EPF identified new variations. EPF achieved previous 2D-to-2D approaches using an intermediate 3D step. The 3D-based approaches are quite different algorithmically than their 2D counterparts. The plane or surface that holds the two-dimensional representation is manipulated in three dimensions, and then viewed through single-point perspective projection. The transformation function results from the combination of the manipulation of the surface and the perspective projection. This combination simplifies the mathematics of the relationship between magnification and transformation to the geometry of similar triangles.

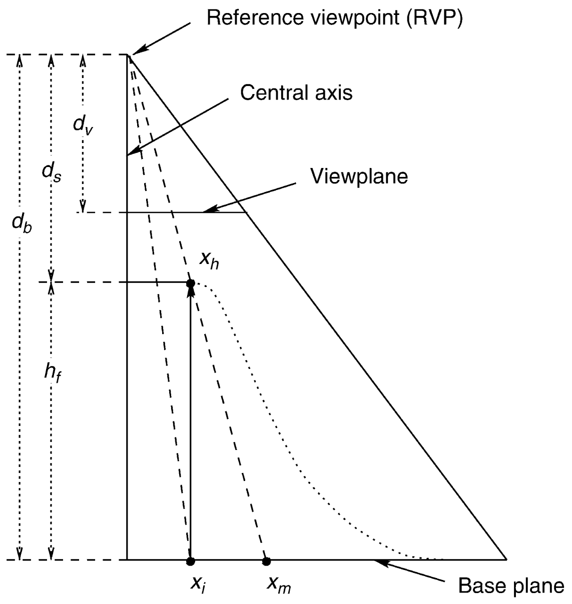
In EPF, data are placed on a 2D plane, which is then placed in a 3D viewing volume on the *baseplane*, parallel to the viewplane at a distance along the *z* axis from the viewpoint which defines unit magnification. The next step is to provide the focal regions with the intended magnification. What is needed is an asymptotic function that relates degree of magnification to *z*-translation that also guarantees fine control as the viewpoint is approached. This function can be derived from similar triangles shown in Fig. 5 as follows:

$$\frac{x_m}{x_i} = \frac{d_b}{d_s}$$

$$mag = \frac{x_m}{y_i}$$

$$h_f = d_b - d_s$$

where x_i is a point on the baseplane that is raised to a height h_f providing a magnification of *mag*. The



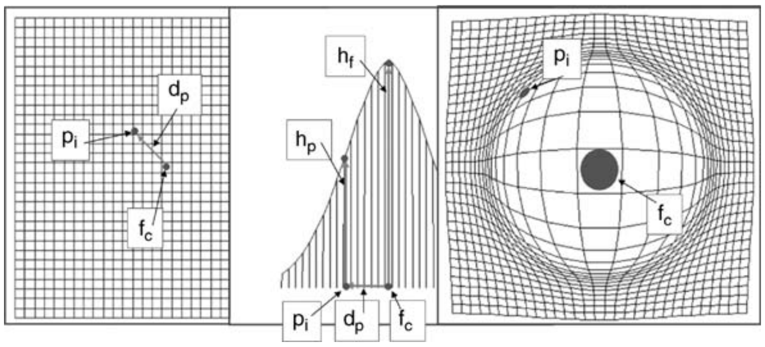
Distortion Techniques. Figure 5. The relationships between the displaced points and the apparent magnification.

position x_m is the apparent location after the displacement of the point x_i to a height h_f:

$$h_f = d_b - \left(\frac{d_b}{mag}\right)$$

This function offers infinite magnification control, which is limited only by the numerical resolution of the computer. The coordinates (x_m, y_m) allow the option of performing transformations directly by translating the point in *x* and *y*, or through perspective by adjusting the height. To ensure full visibility for multiple foci, the foci are viewer-aligned. That is the focal center is aligned to the viewpoint. To ensure uniform magnification response throughout the display the translation vectors are normalized in *z* (see [3]).

Now that focal magnification is obtained, an appropriate distortion that will link the foci to the context is needed. This is achieved through a drop-off function. Points on the surface are translated depending on the value of the drop-off function when applied to the distance of the point from the focal region. The extent of the spread of the distortion into the context can be controlled by the viewer through adjustments to the domain and range of the drop-off function. In Figure 6, the focal point f_c is raised to height h_f according to the



Distortion Techniques. Figure 6. Providing an integrated context.

magnification required. The height h_p of any point p_i at a distance d_p from the focal point f_c can be calculated.

Many drop-off functions are effective. Perhaps the simplest is a linear drop-off. In an EPF linear drop-off function, the surface height h_p of a point $_p$, that is a distance $d_p$from the focal centre f_c with a lens radius l_r, and a focal height h_f is calculated by:

$$h_p = h_f * (1 - (d_p/l_r))$$

Varying the lens radius l_r affects the limits of the lens and the resulting slope. A Gaussian drop-off function offers a smooth visual integration with its bell curve.

$$h_p = h_f * \exp^{-((d_p^2/\sigma)}$$

An inverse power function offers a very good drop-off function for achieving high focal magnifications while still offering full visual integration in the distorted region.

$$h_p = h_f * (1 - (d_p/l_r))^k$$

Note that if $k = 1$, this is the equivalent to the linear drop-off function. Alternatively, setting $k = 2.7$ results in high-magnification focal regions. Having chosen the focal magnification and the drop-off, the manipulated surface is then viewed through perspective projection. Single-point perspective projection preserves angles, proximity, and parallelism on all x, y planes.

Many other possibilities exist for varying the distortion technique. For instance, different distance metrics can be used. An L_p metric offers a continuum between radial and orthogonal layouts (See Fig. 7). For 2D distances between points $p_1(x_1, y_2)$ and $p_2(x_2, y_2)$, L_p metrics are defined as:

$$L(P) = \sqrt[P]{|x_1 - x_2|^P + |y_1 - y_2|^P}$$

where $L(2)$ is Euclidean distance.
The $L(\infty)$ metric is:

$$L(\infty) = \sqrt[\infty]{|x_1 - x_2|^\infty + |y_1 - y_2|^\infty}$$

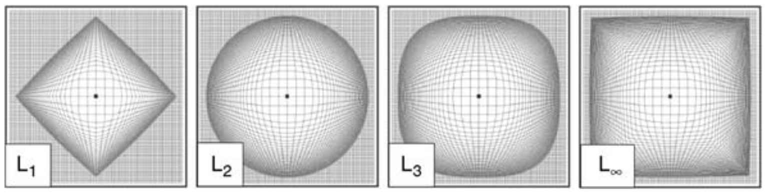
which resolves to:

$$L(\infty) = \max(|x_1 - x_2|, |y_1 - y_2|)$$

While there is continued interest in developing better distortion techniques – a recent new framework [14] offers an image-based approach and incorporates transparency blending into distortion techniques – the focus within the research community has increasingly shifted to empirical work. Distortion techniques have been shown to offer advantages for command and control [17], web navigation [13], and menu selection [1]. However, usability problems have also been shown to exist in important tasks such as targeting, steering, and layout [6,7].

Key Applications

There has not yet been widespread acceptance of distortion techniques in readily-available applications (notable exceptions include the Mac OS X Dock and www.idelix.com). In considering potential applications, it is important to return to the definition of distortion techniques, which involved a willingness to make use of distortion to achieve a data or task goal. In this light, it is probable that current distortion frameworks and lenses will in the future be considered as relatively crude functions. Consider a practical



Distortion Techniques. Figure 7. L-Metrics.

example: drawing a map in which two quite distant coastal towns are connected by two roads. One road runs along the shoreline and the other road runs high above it along a cliff top. Because of the cliff, there are no connections between these two roads except at the towns. Within the available display space, drawing the map faithfully and to scale would place the roads overtop each other at several points along the way. This is a very familiar cartographic problem and distortion has been used for centuries to solve it. Here, the data accuracy that is preserved is the fact that no connection between the roads exists. The spatial distortion is a small price to pay. However, most current distortion techniques would not provide a subtle and graceful adjustment. There is considerable open research space for more subtle and controllable distortion techniques. Also, as Furnas [4] recently noted, some of the concepts like DOIs lend themselves to integrated solutions involving data as well as spatial adjustments.

Cross-references

► Context
► Contextualization
► Data Transformation Methods
► Information Navigation
► Visualization for Information Retrieval
► Zooming Techniques

Recommended Reading

1. Bederson B. Fisheye menus. In Proc. 13th Annual ACM Symp. on User Interface Software and Technology, 2000, pp. 217–225.
2. Carpendale S., Cowperthwaite D., and Fracchia F.D. Making distortions comprehensible. In Proc. 1997 IEEE Symp. on Visual Languages (VL '97). 1997, pp. 36–45.
3. Carpendale S. and Montagnese C. A Framework for unifying presentation space. In Proc. 14th Annual ACM Symp. on User Interface Software and Technology, 2001, pp. 61–70.
4. Furnas G. A fisheye follow-up: further reflections on focus+context. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2006, pp. 999–1008.
5. Furnas G.W. Generalized fisheye views. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1986, pp. 16–23.
6. Gutwin C. Improving focus targeting in interactive fisheye views. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2002, pp. 267–274.
7. Gutwin C. and Fedak C. Interacting with big interfaces on small screens: a comparison of fisheye, zoom, and panning techniques. In Proc. Graphics Interface 2006, 2004, pp. 213–220.
8. Keahey A. The generalized detail-in-context problem. In Proc. IEEE Symp. on Information Visualization, 1998, pp. 44–51.
9. Lamping J., Rao R., and Pirolli P. A focus+context technique based on hyperbolic geometry for visualising large hierarchies. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1995, pp. 401–408.
10. Leung Y. and Apperley M. A review and taxonomy of distortion-oriented presentation techniques. ACM Transactions on Computer Human Interaction, 1(2):126–160, 1994.
11 Mackinlay J., Robertson G., and Card S. Perspective wall: detail and context smoothly integrated. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1991, pp. 173–179.
12. Misue K., Eades P., Lai W., and Sugiyama K. Layout adjustment and the mental map. J. Visual Lang. Comput., 6(2):183–210, 1995.
13. Munzner T. and Burchard P. Visualizing the structure of the world wide web in 3D hyperbolic space. In ACM VRML '95, 1995, pp. 33–38.
14. Pietriga E. and Appert C. Sigma lenses: focus-context transitions combining space, time and translucence. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 2008, pp. 1343–1352.
15. Robertson G.G. and Mackinlay J.D. The Document lens. In Proc. 6th Annual ACM Symp. on User Interface Software and Technology, 1993, pp. 101–108.
16. Sarkar M. and Brown M.H. Graphical fisheye views. Commun. ACM, 37(12):73–84, 1994.
17. Schaffer D., Zuo Z., Greenberg S., Bartram L., Dill J., Dubs S., and Roseman M. Navigating hierarchically clustered networks through fisheye and full-zoom methods. ACM Transactions on Computer Human Interaction. 3(2):162–188, 1996.
18. Spence R. and Apperley M.D. Data base navigation: an office environment for the professional. Behav. Inform. Technol., 1(1):43–54, 1982.

Distributed Architecture

Tore Risch
Uppsala University, Uppsala, Sweden

Synonyms

Parallel database; Federated database; Multi-database; Peer-to-peer database

Definition

A distributed database [3] is a database where data management is distributed over several nodes (computers) in a computer network. In a central DBMS the data is managed by one node whereas in a distributed DBMS the data is managed by several nodes. A *distributed DBMS* is a database manager consisting of several nodes distributed over a network. Each node is a database manager by itself that communicates with other nodes in the network. In a regular distributed DBMS it is up to the database administrator to manually specify how data collections (e.g. relational tables) are distributed over the nodes when a distributed database is designed. Queries and updates to the distributed relations are transparently translated by the distributed DBMS into data operations on the affected nodes giving the user the impression of using a single database, called *query* and *update transparency*. Thus the distributed DBMS provides distribution transparency for database users but not for the database administrator.

Closely related to distributed DBMSes are *parallel databases* where a parallel DBMS engine runs on usually a cluster. The parallel DBMS automatically determines how data structures are internally distributed over the nodes providing distribution transparency also for the database administrator, called *schema transparency*.

The purpose of *heterogeneous databases* is to be able to combine data from several independently developed autonomous databases. Heterogeneous databases can be divided into federated databases, mediators, and multi-databases. In a *federated database* the database administrator defines a single *global integration schema* describing how data in underlying databases are mapped to the integration schema view. This provides distribution transparency for integrated data. *Mediators* allow the definition of several views over data from different data sources. Since it may be difficult to define integration schemas and views when there are many participating autonomous databases, *multi-databases* relax the distribution transparency also for the database users who there specify queries and updates using a multi-database query language where individual data collections in the participating nodes can be explicitly referenced.

A related technology is *peer-to-peer systems* where networks of files are distributed over the Internet. Meta-data is associated with the files and the user can search for files satisfying conditions. Peer-to-peer search is usually made by propagating queries between the peers. The consistency and correctness of queries are relaxed compared to regular databases in order to provide better performance and node autonomy.

Historical Background

Distributed DBMSs were pioneered by System R and Ingres* in the beginning of the 1980s. Early distributed DBMSs assumed slow communication between nodes having limited amounts of main memory geographically distributed in a wide area network. The database administrator instructed the distributed DBMS where to place data, while the user could specify transparent queries to the distributed DBMS without detailed knowledge of where data was placed.

The evolvement of computer clusters provided hardware resources for very high performing database servers running on clusters, *parallel databases*. Since the communication between cluster nodes is very fast and not geographically distributed, the database administrator need not provide manual placement rules of distributed data, i.e. the parallel DBMS provides full distribution transparency also for the database administrator. With the evolvement of fast wide area computer networks parallel DBMS technology can be used also for some geographically distributed databases. However, it should be noted that update latency has to be taken into account for large geographical distances because of the speed of light. In general geographically distributed databases still requires manual distribution.

Not least the development of the Internet has caused the need to integrate data from many pre-existing databases. The area of *heterogeneous databases* [4,2] deals with tools and methodologies to combine data from several autonomous databases. While distributed and parallel databases assumed all data managed by one distributed DBMS, heterogeneous databases integrate databases using different DBMS and different schemas.

There are several flavors of heterogeneous databases:

1. *Federated databases* require the definition of a *global integration schema* containing mappings to the participating databases' schemas. The federated database becomes a central server on top of the participating autonomous databases.
2. As the number of databases to integrate increases it becomes very difficult or impossible to define a global integration schema over the large numbers of autonomous databases. *Multi-databases* [2] provide no global conceptual schemas and instead a *multi-database query language* allows specification of queries searching through many participating databases.
3. Mediators [5] provide a middle ground between a single integration schema and no schema at all. Instead the user can define mediator views that combine and reconcile data from different data sources. Such views require a query language that can express queries over several databases, i.e. a multi-database query language. The mediator system becomes a middleware between users and wrapped data sources.

While distributed databases could handle transparent queries and updates for a small number of nodes, the evolvement of the Internet requires technologies to deal with geographically distributed databases having 1,000s of nodes. *Peer-to-peer systems* enable such highly distributed file access where users search for data stored in peers. In a *peer-to-peer database* queries are propagated between the participating peer nodes. To improve performance at the expense of query correctness the propagation may stop after a certain number of hops. This is sufficient for many modern applications that do not have strict consistency requirements; for example Internet search engines do not guarantee the full correctness of answers.

Foundations

Autonomy and Heterogeneity

Different distributed DBMS architectures provide different levels of autonomy for the participating nodes.

A *homogeneous* distributed database is a distributed database where all nodes are managed by the same distributed DBMS. A homogeneous distributed database can be regarded as a central database distributed over many nodes where data and processing is internally transparently distributed over several nodes. By contrast, a heterogeneous database is a (distributed or central) database where data originates from participating autonomous databases possibly using different DBMSs.

Regular distributed and *parallel* databases are homogeneous. One distributed DBMS manages all data. *Distributed database design* involves designing the schema in a top-down fashion as for a conventional central database. Parallel databases provide automatic and transparent data placement without user intervention, while regular distributed databases require the database administrator to specify how data should be distributed over nodes. In regular distributed and parallel database the nodes have no autonomy at all.

Federated databases are central database servers that integrate data from several participating databases. Federated databases are thus heterogeneous. *Global integration schemas* are defined that integrate data originated in the participating databases. The design of the integrated schema needs to deal with data integration issues on how to combine the same or similar data represented differently in different participating databases. Different participating databases may use different DBMSs. The schemas of the participating databases are designed before the integrated database schema is designed. Thus the design process for heterogeneous databases becomes bottom-up, whereas homogeneous databases are usually designed top-down. The design of the integrated schema needs to deal with data integration issues on how to combine the same or similar data represented differently in different participating databases.

Both *federated databases*, *mediators*, and *multi-databases* are heterogeneous. The main difference between them is how integration schemas are defined. Federated database assume one global integration schema. If there are many different participating databases it is difficult to define such a global integration schema. This is relaxed in mediators, which allow the definition of many integration schemas as views over wrapped underlying data sources of different kinds. In multi-databases the user is given access to a *multi-database query language* where he can specify queries

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:55

over many sources. A multi-database query language provides the basis for defining mediator views.

Finally, the aim of peer-to-peer databases is distributed queries in a widely distributed network of heterogeneous nodes. Unlike parallel and distributed databases the individual nodes are not managed by a single system, but independently.

Transparency

Distributed databases can be classified according to what kinds of transparency they provide w.r.t. distribution of data. Three different kinds of transparency can be identified for different kinds of services provided by the distributed DBMS, *schema transparency*, *query transparency*, and *update transparency*.

Schema transparency means that the distributed DBMS decides completely on its own where to place data on different nodes. The database administrator has the impression of using a single database and specifies the logical schema without considering any distribution at all. However, often it is desirable to allow the database administrator to specify how to distribute data, and thus relax schema transparency. For example, for performance and to allow local control, a geographically distributed database for a large enterprise may need to cluster employee data according to the countries where departments are located. Therefore full schema transparency is often provided only on local area networks or cluster computers where the communication between nodes is very fast.

With *query transparency* the distribution of data is not reflected in user queries. Queries are transparently translated by a distributed query optimizer into queries and updates to the affected nodes giving the user the impression of using a single database. By analyzing a given user query the distributed query optimizer can often statically determine which nodes to access. Query execution plans can execute in parallel on different nodes with partial results transported between nodes and combined on other nodes. Query transparency is very important for distributed databases since it is very difficult and error prone to manually implement distributed communicating execution plans.

Update transparency allows database updates to be specified without taking distribution into account. A distributed transaction manager propagates updates to affected nodes.

Distributed or Parallel DBMS Provide Update Transparency

In the classification above, only *parallel DBMSs* provide complete transparency for everyone using the database, database administrators as well as users. The term *regular distributed database* refers to a distributed DBMS with query and update transparency but without schema transparency.

With *naming transparency*, users are provided with a single name for a distributed relation defined in terms of several internal relations stored on separate nodes. Regular distributed, parallel, federated, and peer-to-peer databases provide naming transparency, which is relaxed for mediators and multi-databases.

Distributed database design involves manual specification to the distributed DBMS of the distribution of data collections. The database administrator can tune the data placement in a wide area computer network. The two fundamental methods for such manual data distribution are *fragmentation* and *replication.* Fragmentation splits a collection (e.g. Table 1) into separate non-overlapping segments on different nodes, while replication stores identical copies of a collection on different nodes. The distributed DBMS guarantees that queries and updates of fragmented or replicated collections are transparent so the user need not be aware of how data is distributed.

Fragmentation (or partitioning) allows the administrator of a distributed database to manually specify on which nodes the DBMS should place different sections of each distributed data collection. In a distributed relational database tables are fragmented. For example, the placement of employee records in a relation can be fragmented according to in which countries different employees work. Fragmentation speeds up database queries and updates since it allows parallel access to distributed fragments. Furthermore, by analyzing queries and updates the query optimizer can often determine exactly which nodes are affected and send the query/update statements only to those nodes.

Replication allows the DBA to declare to the DDBMS to place the same data collections on more than one node. For example, a relational table may be replicated on several nodes. Replication speeds up data access at the expense of update cost. However, as explained below, if consistency is relaxed the update cot may be reduced.

Federated databases also provide query and update transparency by allowing the database administrator to

Distributed Architecture. Table 1. The Architectures of DDBMSs can be Classified Along Different Dimensions. The Following Table Classifies Different Kinds of Distributed DBMS Architectures

	Autonomy	Schema transparency	Query transparency	Update transparency	Naming transparency	Central schema
Parallel	No	Yes	Yes	Yes	Yes	Yes
Regular Distributed	No	No	Yes	Yes	Yes	Yes
Federated	Yes	No	Yes	Limited	Yes	Yes
Mediators	Yes	No	Yes	Limited	No	No
Multi-databases	Yes	No	No	No	No	No
Peer-to-peer	Yes	No	Yes	Yes	Yes	No

define a *global integration schema* that hides the underlying integrated databases.

Mediators provide some query transparency by allowing users to define views over integrated databases. Update transparency is more problematic as it requires updatable views.

With *multi-databases* transparency is further relaxed so the user can reference individual databases explicitly in queries and updates.

Finally, *peer databases* [1] provide query and update transparency in widely distributed systems but do not require fully correct query answers.

Consistency

If data is widely distributed over many nodes in a network, the cost of maintaining data consistency may be very high. The transaction manager must guarantee that all transactions are atomic and updates propagated to affected nodes so that the database is kept consistent. Two and three phase commit protocols are needed when more than one node is affected by an update to guarantee full update transparency. These protocols are expensive when many nodes are involved and relaxed update transparency may suffice to enable higher transaction performance. If the same kind data is present on many nodes updates must be propagated to all replicas, which can be very expensive in a geographically distributed database.

Regular distributed databases usually provide transaction atomicity as an option. However, because of the high cost of transaction atomicity modern distributed DBMS also provide the option to propagate updates lazily, thus compromising the consistency.

In a parallel DBMS running on a cluster, the nodes inside the cluster run DBMS kernel software which is completely controlled by the parallel DBMS. From the user's point of view it looks like a central DBMS; the main difference being the higher performance provided by parallelization of DBMS kernel software.

In regular distributed and parallel DBMSs a single database kernel manages all distributed data. All individual nodes are running the same distributed DBMS software. Different nodes may have different roles, e.g. some nodes handle query processing, some nodes handle locking, some nodes handle recovery, etc. The DBMS is a monolithic systems distributed over several nodes controlling the consistency of the individual nodes.

In general, consistent updates are difficult to achieve with heterogeneous databases since the participating databases are autonomous and the integrating DBMS may not have access to transaction managers of the participating databases.

In peer-to-peer databases the data consistency is relaxed for higher update and query performance. Data can be partly replicated for efficiency but the system does not guarantee consistency among the replicas so updates need not always be propagated to all replicas at every update. This means that queries may return less reliable result, which is often acceptable in a widely distributed database. This is similar to how search engines compromise query quality for performance.

Distributed Catalog Management

A particular problem for distributed databases is how and where to handle catalog data, such as the overall schema, statistics about data collections, the location of data collections, and how data collections are replicated and partitioned. The catalog information is accessed intensively by database users in queries and updates. On the other hand, in most DBMSs it is assumed that schema and catalog information changes slowly, which, for example, permits pre-compilation of (distributed) database queries. The assumption that catalog data changes slowly but is intensively accessed is a case for replicating catalog information on many nodes, in particular on those coordinating nodes with which the users interact. On the other hand, in a heterogeneous database with many participating autonomous nodes, the assumption that schemas and data placements do not change usually does not hold.

Regular distributed and parallel databases assume few participating non-autonomous nodes and the catalog is therefore replicated. Federated databases have a central architecture where all interaction with the database is through the global schema and it contains replications of catalog information from the participating databases. For mediators, multi-databases, and peer-to-peer there is no central global schema and the query processing nodes are autonomous. Therefore the catalogue data cannot be fully replicated and it will be up to different nodes to cache catalog data when needed. The validity of cached catalog data needs to be properly handled though; otherwise queries may fail or even return the wrong data.

Cross-references

► Data partitioning
► Data dictionary
► Data replication
► Distributed concurrecy control
► Distributed database design
► Distributed database systems
► Distributed query optimization
► Distributed transaction management
► Distributed DBMS
► Information Integration
► Mediation
► Parallel query processing
► Parallel Database Management
► Peer Data Management System
► Shared-Nothing Architecture
► View-based data integration

Recommended Reading

1. Beng Chin Ooi and Kian-Lee Tan (guest eds.). Introduction: special section on peer-to-peer-based data management. IEEE Trans. Knowl. Data Eng., 16(7):785–786, 2004.
2. Litwin W., Mark L., and Roussopoulos N. Interoperability of multiple autonomous databases ACM Comput. Surv., 22(3):267–293, 1990.
3. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems (2nd edn.). Prentice Hall, NJ, 1999.
4. Sheth A.P. and Larson J.A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv., 22(3):183–235, 1990.
5. Wiederhold G. Mediators in the architecture of future information systems. IEEE Comput., 25(3):38–49, 1992.

Distributed Commit Protocol

► Two-Phase Commit Protocol

Distributed Component Object Model

► DCOM

Distributed Computing Environment

► DCE

Distributed Concurrency Control

MATHIAS WESKE
University of Potsdam, Potsdam, Germany

Synonyms

Synchronizing distributed transactions

Definition

Distributed concurrency control provides concepts and technologies to synchronize distributed transactions in a way that their interleaved execution does not violate the ACID properties. Distributed transactions are executed in a distributed database environment, where a set of connected data servers host related data. A distributed transaction consists of a set of subtransactions, each of which is executed by one data server. Distributed concurrency control makes sure that all subtransactions of a set of distributed transactions are serialized identically in all data servers involved. Therefore, not only local dependencies need to be taken into account, but also dependencies involving multiple data servers. Concurrency control techniques known from centralized database systems need to be extended to cope with the new requirements imposed by the distribution aspect.

Historical Background

Distributed concurrency control was an emerging research topic in the early 1980's. Based on the seminal work on concurrency control and locking protocols in centralized database systems by Eswaran et al. [3], Gray proposes implementation approaches for transactions [4], and Spector and Schwarz investigate transactions in the context of distributed computing [9].

In distributed database environments, data relevant to a specific application domain is spread across a set of data servers, each of which hosts a partition of the data items [2,8]. The data servers form a distributed database federation. Distributed database federations are homogenous if they run the same database software; they are heterogeneous if they do not.

To explain the core of distributed concurrency control, homogeneous database federations are addressed first. Since transactions are representations of application programs and data is spread across a set of data servers in a distributed environment, transactions need to access multiple data servers. Transactions with this property are called distributed transactions. Distributed transactions consist of a set of subtransactions, each of which runs at one specific data server of the federation.

Distributed transactions – just like centralized transactions – need to satisfy the ACID properties. Conflict serializability is a theoretically proven and practically relevant technique to ensure isolation of centralized transactions. Given a pair of transactions, all conflicting data access operations of these transactions occur in the same order, i.e., the serialization order of the transactions involved.

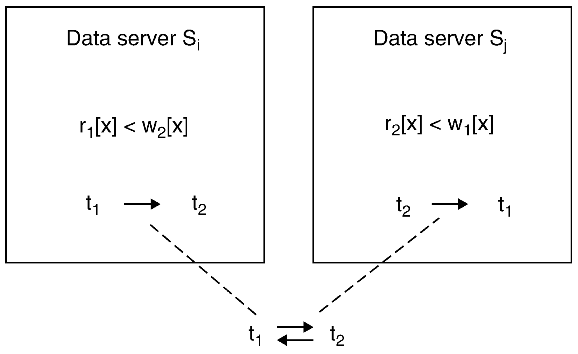
The complicating factor in the case of distribution is the lack of global knowledge. Assume transactions t_1 and t_2 running on data servers S_i and S_j. Subtransactions for t_1 and t_2 are created and started in S_i and S_j. These subtransactions are executed in the data sites according to the concurrency control technique in place. Since each data server is aware of the transactions and subtransactions that run locally, any serialization ordering between these transactions is fine, as long as no cycles in the global serialization graph are introduced.

As a result different serialization orders might emerge in the different data servers of the database federation. This is the case, if, for example, t_1 is serialized in S_i before t_2, while in S_j, t_2 is serialized before t_1. The serialization graph in S_i contains $t_1 \rightarrow t_2$, indicating that t_1 is serialized before t_2, while in S_j the serialization ordering is in opposite direction. Due to the lack of global knowledge, there is a cyclic serialization dependency, a violation of conflict serializability that none of the sites involved can detect. The distributed concurrency control problem is depicted in Fig. 1.

Foundations

In homogeneous database federations, distributed two phase locking is a variant of centralized two phase locking: All locks that a distributed transaction holds need to be held until no more locks need to be acquired by that transaction. This means, for instance, that a subtransaction of distributed transaction t_1 can only release its locks, if no subtransaction of t_1 will acquire additional locks. Local knowledge of a particular data server is insufficient to decide about unlocking data objects, because there can still be locking operations by other subtransactions of the transaction that are active in other data servers.

To provide a solution to this problem, information about distributed transaction needs to be



Distributed Concurrency Control. Figure 1. Distributed concurrency control problem: global serialization graph contains a cycle, although serialization graphs of data servers involved are acyclic.

communicated between the data servers involved. There are several approaches to providing this information. When atomic commitment protocols – such as the two phase commit protocol – are in place, then distributed two phase locking can take advantage from the commit protocols. To commit a distributed transaction *t*, the two phase commit protocol sends a prepare-to-commit message to all sites that host subtransactions of *t*. When a data server receives a prepare-to-commit message for a transaction *t*, *t* will no longer acquire new locks. When there is agreement on committing the distributed transaction, each data server commits the changes of its subtransaction of *t* and releases its locks.

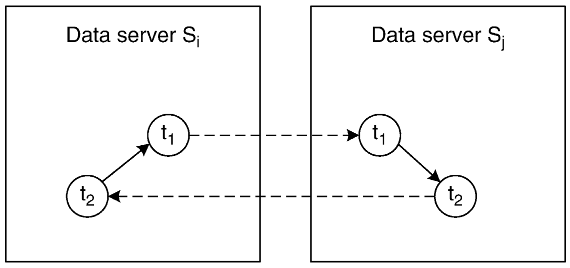
While distributed two phase locking solves the distributed concurrency control problem, the deadlock problem re-appears. Distributed deadlocks occur if conflicting locks are set by subtransactions of different transactions in a cyclic manner, and these locks are distributed among multiple sites of the database federation. As a result, no transaction can ever terminate successfully. In the example introduced above, if the subtransaction of t_1 in S_i holds a lock that the subtransaction of t_2 in S_i needs, in S_i, t_2 waits for the completion of t_1. A distributed deadlock exists if at the same time, t_1 waits for the completion of t_2 in S_j.

In this situation, there are local waiting conditions involving the subtransactions of t_1 and t_2, but there are also non-local waiting conditions: the subtransaction of t_1 in S_i waits for the completion of its subtransaction in S_j. Distributed deadlocks involve both local and non-local waiting conditions. This situation is shown in Fig. 2, marking local waiting conditions by solid edges and non-local waiting conditions by dotted edges.

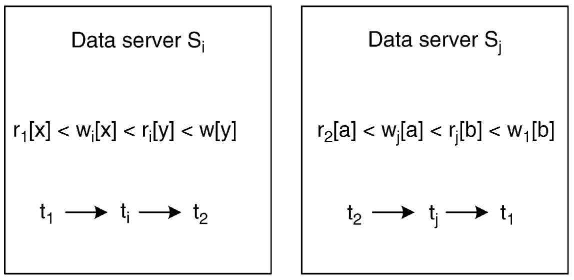
A simple approach to handling distributed deadlocks is timeout, where the time span that a lock can be held by a transaction is limited by a timeout value. If the timeout expires, the subtransaction is aborted. While distributed deadlocks are not possible using time-outs, potentially high abortion rates are introduced due to aborting transactions that are actually not involved in a deadlock cycle. More elaborate techniques are based on detecting distributed deadlocks, which is done by communicating information about waiting conditions between the data servers of a federation and aborting only those transactions that are actually involved in a distributed deadlock situation [6].

Timestamp ordering does not suffer from the deadlock problem; in this concurrency control technique, a strict ordering on the set of transactions is introduced. This ordering defines the order in which the transactions are serialized. Conflicting operations $p_i[x]$ and $q_j[x]$ are executed in the order $p_i[x]$ $q_j[x]$, if and only if $ts(t_i)$ $ts(t_j)$, where $ts(t)$ denotes the timestamp of transaction *t*. In the centralized case, assigning timestamps to transactions is performed by the scheduler that uses a logical clock to assign unique timestamps to newly arriving transactions Fig. 3.

In the distributed case, the definition of globally unique timestamps can be achieved by so called Lamport clocks, introduced in [7]. Lamport clocks use the axiom that a message always arrives after it was sent. This axiom is used to synchronize clocks in a distributed system. Each message carries a timestamp that reflects the local clock value of the sending system when the message was sent. When a message arrives with, for instance, timestamp 8, the receiving system sets its local clock to 9, should it be lower than that at the time of message arrival. To provide globally



Distributed Concurrency Control. Figure 2. Distributed deadlock situation due to waiting conditions of subtransactions involving multiple sites of a database federation.


Data server S_i
$r_1[x] < w_i[x] < r_i[y] < w[y]$
$t_1 \longrightarrow t_i \longrightarrow t_2$
Data server S_j
$r_2[a] < w_j[a] < r_j[b] < w_1[b]$
$t_2 \longrightarrow t_j \longrightarrow t_1$

Distributed Concurrency Control. Figure 3. Due to indirect conflicts with local transactions in heterogeneous database federations, global transaction manager cannot detect violations of global serialization orderings.

unique timestamps for transactions, each timestamp also contains the identifier of the data server from which the transaction originated. If these mechanisms regarding timestamps are in place in the distributed database federation, concurrency control can be done by the timestamp ordering technique, just like in the centralized case.

To summarize, distributed concurrency control in homogeneous database federations provides synchronization transparency to application programs by using variations of concurrency control techniques that are in place in centralized database systems [1]. The situation is more complex in heterogeneous federations, where data servers run different database management systems or are legacy systems that provide their functionality via application programming interfaces that allow us to access data. Regardless of the particular system in place, however, it must allow subtransactions of distributed transactions to read and write data items.

The techniques that work well in the case of homogeneous federations cannot immediately be used in the heterogeneous case. To cope with the heterogeneity issue, a software component is introduced, responsible for controlling the order in which subtransactions and their operations are issued to the data servers. This software component is called global transaction manager.

While the global transaction manager can control the ordering of subtransactions and their operations, it cannot influence the ordering of local transactions and their operations in the data servers. These local transactions can introduce serialization problems that the global transaction manager cannot take care of. These problems are due to indirect conflicts between subtransactions of distributed transactions. Indirect conflicts involve local transactions, each of which has a direct conflict with a subtransaction of a distributed transaction. These conflicts lead to serialization orderings between subtransactions that are not visible by the global transaction manager.

Assume data servers S_i and S_j are in place, global transactions t_1,t_2, that access both data servers. There are local transactions t_i in S_i and t_j in S_j. In data server S_i, the local transaction manager can serialize $t_1 t_i t_2$, while data server S_j serializes the transactions as t_2,t_j, t_1. Assuming that there are no direct conflicts between the subtransactions of t_1 and t_2 in place, the non-matching serialization cannot be detected by the global transaction manager.

There are several approaches to deal with this problem. Global serializability can be achieved by local guarantees, for instance the property of rigorousness, discussed in [10]. There are also approaches that introduce direct conflicts between distributed transactions in the data servers and, thus, make sure that the local serialization orderings of the data servers are in line.

An approach to solve the distributed concurrency control problem in heterogeneous federations is based on global transaction managers creating direct conflicts between subtransactions of distributed transactions by introducing additional data access operations. These operations force local direct conflicts that are handled by the individual data servers, using their respective concurrency control techniques. These data items are known as tickets, and the operations as take-ticket operations.

To start accessing local data items, each subtransaction of a distributed transactions first needs to take a ticket and issues the respective operation. This operation reads the current value of the ticket (a data item stored in the data server) and increments it, thus forcing direct conflicts between any two subtransactions accessing data in a particular data server of a heterogeneous federation. This technique makes sure that there are always direct conflicts between subtransactions of distributed transactions. However, the conflicts can still have different orientation, and the serialization orderings might still be different.

The information about the conflicts needs to be communicated to the global transaction manager; there are two variants to do so, an optimistic and a conservative one. In the optimistic variant, there is no restriction on the order in which subtransactions take their tickets in the data servers. The global transaction manager maintains a global ticket graph that represents the ordering in which distributed transactions take tickets. This graph contains an edge $t_i \rightarrow t_j$ if and only if both distributed transactions have subtransactions in one data server, and the ticket of t_i has a lower value than the ticket of t_j at that site. In this case t_i is serialized before t_j.

On commit of a distributed transaction t, the global transaction manager checks the global ticket graph. A loop in this graph indicates a serialization violation, leading to aborting *t*. Just like in the centralized case, if there is little contention, and few conflicts are in place, the optimistic method is advantageous. However, high contention leads to many conflicts, and the probability of transaction abortions increases.

In the conservative ticketing method, the order in which tickets are taken is restricted by the global transaction manager. A subtransaction is allowed to take a ticket only if the global serialization ordering is not violated by this taking a ticket. While at run time there is additional overhead in checking whether a take-ticket operation can be permitted, the conservative approach reduces the number of abort operations. Therefore, the conservative ticketing method is advantageous in case of high contention, while the optimistic approach is likely to perform better in low contention scenarios.

Future Directions

The internet has and is continuing to pose new requirements to existing solutions in various areas of database technology, including distributed concurrency control. If applications use resources and data provided by different organizations on the internet, transactional properties are ver/y hard, if not impossible, to satisfy. As a result, new concepts and technologies are sought to provide application level consistency. As one representative of these future directions of research, the work by Pat Helland in the context of internet scale applications and their challenges is mentioned [5].

Cross-references

► ACID Properties
► Conflict Serializability
► Timestamp Ordering
► Two Phase Commit
► Two Phase Locking

Recommended Reading

1. Bernstein P.A. and Goodman N. Concurrency control in distributed database systems. ACM Comput Surv., 13 (2):185–221, 1981.
2. Ceri S. and Pelagatti G. Distributed Databases: Principles and Systems. McGraw-Hill, NY, USA, 1984.
3. Eswaran K.P., Gray J.N., Lorie R.A., and Traiger I.L. The notions of consistency and predicate locks in a database system. Commun. ACM, 19(11):624–633, 1976.
4. Gray J.N. The transaction concept: virtues and limitations. In Proc. 7th Int. Conf. on Very Data Bases, 1981, pp 144–154.
5. Helland P. Life beyond distributed transactions: an Apostate's opinion In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 132–141.
6. Knapp E. Deadlock detection in distributed databases. ACM Comput. Surv., 19(4):303–328, 1987.
7. Lamport L. Time, clocks, and the ordering of events in a distributed system. Commun. ACM, 21(7):558–565, 1978.
8. Öszu T. and Valduriez P. Principles of distributed database systems. 2nd edn. Prentice-Hall, 1999.
9. Spector A.Z. and Schwarz P.M. Transactions: a construct for reliable distributed computing. ACM Operat. Syst. Rev., 17 (2):18–35, 1983.
10. Weikum G. and Vossen G. Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, 2002.

Distributed Data Streams

Minos Garofalakis
Technical University of Crete, Chania, Greece

Definition

A majority of today's data is constantly evolving and fundamentally distributed in nature. Data for almost any large-scale data-management task is continuously collected over a wide area, and at a much greater rate than ever before. Compared to traditional, centralized stream processing, querying such large-scale, evolving data collections poses new challenges, due mainly to the physical distribution of the streaming data and the communication constraints of the underlying network. Distributed stream processing algorithms should guarantee efficiency not only in terms of *space* and *processing time* (as conventional streaming techniques), but also in terms of the *communication load* imposed on the network infrastructure.

Historical Background

The prevailing paradigm in database systems has been understanding the management of *centralized* data: how to organize, index, access, and query data that is held centrally on a single machine or a small number of closely linked machines. Work on parallel and distributed databases has focused on different notions of consistency and methods for effectively distributing query execution plans over multi-node architectures – the issues of monitoring or querying distributed, high-speed data streams in a space-, time- and communication-efficient manner were not addressed in this realm. Similarly, the bulk of early research on data-streaming algorithms and systems has concentrated on a *centralized* model of computation, where the stream-processing engine has direct access to all the streaming data records. Centralized stream-processing models can obviously ignore communication-efficiency issues; still, such models are also painfully inadequate for many of the prototypical data-streaming applications, including IP-network and sensornet monitoring.

Foundations

Tracking and querying large-scale, evolving data collections poses a number of challenges. First, in contrast with conventional, centralized models of data-stream processing, the task is inherently *distributed*; that is, the underlying infrastructure comprises several remote sites (each with its own local data source) that can exchange information through a communication network. This also means that there typically are important *communication constraints* owing to either network-capacity restrictions (e.g., in IP-network monitoring, where the volumes of collected utilization and traffic data can be huge [7]), or power and bandwidth restrictions (e.g., in wireless sensor networks, where communication overhead is the key factor in determining sensor battery life [18]). Second, each remote site may see a *high-speed stream* of data and, thus, must solve a local (centralized) stream-processing problem within its own local resource limitations, such as *space* or *CPU-time* constraints. This is certainly true for IP routers (that cannot possibly store the log of all observed packet traffic at high network speeds), as well as wireless sensor nodes (that, even though may not observe large data volumes, typically have very little memory on-board). Finally, applications often require *continuous monitoring* of the underlying streams (i.e., real-time tracking of measurements or events), not merely one-shot responses to sporadic queries.

To summarize, the focus is on techniques for processing queries over collections of remote data streams. Such techniques have to work in a distributed setting (i.e., over a communication network), support one-shot or continuous query answers, and be space, time, and communication efficient. It is important to note that, for most realistic distributed streaming applications, the naive solution of collecting all the data in a single location is simply *not* a viable option: the volume of data collection is too high, and the capacity for data communication relatively low. Thus, it becomes critical to exploit local processing resources to effectively minimize the burden on the communication network. This establishes the fundamental concept of "*in-network processing*:" if more computational work can be done *within* the network to reduce the communication needed, then it is possible to significantly improve the value of the network, by increasing its useful life and communication capacity, and extending the range of computations possible over the network. This is a key idea that permeates the bulk of existing work on distributed data-stream processing – this work can, in general, be characterized along three (largely orthogonal) axes:

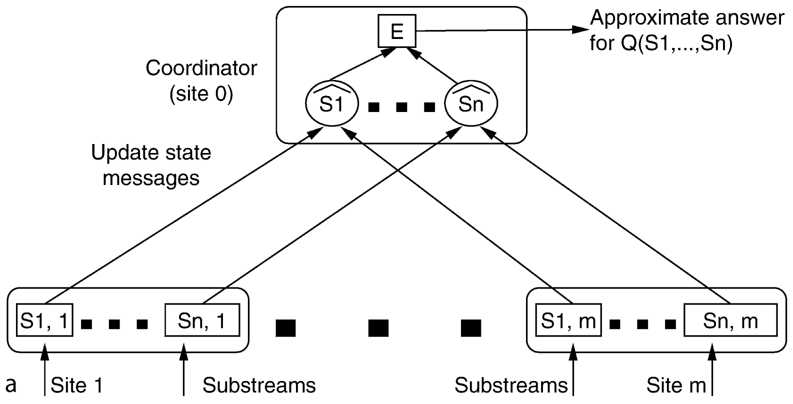
1. *Querying Model:* There are two broad classes of approaches to in-network query processing, by analogy to types of queries in traditional DBMSs. In the *one-shot* model, a query is issued by a user at some site, and must be answered by "pulling" the current state of data in the network. For simple aggregates, this can be done in a few rounds of communication where only small, partial-aggregate messages are exchanged over a spanning tree of the network. For more complex, *holistic* aggregates (that depend on the complete data distribution, such as quantiles, topk-*k*, count-distinct, and so on), simple combination of partial results is insufficient, and instead clever composable summaries give a compact way to accurately approximate query answers.

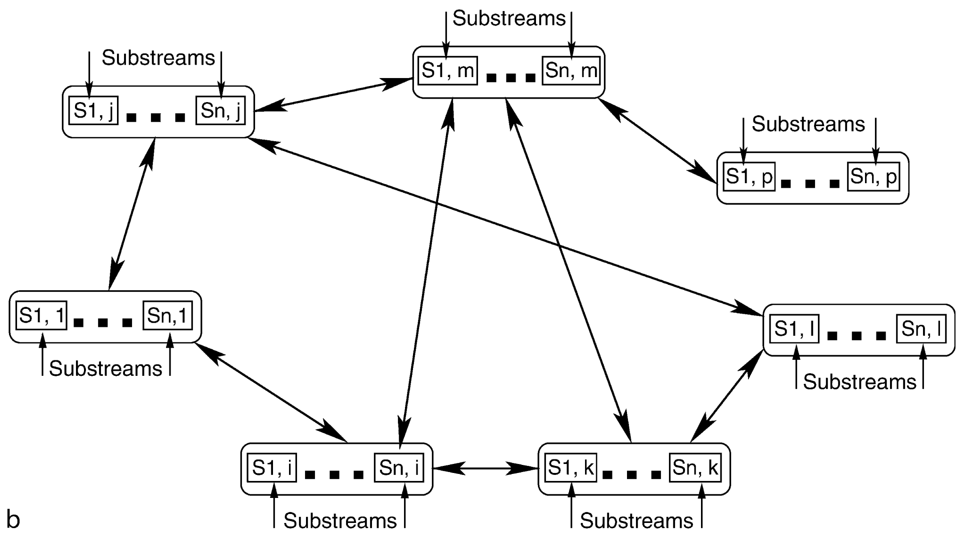
In the *continuous* model, users can register a query with the requirement that the answer be tracked continuously. For instance, a special case of such a continuous query is a *distributed trigger* that must fire in (near) real-time when an aggregate condition over a collection of distributed streams is satisfied (e.g., to catch anomalies, SLA violations, or DDoS attacks in an ISP network). This continuous monitoring requirement raises further challenges, since, even using tree computation and summarization, it is still too expensive to communicate every time new data is received by one of the remote sites. Instead, work on continuous distributed streams has focused on "push-based" techniques that tradeoff result accuracy for reduced communication cost, by apportioning the error in the query answer across *filter* conditions installed locally at the sites to reduce communication.

Approximation and *randomization* techniques are also essential components of the distributed stream querying model, and play a critical role in minimizing communication. Approximate answers are often sufficient when tracking the statistical properties of large-scale distributed systems, since the focus is typically on indicators or patterns rather than precisely-defined events. This is a key observation, allowing for techniques that effectively tradeoff efficiency and approximation accuracy.

2. *Communication Model:* The architecture and characteristics of the underlying communication network have an obvious impact on the design of effective distributed stream processing techniques. Most existing work has focused on *hierarchical* (i.e., tree) network architectures, due to both their conceptual simplicity and their importance for practical scenarios (e.g., sensornet routing trees [18]). As an example, Fig. 1a depicts a simple *single-level* hierarchical model with $m + 1$ sites and n (distributed) update streams. Stream updates arrive continuously at the remote sites 1,. . .,m, whereas site 0 is a special *coordinator* site that is responsible for generating answers to (one-shot or continuous) user queries Q over the n distributed streams. In this simple hierarchical model, the m remote sites do not communicate with each other; instead, as illustrated in Fig. 1a, each remote site exchanges messages only with the coordinator, providing it with state information for (sub)streams observed locally at the site.

More general, *multi-level* hierarchies have individual substream-monitoring sites at the leaves and internal nodes of a general *communication tree*, and the goal is to effectively answer or track a stream query $Q(S_1,\ldots, S_n)$ at the *root node* of the tree. The most general setting are *fully-distributed* models, where individual monitor sites are connected through an arbitrary underlying *communication network* (Fig. 1b);





Distributed Data Streams. Figure 1. (a) Single-level hierarchical stream-processing model. (b) Fully-distributed model.

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:56

this is a distinctly different distributed system architecture since, unlike hierarchical systems, no centralized authority/coordination exists and the end goal is for all the distributed monitors to efficiently reach some form of *consensus* on the answer of a distributed stream query.

Besides the connectivity model, other important network characteristics for distributed stream processing include: the potential for broadcasting or multicasting messages to sites (e.g., over a limited radio range as in wireless sensornets), and the node/link-failure and data-loss characteristics of the supporting hardware.

3. *Class of Queries:* The key dichotomy between simple, *non-holistic* aggregate queries (e.g., MIN, SUM, AVG) and *holistic* aggregates (e.g., median) has already been discussed; clearly, holistic aggregates introduce many more challenges for efficient distributed streaming computation. Another important distinction is that between *duplicate-sensitive* aggregates (that support bag/multi-set semantics, such as median, SUM, or top-k) and *duplicate-insensitive* aggregates (that support set semantics, such as MIN or count-distinct). Finally, another important class is that of complex *correlation* queries that combine/correlate streaming data across different remote sites (e.g., through a streaming *join* computation). Such correlations can be critical in understanding important trends and making informed decisions about measurement or utilization patterns. Different classes of streaming queries typically require different algorithmic machinery for efficient distributed computation.

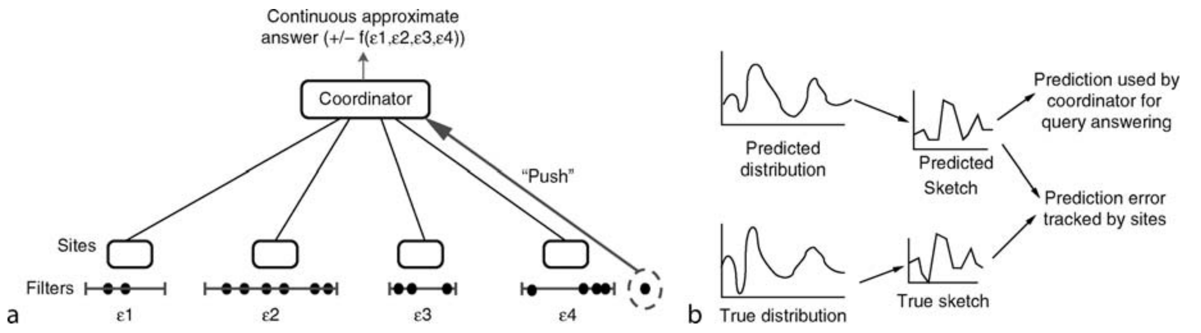
The remainder of this section provides a brief overview of some key results in distributed data streaming, for both the one-shot and continuous querying models, and concludes with a short survey of systems-related efforts in the area.

One-Shot Distributed Stream Processing. Madden et al. [18] present simple, exact *tree-based aggregation* schemes for sensor networks and propose a general framework based on *generate*, *fuse*, and *evaluate* functions for combining partial results up the aggregation tree. They also propose a classification of different aggregate queries based on different properties, such as duplicate in/sensitivity, example or summary results, monotonicity, and whether the aggregate is *algebraic* or *holistic* (which essentially translates to whether the intermediate partial state is of constant size or growing). While the exact computation of holistic aggregates requires linear communication cost, guaranteed-quality *approximate* results can be obtained at much lower cost by approximating intermediate results through *composable data synopses* [1,9].

Robustness is a key concern with such hierarchical aggregation schemes, as a single failure/loss near the root of the tree can have a dramatic effect on result accuracy. *Multi-path routing* schemes mitigate this problem by propagating partial results along multiple different paths. This obviously improves reliability and reduces the impact of potential failures; in addition, this improved reliability often comes essentially "for free" (e.g., in wireless sensornets where the network is a natural broadcast medium). Of course, multi-path routing also implies that the same partial results can be accounted for multiple times in the final aggregate. As observed by Nath et al. [20], this duplication has no effect on aggregates that are naturally *Order and Duplicate Insensitive (ODI)*, such as MIN and MAX; on the other hand, for non-ODI aggregates, such as SUM and COUNT, *duplicate-insensitive sketch synopses* (e.g., based on the Flajolet-Martin sketch [9]) can be employed to give effective, low-cost, multi-path approximations [20]. Hybrid approaches combining the simplicity of tree aggregation (away from the root node) and the robustness of multi-path routing (closer to the root) have also been explored [19].

Gossip (or, *epidemic*) protocols for spreading information offer an alternative approach for robust distributed computation in the more general, fully-distributed communication model (Fig. 1b). ODI aggregates (and sketches) naturally fit into the gossiping model, which basically guarantees that all n nodes of a network will converge to the correct global ODI aggregate/sketch after O(logn) rounds of communication. For non-ODI aggregates/sketches, Kempe et al. [15] propose a novel gossip protocol (termed *push-sum*) that also guarantees convergence in a logarithmic number of rounds, and avoids double counting by splitting up the aggregate/sketch and ensuring "conservation of mass" in each round of communication.

Continuous Distributed Stream Processing. The continuous model places a much more stringent demand on the distributed stream processing engine, since remote sites must collaborate to *continuously* maintain a query answer that is accurate (e.g., within specified error bounds) based on the current state of the stream(s). Approximation plays a critical role in the design of communication-efficient solutions for such

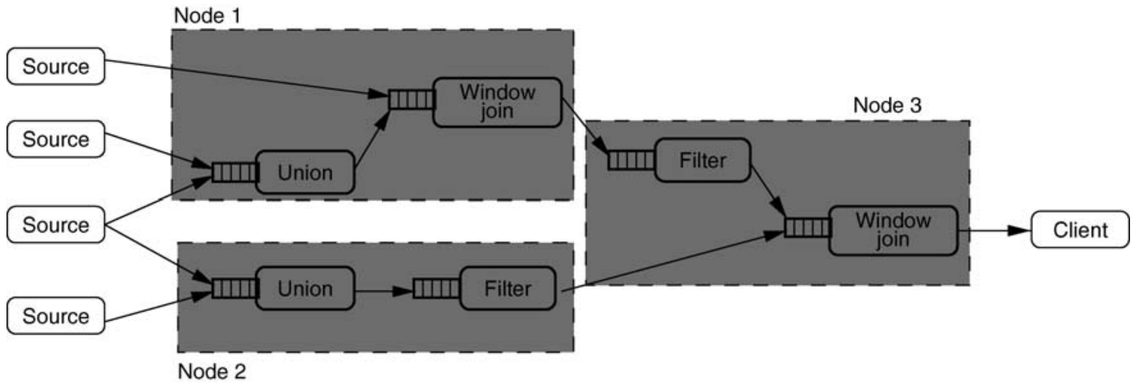

Continuous approximate answer (+/− f(ε1,ε2,ε3,ε4))
Coordinator
"Push"
Sites
Filters
a
ε1
ε2
ε3
ε4
Predicted distribution
Predicted Sketch
Prediction used by coordinator for query answering
Prediction error tracked by sites
b
True distribution
True sketch

Distributed Data Streams. Figure 2. (a) Using local filters for continuous distributed query processing: Most updates fall within the local-filter ranges and require no communication with the coordinator (that can provide approximate answers with guarantees depending on the filter "widths"); only updates outside the local-filter range require new information to be "pushed" by the local site to the coordinator. (b) Prediction-based approximate query tracking: Predicted sketches are based on simple prediction models of local-stream behavior, and are kept in-sync between the coordinator (for query answering) and the remote sites (for tracking prediction error).

continuous monitoring tasks. In a nutshell, the key idea is to tradeoff result accuracy and local processing at sites for reduced communication costs, by installing *local filters* at the remote sites to allow them to only "push" significant updates to the coordinator; of course, these distributed local filters would have to be *safe*, that is, they should guarantee the overall error bound for the global query result (based on the exact current state) at the coordinator. This idea of local traffic filtering for continuous distributed queries is pictorially depicted in Fig. 2a.

A key concept underlying most continuous distributed monitoring schemes is that of *adaptive slack allocation* – that is, adaptively distributing the overall "slack" (or, error tolerance) in the query result across the local filters at different participating sites based on observed local update patterns. Obviously, the complexity of such slack-distribution mechanisms depends on the nature of the aggregate query being tracked. Olston et al. [21] consider the simpler case of algebraic aggregates (where breaking down the overall slack to safe local filters is straightforward), and discuss adaptive schemes that continuously grow/shrink local filters based on the frequency of observed local violations. As expected, the situation is more complicated in the case of holistic aggregates: Babcock and Olston [2] discuss a scheme for tracking an approximate global top-k set of items using a cleverly-built set of local constraints that essentially "align" the *local* top-k set at a site with the *global* top-k; furthermore, their algorithm also retains some amount of slack at the coordinator to allow for possible localized resolutions of constraint violations. Das et al. [8] consider the problem of monitoring distributed set-expression cardinalities and propose tracking algorithms that take advantage of the set-expression semantics to appropriately "charge" updates arriving at the local sites.

Simple slack-allocation schemes are typically based on a naive *static model* of local-site behavior; that is, the site's "value" is assumed constant since the last update to the coordinator, and communication is avoided as long as this last update value stays within the slack bounds. Cormode and Garofalakis [5] propose the use of more sophisticated, *dynamic prediction models* of temporal site dynamics in conjunction with appropriate sketching techniques for communication-efficient monitoring of complex distributed aggregate queries. Their idea is to allow each site and the coordinator to share a prediction of how the site's local stream(s) (and, their sketch synopses) evolve over time. The coordinator uses this prediction to provide continuous query answers, while the remote site checks locally that the prediction stays "close" to the actual observed streaming distribution (Fig. 2b). Of course, using a more sophisticated prediction model can also impose some additional communication to ensure that the coordinator's view is kept in-sync with the up-to-date local stream models (at the remote sites). Combined with intelligent sketching techniques and methods for bounding the overall query error, such approaches can be used to track a large class of complex, holistic queries, only requiring concise communication



Distributed Data Streams. Figure 3. Distributed stream-processing dataflow.

exchanges when prediction models are no longer accurate [5]. Furthermore, their approach can also be naturally extended to multi-level hierarchical architectures. Similar ideas are also discussed by Chu et al. [4] who consider the problem of *in-network probabilistic model maintenance* to enable communication-efficient approximate tracking of sensornet readings.

A common feature of several distributed continuous monitoring problems is continuously evaluating a condition over distributed streaming data, and *firing* when the condition is met. When tracking such *distributed triggers*, only values of the "global" continuous query that are above a certain threshold are of interest (e.g., fire when the total number of connections to an IP destination address exceeds some value) [13]. Recent work has addressed versions of this distributed triggering problem for varying levels of complexity of the global query, ranging from simple counts [16] to complex functions [26] and matrix-analysis operators [12]. Push-based processing using local-filter conditions continues to play a key role for distributed triggers as well; another basic idea here is to exploit the threshold to allow for even more effective local traffic filtering (e.g., "wider" yet safe filter ranges when the query value is well below the threshold).

Systems and Prototypes. Simple, algebraic in-network aggregation techniques have found widespread acceptance in the implementation of efficient sensornet monitoring systems (e.g., TAG/TinyDB [18]). On the other hand, more sophisticated approximate in-network processing tools have yet to gain wide adoption in system implementations. Of course, Distributed Stream-Processing Engines (DSPEs) are still a nascent area for systems research: only a few research prototypes are currently in existence (e.g., Telegraph/TelegraphCQ [25], Borealis/Medusa [3], P2 [17]). The primary focus in these early efforts has been on providing effective system support for *long-running stream-processing dataflows* (comprising connected, pipelined query operators) over a distributed architecture (Fig. 3). For instance, Balazinska et al. [3] and Shah et al. [25] discuss mechanisms and tools for supporting parallel, highly-available, fault-tolerant dataflows; Loo et al. [17] propose tools for declarative dataflow design and automated optimizations; Pietzuch et al. [22] consider the problem of distributed dataflow operator placement and propose techniques based on a cost-space representation that optimize for network-efficiency metrics (e.g., bandwidth, latency); finally, Xing et al. [27] give tools for deriving distributed dataflow schedules that are resilient to load variations in the input data streams. To deal with high stream rates and potential system overload, these early DSPEs typically employ some form of *load shedding* [3] where tuples from operators' input stream(s) are dropped (either randomly or based on different QoS metrics). Unfortunately, such load-shedding schemes cannot offer any hard guarantees on the quality of the resulting query answers. A mechanism based on *revision tuples* can be employed in the Borealis DSPE to ensure that results are *eventually correct* [3]. AT&T's Gigascope streaming DB for large-scale IP-network monitoring [7] uses approximation tools (e.g., sampling, sketches) to efficiently track "line-speed" data streams at the monitoring endpoints, but has yet to explore issues related to the physical distribution of the streams and holistic queries.

Key Applications

Enterprise and ISP Network Security: The ability to efficiently track network-wide traffic patterns plays a key role in detecting anomalies and possible malicious attacks on the network infrastructure. Given the sheer volume of measurement data, continuously centralizing all network statistics is simply not a feasible option, and distributed streaming techniques are needed.

Sensornet Monitoring and Data Collection: Tools for efficiently tracking global queries or collecting all measurements from a sensornet have to employ clever in-network processing techniques to maximize the lifetime of the sensors.

Clickstream and Weblog Monitoring: Monitoring the continuous, massive streams of weblog data collected over distributed web-server collections is critical to the real-time detection of potential system abuse, fraud, and so on.

Future Directions

The key algorithmic idea underlying the more sophisticated distributed data-stream processing techniques discussed in this article is that of effectively trading off space/time *and communication* with the quality of an approximate query answer. Exploring some of the more sophisticated algorithmic tools discussed here in the context of real-life systems and applications is one important direction for future work on distributed streams; other challenging areas for future research, include:

- Extensions to other application areas and more complex communication models, e.g., monitoring P2P services over shared infrastructure (OpenDHT [23] over PlanetLab), and dealing with constrained communication models (e.g., intermittent-connectivity and delay-tolerant networks (DTNs) [14]).
- Richer classes of distributed queries, e.g., set-valued query answers, machine-learning inference models [11].
- Developing a theoretical/algorithmic foundation of distributed data-streaming models: What are fundamental lower bounds, how to apply/extend information theory, communication complexity, and distributed coding. Some initial results appear in the recent work of Cormode et al. [6].
- Richer prediction models for stream tracking: Can models effectively capture site correlations rather than just local site behavior? More generally, understand the model complexity/expressiveness tradeoff, and come up with principled techniques for capturing it in practice (e.g., using the MDL principle [24]).
- Stream computations over an *untrusted* distributed infrastructure: Coping with privacy and authentication issues in a communication/computation-efficient manner. Some initial results appear in [10].

Data Sets

Publicly-accessible network-measurement data collections can be found at the Internet Traffic Archive: (http://ita.ee.lbl.gov/), and CRAWDAD (the Community Resource for Archiving Wireless Data at Dartmouth, http://cmc.cs.dartmouth.edu/data/dartmouth.html).

Cross-references

► AMS Sketch
► Continuous Query
► Count-Min Sketch
► Data Streams
► Load Shedding
► Scheduling Strategies
► Stream Models
► Stream Processing
► Stream Sampling
► Streaming Applications
► Synopsis Structures

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th Annual ACM Symp. on the Theory of Computing. Philadelphia, Pennsylvania, May 1996, pp. 20–29.
2. Babcock B. and Olston C. Distributed top-K monitoring. In Proc. 2003 ACM SIGMOD Int. Conf. on Management of Data. San Diego, California, June 2003.
3. Balazinska M., Balakrishnan H., Madden S., and Stonebraker M. Fault-tolerance in the borealis distributed stream processing system. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
4. Chu D., Deshpande A., Hellerstein J.M., and Hong W. Approximate data collection in sensor networks using probabilistic models. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
5. Cormode G. and Garofalakis M. Sketching streams through the net: distributed approximate query tracking. In: Proc. 31st Int. Conf. on Very Large Data Bases, 2005.
6. Cormode G., Muthukrishnan S., and Yi K. Algorithms for distributed functional monitoring. In Proc. 19th Annual ACM-SIAM Symp. on Discrete Algorithms, 2008.
7. Cranor C., Johnson T., Spatscheck O., and Shkapenyuk V. Gigascope: a stream database for network applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
8. Das A., Ganguly S., Garofalakis M., and Rastogi R. Distributed set-expression cardinality estimation. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.

9. Flajolet P. and Nigel Martin G. Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci., 31:182–209, 1985.
10. Garofalakis M., Hellerstein J.M., and Maniatis P. Proof sketches: verifiable in-network aggregation. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
11. Guestrin C., Bodik P., Thibaux R., Paskin M., and Madden S. Distributed regression: an efficient framework for modeling sensor network data. Inform. Process. Sensor Networks, 2004.
12. Huang L., Nguyen X., Garofalakis M., Hellerstein J.M., Jordan M.I., Joseph A.D., and Taft N. Communication-efficient online detection of network-wide anomalies. In Proc. 26th Annual Joint Conf. of the IEEE Computer and Communications Societies, 2007.
13. Jain A., Hellerstein J., Ratnasamy S., and Wetherall D. A wakeup call for internet monitoring systems: The case for distributed triggers. In Proc. Third Workshop on Hot Topics in Networks (Hotnets). 2004.
14. Jain S., Fall K., and Patra R. Routing in a delay tolerant network. In Proc. ACM Int. Conf. of the on Data Communication, 2005.
15. Kempe D., Dobra A., and Gehrke J. Gossip-based computation of aggregate information. In Proc. 44th Annual IEEE Symp. on Foundations of Computer Science. 2003.
16. Keralapura R., Cormode G., and Ramamirtham J. Communication-efficient distributed monitoring of thresholded counts. In - Proc. 2006 ACM SIGMOD Int. Conf. on Management of Data. Chicago, Illinois, June 2006, pp. 289–300.
17. Loo B.T., Condie T., Garofalakis M., Gay D.E., Hellerstein J.M., Maniatis P., Ramakrishnan R., Roscoe T., and Stoica I. Declarative networking: language, execution, and optimization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.
18. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. TAG: a tiny aggregation service for ad-hoc sensor networks. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002.
19. Manjhi A., Nath S., and Gibbons P. Tributaries and deltas: efficient and robust aggregation in sensor network streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
20. Nath S., Gibbons P.B., Seshan S., and Anderson Z.R. Synopsis diffusion for robust aggrgation in sensor networks. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems. 2004.
21. Olston C., Jiang J., and Widom J. Adaptive filters for continuous queries over distributed data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
22. Pietzuch P., Ledlie J., Schneidman J., Roussopoulos M., Welsh M., and Seltzer M. Network-aware operator placement for stream-processing systems. In: Proc. 22nd Int. Conf. on Data Engineering, 2006.
23. Rhea S., Godfrey B., Karp B., Kubiatowicz J., Ratnasamy S., Shenker S., Stoica I., and Yu H.Y. OpenDHT: a public dht service and its uses. In Proc. ACM Int. Conf. of the on Data Communication, 2005.
24. Rissanen J. Modeling by shortest data description. Automatica, 14:465–471, 1978.
25. Shah M.A., Hellerstein J.M., and Brewer E. Highly available, fault-tolerant, parallel dataflows. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
26. Sharfman I., Schuster A., and Keren D. A geometric approach to monitoring threshold functions over distributed data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 301–312.
27. Xing Y., Hwang J.-H., Cetintemel U., and Zdonik S. Providing resiliency to load variations in ditributed stream processing. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.

Distributed Database Design

Kian-Lee Tan
National University of Singapore, Singapore, Singapore

Synonyms

Horizontal fragmentation; Vertical fragmentation; Data replication

Definition

Distributed database design refers to the following problem: given a database and its workload, how should the database be split and allocated to sites so as to optimize certain objective function (e.g., to minimize the resource consumption in processing the query workload). There are two issues: (i) Data fragmentation which determines how the data should be fragmented; and (ii) Data allocation which determines how the fragments should be allocated. While these two problems are inter-related, the two issues have traditionally been studied independently, giving rise to a two-phase approach to the design problem.

The design problem is applicable when a distributed database system has to be built from scratch. In the case when multiple existing databases are to be integrated (e.g., in multi-database context), there is no design issue.

Historical Background

In a distributed database system, relations are typically fragmented and stored at multiple sites. Fragmentation of a relation is useful for several reasons. First, an application typically accesses only subsets of relations. Moreover, different subsets are naturally needed at different sites. As such, fragmenting a relation to facilitate locality of accesses of applications can improve performance (otherwise, the overhead of shipping relations from one site to another may be unnecessarily high). Second, as applications may operate on different fragments, the degree of concurrency is thus enhanced. Even within a single application, accessing multiple fragments that are located at different sites essentially facilitates parallelism. Third, for applications that require multiple fragments from different relations, these fragments can be colocated to minimize communication overhead. However, like normalization, decomposing a relation to fragments may lead to poorer performance when multiple fragments need to be joined. In addition, if two attributes with a dependency are split across fragments, then it becomes more costly to enforce the dependency. Most of the works on fragmentation were done in the early 1980s [4,5,11,12,13]. One of the fundamental task in vertical fragmentation is to identify attributes that should be grouped together. In [11,9], the bond energy algorithm (BEA) was first proposed to cluster together attributes with high affinity for each other; based on this, the relation is then partitioned accordingly.

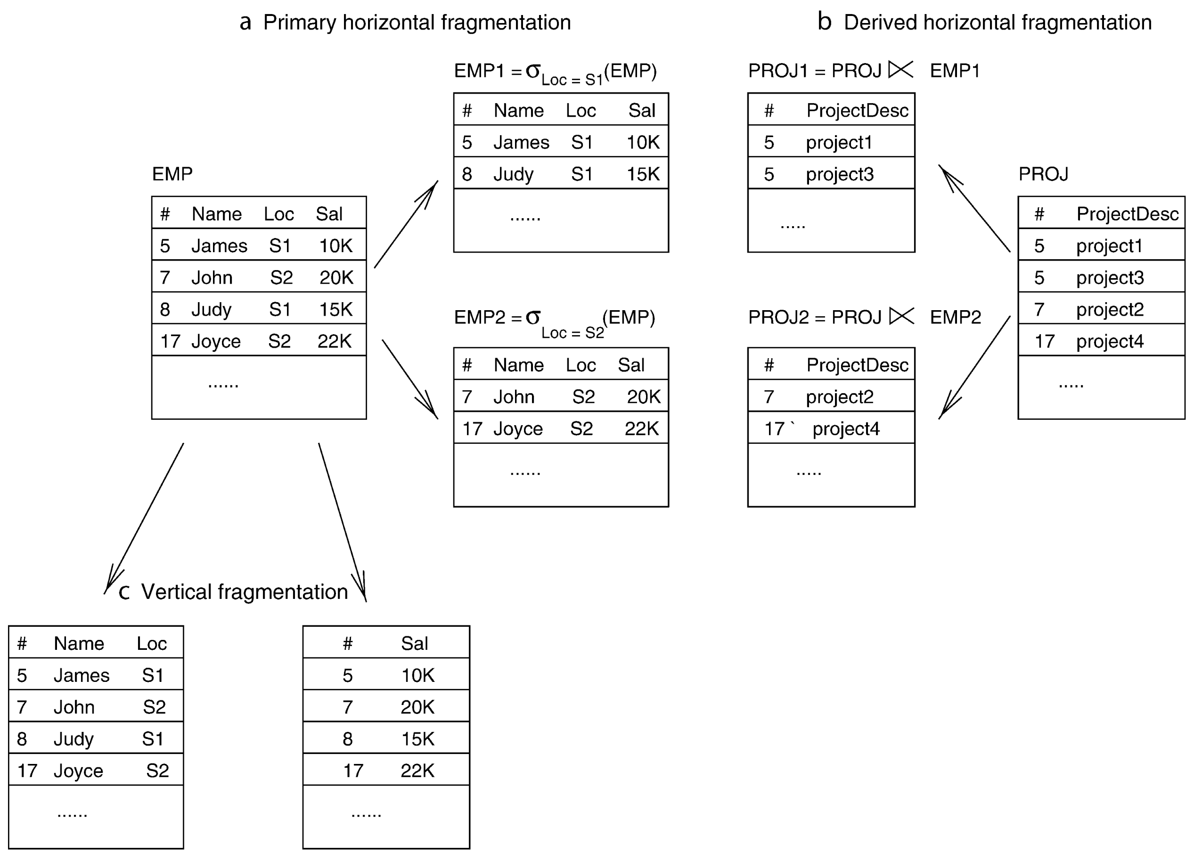
The allocation problem received much more attention. Initial work dated back to as early as 1969 where the file allocation problem was investigated [7]. In [1,2,6], the data allocation problem was shown to be NP-hard. In a dynamic environment, the workload and access pattern may change. In [3,8], dynamic data allocation algorithms were studied. These techniques change the initial data allocation to adapt to changing access patterns and workload.

There were also several works that combine both fragmentation and allocation into an integrated solution [13,14].

Foundations

Fragmentation

In a distributed database system, a relation R may be split in a number of fragments $F = \{R_1, R_2, ..., R_n\}$ in such a way that R can be reconstructed from them. There are essentially three fragmentation schemes.


a Primary horizontal fragmentation
b Derived horizontal fragmentation
c Vertical fragmentation
EMP
EMP1 = $\sigma_{Loc = S1}$(EMP)
EMP2 = $\sigma_{Loc = S2}$(EMP)
PROJ
PROJ1 = PROJ ⋉ EMP1
PROJ2 = PROJ ⋉ EMP2

EMP

#	Name	Loc	Sal
5	James	S1	10K
7	John	S2	20K
8	Judy	S1	15K
17	Joyce	S2	22K
......			

EMP1 = $\sigma_{Loc = S1}$(EMP)

#	Name	Loc	Sal
5	James	S1	10K
8	Judy	S1	15K
......			

EMP2 = $\sigma_{Loc = S2}$(EMP)

#	Name	Loc	Sal
7	John	S2	20K
17	Joyce	S2	22K
......			

PROJ

#	ProjectDesc
5	project1
5	project3
7	project2
17	project4
.....	

PROJ1 = PROJ ⋉ EMP1

#	ProjectDesc
5	project1
5	project3
.....	

PROJ2 = PROJ ⋉ EMP2

#	ProjectDesc
7	project2
17`	project4
.....	

#	Name	Loc
5	James	S1
7	John	S2
8	Judy	S1
17	Joyce	S2
......		

#	Sal
5	10K
7	20K
8	15K
17	22K
......	

Distributed Database Design. Figure 1. Fragmentation schemes.

In *primary horizontal* fragmentation, each fragment is essentially a subset of the tuples in the original relation (see Fig. 1a). In general, a fragment is defined as a selection on the original relation, i.e., $R_i = \sigma_{C_i}(R)$ where C_i is a predicate used to construct R_i. A good primary horizontal fragmentation of a relation typically has three desirable properties:

1. *Completeness:* $\forall t \in R$, $\exists R_i \in F$ such that $t \in R_i$. This property states that every tuple in the original relation must be assigned to a fragment.
2. *Disjointness:* $\forall t \in R_i$, $\nexists R_j$ such that $t \in R_j$, $i \neq j$, R_i, $R_j \in F$. This property states that all fragments are disjoint, i.e., a tuple is assigned to only one fragment. This property is useful as it leaves the decision to replicate fragments to the allocation phase.
3. *Reconstruction:* $R = \bigcup_{i=1}^{n} R_i$. This property states that the original relation can be reconstructed by simply union-ing all its fragments.

In *derived horizontal fragmentation*, a relation S is fragmented based on the fragments of another relation R. R is called the owner relation and S the member relation. Each fragment of S is obtained using a semi-join between S and a corresponding fragment of R, i.e., $S_i = S \ltimes_{S.k=R_i.k} R_i$ where k is the join attribute (see Fig. 1b). Derived fragmentation is useful when the fragments of R need to be combined with the records of S with matching join keys. As such, the corresponding derived fragment is the necessary subset, and can be co-located with the R fragment. In order to ensure completeness, referential integrity constraint has to be enforced, i.e., the set of distinct join attribute values of the member relation must be a subset of the set of distinct join attribute values of the owner relation. Moreover, for disjointness property to be achievable, the join attribute should be the key of the owner relation.

In *vertical fragmentation*, a relation $R[T]$ (where T is the set of attributes in R's schema) is decomposed into, say k fragments $R_1[T_1]$, $R_2[T_2]$,...$R_k[T_k]$ (where T_i is the set of attributes in fragment R_i) (see Fig. 1c). Each fragment R_i is defined as the projection of the ordinal relation on the set of attributes T_i, i.e., $R_i = \pi_{T_i}(R)$. For completeness, $T = \cup_{i=1}^{k} T_i$. In addition, to reconstruct R from its fragments, the lossless join property must be enforced, i.e., $R = \bowtie_{i=1}^{k} R_i$. One way to achieve the lossless join property is to repeat the key attributes in all the fragments, i.e., $\forall i$, $key \subseteq T_i$.

In practice, a combination of all the above fragmentation schemes can be applied on a single table. For example, the EMP table can be first horizontally partitioned based on the location, and then the fragment at location S1 (EMP1) may be further vertically partitioned into $P1 = \pi_{\#,\ Name,\ Loc} EMP1$ and $P2 = \pi_{\#,\ Sal} EMP1$. Now a query over the original relation EMP can then be evaluated as a query over the corresponding fragments (cross-reference Distributed Query Processing). For example, a query to find the names of employees in location S1, expressed as $\pi_{Name}\sigma_{Loc=S1}EMP$, can be reduced to a query over one fragment: $\pi_{Name}P1$. As can be seen, fragmentation can lead to more efficient query processing.

To generate the set of fragments, the following strategy can be adopted. Given a set of simple predicates $P = \{P_1, P_2, ..., P_m\}$ (each P_i is of the form `attribute` θ `value` where $\theta \in \{<, \leq, =, >, \geq\}$), a set of minterm predicates M is generated. M is defined as follows:

$$M = \{m | m = \wedge_{P_k \in P} {P_k}^*, 1 \leq k \leq m\}$$

where P_k^* is P_k or $\neg P_k$. After eliminating useless minterm prediates, the resultant minterm predicates can be used to produce the set of disjoint fragments: $\sigma_m(R)$ for all $m \in M$. As an example, suppose $P = \{A < 10, A > 5, Loc = S_1, Loc = S_2\}$ is the set of simple predicates. Moreover, assume that there are only 2 locations, S_1 and S_2. Then, there will be a total of 16 minterm predicates; several of these are empty set, e.g., $\{A < 10 \wedge A > 5 \wedge Loc = S_1 \wedge Loc = S_2\}$ and $\{A < 10 \wedge \neg A > 5 \wedge Loc = S_1 \wedge Loc = S_2\}$. The resultant set of minterm predicates consists of 6 predicates: $\{5 < A < 10 \wedge Loc = S_1\}$, $\{5 < A < 10 \wedge Loc = S_2\}$, $\{A \leq 5 \wedge Loc = S_1\}$, $\{A \leq 5 \wedge Loc = S_2\}$, $\{A \geq 10 \wedge Loc = S_1\}$, $\{A \geq 10 \wedge Loc = S_2\}$. Each of these predicates will result in a fragment.

Allocation

Once a database has been fragmented, the fragments are allocated to the sites. This gives rise to the allocation problem: Given a set of fragments $F = \{F_1, F_2, ..., F_n\}$ and a number of sites $S = \{S_1, S_2, ..., S_m\}$ on which a number of applications $Q = \{Q_1, Q_2, ..., Q_p\}$ is running, allocate $F_i \in F$ to $S_j \in S$ such that some optimization criterion is met (subject to certain constraints, e.g., available storage at certain sites). Some optimization criteria include maximizing throughput, minimizing the average response time or minimizing the total cost of serving *Q*. This (allocation) problem is complex because the optimal solution depends on many factors, for example, the location in which a query originates, the query processing strategies (e.g., join methods) that are used, the hardware at the various sites and so on. As such, for the problem to be tractable, the problem is typically simplified with certain assumptions, e.g., only communication cost is considered.

As an example, consider the following simple cost models which determine the read, write and storage cost of an arbitrary fragment *f*. (Note that a more complex model will need to consider other factors like fragment size, queries involving multiple fragments, and so on.). The read cost of *f* is given by:

$$\sum_{i=1}^{m} t_i \times MIN_{j=1}^{m} C_{ij}$$

where i is the originating site of the request, t_i is the read traffic at S_i and C_{ij} is the cost to access fragment f (stored) at S_j from S_i. If only the transmission cost is considered, then $C_{ii} = 0$ if f is stored at site S_i, and $C_{ij} = \infty$ if f is not stored at site S_j. The update cost is given by:

$$\sum_{i=1}^{m}\sum_{j=1}^{m} X_j \cdot u_i \cdot C_{ij}$$

where i is the originating site of the request, j is the site being updated, $X_j = 1$ if f is stored at S_j and 0 otherwise, u_i is the write traffic at S_i and C_{ij} is the cost to update f at S_j from S_i. Finally, the storage cost is given by:

$$\sum_{i=1}^{m} X_i \cdot d_i$$

where $X_i = 1$ if f is stored at S_j and 0 otherwise, and d_i is the storage cost at S_i.

Fragments can be allocated to minimize a combination of the above costs. Based on these cost models, any optimization algorithm can be easily adapted to solve it. For example, a randomized algorithm (such as simulated annealing, iterative improvement) can allocate fragments to sites and the allocation that gives the best cost is the winner.

Heuristics have also been developed. For example, a best-fit heuristic for non-replicated allocation works as follows: For each fragment *f*, place it at the site *j* where the total cost (for this fragment only) is minimum. While this scheme is computationally efficient, it ignores the effect of other fragments which may render the allocation sub-optimal.

Future Directions

Even though the distributed database design has been extensively studied, similar data partitioning and placement (allocation) issues continue to be surfaced for new architectural design. The focus here will be more on load-balancing and dynamic data migration. For example, in a highly dynamic and loosely connected distributed systems like a peer-to-peer system, there is yet no effective solution to dynamically adapt the placement of data for optimal performance. Similarly, in a distributed publish/subscribe environment, the problem of migrating data (subscriptions) for optimal performance has not been adequately explored. Moreover, in these systems, the logical network connection between sites may be dynamically changing, and this will influence the allocation of data.

Cross-references

► Data Replication
► Parallel Database Management
► Peer Data Management System

Recommended Reading

1. Apers P.M. Data Allocation in distributed database systems. ACM TODS, 13(2):263–304, 1988.
2. Bell D.A. Difficult data placement problems. Comput. J., 27(4):315–320, 1984.
3. Brunstrom A., Leutenegger S.T., and Simha R. Experimental evaluation of dynamic data allocation strategies in a distributed database with changing workloads. In Proc. Int. Conf. on Information and Knowledge Management, 1995, pp. 395–402.
4. Ceri S., Negri M., and Pelagatti G. Horizontal data partitioning in database design. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1982, pp. 128–136.
5. Ceri S. and Pelagatti G. Distributed Databases: Principles and Systems. McGraw-Hill, NY, USA, 1984. ISBN 0-07-010829-3.
6. Chang C.C. and Shieh J.C. On the complexity of file allocation problem. In Proc. Int. Conf. on the Foundations of Data Organization, 1985, pp. 177–181.
7. Chu W.W. Optimal file allocation in a multiple computer network. IEEE Trans. Comput., C-18(10):885–889, 1969.
8. Karlapalem K. and Ng M.P. Query-driven data allocation algorithms for distributed database systems. In Proc. 8th Int. Conf. Database and Expert Syst. Appl., 1997, pp. 347–356.
9. McCormick W.T., Schweitzer P.J., and White T.W. Problem decomposition and data reorganization by a clustering technique. Oper. Res., 20(5):993–1009, 1972.

10. Muri S., Ibaraki T., Miyajima H., and Hasegawa T. Evaluation of file redundancy in distributed database systems. IEEE Trans. Software Eng., 11(2):199–205, 1995.
11. Navathe S., Ceri S., Wiederhold G., and Dou J. Vertical partitioning of algorithms for database design. ACM TODS, 9(4):680–710, 1984.
12. Ozsu M.T. and Valduriez P. Principles of Distributed Database Systems, 2nd edn. Prentice-Hall, ISBN 0-13-659707-6, 1999.
13. Sacca D. and Wiederhold G. Database partitioning in a cluster of processors. ACM TODS, 10(1):29–56, 1985.
14. Yoshida M., Mizumachi K., Wakino A., Oyake I., and Matsushita Y. Time and cost evaluation schemes of multiple copies of data in distributed database systems. IEEE Trans. Software Eng., 11(9):954–958, 1985.

Distributed Database Management System (DDBMS)

► Distributed DBMS

Distributed Database Systems

Kian-Lee Tan
National University of Singapore, Singapore, Singapore

Synonyms

Homogeneous distributed database systems; Heterogeneous distributed database systems; Federated database systems; Multidatabases

Definition

A distributed database (DDB) is an integrated collection of databases that is physically distributed across sites in a computer network. A distributed database management system (DDBMS) is the software system that manages a distributed database such that the distribution aspects are transparent to the users. To form a distributed database system (DDBS), the files must be structured, logically interrelated, and physically distributed across multiple sites. In addition, there must be a common interface to access the distributed data.

Historical Background

There are many reasons that motivated DDBS. First, distributed databases reflect organizational structure. In many organizations, the data are naturally distributed across departments/branches where each department/branch maintains its own local data. Moreover, it is not always possible to build a centralized system to consolidate these data. In addition, by keeping these data at their respective remote sites facilitates autonomy where each site retains control over the data that it generates/possesses.

Next, a DDBS is expected to offer better performance – data are placed at locations where they are frequently accessed, and hence communication overhead can be minimized; moreover, parallelism can be exploited to process a query in parallel. Had data been stored in a centralized site, the centralized site may become a bottleneck, and the communication overhead may be significant.

A DDBS also offers better availability – when a site fails, the other operational sites can potentially still be available for data retrieval and query processing. A centralized site is vulnerable to single point of failure.

The concepts behind distributed DBMS were pioneered during the late 1970s through several research projects including SDD-1 [8] developed by Computer Corporation of America, Distributed INGRES [11] started at the University of California at Berkeley, and R*STAR [9] designed at IBM research lab.

The first well-publicized distributed DBMS product was INGRES/Star, announced in 1987. Oracle also announced its distributed DBMS capabilities in 1987, and the first Oracle product to reasonably support distributed database processing is Oracle 7. IBM's distributed DBMS products, based on the distributed relational data architecture, are largely systems for integrating data sets across the different versions of DB2 that run on AIX, OS/2, OS/400, VM and MVS.

More recent trends have focused on multi-databases (heterogeneous databases) and distributed systems that offer more autonomy to individual system [10,12].

Foundations

In a distributed database system, data are distributed across a number of sites [1,6]. A relation can be horizontally or vertically fragmented (cross-reference Distributed Database Design), and/or replicated. These fragments/replicas are allocated to the sites to be stored there. In a multi-database, where multiple databases are to be integrated, one can view the local database as a fragment of the integrated database.

One key consideration in the design of a DDBS is the notion of data transparency. With data transparency, the user accesses the database thinking that (s)he is working with one logical centralized database. There are several forms of transparency: (i) distribution transparency; (ii) replication transparency; (iii) Location transparency; and (d) Transaction transparency. In distribution transparency, the user is not aware of how a relation has been fragmented. In replication transparency, the user sees only one logically unique piece of data. (S)he is masked from the fact that some fragments may be replicated and that replicas reside at different locations.

In location transparency, the user is masked from the use of the location information when issuing the query. In other words, there is no need for the user to specify the location of data that (s)he is retrieving. This follows from the distribution/replication transparencies. In transaction transparency, the user is masked from coordination activities required to achieve consistency when processing queries. In fact, each transaction maintains database consistency and integrity across the multiple databases/replicas (without user knowledge). A global transaction that accesses data from multiple sites has to be divided into subtransactions that process data at one single site.

A DDBMS comprises a number of components [2]: (i) The database management component is a centralized DBMS that manages the local database; (ii) The data communication component handles the communication between sites in a DDBS; (iii) The data dictionary which is extended to represent information about the distribution of data (and sites) in the DDBS; (iv) the distributed database component (DDB) that manages the DDBS to ensure that the system functions correctly.

In particular, the DDB performs a number of tasks. First, in processing a global transaction, the DDB needs to determine a distributed query plan that is most cost-effective. Distributed query processing algorithms including semijoins are often used to reduce the communication overhead.

Second, there is a need to synchronize the accesses from multiple users to the distributed databases in order to maintain the consistency and integrity of the database. Unlike centralized database systems, the database consistency has to be guaranteed at each site as well as across all sites.

Third, as a global transaction may require accessing data from multiple sites, the competition for these data can result in deadlocks (for locking-based synchronization mechanisms, which is the most commonly used). In a DDBS, deadlocks may not occur within a site, but occur across sites. As such, having a “global” view is critical in order to detect and to recover from deadlocks.

Fourth, atomicity of global transactions is an important issue. Clearly, if a subtransaction running at one site commits while another substransaction running at another site aborts, then the database will become inconsistent. As such, distributed transaction commit protocols must ensure that such a situation cannot arise. The two-phase commit protocol is the most widely used.

Finally, as in any distributed systems, sites may fail or become inaccessible as a result of network problems. As a result, it is necessary to ensure the atomicity of global transactions in the midst of failures. Moreover, mechanisms must be provided to ensure the consistency of the database, and to bring the recovered failed nodes up-to-date.

While DDBS offers a host of promises and advantages, it is very difficult to realize a full-fledge DDBS. For example, it has been argued that full transparency makes the management of distributed data very difficult so much so that it can lead to poor manageability, poor modularity and poor message performance [3]. As another example, it is clearly more complex to design the DDB component and the respective mechanisms and protocols for the DDBS to operate as promise and to ensure consistency and reliability. Being connected to a network also means that it is more vulnerable to security threats. Nevertheless, it is an exciting field that offers many challenges for researchers to work on.

Key Applications

Many of today’s applications are naturally distributed. For example, in supply-chain management, there is a need to access information concerning parts, products, suppliers which are housed by different organizations. As another example, in financial applications, a bank typically has many branches locally or overseas, and each such branch maintains its own databases to better serve their local customers; there is, however, a need to have an integrated view of the bank as well. Other applications include airline reservation, government agencies, health care, and so on.

Future Directions

Distributed databases continue to be an interesting area of research. In particular, the internet has made large scale distributed systems possible and practical. There are still many problems that have not been adequately addressed. For example, it remains a challenge to find the optimal query optimization plan especially when site autonomy are to be enforced. Some recent works have examined these using microeconomics principles [7]. Another direction is the semantic interoperability problem [4]. Yet another direction is in the design of advanced distributed data management systems, e.g., peer-based data management [5] (where peers are highly dynamic and may join and leave the network anytime), mobile data management (where nodes are mobile), and sensornet databases (where sensor nodes are battery-powered, has limited storage and processing capabilities).

Cross-references

► Data Replication
► Distributed Architecture
► Distributed Database Design
► Distributed Query Optimization
► Distributed Query Processing
► Parallel Database Management
► Peer Data management Systems

Recommended Reading

1. Bell D. and Grimson J. Distributed Database Systems. Addison-Wesley, 1992.
2. Ceri S. and Pelagatti G. Distributed Databases: Principles and Systems. McGraw-Hill, 1984.
3. Gray J. Transparency in Its Place – The Case Against Transparent Access to Geographically Distributed Data. Technical Report TR89.1, Cupertino, Calif.: Tandem Computers Inc., 1989.
4. Halevy A.Y., Ives Z.G., Suciu D., and Tatarinov I. Schema mediation for large-scale semantic data sharing. VLDB J., 14(1):68–83 2005.
5. Ng W.S., Ooi B.C., Tan K.L., and Zhou A. PeerDB: A P2P-based System for Distributed Data Sharing. Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 633–644.
6. Ozsu M.T. and Valduriez P. Principles of Distributed Database Systems, 2nd edn. Prentice-Hall, 1999.
7. Pentaris F. and Ioannidis Y.E. Query optimization in distributed networks of autonomous database systems. ACM Trans. Database Syst., 31(2):537–583, 2006.
8. Rothnie Jr., Bernstein P.A., Fox S., Goodman N., Hammer M., Landers T.A., Reeve C.L., Shipman D.W., and Wong E. Introduction to a system for distributed databases (SDD-1). ACM Trans. Database Syst., 5(1):1–17, 1980.
9. Selinger P.G. An architectural overview of R*: a distributed database management system. Berkeley Workshop, 187, 1981.
10. Sheth A.P. and Larson J.A. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv., 22(3):183–236., 1990.
11. Stonebraker M. The Design and Implementation of Distributed INGRES. The INGRES Papers, 1986, pp. 187–196.
12. Stonebraker M., Aoki P.M., Pfeffer A., Sah A., Sidell J., Staelin C., and Yu A. Mariposa: a wide-area distributed database system. VLDB J., 5(1):48–63, 1996.

Distributed Databases

► Storage Grid

Distributed DBMS

Sameh Elnikety
Microsoft Research, Cambridge, UK

Synonyms

Distributed Database Management System (DDBMS)

Definition

A distributed DBMS is a software system that manages a distributed database, which consists of data that are partitioned and replicated among interconnected server sites. The primary objective of a distributed DBMS is to hide data distribution so that it appears as one logical database system to the clients.

Historical Background

Distributed DBMS started in the late 1970s [2,10,12] with shared-nothing parallel database systems [13], which were designed for achieving higher performance by exploiting parallelism in transaction workloads. Work on distributed DBMS was mainly motivated by the need to manage data for large organizations having different offices and subsidiaries but slow computer networks hampered the adoption of DDBMS [14]. In the 1990s, advances in computer networking coupled with the growing business needs to manage distributed data fueled the work on distributed database systems.

Foundations

A distributed DBMS (DDBMS) manages a distributed database that is accessed at multiple sites, where each site contains a partition of the database. A single partition can be replicated across multiple sites. At one end of the spectrum, in a fully-replicated database, each site has a full copy of the database. At the other end of the spectrum, a fully-partitioned database is divided into disjoint partitions, also called fragments, and each is placed at only one site. Hence, a DDBMS typically maintains a data directory which maps each data item to the sites at which the data item is maintained. This design raises three key challenges for DDBMSs. The first challenge is the distribution and placement of the data. Second, the distributed DBMS must update all copies of modified data items when executing update transactions. The third challenge arises when a query needs to access data items at multiple sites, requiring coordination and communication among the sites. Despite these challenges, there is a potential gain in performance due to the proximity of data to applications and due to inter-transaction and intra-transaction parallelism in the workload.

Data Placement among Sites

As far as data placement is concerned, there is a performance tradeoff. Applications running at a site prefer to access all needed data items locally. At the same time, this naturally leads to an increasing number of data replicas, and updates become more expensive as they need to reach more replicas. This tradeoff makes deciding how the database is distributed one of the main design decisions. The database designer typically determines which relations are allocated to which sites. Data placement and distribution depend primarily on the applications that access the database as well as on the server network. More fine-grained data distribution as used in parallel database systems [3,5] is also possible. For example, a single relation can be fragmented horizontally (e.g., using a selection operator) or vertically (e.g., using a projection operator) into several sub-relations and each sub-relation is allocated to at least one site.

Data placement can also be performed dynamically [6]. Most approaches to dynamic data placement are adaptive. They keep statistics on the workload and move or copy data at different sites so as to adjust the data placement to the current workload. For example, a new copy of a set of data items could be established to help balance the load among servers, or to reduce wide-area network communication costs. When a new copy of a data item is made, the copy is designated as either a replica or a cache. A replica of a data item is long-lived and maintained by reflecting the item's modification to the copy. On the other hand, a cache is typically short-lived and invalidated on changes to the data item. Due to the complexity of the dynamic data placement problem, some research work targets using economic models [6] to optimize dynamic data placement. One can no longer discuss caching and assume that the DDBMS maintains all data item copies.

Propagating the Effects of Update Transactions

When an update transaction modifies the value of a database item, the DDBMS is responsible for reflecting this change to all copies of the data item; otherwise the copies become inconsistent and consequently the database servers may diverge, violating system correctness properties. There is a large body of work on handling updates, mainly inspired by the work in distributed systems (e.g., quorum protocols). However, few ideas have impacted commercial DDBMS products as they mainly use ROWAA (Read One Write All Available) protocols. In a ROWAA protocol, the update of a data item has to reach all copies of that data item. It is, however, sufficient to access only a single copy to read any data item. Here the discussion focuses on a key correctness property for ROWAA protocols: one-copy semantics.

When a distributed system offers one-copy semantics, clients cannot tell whether they are communicating with a single DBMS or a distributed DBMS. Without one-copy semantics, applications have to deal with inconsistencies while accessing the database. In practice, some applications require one-copy semantics (e.g., airline reservation, banking, and e-commerce systems). However, a large class of applications can tolerate inconsistencies (e.g., reporting applications).

The process of reflecting the changes of modified data items is called update propagation, and there are several protocols that implement it. For example in eager protocols, all live copies of each modified data item are changed as part of the update transaction. In contrast, lazy protocols propagate the changes after the commit of the update transaction.

One-copy semantics can be implemented using eager or lazy update propagation protocols, but it requires a total order on the propagated updates. This total order is typically the commit order of update transactions and each site applies its relevant updates in an order consistent with the total order.

Distributed Query Execution

When a distributed DBMS receives a query, it generates a query execution plan that may access data at multiple sites. Each site executes part of the plan and may need to communicate with other sites. Communication among sites is expensive and good plans generally minimize the amount of data exchanged between sites [8]. In wide-area networks, the cost of communication is generally higher than the cost of local processing. In contrast, the cost of communication is equal to or less than the cost of local processing in local-area networks. Optimizing distributed queries is challenging [6] because the search space is large. It includes the selection of data fragments to be accessed, the order of operations in the plan, and the cost of communication among the sites. The problem becomes harder when optimizing multiple queries at a time.

A DDBMS provides global transaction isolation (e.g., one-copy serializability [9] or generalized snapshot isolation [4]) through distributed concurrency control. To provide global isolation for transactions, local concurrency control protocols alone are insufficient. For example, if each site individually ensures local serializability, the global transaction execution may violate global serializability since two sites could locally serialize conflicting global transactions in different orders. Protocols that implement distributed concurrency control (e.g., distributed two-phase locking or multi-version timestamp ordering) require coordination among the sites.

For example in locking-based distributed concurrency control protocols, the primary-site [1] method uses a central lock controller to manage the locks. Alternatively, in the primary-copy [15] method each data item has one copy that is designated as the primary copy, and only the primary copy is locked. This design allows locks to be distributed with the data among several sites.

Coordination among several sites can become a significant source of overhead. Deadlocks and aborts become more frequent in a distributed DBMS compared to centralized DBMS because distributed transactions require communication among multiple sites and therefore take longer to execute.

Handling Failures

Fault-tolerance is an important aspect in any distributed system. A DDBMS contains multiple sites and each site can fail independently. The DDBMS, therefore, needs to cope with and recover from site failures. The DDBMS always guarantees the safety properties – including atomicity and durability of distributed update transactions – despite site failures [9,12]. To terminate an update transaction while ensuring the safety properties, the DDBMS employs a distributed commit protocol [1] (e.g., distributed two-phase commit, and three-phase commit) that are specifically designed to handle site failures. Liveness properties, which is concerned with the ability to process transactions, depend on which and how many sites are still connected and operating normally. Liveness properties encompass system performance and availability and usually require high degree of DDBMS customization to meet the desired targets. For example instead of using primitive techniques to recover from a crash and bring a site's database up-to-date, data streaming from multiple sources at several sites substantially reduces database recovery time and therefore improves the DDBMS availability.

Key Applications

Today, distributed DBMSs are used to manage distributed databases, such as in geographically distributed systems (e.g., hotel chains and multi-plant manufacturing systems), and in databases under several administrative or autonomous domains. Distributed DBMSs are also used to achieve fault-tolerance and scalability when a centralized DBMS is not satisfactory, for instance in financial transaction processing and airline reservation systems.

In modern data centers that host web services, DDBMS technology appears in two forms. First in fully replicated database systems, which consist of a cluster of servers interconnected through a fast local-area network. The objective of such replicated databases is to achieve both higher availability since data is available at several server nodes as well as higher performance as transactions can be processed in parallel at the database servers.

The second form is partitioned database systems in which relations are striped (fragmented) among several servers. A single query can be divided into smaller subqueries that execute on the servers, leading to a shorter response time.

Future Directions

Currently, DDMBS technology is rarely used in large scale information systems without extensive customization necessary to obtain adequate performance. At the same time, networking and storage technologies are advancing at a pace much higher than the improvement in microprocessors clock speeds. These trends suggest that DDBMS use will increase in the future. But further research is needed to ease the deployment and management of DDBMS.

Cross-references

► DBMS
► Distributed Concurrency Control
► Distributed Database
► Distributed Database Design
► Parallel Database Management
► Replication for High Availability
► Replication for Scalability

Recommended Reading

1. Bernstein P. and Goodman N. Concurrency control in distributed database systems. ACM Comput. Surv., 13 (2):185–221, 1981.
2. Bernstein P., Shipman D., and Rothnie J. Concurrency control in a system for distributed databases (SDD-1). ACM Trans. Database Syst., 5(1):18–51, 1980.
3. DeWitt D. and Gray J. Parallel database systems: the future of high performance database systems. Commun. ACM., 35(6):85–98, 1992.
4. Elnikety S., Pedone F., and Zwaenepoel W. Database replication using generalized snapshot isolation. In Proc. 24th Symp. on Reliable Distributed Syst., 2005.
5. Ghandeharizadeh S., Gao S., Gahagan C., and Krauss R. High performance parallel database management systems. In Handbook on Data Management in Information Systems, J. Blazewicz, W. Kubiak, T. Morzy, M. Rusinkiewicz (eds.). Springer, 2003, pp. 194–220.
6. Kossmann D. The state of the art in distributed query processing. ACM Comput. Surv., 32(4):422–469, 2000.
7. Muffin S.M. A Distributed Database Machine. ERL Technical Report UCB/ERL M79/28, University of California at Berkeley, CA, 1979.
8. Özsu T. and Valduriez P. Principles of Distributed Database Systems. Prentice-Hall, Englewood Cliffs, NJ, 1991.
9. Pacitti E., Coulon C., Valduriez P., and Özsu T. Preventive replication in a database cluster. Distrib. Parallel Databases., 18(3):223–251, 2005.
10. Papadimitriou C. The theory of database concurrency control. CS Press, AB, 1988.
11. Ries D. and Epstein R. Evaluation of Distribution Criteria for Distributed Database Systems. UCB/ERL Technical Report M78/22, UC Berkeley, CA, 1978.
12. Skeen D. and Stonebraker M. A formal model of crash recovery in a distributed system. IEEE Transactions on Software Engineering 9(3):219–228, 1983.
13. Stonebraker M. The case for shared nothing. IEEE Database Eng. Bull., 9: 4–9, 1986.
14. Stonebraker M. Readings in Database Systems (2nd ed.). Morgan Kaufmann Publishers, Scan Mateo, CA, 1994.
15. Zaslavsky A., Faiz M., Srinivasan B., Rasheed A., and Lai S. Primary copy method and its modifications for database replication in distributed mobile computing environment. In Proc. 15th Symp. on Reliable Distributed Syst., 1996.

Distributed Deadlock Management

WEE HYONG TOK
National University of Singapore, Singapore, Singapore

Synonyms

Deadlocks in distributed database systems

Definition

In a database that supports locking protocol, accesses to data are controlled using locks. Whenever a transaction needs to access a shared object, it will be granted a lock (and hence access) to the object if there is no other conflicting locks on the object; otherwise, the requesting transaction has to wait. A deadlock occurs when transactions accessing shared data objects are waiting *indefinitely* in a circular fashion until a special action (such as aborting one of the transactions) is taken. In a distributed database environment, deadlocks can occur locally at a single site, or across sites where a chain of transactions may be waiting for one another to release the locks over a set of shared objects.

For example, consider two data objects o_1 and o_2 stored at site 1 and site 2 respectively. Suppose two transactions, T_1 and T_2, initiated at site 1 and site 2, are updating o_1 and o_2 concurrently. As T_1 is updating o_1 at site 1, it holds a write lock on o_1. Similarly, T_2 holds a write lock on o_2 at site 2. When T_1 attempts to access o_2, it has to wait for T_2 to release the lock on o_2. Likewise, T_2 has to wait for T_1 to release the lock on o_1. This leads to a deadlock as both transactions cannot proceed.

Three techniques are commonly used to handle deadlocks: prevention, avoidance, and detection and resolution. Deadlock prevention methods ensure that deadlocks will not occur in the database system. This is achieved by preventing transactions which will cause deadlocks from being executed. Deadlock avoidance schemes preemptively detect potential deadlocks during the scheduling of multiple transactions. Deadlock detection protocols detect deadlocks in running transactions by using a transaction wait-for graph. Whenever deadlocks are detected, a resolution strategy aborts some of the transactions in order to break the deadlock.

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:29:59

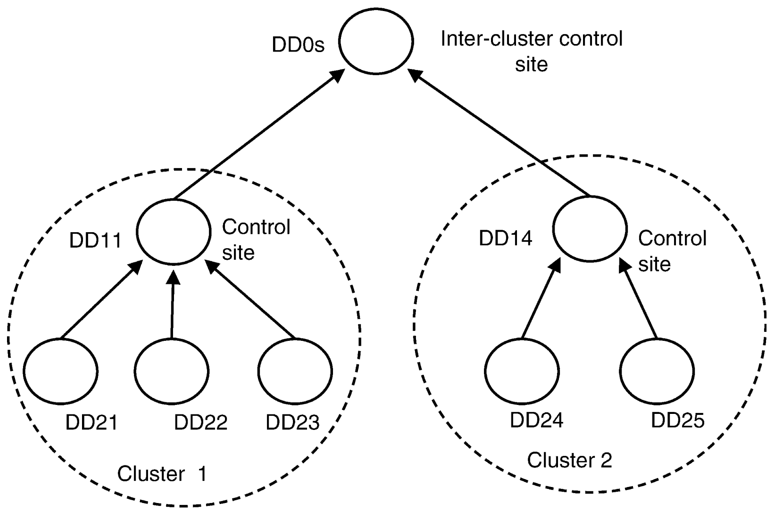
Historical Background

When transactions access shared objects, there is a dependency relationship between the transactions. The dependency relationship captures how the transactions wait for one another, and is commonly represented as a directed graph, called the wait-for graph (WFG). Each node represents a transaction. An edge (i.e. arc) in the graph is used to capture the wait-for relationship. For example, if a directed edge is found between node A and node B, then the transaction represented by node A is waiting for another transaction which is represented by node B. In distributed database systems, the WFG is local if it involves only the data at a single site, and is global if it involves data from multiple sites. When a cycle exists in a WFG, a deadlock occurs.

Deadlock detection algorithms for distributed database systems can be categorized as: centralized, hierarchical, and distributed [12].

Centralized deadlock detection algorithms [5,6] use a central site for detecting deadlocks in a distributed database system. This site is referred to as the central deadlock detection site, *C*. *C* is responsible for building the global WFG. Other sites transmit their local WFG to *C*. Subsequently, only the changes to the local WFG are transmitted to *C*. These changes include new or deleted edges in the local WFG. *C* continuously checks for cycles in the global WFG and performs deadlock resolution whenever cycles are detected. The implementation of centralized deadlock detection is simple and straightforward. However, due to the need to continuously transmit new changes in the local TWG to *C*, it can cause a high communication overhead. In addition, the use of a single site makes it susceptible to overloading and being a single point of failure. Özsu and Valduriez [12] noted that centralized two-phase locking (2PL) and deadlock detection is a good, natural combination. Centralized deadlock detection is implemented in Distributed INGRES [15].

Hierarchical deadlock detection algorithms [10,6] rely on a hierarchical organization of the sites in a distributed database system to detect deadlocks. Each internal node (site) merges the WFG from its child nodes into a WFG that captures the dependency relationship among the descendant nodes. It can thus detect deadlocks in its descendant nodes. It also transmits the combined WFG to its parent node. The key idea in hierarchical deadlock detection is to ensure that a deadlock can be detected as early as possible by a nearby site. This reduces the need to escalate the detection to the root site. As deadlock detection is spread amongst several sites, hierarchical deadlock detection algorithms incur less communication overheads compared to centralized algorithms. The implementation of hierarchical deadlock detection algorithms is more complex due to the need to coordinate between the multiple deadlock detectors. Figure 1 shows an example of how five sites in a distributed database system is organized in a hierarchy for deadlock detection. Some sites might function as the deadlock detector for multiple levels of the hierarchy. Each deadlock detector is denoted as DD_{ls}, where *l* and *s* denote the level and

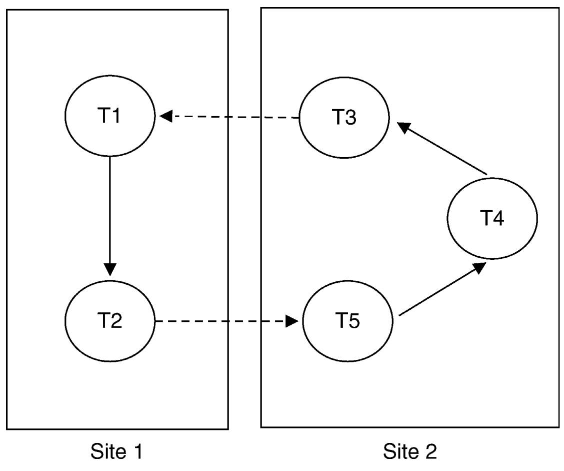


Distributed Deadlock Management. Figure 1. Hierarchical deadlock detection.

the site of the deadlock detector respectively. From the figure, observe that in cluster 1, site 1 is the control site, and detects deadlocks for sites 1, 2, and 3. Thus, sites 1, 2, and 3 will need to update the deadlock detector at the next level (i.e. DD$_{11}$) with their respective local WFG. If the deadlock needs to be detected between site 1 and site 5, then it will be detected by the deadlock detector at the root of the tree (i.e. DD$_{0s}$, where $1 \leq s \leq 5$).

Distributed deadlock detection algorithms [10,11] rely on the cooperation of all the sites in the distributed database system in order to detect deadlocks. Each site in the distributed database system consists of a deadlock detector. Potential deadlock information is transmitted from one site to another. Consequently, the deadlock detector modifies its local WFG using information about the potential deadlocks, as well as whether a local transaction is waiting for transactions at another sites. Distributed deadlock detection is implemented in System R* [11].

In distributed database systems, effective global deadlock detection rely on the the timely propagation of local information from all the sites. However, some sites might be slower in propagating local information. As a result, this might lead to the detection of phantom deadlocks. A phantom deadlock is a deadlock which does not exist. Hence, in order to break the phantom deadlocks, transactions might be aborted. Both the centralized and hierarchical algorithms are prone to the phantom deadlock problem. For example, in Fig. 2, a global deadlock exists. However, at site 2, the transaction T_5 might be aborted. T_5 can aborted due to the business logic encoded in the transaction. This changes the local WFG for site 2. Consequently, no more cycle exists in the global WFG. However, if the changes in the local WFG are not propagated in a timely manner to update the global WFG, a phantom deadlock arises.



Distributed Deadlock Management. Figure 2. Global wait-for graph.

The book [12] provides a good overview of distributed deadlock management techniques. The surveys [4,7,14,1] provide a good discussion of various distributed deadlock detection algorithms. Krivokapić et al. [8] further categorizes deadlock detection approaches as path-pushing [10], probe-based [13] or having a global state strategy [2,3]. Using the categorization, Krivokapić et al. [8] presented a detailed performance analysis of representative algorithms for each of the categories.

Foundations

A distributed database system consists of a collection of database sites. A centralized database system is located at each of the sites. The database sites communicate with each other by sending messages via a communication network. A transaction consists of a sequence of operations (e.g. read, write) that are performed on the data objects. Whenever a transaction needs to perform an operation on a data object, it sends a resource request to a transaction manager (TM). The resource request can refer to operations that are performed on local or remote data objects.

Transaction Wait-for Graph

Given N active transactions, $T_1 \ldots T_N$, in a database system, a directed graph, called a transaction Wait-for Graph (WFG), can be built. Each vertex of the graph corresponds to an active transaction. A wait-for relationship exists between two transactions, T_i and T_j $(i \neq j)$, if T_i is waiting for T_j to release a lock. This is denoted as a directed edge between two vertices, V_i and V_j $(i \neq j)$ in the graph. In most deadlock detection algorithms, the WFG is used for analyzing deadlocks. Deadlocks occur when cycles are detected in the graph.

In order to analyze deadlocks in a distributed database system, a global transaction wait-for-graph (WFG) is commonly used. The global WFG is constructed by taking the union of the local WFGs for all the sites in the distributed database system. The main difference between the local and global WFG is that the global WFG captures inter-site wait-for relationship. Inter-site waiting occurs when a transaction, executing on one site, is waiting for the release of a lock on a data object by another transaction executing on another site. For example, in a distributed database system, five transactions, T_1 to T_5 are active on two sites. Figure 2 shows the global WFG, which captures the wait-for relationship between the transactions. The dashed lines are used to denote inter-site waiting. From the figure, several observations can be made. Firstly, one can observe that at site 1, transaction T_1 is waiting for T_2. At site 2, T_5 is waiting for T_4, and T_4 is waiting for T_3. In addition, the transaction T_2 (site 1) is waiting for T_5 (site 2), and T_3 (site 2) is waiting for T_1 (site 1). As a cycle is detected in the global WFG, a deadlock is detected.

Deadlock Models

Different types of models of deadlock are presented in [7]. The models are used to capture the type of resource requests that need to be processed by application programs. The models include: one-resource, *AND*, *OR*, *AND-OR*, (nk), and the unrestricted model. In the one-resource model, a transaction can have at most one resource request. Thus, the maximum outdegree for a WFG vertex is 1. In both the *AND* and *OR* models, a transaction requests for access to a set of shared data objects (i.e. resources). The main difference between the two models is that in the *AND* model, the transaction blocks until all the requested resources are available, whereas in the *OR* model, a transaction blocks until any one of the resources is available. The *AND-OR* deadlock model is a generalization of the *AND* and *OR* models. In order to further generalize the *AND-OR* model, the $\binom{n}{k}$ model is used. In this model, a transaction can request for any k available resources from a set of n available resources. The unrestricted model does not impose any constraints on the number of resources that are requested by a transaction. In the one resource and *AND* deadlock model, a deadlock occurs whenever there is a cycle in the WFG. The detection of deadlocks in the other models require more complex computation, and is discussed in details in [7].

Static Vs Dynamic Deadlock Detection

Deadlock detection can be classified as static or dynamic. In static deadlock detection, the overall strategy in which deadlocks is detected is fixed. For example, in centralized deadlock detection, the central site is pre-determined. Similarly, in hierarchical deadlock detection, the hierarchical organization of the sites is also pre-determined. In distributed deadlock detection, all sites have individual deadlock detection mechanisms. In dynamic deadlock detection, deadlock detection agents (DDA) are dynamically created for transactions that access the same data objects. This is first proposed in [8]. This allows the DDA scheme to adapt or self-tune to the system transaction load.

Deadlock Resolution

Whenever deadlocks are detected, deadlock resolution is used to remove the deadlocks. The key idea in deadlock resolution is to minimize the cost incurred as a result of resolving the deadlock. There are two possible strategies for deadlock resolution. In the first strategy, a deadlock resolver aborts one or more transactions that caused the deadlock. In the second strategy, transactions are timed-out whenever deadlock occurs.

In the first strategy, one or more transactions are selected to be aborted. These transactions are referred to as the *victim transaction(s)*. Singhal [14] presents a general strategy for deadlock resolution in distributed database systems. A victim transaction which will optimally resolve the deadlock is selected. The victim transaction is aborted, and the locks that are held by the transaction are released. Deadlock detection information that are related to the victim transaction is removed from the system. In a distributed database system, several issues need to be considered when aborting transactions. First, whenever a deadlock is detected by a site, the site might not have access to the global deadlock information. Second, multiple sites might independently detect the same deadlock. This might cause the sites to independently resolve the deadlock. Consequently, this causes more transactions to be aborted than necessary. To solve this issue, a deadlock resolver can be selected from amongst the sites or the deadlock detection by various sites can be prioritized. In order to determine the set of victim transactions, various heuristics can be used. Intuitively, the heuristics ensure that the cost of aborting transactions is minimized. Some of these heuristics include: choosing the youngest transaction in the cycle [8] or choosing the transaction that causes the maximum number of cycles [9]. The first heuristic is motivated by the observation that the youngest transaction has just started execution. Hence, it is less costly to abort. In contrast, an older transaction has executed for some time, and will be costly to abort. The second heuristic is motivated by the observation that when deadlocks occur, it is important to break the cycles in the global WFG. If a transaction that caused the maximum number of cycles is aborted, it can potentially remove more cycles in the WFG. Hence, deadlock can be resolved faster.

In the second strategy, deadlocks are resolved using time-outs. This strategy is suitable for the case where deadlocks are infrequent. The time-out interval determines the waiting time for a transaction to be aborted. Thus, the selection of an optimal time-out interval is important. A short time-out interval will cause transactions to be aborted unnecessarily. On the other hand, a long time-out interval will result in the slow resolution of deadlocks might not be resolved quickly. Consequently, this impacts on the responsiveness of the applications.

Key Applications

Deadlocks occur whenever locking-based protocols are used to manage shared data objects or resources. Distributed deadlock management is more challenging to handle because none of the sites have global knowledge of the entire system. The techniques that are described can be applied and adapted for deadlock management for various types of distributed systems.

Future Directions

The emergence of new computing platforms (e.g. Peer-to-Peer (P2P), cloud computing) present new interaction modalities with distributed data. As applications built on these paradigms mature, a natural progression would be the need for transactions which access shared resources. This compels the need for locking-based protocols to be used for accessing the shared resources. Consequently, many open issues arise for distributed deadlock detection in these new computing platforms.

Cross-references

► Distributed Concurrency Control
► Distributed Database Design
► Distributed Database Systems
► Two-Pheese Loching

Recommended Reading

1. Abonamah A.A. and Elmagarmid A. A survey of deadlock detection algorithms in distributed database systems. Advances in Distributed and Parallel Processing (vol. one): system paradigms and methods, pages 310–341, 1994.
2. Bracha G. and Sam T. Distributed deadlock detection. Distributed Computing, 2(3):127–138, 1985.
3. Chandy K.M and Lamport L. Distributed snapshots: Determining global states of distributed systems. ACM Trans. Comput. Syst., 3(1):63–75, 1986.
4. Elmagarmid A.K. A Survery of distsributed deadlock algorithms. SIGMOD Record, 15(3):37–45, 1986.
5. Gray J. Notes on data base operating systems. In: Advanced Course: Operating Systems, pages 393–481, 1978.
6. Ho Gray S. and Ramamoorthy C.V. Protocols for deadlock detection in distributed database systems. IEEE Trans. Softw. Eng., 8(6):554–557, 1982.
7. Knapp E. Deadlock detection in distributed databases. ACM Comput. Surv., 19(4):303–328, 1987.
8. Krivokapić., N. Kemper A. and Gudes E. Deadlock detection in distributed database systems: a new algorithm and a comparative performance analysis. The VLDB Journal, 8(2):79–100, 1999.
9. Makki K. and Pissinou N. Detection and resolution of deadlocks in distributed database systems. In Proc. Int. Conf. on Information and Knowledge Management, 1995, pp. 411–416.
10. Menascé D.A. Muntz R. Locking and deadlock detection in distributed data bases. IEEE Trans. Softw. Eng., 5(3):195–202, 1997.
11. Mohan C., Lindsay., and Bruce G. Obermarck Ron Transaction management in the R* distributed database management system. ACM Trans. Database Syst., 11(4):378–396, 1986.
12. Özsu T.M. and Valduriez P. Principles of Distributed Database Systems, Second Edition. Prentice-Hall, 1999.
13. Roesler M., Burkhard W.A. and Cooper K.B. Efficient deadlock resolution for lock-based concurrency control schemes. In: Proc. 18th Int. Conf. on Distributed Computing Systems, pages 224–233, 1998.
14. Singhal M. Deadlock detection in distributed systems. Computer, 22(11):37–48, 1989.
15. Stonebraker M. The design and implementation of distributed ingres. The INGRES Papers: anatomy of a Relational Database System, Pages 187–196, 1986.

Distributed Hash Table

Wojciech Galuba, Sarunas Girdzijauskas
EPFL, Lausanne, Switzerland

Synonyms

DHT

Definition

A *Distributed Hash Table* (DHT) is a decentralized system that provides the functionality of a hash table, i.e. insertion and retrieval of key-value pairs. Each node in the system stores a part of the hash table. The nodes are interconnected in a *structured overlay network*, which enables efficient delivery of the key lookup and key insertion requests from the requestor to the node storing the key. To guarantee robustness to arrivals and departures of nodes, the overlay network topology is *maintained* and the key-value pairs are *replicated* to several nodes.

Key Points

Every DHT defines its key space. For example, in many DHTs the keys are 160-bit integers, which is the output of the popular SHA1 hash function. Each node in the system has a specific location in the key space and stores the key-value pairs that are close to that location. The different DHT systems vary in the exact algorithms for deciding which node should store which key [2].

The DHT relies on a structured overlay network for some of its functionality. The overlay network uses the DHT keys for addressing its nodes, i.e. the overlay is content (key) addressable. The structured overlay network provides the routing primitive which allows scalable and reliable delivery of key lookup and key insertion messages. The structured overlay implementations maintain the overlay topology and ensure routing efficiency as the nodes arrive and depart from the system (node churn) [1].

Not only the overlay topology but also the DHT data storage must be tolerant to churn. To achieve that, the key-value pairs are replicated across several nodes. A sufficient number of replicas needs to be maintained to prevent data loss under churn. There are a number of approaches to replication (cf. *P*eer-to-peer storage systems). In the most common replication strategy a node joining the DHT contacts the nodes that are close to it in the key space and replicates the key-value pairs that they store.

Cross-references

- ► Consistent Hashing
- ► Hash Table
- ► Overlay Network
- ► Peer-to-Peer Overlay Networks
- ► Peer-to-Peer System
- ► Replication in DHTs

Recommended Reading

1. Rhea S.C., Geels D., Roscoe T., and Kubiatowicz J. Handling churn in a DHT. In Proc. USENIX 2004 Annual Technical Conf., 2004, pp. 127–140.
2. Risson J. and Moors T. Survey of research towards robust peer-to-peer networks: Search methods. Comput. Networks, 50 (17):3485–3521, 2006.

Distributed Join

KAI-UWE SATTLER
Technical University of Ilmenau, Ilmenau, Germany

Synonyms

Join processing; Distributed query

Definition

The distributed join is a query operator that combines two relations stored at different sites in the following way: each tuple from the first relation is concatenated with each tuple from the second relation that satisfies a given join condition, e.g., the match in two attributes. The main characteristics of a distributed join is that at least one of the operand relations has to be transferred to another site.

Historical Background

Techniques for evaluating joins on distributed relations have already been dy discussed in the context of the first prototypes of distributed database systems such as SDD-1, Distributed INGRES and R*. In [6] the basic strategies ship whole vs. fetch matches were discussed and results of experimental evaluations were reported. Another report on an experimental comparison of distributed join strategies was given in [5].

Special strategies for distributed join evaluation that aim at reducing the transfer costs were developed by Bernstein et al. [2] (semijoin) as well as Babb [1] and Valduriez [11] (hashfilter join) respectively. The problem of delayed and bursty arrivals of tuples during join processing was addressed by particular techniques like the XJoin [10]. Moreover, a technique for dealing with limited query capabilities of wrappers in heterogeneous databases was introduced in [8] as a special variant of the fetch matches strategy.

Foundations

For evaluating a distributed join several tasks have to be addressed. First, the site where the actual join is to be processed, has to be chosen. Second, the operand relations have to be transferred to the chosen site if they are not already available there. Finally, the join between the two relations has to be computed locally at that site. In the following, these different issues are described in more details.

Site Selection

Choosing the site where the join operation is to be performed on is usually part of query optimization. For the simplest case of joining two relations *R* and *S* there are three possible join sites: the site of *R*, the site of *S*, and a third site at which the result is eventually needed. For the decision several aspects must be taken into account: the cost for transferring the operand relations, the join ordering (in case of multi-way joins) as well as subsequent operators in the query, e.g. at which site the join result is needed for further processing.

Relation Transfer

Given that one of the relations is available at the join site, there are two basic strategies to transfer the other relation. The *ship whole* approach works as follows:

1. The remote relation is shipped to the join site at a whole.
2. After the relation is received by the join site, it can be stored and used to perform the join locally.

In contrast, the *fetch matches* or *fetch as needed* strategy consists of the following steps:

1. The relation at the join site is scanned.
2. Based on the current value of the join attribute, the matching tuples from the other relation are requested.
3. If the other site has matching tuples, they are sent back and joined with the currently considered tuple.

The following example illustrates the costs (in terms of transferred data and number of messages) of these two strategies. Given two relations *R* (site 1) and *S* (site 2) with cardinalities $|R| = 2.000$ and $|S| = 5.000$, where the tuple sizes in bytes are $width(R) = width(S) = 100$ and the join attributes $A \in attr(R)$ and $C \in attr(S)$ have $width(A) = width(C) = 10$. Furthermore, a foreign key constraint $A \rightarrow C$ is assumed meaning that $|R \bowtie_{A=C} S| = 2.000$.

Using the ship whole approach the complete relation has to be shipped, i.e., in case of *R* the transfer volume is $|R| \cdot width(R) + |R \bowtie_{A=C} S| \cdot (width(R) + width(S))$, but only two messages are required (assuming that the relation is sent using a single message). With the fetch matches strategy the join attribute value of each tuple is shipped to the other site and for each tuple a result message is sent back. Thus, for relation R $2|R|$ messages are needed. However, in the first step $|R| \cdot width(A)$ bytes are transferred and in the second step $|R \bowtie_{A=C} S| \cdot width(S)$ bytes are sent back. The results for all strategies are shown in Table 1. Note, that only that cases are shown, where the result is sent back to the query issuer. Furthermore, for simplicity the message sizes are not considered in the transfer volume.

Distributed Join. Table 1. Transfer costs for join strategies

Strategy	Query issuing site	Transfer volume (KBytes)	Number of messages
Ship whole	Site 1	600	2
Ship whole	Site 2	900	2
Fetch matches	Site 1	220	4.000
Fetch matches	Site 2	250	10.000

The results show that:

- The ship whole approach needs a larger transfer volume.
- The fetch matches strategy requires more messages.
- The smaller relation should always be shipped.
- The transfer volume in the fetch matches strategy depends on the selectivity of the join.

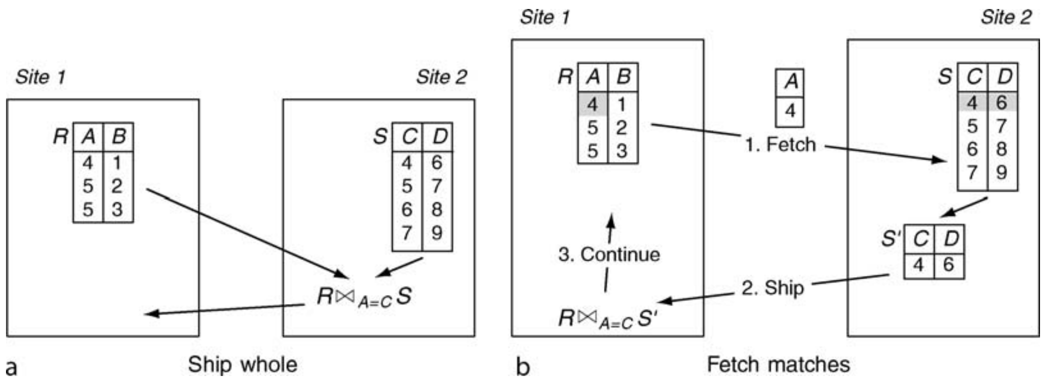
These basic strategies can further be improved by exploiting *row blocking*, i.e., sending several tuples in a block. Second, in case of the fetch matches strategy the relation can be sorted first in order to avoid fetching tuples for the same join values multiple times.

Local Join Processing

After having transferred the operand relations to the join site, the join is evaluated using any of the conventional algorithm known from centralized DBMS, e.g. nested-loops join, sort merge join, or hash join. If local indexes on the join attributes are available, they can be exploited. Furthermore, for the ship whole strategy a temporary index on the shipped relation can be created and exploited. In any case, the decision which strategy to use for the join evaluation is made in the local optimization step.

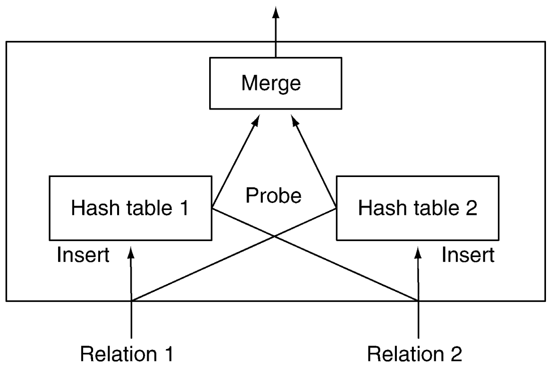
Sequential vs. Pipelined Processing

Another aspect of local processing is the strategy for dealing with delayed or bursty tuple arrivals. A first



Distributed Join. Figure 1. Basic strategies for join processing.

approach is to simply ignore this issue and assume a constant and on time arrival of tuples. Each incoming tuple is processed according to the chosen join strategy. In case of the sort merge join this could mean to wait for all tuples of the relation. With other strategies (e.g., if the shipped relation is the outer relation of a nested-loops join) the incoming tuple is directly processed in a pipelined fashion. However, if no tuple arrives, e.g., due to a network delay, no join result can be produced. An alternative solution is to use a double pipelined hash join, which exploits the inherent parallelism of distributed processing and in this way allows to reduce the overall response time.



Distributed Join. Figure 2. Double pipelined hash join.

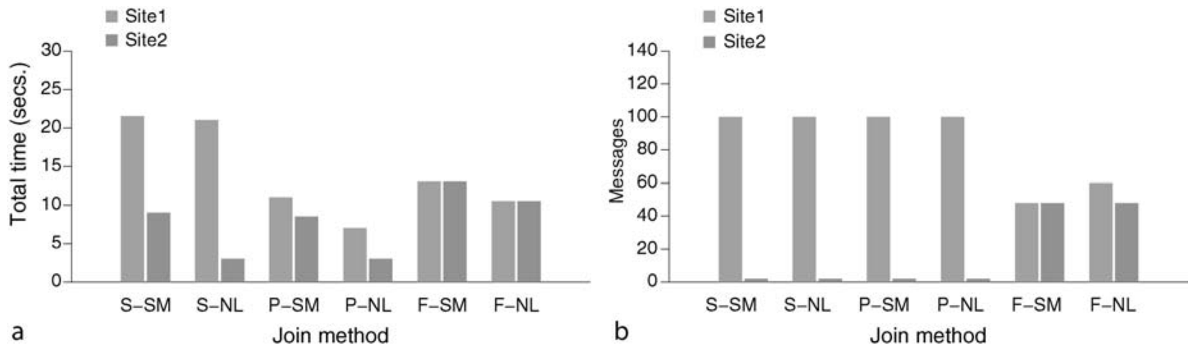
Algorithms

In the following some special approaches of distributed joins are described that extend the basic model of ship whole and fetch matches.

Semijoin/hashfilter join. This join algorithms can be regarded as a special variant of the fetch matches join with the aim to reduce transfer costs. For this purpose, only the projected join column or a compact bitmap representation (computed by applying a hash function) of that column of the first relation is sent to the second site as a whole. Next, all matching tuples are determined and sent back to the first site, where the actual join is computed.

XJoin. In wide-area networks with unpredictable response and transfer times, transmission delays may result in delayed or bursty arrivals of data. Furthermore, slow data sources can also delay or even block join processing. To address this problem and to allow to deliver join results as early as possible and continuously the XJoin [10] was proposed. It is based on the symmetric hash join, which works as follows: For each operand relation a hash table is maintained. Each incoming tuple is first inserted into the corresponding hash table. Next, it is used for probing the hash table of the other relation to find matching tuples, compute the join with them, and output the result immediately (Fig. 2). The XJoin extends this algorithm by considering the case where the hash tables exhaust the available main memory. For this purpose, a partitioning strategy is applied to swap out portions of the hash tables to disk-resident partitions. These partitions are also used to produce results when the site waits for the next tuples: in this case tuples from the disk are joined with memory-resident partitions. In order to avoid duplicates in the result special precautions are needed.

Bind join. In heterogeneous databases component databases (data sources) are usually encapsulated by wrappers responsible for query and result translation. Depending on the kind of the sources (e.g., a legacy system, a Website, or Web Service) these wrappers sometimes do not allow fetching the whole table or



Distributed Join. Figure 3. Comparison of different join algorithms.

evaluating a join. Instead they support only parameterized selections of the form

select * from *R* where *A* = ''?''

In order to still join another relation with *R* a bind join can be performed which is in fact a special fetch matches strategy. By scanning the outer relation the current value of the join column is passed on as the parameter ''?'' to the wrapper query and the results are collected and combined into the final join result. This can further be improved by precompiling the query plan, e.g., by exploiting prepared statements or cursor caching techniques.

Key Applications

The main application of distributed joins is query processing in distributed databases systems. In order to evaluate a join operation on relations stored at different sites a distributed strategy is needed.

A second area of application are heterogeneous databases, e.g., in the form of mediators or federated database systems. If legacy component databases or their wrappers provide only limited query capabilities (e.g., supporting only selections), a special strategy is required, which was introduced above as bind join. The XJoin presented above is also useful for evaluating joins in this context.

Other application examples for distributed joins include P2P systems for managing structured data, e.g., Peer Data Management Systems (PDMS) and P2P databases as well as distributed data stream processing systems.

Experimental Results

Experimental comparisons of different join strategies have been reported for example by Lu and Carey [5] as well as by Mackert and Lohman [6]. Figure 3 shows some results from [5] for the join between two relations from two different sites, each with 1,000 tuples and a result size of 100 tuples.

The join algorithms considered in this experiment are:

- A sequential combination of the ship whole approach with the nested-loops join for the local processing (S-NL) and the sort-merge join (S-SM). The shipped relation was stored in a temporary table and for the nested-loops variant an index was created before computing the join.
- A pipelined variant of this approach, where incoming tuples were processed on the fly (P-NL and P-SM).
- A strategy equivalent to the fetch matches approach using nested-loops join (F-NL) or sort-merge join (F-SM).

In Fig. 3a the elapsed time for processing the joins is shown, Fig. 3b depicts the number of messages.

Cross-references

► Evaluation of Relational Operators
► Semijoin

Recommended Reading

1. Babb E. Implementing a Relational Database by Means of Specialized Hardware. ACM Transactions on Database Systems 4(1):1–29, 1979.
2. Bernstein P.A., Goodman N., Wong E., Reeve C.L., Rothnie J.B. Query Processing in a System for Distributed Databases (SDD-1). ACM Transactions on Database Systems 6(4): 602–625, 1981.
3. Hevner A.R., Yao S.B.: Query Processing in Distributed Database Systems. IEEE Transactions on Software Engineering, 5(3):177–182, 1979.
4. Kossmann D. The State of the Art in Distributed Query Processing. ACM Computing Surveys 32(4):422–469, 2000.

5. Lu H., Carey M. Some Experimental Results on Distributed Join Algorithms in a Local Network. In Proc. 11th Int. Conf. on Very Large Data Bases, 1985, pp. 229–304.
6. Mackert L.F., Lohman G. R* Optimizer Validation and Performance Evaluation for Local Queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 84–95.
7. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 2nd Edition. Prentice Hall 1999.
8. Roth M.T., Schwarz P. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 266–275.
9. Stonebraker M. The Design and Implementation of Distributed INGRES. In The INGRES Papers, M. (ed.): Stonebraker Addison-Wesley, Reading, MA, 1986.
10. Urhan T., Franklin M.J. XJoin: A Reactively-Scheduled Pipelined Join Operator. Bulletin of the Technical Committee on Data Engineering 23(2):27–33, 2000.
11. Valduriez P. Semi-Join Algorithms for Distributed Database Machines. In Schneider J.-J. (Ed.) Distributed Data Bases, North-Holland, 1982, pp. 23–37.
12. Williams R., Daniels D., Hass L., Lapis G., Lindsay B., Ng. P., Obermarck R., Selinger P., Walker A., Wilms P., and Yost R. R*: An overview of the Architecture. IBM Research Lab, San Jose, CA, 1981.

Distributed Query

► Distributed Join
► Distributed Query Processing

Distributed Query Optimization

Stéphane Bressan
National University of Singapore, Singapore, Singapore

Synonyms

Query optimization in distributed database systems

Definition

Distributed query optimization refers to the process of producing a plan for the processing of a query to a distributed database system. The plan is called a query execution plan. In a distributed database system, schema and queries refer to logical units of data. In a relational distributed relation database system, for instance, logical units of data are relations. These units may be be fragmented at the underlying physical level. The fragments, which can be redundant and replicated, are allocated to different database servers in the distributed system.

A query execution plan consists of operators and their allocation to servers. Standard physical operators, usually implementing the data model's algebra, are used to process data and to consolidate intermediary and final results. Communication operators realize the transfer, sending and receiving, of data from one server to another. In the case of fragmentation the plan uses fragments instead of logical data units. In the case of replication, the plan defines the choice of replicas.

Distributed query optimization, like non-distributed query optimization, involves the enumeration of candidate query execution plans and the selection of an optimal or satisfactory plan with respect to a cost model.

A cost model for distributed query optimization involves not only local processing cost, i.e. the cost of central unit processing and of input/output operations but also the cost of communications.

A distributed query optimization algorithm selects an optimal or satisfactory plan by exploring parts of the combinatorial search space defined as the set of possible query execution plans. The function of cost to be optimized is called the objective function.

Historical Background

The three reference distributed relational database management systems are SDD-1 [1], Distributed Ingres [3] and System R* [4]. Their respective distributed query optimization algorithms are representative of the typical possible strategies.

The first distributed query optimization algorithm is Wong's "hill climbing" algorithm [10]. The algorithm greedily tries and improves an initial feasible solution and reaches, as the name indicates, a local optimum. The algorithm is further refined in the distributed query optimization algorithm of SDD-1 where it is extended to include semi-join programs. SDD-1 supports neither replication nor fragmentation. While Wong's algorithm works with a general objective function, SDD-1's implementation considers total communication cost only. Clearly, the overall focus of SDD-1's optimization approach is to reduce the volume of data transmitted. SDD-1 distributed query optimization is static.

Distributed Ingres' distributed query optimization algorithm [3] deterministically explores the search space of possible plans by making local optimization decisions at each step. Distributed Ingres supports horizontal fragmentation. The objective function is a weighted combination of total time cost and response time. It therefore includes both local and communication cost. Distributed Ingres' distributed query optimization is dynamic and therefore can benefit by the knowledge of the actual size of intermediary results.

System R*'s distributed query optimization algorithm is described in [7]. The algorithm exhaustively explores the search space of all possible query execution plans using dynamic programming to prune regions. The implementation of the algorithm in System R* supports neither replication nor fragmentation. The objective function is total cost and includes local processing and communication cost. System R*'s distributed query optimization is static.

Several approaches to dynamic query optimization have been proposed for parallel and distributed databases (see [5] for a rapid overview). Query scrambling [9], for instance, allows re-organization of the query execution plan during its execution in order to cope with unpredicted delays.

As an alternative to centralized optimization, the first system explicitly proposing an economical model for distributed query optimization is the Mariposa system [8].

While most of today's commercial database management system vendors offer distributed versions of their software, the continuous technological developments and the new application perspectives constantly compel extension and revision of existing distributed query optimization techniques in order to meet the needs of the new systems and applications.

Both textbooks [2,6] present and discuss in details the state of the art of distributed database technology in general and distributed query optimization in particular in the 1980s and 1990s, respectively. The survey [5] is a more current account of the development of these technologies and techniques.

Foundations

A distributed database management system consists of several dispersed database servers interconnected by an either local or wide area network. The database servers can be homogeneous or heterogeneous in hardware and software. The servers and network can be more or less autonomous.

Fragmentation and Replication

Data in a distributed database application may be fragmented. Logical units of data as they appear in the schema of the application may be further decomposed. Fragmentation can generally be expressed by means of logical operators. In the case of relational distributed database applications, a vertical fragment of a relation is the result of a relational projection, while a horizontal fragment is the result of a relational selection. Fragments are allocated to different servers in the distributed database management system. Fragmentation should be a lossless decomposition. However fragments can be redundant and replicated.

Fragmentation independence (often referred to as "fragmentation transparency") assumes that programmers write programs in terms of the logical schema of the application and ignore the details of fragmentation, redundancy and replication, and allocation. This property should be guaranteed for all design and programming purposes except, if necessary, for performance tuning.

The choice among possible fragments and replicas, the allocation of operations together with the cost and time needed for processing, as well as communication with local storage and network communication define the search space for the distributed query optimization problem and its objective function. Given a query to a distributed database system, its optimization, the distributed query optimization problem, is the choice of a query execution plan given fragmentation, redundancy and replication, and allocation, which is optimal or satisfactory with respect to the cost model, i.e. tries and minimizes the objective function.

Plan Enumeration

A query execution plan for a query to a distributed database management system is the decomposition of the query into local plans accessing fragments together with a global plan for the coordination of local executions, communications, and transmission, consolidation and production of results.

Since fragments and replicas are usually defined by logical operations, it is possible to rewrite the original query into a query in the same language involving only actual fragments and replicas. This decomposition is straightforwardly done by replacing each logical unit by a sub-query defining its reconstruction from fragments and replicas. For instance, in the simplest cases, a horizontally decomposed relation is the union of its fragments; a vertically decomposed relation is the natural join of its fragments. Because of redundancy and replication, there can be several possible decompositions using alternative fragments and replicas and therefore several alternative rewritings.

As with non-distributed query optimization, there may be several candidate access and execution methods, therefore several possible physical operators, for each logical operation. Furthermore, the data necessary to an operation may not be available at the server where the operation is allocated. Data needs to be transmitted from the server where it is located to the server where the operation is executed. When data transmission is a dominant cost, an extended set of operations can be considered to reduce data before transmission: this is how the semi-join operator is used in semi-join programs to favor transmission of small amounts of data and local operations to transmission of large amounts of data. One strategy consists in trying and fully reducing (when queries are acyclic) fragments before transmitting them.

A query execution plan is a tree of operators for processing, as in non-distributed database management systems, and, specifically to distributed database management systems, for communication and transmission of data. Every operator must be allocated to one database server. Non-distributed query optimizers are generally concerned with left- (or right-) deep trees of operators, i.e. sequence of operations. Distributed query optimizers need to consider bushy trees which contain opportunities for parallel execution. Indeed sibling sub-trees, when allocated to different servers, can be executed in parallel.

In summary, the plan enumeration involves the enumeration of alternative fragments and replicas, the choice of local execution methods, operations and communications, and of the choice of order and allocation of execution. As with non-distributed query optimization, the size of the search space is combinatorial. Strategies exploring the entirety of the search space are unlikely to be efficient. One can either use dynamic programming techniques to prune the search space yet finding an optimal solution, or use heuristics or randomized algorithms, such as simulated annealing, to find near or local optima. The exploration of the search space is guided by the objective function, i.e. the cost associated to each plan.

Total Cost Model and Response Time

For each query execution plan the optimizer tries and estimates its total cost, response time or a combination of both. This is the objective function of the optimization algorithm. The optimizer chooses the plan which minimizes the cost, the response time or finds the best combination or compromise.

At a coarse granularity, total cost is the sum of local cost and communication cost. At a finer granularity, the total cost model can be seen as an extension of the standard cost model for centralized databases to which is added the cost of network communications. The total cost is the sum of the unit cost of central unit processing, C_{cpu}, multiplied by the number of cycles, *#cycles*, and the unit cost an input/output operation, $C_{I/O}$, multiplied by the number of such operations, *#I/O*, with the cost of communications.

$$C_{total} = C_{cpu} \times \#cycles + C_{I/O} \times \#I/O + C_{comm}$$

It is commonly assumed that the cost of communication is linear in the number of bytes transmitted. The cost of communication combines the cost of initiating a message C_{s_mess}, the cost of receiving a messages C_{r_mess}, the number of messages *#mess*, the cost of transmitting one byte C_{tr_byte}, and the number of bytes transmitted *#bytes* as follows.

$$\begin{aligned} C_{comm} = & ((C_{s\_mess} + C_{r\_mess}) \times \#mess) \\ & + (C_{tr\_byte} \times \#bytes) \end{aligned}$$

Notice that the above formula requires the knowledge or estimation of the size of the elementary data units being processed as well as the selectivity of operations to estimate the size of intermediary results. The unit of cost is commonly time (in which case the model is referred to as the "total time cost model"). It is however probably more accurate to assume a Dollar cost since the total cost model is a measure of resource utilization.

The total cost model accounts for the usage of resources but does not account for the benefits of processing parallelism, input/output parallelism, and network parallelism naturally available in distributed database systems. Namely, processing, communication with local storage and network communications can happen in parallel since different machines interconnected by a physical network are involved.

Response time is the time, on the user's clock, needed by the system to execute a query. A simple example is a sequential scan operation (for instance in the case of a selection operation on a condition for which neither indexing nor fragmentation can be leveraged to rule out some fragments) of a relation fragmented into n fragments allocated to n different servers. The scan operation is decomposed into n scan operations: one for each individual fragments. If, for simplicity, it is assumed that all sequential scan operations start simultaneously, the response time is the time of the longest scan operation plus the time of longest request communication and the time of the last response communication. The total time is the sum of the times of individual sequential scans plus the sum of the times of all point-to-point (or broadcast, as possible in Distributed Ingres) communications.

Static versus Dynamic Distributed Query Optimization

Static query optimization is performed when compiling the application program without knowledge of the actual size of the elementary units of data. Dynamic query optimization is performed just before query execution thus allowing the optimizer to take such knowledge into account. A simple solution consists in statically generating several candidate plans for the same query and then dynamically choosing the plan with the best potential for efficient execution. Another solution, intermediary between static and dynamic query optimization prepares a query plan which is later allocated.

Furthermore modern distributed database applications leveraging the Internet are confronted with the unpredictable nature of a best effort communication network and of typically autonomously managed servers. Unpredictable delays result in queries executions being blocked sine die. Query scrambling is a dynamic approach that considers re-planning of the query during its execution. Initially a query execution plan for the distributed query is produced as described above. Re-planning takes the form of the rescheduling of transmissions and physical algebraic operators. If a plan execution is blocked because of a stalled communication, query scrambling attempts to perform other data transmission and operations initially scheduled for later in the original query execution plan. Query scrambling is a form of re-optimization at execution time (on-the-fly).

Global versus local, Centralized versus Distributed Query Optimization, andEconomical Models

The optimization process consists of both global (to the system) and local (to each database server) optimization. Global optimization is usually centralized at the server initially receiving the query. Global optimization needs to know or estimate the values required for the estimation of local cost, response time and communication cost, such as, for instance, the selectivity of sub-trees locally executed in order to estimate the number of bytes transmitted in the result. Details of local optimization, such as the choice of access methods, can be left to the component database servers.

This assumes that the database server performing the global optimization has or can collect sufficient information about the network and the other servers to properly estimate costs and to devise and choose a plan. Yet this information (including statistics about the network and local processing) may be difficult to collect and maintain if servers and network are autonomous, if the overall system is dynamic in nature, as well as if external independent factors influence its performance (e.g., general traffic of a public network).

Naïve attempts to distribute the classical distributed query optimization algorithms do not scale and rapidly lead to unsatisfactory sub-optimal solutions. One family of models for distributed optimization among autonomous agents is the one of economical models. These models try and simulate a free market regulated by supply and demand with negotiations based on bidding, auctioning or bargaining.

In a simple and simplified model with bidding for "distributed" distributed query optimization, the usage of resources is first monetized. Users (or user programs) assign a budget to their query and ask for the most economical execution available. The budget of a query is a decreasing function of response time, thus naturally expressing the acceptable compromise between resource usage and response time for the particular user. Brokers perform the (possibly partially) global by asking for and purchasing resources but remaining below the budget curve. Servers can actively try and maximize profit from selling resources by bidding for execution or buying or selling data.

Future Directions

The evolution of technology together with new applications places the optimization requirements for different new and future distributed database applications at various points on the spectrum between approaches conventionally referred to as centralized databases, parallel databases, distributed database, federated databases and peer-to-peer databases. Varying degrees of autonomy and heterogeneity, together with the relative importance of the elements defining cost change the requirements. For instance, while peer-to-peer database applications may be primarily concerned with response time and network congestion, mobile ad hoc network databases applications (distributed databases applications on mobile devices) may be primarily concerned with communication costs.

Cross-references

► Distributed Databases
► Distributed Database Design
► Distributed Query Processing
► Query Optimization
► Query Processing
► Semi-Join
► Semi-Join Program

Recommended Reading

1. Bernstein P.A., Goodman N. Wong E. Reeve C.L., and Rothnie Jr. Query processing in a system for distributed databases (SDD-1). ACM Trans. Database Syst., 6(4):602–625, 1981.
2. Ceri S. and Pelagatti G. Distributed Databases Principles and Systems. McGraw-Hill, New York, NY, USA, 1984.
3. Epstein R.S., Stonebraker M., and Wong E. Distributed query processing in a relational data base system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1978, pp. 169–180.
4. Haas L.M., Selinger P.G., Bertino E., Daniels D., Lindsay B.G., Lohman G.M., Masunaga Y., Mohan C., Ng P., Wilms P.F., and Yost R.A. R*: a research project on distributed relational dbms. IEEE Database Eng. Bull., 5(4):28–32, 1982.
5. Kossmann D. The state of the art in distributed query processing. ACM Comput. Surv., 32(4):422–469, 2000.
6. Özsu T. and Valduriez P. Principles of Distributed Database Systems, 2nd edn. 1999.
7. Selinger P.G. and Adiba M.E. Access Path Selection in Distributed Database Management Systems. In ICOD. 1980, pp. 204–215.
8. Stonebraker M., Devine R., Kornacker M., Litwin W., Pfeffer A., Sah A., and Staelin C. An economic paradigm for query processing and data migration in mariposa. In PDIS. IEEE Computer Society, 1994, pp. 58–67.
9. Urhan T., Franklin M.J., and Amsaleg L. Cost based query scrambling for initial delays. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 130–141.
10. Wong E. Retrieving dispersed data from SDD-1: a system for distributed databases. In Berkeley Workshop, 1977, pp. 217–235.

Distributed Query Processing

KAI-UWE SATTLER
Technical University of Ilmenau, Ilmenau, Germany

Synonyms

Distributed query; Distributed query optimization

Definition

Distributed query processing is the procedure of answering queries (which means mainly read operations on large data sets) in a distributed environment where data is managed at multiple sites in a computer network. Query processing involves the transformation of a high-level query (e.g. formulated in SQL) into a query execution plan (consisting of lower-level query operators in some variation of relational algebra) as well as the execution of this plan. The goal of the transformation is to produce a plan which is equivalent to the original query (returning the same result) and efficient, i.e. to minimize resource consumption like total costs or response time.

Historical Background

Motivated by the needs of large companies and organizations that manage their data at different sites, distributed database systems are subject of research since the late 1970s. In these years, three important prototype systems were developed which already introduced fundamental techniques of distributed query processing. The first system was SDD-1 [1], developed at Computer Corporation of America between 1976 and 1980, that run on PDP-10 mainframes connected by Arpanet. SDD-1 pioneered among others optimization techniques and semijoin strategies. The two others, Distributed INGRES [8] (University of Berkeley, 1977–1981) and R* [11] (IBM Research, 1981–1985) contributed further techniques for distributed databases but none of these approaches was commercially successful. In [10], Stonebraker and Hellerstein explained this by the lack of an adequate and stable networking environment at this time and the slow market acceptance of distributed DBMS. Today, this has radically changed: the major DBMS vendors offer solutions that include query processing facilities allowing to query and combine remote tables. Typically, this is achieved by extensions of the standard query processor such as special operators for executing remote (sub-)queries, adding distributed queries to the search space of the query optimizer as well as cost functions dealing with network communication.

Besides these classic techniques, several new approaches have been developed since the early prototypes. The Mariposa system [9] was based on a micro-economic model for query processing where sites (servers) bid in an auction to execute parts of a query which is payed by the query issuer (client). This allows different bidding strategies for each server in order to maximize its own profit and may better work in a non-uniform, wide-area environment with a large number of sites. Another line of research addressed the problem of changing or unpredictable communication times and data arrival rates in a distributed environment by dynamic or adaptive approaches. Proposed solutions range from operator-level adaptation by introducing dedicated operators (e.g. join strategies) to adaptation at query level by interleaving of query optimization and execution [4].

Foundations

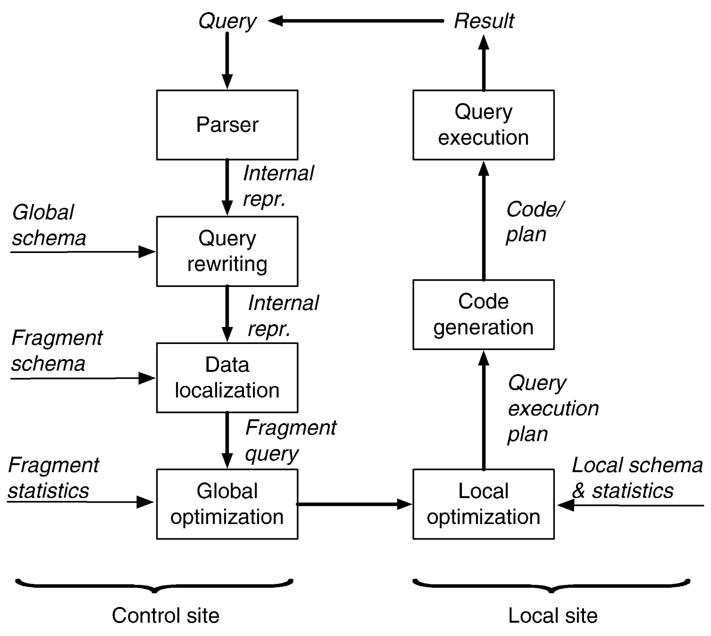
The subject of distributed query processing is to answer a query on data managed at multiple sites. This involves several steps for transforming a high-level query into an efficient query execution plan and opens various alternative ways for executing query operations of this plan. In this rest of this section, these phases are described and the most important techniques for each step are discussed.

Phases of Distributed Query Processing

The procedure of distributed query processing (Fig. 1) follows the approach of conventional query processing in database systems: In the first phase, a given query is parsed and translated into an internal representation (e.g. a query graph with nodes representing operators of an extended relational algebra). Next in the rewriting phase, the query is further transformed by applying equivalence rules in order to normalize, unnest, and simplify it without taking physical aspects such as cardinalities of relations, existence of indexes etc. into account. In the next step, the query is optimized by replacing the logical query operators by specific algorithms (plan operators) and access methods as well as by determining the order of execution. This is typically done by enumerating alternative but equivalent plans, estimating their costs and searching for the best solution. Finally, the chosen query execution plan is sent to the query execution engine, possibly after generating executable code.

In the distributed case, the phases of rewriting and optimization are extended. During query rewrite global relations referenced in the query have to be replaced by the corresponding fragmentation and reconstruction expressions resulting in fragment queries. Furthermore, reduction techniques are applied to eliminate redundant fragment queries or queries producing empty results. These steps are called *data localization.*

The optimization phase is split into a global step performed at the control site where the query was submitted and a local step which is done at each site maintaining a fragment relation referenced in this query. *Global optimization* involves the decision at which site the operation is to be executed as well inserting communication primitives into the plan. Global optimization typically ignores access methods like index usage for fragment relations – this is part of the local optimization.



Distributed Query Processing. Figure 1. Phases of Distributed Query Processing.

Data Localization

Usually, a distributed database is designed in a way that fragments of a global relation are stored at different sites. These fragments are defined by fragmentation expressions expressed for example as relational queries. In order to provide location and fragment transparency a query on a global relation R has to be transformed to a query that accesses only the fragments of R. This is achieved by replacing the global relation by the expression for reconstructing R from the fragments. Assuming a global relation $R(A, B, C)$ which is horizontally fragmented into R_1, R_2, R_3 as follows:

$$R_1 = \sigma_{A<100}(R)$$
$$R_2 = \sigma_{100 \le A \le 200}(R)$$
$$R_3 = \sigma_{A>200}(R)$$

Then, R can be reconstructed by $R = R_1 \cup R_2 \cup R_3$. Using this fragmentation information, a global query $q_1 := \sigma_{A>150}(R)$ is now transformed into the query

$$q_1' := \sigma_{A>150}(R_1 \cup R_2 \cup R_3)$$

Obviously, this is not an efficient query because R_1 does not contribute to the result. Therefore, reduction techniques are applied which exploit equivalence transformations of relational algebra expressions in combination with rules for identifying fragment queries. This produces empty results due to useless selections or joins. These rules work mainly by analyzing predicates of fragment expressions and queries and finding contradictions [2].

For instance, for horizontal fragmentation the following rule can be used:

$$\sigma_{p_i}(R_j) = \emptyset \text{ if } \forall \mathrm{t} \in \mathrm{R} : \neg(\mathrm{p_i(t)} \wedge \mathrm{p_j(t)}) \qquad (1)$$

Considering query q'_1, the selection operator can be pushed down to the fragments. Then, the rule finds the contradicting predicate $A < 100 \wedge A > 150$ resulting in an empty result:

$$\underbrace{\sigma_{A>150}(R_1)}_{\emptyset} \cup \sigma_{A>150}(R_2) \cup \sigma_{A>150}(R_3)$$

Thus, the fragment query $\sigma_{A>150}(R_1)$ can be eliminated.

A similar technique is used for identifying useless joins using the following rule:

$$R_i \bowtie R_j = \emptyset \text{ if } \forall \mathrm{t_1} \in \mathrm{R_i}, \forall \mathrm{t_2} \in \mathrm{R_j} : \neg(p_i(t_1) \wedge p_j(t_2)) \qquad (2)$$

Assume a second relation $S(A, D)$ with the fragments:

$$S_1 = \sigma_{A \le 200}(S)$$
$$S_2 = \sigma_{A>200}(S)$$

A local query $q_2 := R \bowtie S$ has to be transformed into:

$$q_2' := (R_1 \cup R_2 \cup R_3) \bowtie (S_1 \cup S_2)$$

By distributing the join over the unions and applying rule (2) query q'_2 can be rewritten into the following form where three expressions can be identified as useless joins due to contradicting predicates:

$$(R_1 \bowtie S_1) \cup (R_2 \bowtie S_1) \cup \underbrace{(R_3 \bowtie S_1)}_{\emptyset} \cup \underbrace{(R_1 \bowtie S_2)}_{\emptyset} \cup \underbrace{(R_2 \bowtie S_2)}_{\emptyset} \cup (R_3 \bowtie S_2)$$

This is done by looking only at the predicates of the fragment expressions and not at the relations.

For vertical fragmentation the main goal of reduction is to identify useless projections. Assuming the following fragmentation of relation R:

$$P_1 = \pi_{A,B}(R)$$
$$P_2 = \pi_{A,C}(R)$$

the reconstruction expression is $R = P_1 \bowtie P_2$. Thus, a query $q_3 := \pi_{A,C}(R)$ has to be transformed into the following query:

$$q_3' := \pi_{A,C}(P_1 \bowtie P_2) = \pi_{A,C}(P_1) \bowtie \pi_{A,C}(P_2)$$

A projection on a vertical fragment is useless, if there are no common attributes between the projection and the fragment beside the key attribute. Note, this does not results in an empty relation – however, the result is redundant. This can be formulated in the following rule. Given a fragmentation $R_i = \pi_B(R)$ of relation $R(K, A_1,\ldots,A_n)$ where $B \subseteq \{K, A_1,\ldots,A_n\}$.

$$\pi_{K,B'}(R_i) \text{ is useless, if } B' \cap B = \emptyset \qquad (3)$$

Using this rule for query q'_3, the projection on fragment P_1 can be identified as useless and eliminated from the query:

$$\underbrace{\pi_{A,C}(P_1)}_{\text{useless}} \pi_{A,C}(P_2)$$

For derived and hybrid fragmentation similar techniques exist which are described e.g. in [7].

Optimization of Distributed Queries

Whereas the local optimization step is the same as in a centralized database system, the global optimization comprises some special tasks.

As in conventional query optimization the *ordering of joins* is one of the main challenges. Typically, in a distributed DBMS bushy join trees are considered by the optimizer in addition to linear (left or right deep) trees. Bushy trees are a shape of join trees where both operands can be intermediate results instead of requiring one base relation operand. These bushy trees allow better exploiting parallelism in distributed queries. Furthermore, in a distributed database the problem of *site selection* is added, i.e. at which site an operation (e.g. a distributed join) is to be executed. In the simple case of joining two relations stored at different sites this can be decided simply by comparing the size of the relations: the smaller relation is sent to the other site. However, for a multi-way join the number of possible strategies is much larger and the sizes of the intermediate join results have to be estimated and considered for finding the optimal join ordering.

Basically, there are two fundamental options in site selection: either the data is retrieved from the storing site to the site executing the query (called *data shipping*) or the evaluation of the query is delegated to the site where the data is stored (*query shipping*). These two models are shown in Fig. 2. Data shipping can avoid bottlenecks on sites with frequently used data if many queries have to be evaluated by exploiting client resources. On the other hand, query shipping is the better choice if otherwise large amount of data has to be transferred to the processing site. Thus, a hybrid strategy is sometimes the best solution [3].

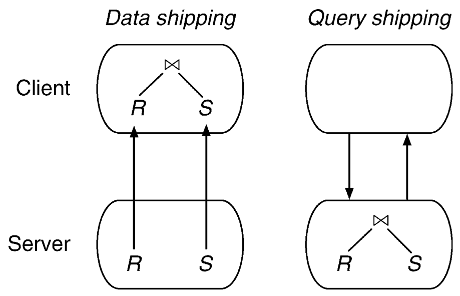
A further task in the global optimization step is to decide which strategies or algorithms are to be used to process the query operators. For example, a join between two relations stored at different sites can be processed in different ways: e.g. by shipping one relation to the site of the other relation as a whole (ship whole), by scanning one of the relations and fetching matching tuples from the other site, or by exploiting a semijoin strategy. Thus, during plan enumeration all these alternatives have to be considered by the optimizer.

Choosing the best query execution plan from the set of alternative plans requires to *estimate costs*. The conventional approach in centralized DBMS focuses on total resource consumption by estimating I/O and CPU costs. In a distributed environment, communication costs depending on the network speed have to be taken into account, too. However, the total resource consumption approach optimizes the throughput of a system, but ignores the inherent parallelism of a distributed query. For example, a scan on two fragments can be executed in parallel which reduces the query execution time to the halve. An alternative approach is the *response time model* which estimates the time between initiating the query and the receipting of the query result. Figure 3 illustrates the difference between these two models. Given two relations R and S, a query $R \bowtie S$ submitted at site 3 and assume that the costs are measured in time units.

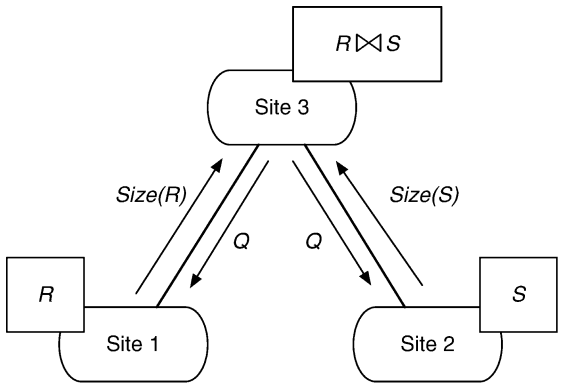
Then, the total resource consumption T_{total} of the query plan is

$$T_{total} = 2 \cdot T_Q + T_S(size(R) + size(S))$$

where T_Q is the time for sending a message (in this case the query request), T_s is the time for shipping a data



Distributed Query Processing. Figure 2. Data shipping vs. query shipping.



Distributed Query Processing. Figure 3. Total resource consumption vs. response time.

unit from one site to another, and *size*(*R*) is the size of relation *R*. In contrast, using the response time model the cost is

$$T_{response} = \max\{T_Q + T_S \cdot size(R), T_Q + T_S \cdot size(S)\}$$

This would prefer execution plans exploiting parallel processing as well as resources of other sites and therefore is better suited for distributed environments if communication is rather cheap.

Query Execution

Basically, the query execution phase of distributed processing is very similar to centralized DBMS. However, there are some specialties which are

- The need for send/receive operators for submitting fragment queries to other sites and shipping back results to the query issuer.
- Dedicated algorithms are required particularly for join processing, e.g. semijoin filtering or non-blocking hash joins.

Finally, there are some special optimization techniques which can be applied. First, in order to reduce the communication overhead caused by network latency, tuples are transferred in a block-wise manner (*row blocking*) instead of sending each tuple at a time. Second, *caching results* for reusing in future queries may help to reduce communication costs, too. Caching requires to balance the costs for loading and maintaining the data and the benefits of answering a query (partially) from the cache [6]. Furthermore, the cached data chunks have to be described in order to be able to check the containment of queries efficiently.

Key Applications

The main application of distributed query processing are classic distributed databases allowing to store and query data at different sites transparently. Another application domain are client/server database systems where the data is stored at a server and queries are submitted at client machines. Often, the computing resources of the client machines can be used for processing (portions of) the queries, too. Typical examples of such systems are object-oriented database systems but also middleware technologies such as application servers.

The existence of legacy databases of all flavors in many companies leads to the need of accessing and integrating them using queries in a standardized (e.g. relational) language on a composite schema. For this purpose, heterogeneous DBMS and mediators were developed that provide wrappers/gateways for encapsulating the system-specific transformation of queries and results between the global level and the local component system. Heterogeneous DBMS exploit distributed query processing techniques but have to deal with the heterogeneity of schema and data as well as with the possibly limited query capabilities of the component system. Commercial DBMS vendors have quickly adopted these techniques and offer now their own gateway solutions in addition to distributed query processing features, e.g. Oracle Database Gateway and IBM Information Server.

P2P systems and sensor networks are a current application area of distributed query processing. In these approaches, a network of nodes or sensors can be seen as a distributed database. Answering queries requires to retrieve and process data from different nodes using techniques described above. The challenges in these areas are mainly scalability (e.g. querying thousands of nodes) as well as dynamics and unreliability of the network.

Cross-references

► Distributed DBMS
► Distributed Join
► Distributed Query Optimization
► Query Processing

Recommended Reading

1. Bernstein P.A., Goodman N., Wong E., Reeve C.L., and Rothnie Jr., J.B. Query processing in a system for distributed databases (SDD-1). ACM Trans. Database Syst. 6(4):602–625, 1981.
2. Ceri S. and Pelagatti G. Correctness of query execution strategies in distributed databases. ACM Trans. Database Syst. 8 (4):577–607, 1983.
3. Franklin M., Jonsson B., and Kossmann D. Performance tradoffs for client-server query processing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 149–160.
4. IEEE Computer Society. Bull. Tech. Committee Data Eng., 23 (2), June 2000.
5. Kossmann D. The state of the art in distributed query processing. ACM Comput. Surv., 32(4):422–469, 2000.
6. Kossmann D., Franklin M., Drasch G., and Ag W. Cache investment: integrating query optimization and distributed data placement. ACM Trans. Database Syst. 25(4):517–558, 2000.
7. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 2nd edn. Prentice-Hall, USA, 1999.

8. Stonebraker M. The Design and Implementation of Distributed INGRES. In stonebraker M. (ed.). The INGRES Papers, Addison-Wesley, Reading, MA, 1986.
9. Stonebraker M., Aoki P., Litwin W., Pfeffer A., Sah A., Sidell J., Staelin C., and Yu A. Mariposa: a wide-area distributed database system. VLDB J. 5(1):48–63, 1996.
10. Stonebraker M. and Hellerstein, J.M. Distributed Database Systems. In: M. Stonebraker and J.M. Hellerstein (eds.). Readings in Database Systems, 3rd edn. Morgan Kaufmann, San Francisco, CA, 1998.
11. Williams R., Daniels D., Hass L., Lapis G., Lindsay B., Ng. P., Obermarck R., Selinger P., Walker A., Wilms P., and Yost R. R*: An overview of the Architecture. Technical Report RJ3325, IBM Research Lab, San Jose, CA, 1981.
12. Yu C.T. and Chang C.C. Distributed query processing. ACM Comput. Surv., 16(4):399–433, 1984.
13. Yu C.T. and Meng W. Principles of Database Query Processing for Advanced Applications. Morgan Kaufmann, 1998.

Distributed Recovery

Kian-Lee Tan
National University of Singapore, Singapore, Singapore

Synonyms

Recovery in distributed database systems
Recovery in distributed commit protocols
Recovery in replicated database systems

Definition

In a distributed database system (DDBS), failures in the midst of a transaction processing (such as failure of a site where a subtransaction is being processed) may lead to an inconsistent database. As such, a recovery subsystem is an essential component of a DDBS [14]. To ensure correctness, recovery mechanisms must be in place to ensure transaction atomicity and durability even in the midst of failures.

Distributed recovery is more complicated than centralized database recovery because failures can occur at the communication links or a remote site. Ideally, a recovery system should be simple, incur tolerable overhead, maintain system consistency, provide partial operability and avoid global rollback [6].

Historical Background

A DDBS must be reliable for it to be useful. In particular, a reliable DDBS must guarantee transaction atomicity and durability when failure occurs. In other words, a committed transaction's actions must persist across all the sites at which it is processed, and an aborted transaction's actions must not be allowed to persist. A transaction is aborted either explicitly (through an abort command) or implicitly as a result of a failure prior to its completion (through the recovery mechanism).

However, in a DDBS, besides traditional failures types which occur in a centralized system (such as media and power failures), new types of failures may occur, e.g., communication link failure, network partitioning, delayed/lost messages and remote site failure. While existing recovery mechanisms on centralized systems can be employed to handle the former type of failures, the latter kind is more complicated to deal with.

To ensure transaction atomicity, distributed commit protocols have been proposed. These include Two-Phase Commit and its variants [3,7,12] and Three-Phase Commit [13]. To recover from failures, a log is maintained at each site. As in centralized system, the log contains information such as the before and after view of an updated record. In addition, to facilitate distributed recovery, actions of the distributed commit protocol are also logged. In this way, each site knows the execution status of a transaction prior to the failure, and can determine if a transaction is committed or not before taking the necessary action. For example, for committed transaction, the log facilitates replaying the operations in the same order as they did before the failure; for uncommitted transactions, the operations has to be undone.

Log-based recovery protocols incur high overhead and may result in low transaction throughput. This is because log records have to be forced-written out to disks during logging, and have to be read from disks again during recovery. In order for a failed site to be recovered speedily, checkpointing techniques have also been proposed [1,10,11]. These schemes periodically maintain consistent states so that a failed site needs to rollback to a recent consistent state to reduce actions to be undone or redone.

There have been some efforts to reduce recovery overhead, and to perform online recovery so as not to disrupt transaction processing. In [15], an agent-based recovery protocol is proposed. The key idea is to buffer new database actions issued at the failed site (as it recovers using an existing log recovery scheme), and then replayed these buffered actions over the recovered state independently. In [5], distributed and redundant logging is used in which a site redundantly logs records to main memory and additionally to a remote site. In this way, the expensive forced-writes are avoided totally. In [9], HARBOR eliminates recovery logs (and hence the expensive disk write operations for log records) by exploiting replication/redundancy. This is done by keeping replicas consistent so that recovering a crashed site can be done by querying remote, online sites for missing updates.

All commercial systems today employ log-based roll-back recovery methods.

Foundations

A DDBS is vulnerable to failures that occur in centralized systems, such as power failure that may result in lost of data at a site. In addition, a DDBS is highly dependent on the availability of all sites, as well as their ability to communicate reliably with one another. When a site fails, sub-transactions running at that site may render the database inconsistent. Likewise, communication failures can result in the network becoming split into two or more partitions whose data may not be consistent. Even timeout may be deemed as a failure. As such, recovery is more complicated for DDBS.

In DDBS, log-based techniques are the pre-dominant schemes used to provide transaction atomicity and durability. Each site maintains a log which extends the widely used write ahead logging (WAL) scheme in centralized systems [3,4]. Under WAL, all modifications to records are written to a log before they are applied. Essentially, each log record contains a four-tuple $<tid, oid, v_{old}, v_{new}>$ where *tid* is the transaction identifier, *oid* is the object identifier, v_{old} and v_{new} are the old and new values of the updated object. There are also additional log records $<tid, start>$ and $<tid, commit>$, $<tid, abort>$ that capture the start of transaction *tid* and that it has committed or aborted. These log records enable the system to redo an operation if a transaction has committed (i.e., if the commit log record is in the log, then it means the transaction has committed, and the object can be updated to the new value v_{new}). Similarly, if the log commit/abort records are not found, then it means that failure has occurred before the transaction is committed/aborted. Hence, it may be necessary to undo some of the operations (by overwriting the object value with v_{old}). In this way, a database can be restored to a consistent state.

The extension arises because of the distributed commit protocols that are introduced to synchronize subtransactions of a transaction during the commit process. This calls for state transitions to be logged so that a failed site knows the status of a transaction's execution and can recover independently without communication from other sites. Consider the two-phase commit protocol which operates as follows: (i) In phase 1, the coordinator requests votes from all participants which may respond with a Yes vote (to commit the transaction) or No vote (to abort the transaction). (ii) In phase 2, the coordinator will make a global decision based on the votes – if all participants vote Yes, then the global decision is to commit the transaction; otherwise, the global decision is to abort it. Failures can occur at different state transitions.

The following log records will be generated at the coordinator: (i) before commit is initiated, the coordinator will generate a $<Tid, start - 2PC>$ log which also contains the list of participants involved in processing the transaction *Tid*. (ii) When the coordinator is ready to commit the transaction (after receiving positive votes from all the participants), it will create a $<Tid, commit>$ log record; alternatively, for a global abort decision, a $<Tid, abort>$ log record is written instead. (iii) Finally, when all participants acknowledged that the commit operations have been performed, the coordinator will add a $<Tid, end - 2PC>$ log.

On the other hand, at each participant, the following log records will be created: (i) a participant that is ready to commit upon receiving a request-for-vote message will log its vote, i.e., a log record $<Tid, Wait - State>$ is created; on the other hand, a participant that will not commit the transaction will create a log record $<Tid, Abort>$. (ii) Upon commit (after receiving the global commit decision), the participant will create a log commit record $<Tid, Commit>$. However, if the global decision is to abort the transaction, if the participant voted Yes, then $<Tid, Abort>$ will be created.

When there is a failure during the commit process, the operational sites will execute a termination protocol that will bring the transaction to a final state (either global abort or commit depending on the states of the operational sites). For two-phase commit, it is possible that the operational sites will need to wait for the failed sites to recover and become operational again before any further action can be taken.

For the failed site, there are two possible cases. First, the failed site is the coordinator. In this case, upon

Comp. by: SIndumathi Stage: Revises1 ChapterID: 000000A824 Date:16/6/09 Time:03:30:01

recovery, the coordinator checks its log. (i) If the failure happens prior to the initiation of the commit procedure (i.e., the failed site did not find the $<Tid, start - 2PC>$ record in its log, then it can initiate the commit procedure. (ii) If the failure occurs in the waiting state (i.e., $<Tid, start - 2PC>$ is found in the log but not $<Tid, commit>$), then it can restart the commit procedure. (iii) Finally, if the failure takes place in the commit/abort state, then if the coordinator has received all acknowledgements, it can complete successfully; otherwise, it will initiate the termination protocol as if it is an operational site.

Second, the failed site is a participant site. Again, there are several possible scenarios. (i) If the site failed before receiving any initiation request, then it can unilaterally abort the transaction. This is because the operational site would have aborted the transaction. (ii) If the site failed in the waiting state (i.e., $<Tid, Wait - State>$ is found in the log but not $<Tid, commit>$), then the recovered site will initiate the termination protocol as an operation site. (iii) Finally, if the $<Tid, commit>$ or $<Tid, abort>$ or log record is found in the log, then no further action is necessary, as it means that the transaction has already committed/aborted.

Now, the recovery process for variants of the two-phase commit protocol is similar. In some case, one can optimize by allowing non-blocking of operational sites, in which case, the failed site may need to verify the status of the distributed commit operation with other operational sites upon recovery.

Key Applications

Since distributed systems will always encounter failure, distributed recovery methods are needed in all distributed systems to ensure data consistency. In particular, techniques developed for DDBS can be adapted and extended to emerging platforms like peer-to-peer (P2P) computing, mobile computing, middleware, and publish/subscribe systems.

Future Directions

Although distributed recovery is a fairly established topic, more work need to be done to support online recovery. In addition, the emergence of new distributed computing platforms (e.g., P2P computing, cloud computing, mobile computing, middleware) brings new challenges that require more effective solutions than what are currently available in the literature. For example, in P2P systems, the dynamism of node join and departure makes it difficult for a failed node to recover fully (since nodes that are operational at the time of the node's failure may no longer be in the network). It remains a challenge to device an effective recovery scheme in this context. As another example, consider mobile computing context (where nodes are mobile). Here, the base stations may be exploited to manage the logs [2]. However, because connections may be intermittent, many transactions will fail. A possible alternative is to allow a longer message delay to be tolerated. In both the P2P and mobile environments, a relax notion of serializability may be more practical. Yet another direction is to design recovery mechanisms for middleware. Initial effort has been done [7] but effective solutions are still lacking. Finally, a networked system is vulnerable to attacks that may corrupt the database. Recovering from such a state is an important subject that has not yet received much attention from the database community.

Cross-references

► Crash Recovery
► Logging and Recovery
► Three-Phase Commit
► Two-Phase Commit

Recommended Reading

1. Chrysanthis P.K., Samaras G., and Al-Houmaily Y.J. Recovery and performance of atomic commit processing in distributed database systems. In Recovery Mechanisms in Database Systems. Kumar Hsu Prentice-Hall, 1998. Chapter 13.
2. Gore M., Ghosh R.K. Recovery of Mobile Transactions. DEXA 2000 Workshop, 23–27, 2000.
3. Gray J. Notes on data base operating systems. In Operating Systems – An Advanced Course. Bayer R., Graham R., Seegmuller G. (eds.). LNCS, Vol. 60, pp. 393–481, Springer, 1978.
4. Gray J. et al. The recovery manager of the system R database manager. ACM Comput. Surv., 3(2):223–243, 1981.
5. Hvasshovd S., Torbjornsen O., Bratsberg S., Holager P. The clustra telecom database: high availability, high throughput, and real-time response. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 469–477.
6. Isloor S.S. and Marsland T.A. System recovery in distributed databases. In COMPSAC. 1979, pp. 421–426.
7. Jimenez-Peris R., Patino-Martinez M., and Alonso G. An algorithm for non-intrusive, parallel recovery of replicated data and its correctness. In Proc. 21st Symp. on Reliable Distributed Syst., 2002, pp. 150–159.
8. Lampson, B. and Sturgis H. Crash recovery in a distributed data storage system. Technical report, Computer Science Laboratory, Xerox Palo Alto Research Center, California, 1976.

9. Lau E. and Madden S. An integrated approach to recovery and high availability in an updatable, distributed data warehouse. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 12–15.
10. Lin J. and Dunham M.H. A low-cost checkpointing technique for distributed databases. Distrib. Parall. Databases, 10(3):241–268, 2001.
11. Lomet D. Consistent timestamping for transactions in distributed systems. Tech. Report CRL90/3, Cambridge Research Laboratory, Digital Equipment Corp., 1990.
12. Mohan C., Lindsay B., and Obermarck R. Transaction management in the R* distributed data base management system. ACM Trans. Database Syst., 11(4):378–396, 1986.
13. Skeen D. Non-blocking commit protocols. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1981, pp. 133–142.
14. Tamer Ozsu M. and Valduriez P. Principles of distributed database systems (2nd edn). Prentice-Hall, 1999.
15. Wang Y. and Liu X. Agent based dynamic recovery protocol in distributed databases. In ISPDC. 2003.

Distributed Sensor Fusion

► Data Fusion in Sensor Networks

Distributed Source Coding

► Data Compression in Sensor Networks

Distributed Spatial Databases

Panos Kalnis
National University of Singapore, Singapore, Singapore

Definition

Distributed spatial databases belong to the broad category of distributed database systems. Data reside in more than one sites interconnected by a network, and query processing may involve several sites. A site can be anything from a server to a small mobile device. The broad definition covers many research areas. This entry gives an overview of the following sub-categories: (i) Distributed spatial query processing, which focuses mainly on spatial joins. (ii) Distributed spatial indexes (e.g., a distributed version of the R-tree). (iii) Spatial queries in large distributed systems formed by devices such as PDAs, mobile phones, or even sensor networks.

Historical Background

Similar to relational databases, in spatial databases the most important operator is the spatial join. In relational databases, distributed joins are often implemented by using the semi-join operator. Let R and S be relations residing in two different sites R_{site} and S_{site}. First R_{site} calculates R' which is the projection of R on the join attribute. R' is transmitted to S_{site} and is joined with S; the result is called *semi-join*. That result is sent back to R_{site} where it is joined with R to produce the final join between R and S.

The semi-join concept can be adapted for joins between spatial datasets. However, the following characteristics of spatial datasets must be taken into account:

1. The relational semi-join is based on the assumption that the projected relation R' will be much smaller than R (since it contains fewer attributes and there are no duplicates), leading to lower transmission cost. In spatial datasets, the equivalent to the join attribute is the spatial object. The size of each spatial object (typically a polygon) may be in the order of hundreds or thousands of bytes, and usually dominates the size of other attributes. Therefore, projecting on the join attribute is not likely to reduce the transmission cost.
2. Evaluation of spatial relationships, such as containment, intersection and adjacency between two polygons, is complex and expensive (in terms of CPU and I/O cost), compared to testing the join condition in relational databases.

To address these issues, existing work [1,14] implements distributed spatial joins by using *approximations* of the spatial objects. This technique is common in spatial databases and involves two phases: the first phase operates on simple approximations of the objects (e.g., Minimum Bounding Rectangle – *MBR*) and produces a superset of the result. Since the approximations are simpler than the original objects, this phase is fast. The second phase removes the false hits of the intermediate result by operating on the exact polygons of the spatial objects.

The previous discussion assumes that the two sites are collaborating and allow access to their internal index structures. However, this is not always the case. Mamoulis et al. [10] and Kalnis et al. [5] assume that a mobile device is interested in the join of two spatial datasets whose sites do not collaborate. To avoid downloading the entire datasets, the mobile device interleaves the join with a statistics acquisition process which prunes parts of the space.

In addition to implementing distributed versions of spatial operators, several papers have focused on distributed spatial data structures (e.g., k-RP*;S [8] and hQT* [7]). A typical problem in distributed tree structures is that the server responsible for the root node is overloaded, since all queries must traverse the tree in a top-down fashion. Mouza et al. [12] proposed the SD-Rtree, which is a distributed version of the R-tree. Overloading the root is minimized by replicating a (possibly outdated) copy of the internal nodes of the tree in all clients. Several researchers [4,6,11] have also developed distributed spatial indices on top of Peer-to-Peer systems.

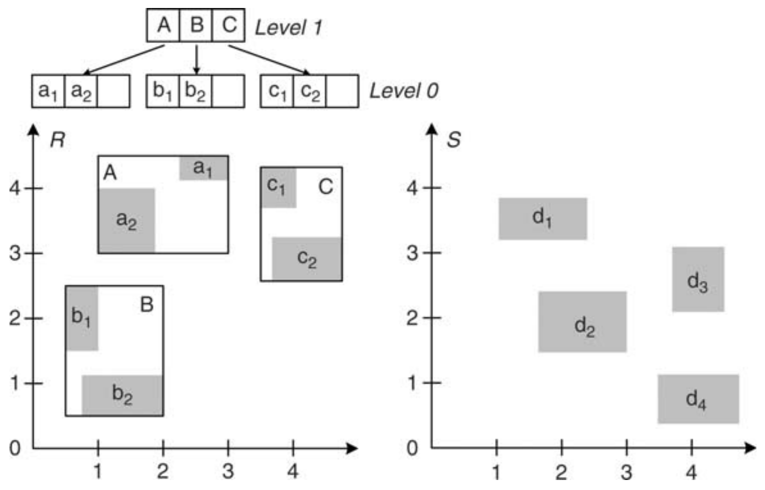
Finally, many applications assume a large number of mobile devices (e.g., PDAs, mobile phones) which store spatial data and ask spatial queries. For example, in the MobiEyes [3] system, mobile clients collaborate to answers continuous range queries. Other papers focus on broadcast-based wireless environments [15], which broadcast periodically the spatial index and data to all clients. Moreover, several sensor networks (e.g., Coman et al. [2]) optimize the query processing by exploiting the spatial properties (e.g., location, communication range and sensing range) of the sensors.

Foundations

The section on "Distributed Query Processing" discusses query processing in distributed spatial databases and focuses mainly on two cases: (i) the sites collaborate and are willing to share their internal indices (i.e., Tan et al. [14]) and (ii) the sites do not collaborate with each other (i.e., Mamoulis et al. [10]). Section "Distributed Spatial Indices" presents distributed spatial indices, the most representative being the SD-Rtree [12]. Finally, Section "Spatial Queries Involving Numerous Mobile Clients" discusses spatial queries that involve numerous and possibly mobile clients (e.g., smart phones, sensor networks, etc).

Distributed Query Processing

As mentioned above, distributed spatial joins are implemented by using approximations of the spatial objects. Tan et al. [14] investigate two approximation methods. The first one assumes that at least one of the datasets is indexed by an R-tree variant. In the example of Fig. 1, let R be the indexed dataset. The method uses the MBRs of the objects at level 0 (i.e., leaf level) or level 1 of the R-tree. Assuming that level 0 is used, R_{site} sends to S_{site} the following set of *MBRs*: $R' = \{a_1,a_2,b_1, b_2, c_1, c_2\}$. S_{site} performs window queries for each object in R' and returns to R_{site} the set $S' =\{ d_1,d_3\}$ of *objects* (i.e., polygons) which intersect with MBRs in R'. Finally R_{site} examines the polygons of the pairs (a_2, d_1) and (c_2,d_3) in order to remove any false hits. A second example assumes that level 1 of the R-tree is used. In this case R_{site} sends the MBR set $R' = \{A, B, C\}$ and S_{site} returns the set of polygons $S' = \{d_1, d_2, d_3\}$. Finally, R_{site} computes the join result from R and S'. It is noted that, if level 0 of the R-tree is used, R' typically contains many MBRs, allowing very refined search in S_{site}; consequently, S' usually contains a small number of objects. On the other hand, if level 1 is used, R' contains less MBRs

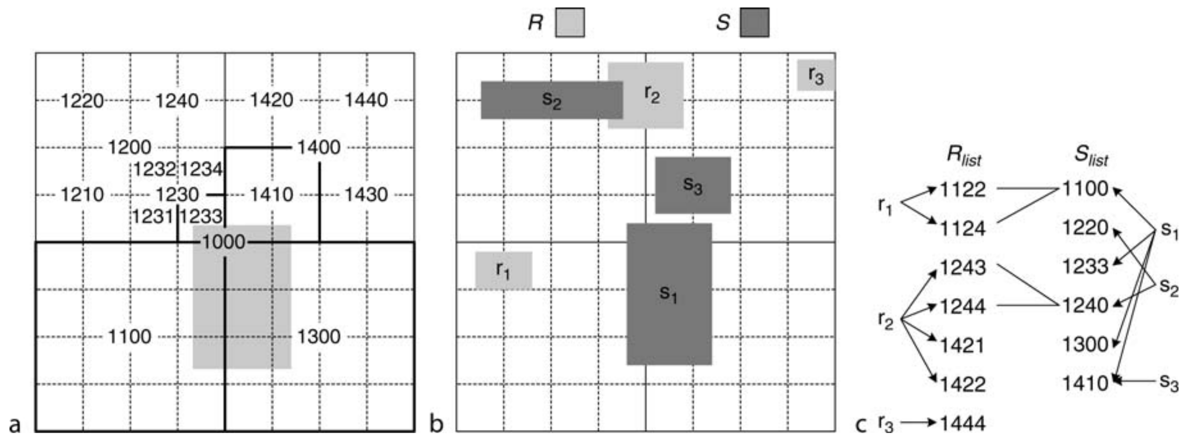


Distributed Spatial Databases. Figure 1. Distributed spatial join based on R-tree approximation.

but S' may include more objects (for instance, d_2 is a false hit). The choice of the appropriate level depends on the average size (in bytes) of the polygons in S.

The second approximation method used by Abel et al. [1] and Tan et al. [14] is similar to the one proposed by Orenstein [13]. The space is recursively divided into cells, and each cell is given a unique base-5 key. The order of the keys follows the *z*-ordering space filling curve. Each object is approximated by up to four cells at various levels of the curve. Figure 2a shows an example; the gray-shaded object is approximated by cells 1100, 1233, 1300 and 1410. The keys of all objects in R_{site} and S_{site} are sorted in ascending order, generating two lists: R_{list} and S_{list}. An example is shown in Fig. 2b and c. Each object has between one (e.g., r_3) and four keys (e.g., r_2); moreover, the keys of an object may not be consecutive in the ordered list (e.g., s_2). R_{site} transmits R_{list} to S_{site}. S_{site} uses *merge-join* to join the two lists. The join condition is cell-containment rather than key equality. For instance, the pair (1122,1100) is generated because cell 1122 is contained in cell 1100. The result may contain duplicates; for example (1122,1100) and (1124,1100) both represent the object pair (r_1,s_1). Duplicates are eliminated and S_{site} sends the semi-join result (together with the polygons of the corresponding S objects) to R_{site}; in the running example, the semi-join contains pairs (r_1, s_1) and (r_2, s_2). Finally, R_{site} eliminates the false hits (i.e., pair (r_1,s_1)) based on the exact object geometry. According to the experimental evaluation in [14], the total cost of the join (in terms of response time) is, in most cases, lower when using the *z*-ordering approximation, compared to the R-tree approach.

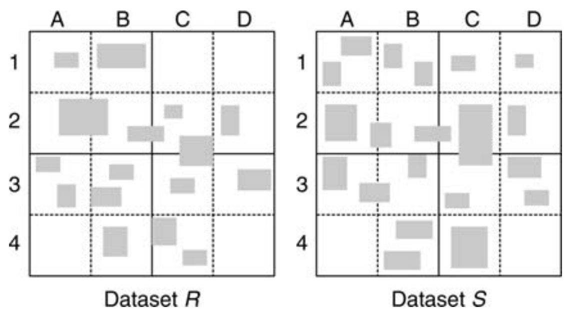
A related problem is studied by Mamoulis et al. [10]. Again, two spatial datasets R and S reside in R_{site} and S_{site}, respectively; however, the two sites do *not* collaborate. As an example, let R be a dataset which stores the location of hotels and other points of interest, whereas S stores information about the physical layers of the region (e.g., rivers, forests, urban areas, etc). Let u be a client with a mobile device (e.g., PDA, smart mobile phone) that asks the query "find all hotels which are at most 2*km* away from a forest." The query is a spatial join. However, neither R_{site} nor S_{site} can evaluate the join, because they do not collaborate. A typical solution is to use a *mediator*. Nevertheless, for ad-hoc queries, it is unlikely that there is a suitable mediator. Therefore, the query must be evaluated by the mobile device. It is assumed that the servers support a simple query interface and can answer *window* (i.e., find all objects in a region) and *count* queries (i.e., how many objects are within a region); the servers do not allow access to their internal indices. Moreover, since telecommunication providers charge by the amount of transferred data, u wants to minimize that quantity and does not consider the query cost at the servers. Figure 3 shows the two datasets. The client partitions conceptually the data by a 2 × 2 grid (i.e., $AB12$,$AB34$,$CD12$,$CD34$) and requests the *number* of objects in each quadrant. Since none of the quadrants is empty, all of them may contain joining pairs. Therefore, the client recursively partitions each quadrant and retrieves the new statistics (i.e., number of objects in each cell). Now, some cells (e.g., $C1$ in R_{site}) are empty; u eliminates these cells, since they cannot contain any solution. For



Distributed Spatial Databases. Figure 2. Distributed spatial join based on *z*-ordering approximation.

the remaining cells u may partition them again and ask for more statistics. However, if the distribution of objects in a cell is roughly uniform, further partitioning is unlikely to eliminate any cells. [3] presents a cost model to estimate when the cost of retrieving refined statistics is more than the potential savings due to pruning. At that point, u performs the spatial join. For each cell, if the number of objects is similar in both datasets, u downloads the objects of both cells and performs hash-based spatial join. On the other hand, if the number of objects differs drastically (e.g., the cell in R has much more objects than the corresponding cell in S), then u downloads only the objects from S and performs nested-loop join by sending a series of window queries to R. Kalnis et al. [5] developed more efficient algorithms for the same problem.

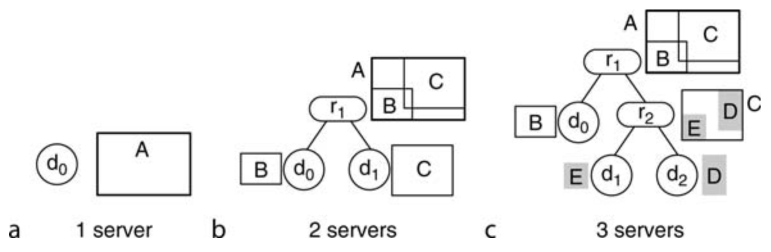
A similar method was used by Liu et al. [9] for evaluating k-nearest-neighbor queries. They assume a client-server architecture (only one server), where the server can execute only window queries. Therefore, the client must estimate the minimum window that contains the result. The authors propose a methodology that either estimates the window progressively, or approximates it using statistics about the data. Statistical information is assumed to be available at the client; hence there is no overhead for retrieving statistics.



Distributed Spatial Databases. Figure 3. MobiHook example.

Distributed Spatial Indices

Several researchers have studied distributed versions of common spatial indices. Litwin and Neimat [8] proposed a distributed version of the kd-tree, called k-RP*S, whereas Karlsson [7] developed hQT*, which is a distributed quad-tree. Both papers focus on point data. Recently, Mouza et al. [12] proposed the SD-Rtree, which is a distributed version of the R-tree capable of indexing regions. SD-Rtree is a binary balanced tree. Initially (Fig. 4a), the entire tree resides in one server S_0. There is a *data* node d_0, which contains all spatial objects and an associated MBR A, which encloses the objects of d_0. d_0 is a conceptual node and typically contains a large number of objects, limited only by the capacity of server S_0. The objects inside d_0 may be indexed by any spatial index (e.g., a local R-tree). During insertion, a split is performed if the capacity of S_0 is reached. A new server S_1 enters the system and receives approximately half of the objects; these are selected using the usual R-tree split algorithms. In Fig. 4b the objects remaining in d_0 are enclosed by MBR B, whereas a new data node d_1 is created in S_1; the corresponding MBR is C. Another node r_1 is created in S_1; r_1 has pointers to d_0 and d_1, together with their MBRs. r_1 is called *routing* node and is similar to the internal nodes of common R-trees. Figure 4c shows one more split of node d_1. A new server S_2 enters the system and creates a data node d_2 and a routing node r_2. The new MBRs of d_1 and d_2 are E and D, respectively. Finally, data is stored in three servers (i.e., S_0 stores d_0, etc) and the routing information is distributed in two servers (i.e., r_1 is in S_1 whereas r_2 is in S_2). Due to splits, the tree may become imbalanced; to achieve balance, node rotations similar to the classical AVL tree are performed.



Distributed Spatial Databases. Figure 4. SD-Rtree example.

The set of all routing nodes is called *image* of the tree. Each client application has a copy of the image. In order to query (or update) the spatial data, the client first searches the local image to identify the server that stores the required data, and contacts that server directly. This is crucial for a distributed structure, since it avoids traversing the tree through the root (i.e., it does not overload the root server). However, a client's image may be outdated. In this case, the wrongly contacted server forwards the query to the correct part of the tree and updates the client's image. Moreover, due to R-tree node overlap, a query may need to traverse multiple paths; some of them are redundant. To avoid this, if two nodes overlap, they are annotated with additional information about their intersection. In Fig. 5a, nodes *A* and *B* store the set of objects in their intersection; in this example the set is empty. The intersection set is updated only if necessary. For instance, the split of node *B* in Fig. 5b does not affect the intersection set. On the other hand, the extension of *D* in Fig. 5c changes the intersection set. If a query *q* arrives at *D*, it can be answered directly by *D* (i.e., without contacting the subtree of *F*), since the intersection set indicates that there is no matching object in *F*.

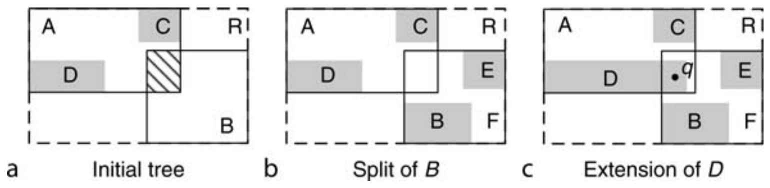
Several researchers have proposed distributed spatial indices on top of Peer-to-Peer (*P2P*) systems. For instance, Mondal et al. [11] proposed a P2P version of the R-tree, whereas Jagadish et al. [4] developed the VBI-tree, a framework for deploying multi-dimensional indices on P2P systems. Also, Kantere and Sellis [6] proposed a spatial index similar to quad-tree, on top of a structured P2P network. These approaches are based on different assumptions for the update and query frequency, and the stability (i.e., mean lifetime) of peers in the network. Therefore, it is not clear which approach is more suitable in practice.

Spatial Queries Involving Numerous Mobile Clients

There exist a variety of spatial distributed systems consisting of numerous mobile clients. Such systems are optimized for the low battery life and the scarce resources (e.g., storage, CPU) of the mobile devices. Gedik and Liu [3] proposed *MobiEyes*, a grid-based distributed system for continuous range queries. MobiEyes pushes part of the computation to the mobile clients, and the server is primarily used as a mediator. The notion of *monitoring regions* of queries was introduced to ensure that objects receive information about the query (e.g., position and velocity). When objects enter or leave the monitoring region, the server is notified. By using monitoring regions, objects only interact with queries that are relevant; hence they conserve precious resources (e.g., storage and computation).

Another architecture is based on the fact that wireless networks are typically broadcast-based. All data are broadcasted periodically and each client listens for its relevant data. In a wireless broadcast environment, an index called *air index* is commonly used to minimize power consumption. A mobile device can utilize the air index to predict the arrival time of the desired data. Therefore, it can reduce power consumption by switching to sleep mode for the time interval that no desired data objects are arriving. The intuition behind the air index is to interleave the index items with the data objects being broadcasted. Zheng et al. [15] proposed two air indexing techniques for spatial data based on the one-dimensional Hilbert Curve and the R-tree, respectively. Using those indices, they discuss how to support Continuous Nearest Neighbor queries in a wireless data broadcast environment.

A related subject is the processing of spatial queries in sensor networks. Such networks are made of a large number of autonomous devices that are able to store, process and share data with neighboring devices. The spatial properties of the sensors (e.g., location, communication range and sensing range) are typically exploited to route queries efficiently. For example, Coman et al. [2] propose a system that answers range queries (e.g., "find the temperature for each point in an area"). The system takes advantage of the fact that the



Distributed Spatial Databases. Figure 5. Example of overlap in SD-Rtree nodes.

sensing ranges of some sensors may completely cover other sensors; consequently, the latter do not need to be contacted.

Key Applications

Nowadays, the trend in databases is the separation of the location of data from the abstract concept of the database itself. Therefore a database may reside in more than one location, but can be queried as a continuous unit. This is made possible due to the increase of network speed. Numerous practical applications can benefit from distributed spatial databases. As an example, consider a spatial database that stores the location of hotels and other points of interest in an area, and a second database which stores information about the physical layers of the region (e.g., rivers, forests, urban areas, etc). Each database is useful by its own. Nevertheless, by combining the two, the value of the data increases, since the distributed system is now able to answer queries such as "find all hotels inside a forest." There are also applications where there is only one spatial dataset, but it is distributed in many servers (or peers). For example, each peer may monitor road congestion in its neighborhood. In order to find routes in a city that avoid congested roads, the spatial data in all peers must be indexed by a distributed data structure.

Cross-references

► Distributed Databases
► Distributed Join
► R-Tree (and Family)
► Spatial Indexing Techniques
► Spatial Join

Recommended Reading

1. Abel D.J., Ooi B.C., Tan K.-L., Power R., and Yu J.X. Spatial Join Strategies in Distributed Spatial DBMS. In Proc. 4th Int. Symp. Advances in Spatial Databases, 1995, pp. 348–367.
2. Coman A., Nascimento M.A., and Sander J. Exploiting Redundancy in Sensor Networks for Energy Efficient Processing of Spatiotemporal Region Queries. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 187–194.
3. Gedik B. and Liu L. MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 67–87.
4. Jagadish H.V., Ooi B.C., Vu Q.H., Zhang R., and Zhou A. VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
5. Kalnis P., Mamoulis N., Bakiras S., and Li X. Ad-hoc Distributed Spatial Joins on Mobile Devices. In Proc. 20th Int. Parallel & Distributed Processing Symp., 2006.
6. Kantere V. and Sellis T.K. A Study for the Parameters of a Distributed Framework That Handles Spatial Areas. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases,. 2007, pp. 385–402.
7. Karlsson J.S. hQT*: A Scalable Distributed Data Structure for High-Performance Spatial Accesses. In Proc. Int. Conf. of Foundations of Data Organization (FODO), 1998, pp. 37–46.
8. Litwin W. and Neimat M.-A. k-RP*S: A Scalable Distributed Data Structure for High-Performance Multi-Attribute Access. In Proc. Int. Conf. on Parallel and Distributed Information Systems (PDIS). 1996, pp. 120–131.
9. Liu D.-Z., Lim E.-P., and Ng W.-K. Efficient k Nearest Neighbor Queries on Remote Spatial Databases Using Range Estimation. In Proc. 14th Int. Conf. on Scientific and Statistical Database Management, 2002, pp. 121–130.
10. Mamoulis N., Kalnis P., Bakiras S., and Li X. Optimization of Spatial Joins on Mobile Devices. In Proc. 8th Int. Symp. Advances in Spatial and Temporal Databases, 2003, pp. 233–251.
11. Mondal A., Lifu Y., and Kitsuregawa M. P2PR-Tree: An R-Tree-Based Spatial Index for Peer-to-Peer Environments. In Proceedings of EDBT Workshops – P2P&DB, 2004, pp. 516–525.
12. du Mouza C., Litwin W., and Rigaux P. SD-Rtree: A Scalable Distributed Rtree. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 296–305.
13. Orenstein J.A. A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 343–352.
14. Tan K.-L., Ooi B.C., and Abel D.J. Exploiting Spatial Indexes for Semijoin-Based Join Processing in Distributed Spatial Databases. IEEE Trans. Knowl. Data Eng., 12(6):920–937, 2000.
15. Zheng B., Lee W.-C., and Lee D.L. Search Continuous Nearest Neighbor on Air. In Proc. Int. Conf. on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous), 2004, pp. 236–245.

Distributed Storage Systems

► Peer-to-Peer Storage

Distributed Transaction Management

Wee Hyong Tok
National University of Singapore, Singapore, Singapore

Synonyms

Transaction management in distributed database systems

Definition

Distributed transaction management deals with the problems of always providing a consistent distributed database in the presence of a large number of transactions (local and global) and failures (communication link and/or site failures). This is accomplished through (i) distributed commit protocols that guarantee atomicity property; (ii) distributed concurrency control techniques to ensure consistency and isolation properties; and (iii) distributed recovery methods to preserve consistency and durability when failures occur.

Historical Background

A transaction is a sequence of actions on a database that forms a basic unit of reliable and consistent computing, and satisfies the ACID property. In a distributed database system (DDBS), transactions may be local or global. In local transactions, the actions access and update data in a single site only, and hence it is straightforward to ensure the ACID property. However, in global transactions, data from more than one site are accessed and updated. A global transaction typically spawns sub transactions at each of the sites in which data are accessed. When multiple global transactions are running, these sub transactions' processing may be interleaved. Thus, distributed transactions must be carefully managed to prevent errors that may corrupt the database. In addition, sites or communication link may fail and result in inconsistent database states across sites.

To ensure that a distributed database is always in a consistent state, the distributed transaction management system must guarantee the ACID property. Transaction atomicity is achieved through distributed commit protocols, e.g., Two-Phase Commit and its variants [8,10,11] and Three-Phase Commit [13]. These protocols allow sub transactions (running at different sites) of a global transaction to reach a final outcome of the execution. In this way, either all sub transactions commit (in which case the global transaction commits) or all sub transactions abort (in which case the global transaction aborts).

To ensure consistency and isolation properties, distributed concurrency control techniques have been designed. In particular, the notion of global serializability has been introduced to ensure that global transactions are serializable. These algorithms determine the management of synchronization primitives (e.g., locks, timestamps, serialization graphs), and the order in which database operations (i.e., read and write) are executed concurrently in a distributed manner. As such, anomalies such as the reading of uncommitted data, unrepeatable reads, and overwriting of uncommitted data can be prevented. Distributed concurrency control algorithms can be classified into lock-based [1,4,11,14,15], timestamp-based [12], and serialization graph testing (SGT) based [2,6,9]. Hybrid methods which use a combination of locks and timestamps are discussed in [5]. For lock-based scheme, (distributed) deadlocks have to be handled. Balter et al. [3] discusses the relationship between deadlock management and concurrency control algorithms.

Finally, when a transaction has committed, its effect must persist in the database regardless of failures. Similarly, if failure happens before a transaction commits, then none of its effects must persist. For example, a power failure that wipes out all main memory content should not nullify the effect of a committed transaction. Similarly, site failures should not affect the distributed commit protocol. To ensure durability and consistency, distributed recovery methods have been developed. These include log-based techniques [8] and checkpointing methods [7] (cross-reference to Distributed Recovery).

The two-phase commit protocol and the distributed 2PL concurrency control algorithm are implemented in the System R* [11] and NonSTOP SQL [15] systems. Distributed INGRES [14] uses the two-phase commit protocol and the primary copy 2PL for concurrency control. Log-based protocols are the main recovery methods employed in commercial systems.

Foundations

A distributed database system consists of a collection of database sites. A centralized database system is located at each of the sites. The database sites communicate with each other by sending messages via a communication network. A transaction consists of a sequence of operations that are performed on the data objects. Two operations, $o_i(x)$ and $o_j(x)$, which are accessing the shared data object x, are conflicting operations if either $o_i(x)$ or $o_j(x)$ is a write operation.

Distributed Transaction Management

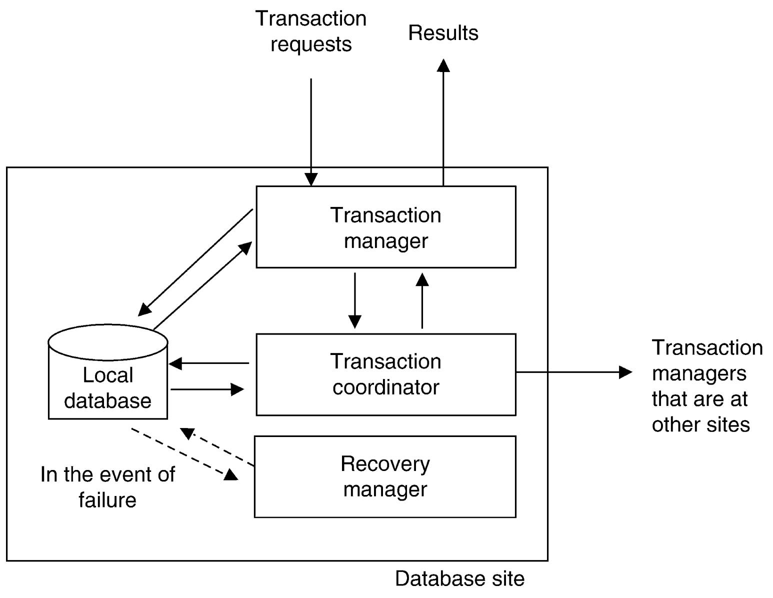
In order to handle both local and global transactions, each site of a distributed database system consists of the following components: transaction manager, transaction coordinator, and recovery manager. The transaction manager handles the execution of transactions at each site, and maintains a log to facilitate recovery. In order to ensure that the ACID properties are guaranteed when multiple transactions are concurrently executed, the transaction manager relies on concurrency control techniques. The transaction coordinator is used to plan and schedule sub transactions that are executed on multiple sites. In addition, the transaction coordinator determines whether the transactions that are executed on multiple sites are committed or aborted. The recovery manager is used to recover the database to a consistent state if failure occurs. Figure 1 shows the various components in the transaction management system at each database site.

Serializability Theory

Serializability theory provides the theoretical foundation for proving the correctness of a concurrency control algorithm, by showing that the overall results of executing a set of concurrent transactions is equivalent to executing the transactions in a serial manner. In serializability theory, this is referred to as a serializable schedule. A serializable schedule ensures the consistency of a database system.

Let T denote a set of concurrently executing transactions. The operations for the various transactions in T can be executed in an interleaved manner. A complete schedule, S, defines the execution order of all operations. A schedule, S', is a prefix of a complete schedule. Formally, $T = \{T_1, T_2, \ldots, T_n\}$, and S is a partial order over T, with ordering relation, $<_S$, where: (i) $S = \bigcup_{i=1}^{n} T_i$, (ii) $<_S \supseteq \bigcup_{i=1}^{n} <_i$, and (iii) for any two conflicting operations, o_i and $o_j \in$ S, either $o_i <_S o_j$ or $o_j <_S o_i$. The first condition states that S consists of the operations which belong to the set of transactions T. The second condition states that the ordering relation is a superset of the ordering relations of each of the transactions. The third condition states that the ordering of every pair of conflicting transaction is given by the ordering relation $<_S$. Two schedules, S_1 and S_2 defined over T are conflict equivalent if for each pair of conflicting operations o_i and o_j (i $\neq$ j), if $o_i <_{S1} o_j$, then $o_i <_{S2} o_j$. A schedule is serializable if and only if it is conflict equivalent to a serial schedule.

In distributed database systems, global serializability is required. For example, consider two transactions, T_1 and T_2, that are initiated at two sites, site 1 and site 2 respectively. Suppose that both transactions access objects x and y. Moreover, suppose that x is stored at site 1, and y is stored at site 2, and that the constraint is that $x + y$ is a constant. Let $W_i(x)$ denote a write action on object x by transaction T_i. Suppose S_1 and S_2 are the local schedules at site 1 and site 2 respectively. Now, consider $S_1 = \{w_1(x), w_2(x)\}$ and $S_2 = \{w_2(y), w_1(y)\}$. Clearly, both S_1 and S_2 are locally serializable. However, while the serial schedule for S_1 is $T_1 \rightarrow T_2$, the serial schedule for S_2 is $T_2 \rightarrow T_1$. These schedules may violate the constraint and hence not globally serializable, if the sequence of actions happens in the following order:



Distributed Transaction Management. Figure 1. Transaction management system at each database site.

$w_1(x)$, $w_2(y)$, $w_2(x)$, $w_1(y)$. However, it has been shown that as long as all the local schedules at each site are serializable with the same serialization order, then the global schedule is serializable. Moreover, the serialization order of the global schedule corresponds to that of the local order. As an example, if $S_1 = \{w_1(x), w_2(x)\}$ and $S_2 = \{w_1(y), w_2(y)\}$, then the global schedule is serializable with $T_1 \rightarrow T_2$.

Concurrency control algorithms are used to ensure that the global schedule is serializable. For replicated distributed databases, more issues need to be considered before serializability theory can be applied. This is because even if the local schedules are serializable, the mutual consistency of the database needs to be considered. For replicated, distributed databases, a one-copy serializable global schedule is desired in order to ensure the mutual consistency of the replicated data. In addition to concurrency control, a replica control protocol (e.g. ROWA) is used to ensure that one-copy serializability can be achieved.

Key Applications

Distributed transaction management forms an integral part of distributed database systems. The concurrency control techniques, commit protocols, and recovery techniques can be applied and adapted for different types of distributed database systems.

Future Directions

The emergence of new computing platforms (e.g. Peer-to-Peer (P2P), and cloud computing) introduces new issues that need to be handled by distributed transaction manager. P2P database systems are inherently distributed systems, and have been studied extensively by the database community. In P2P systems, the absence of a global transaction manager introduces new challenges. In addition, nodes join or leave the P2P network. Transactions are often executed independently on each peer. The maintenance of data consistency in P2P database systems motivates the need for P2P-based concurrency control algorithms. For example, it may be necessary to maintain multiple versions of data objects and/or a weaker notion of serializability than the conventional notion of global serializability. Recovery is also complicated by the fact that a failed site may not find the operational sites (participating in its transactions) available when it recovers (these sites may have left the network). Thus, novel and online recovery mechanisms are needed. Cloud computing is an emerging computing platform for next-generation applications. It provides basic services such as storage, queuing, and computation. Hence, a data store can be easily built using these basic services. The storage services provided by the cloud can be perceived as a very large shared disk. Different applications which use the cloud will require different levels of concurrency control. The need to support concurrent access on the shared disk and different level of concurrency control motivates the need for concurrency control to be offered on a *à la carte* basis. In addition, recovery techniques for cloud computing have not been adequately studied in the literature.

Cross-references

► ACID Properties
► Distributed Architecture
► Distributed Concurrency Control
► Distributed Deadlock Management
► Distributed Recovery
► Three-Phase Commit
► Two-Phase Commit

Recommended Reading

1. Alsberg P. and Day J.D. A principle for resilient sharing of distributed resources. In Proc. 2nd Int. Conf. on Software Eng., 1976, pp. 562–570.
2. Badal D.Z. Correctness of concurrency control and implications for distributed databases. In Proc. Computer Software and Applications Conference (COMPSAC). Chicago, IL, 1979, pp. 588–594.
3. Balter R., Berard P., and Decitre P. Why control of the concurrency level in distributed systems is more fundamental than deadlock management. In Proc. ACM SIGACT-SIGOPS 1st Symp. on the Principles of Dist. Comp., 1982, pp. 183–193.
4. Bernstein P.A. and Goodman N. Concurrency control in distributed database systems. ACM Comput. Surv., 13 (2): 185–221, 1981.
5. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Boston, MA, 1987.
6. Casanova M.A. The concurrency control problem for database systems. Lecture Notes in Computer Science, Springer Berlin 1981.
7. Chrysanthis P.K., Samaras G., and Al-Houmaily Y.J. Recovery and performance of atomic commit processing in distributed database systems, Chapter 13. In Recovery Mechanisms in Database Systems, V. Kumar, M. Hsu Prentice-Hall, Upper Saddle River, NJ, 1998.
8. Gray J. Notes on data base operating systems. In Operating Systems – An Advanced Course, R. Bayer, R. Graham, G. Seegmuller (eds.). Lecture Notes in Computer Science, vol. 60, Springer, Berlin, 1978, pp. 393–481.

9. Hadzilacos T. and Yannakakis M. Deleting completed transactions. In Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1986, pp. 43–46.
10. Lampson B. and Sturgis H. Crash recovery in a distributed data storage system. Technical report, Computer Science Laboratory, Xerox Palo Alto Research Center, CA, 1976.
11. Mohan C., Lindsay B.G., and Obermarck R. Transaction management in the r* distributed database management system. ACM Trans. Database Syst., 11(4):378–396, 1986.
12. Shapiro R. and Millstein R. Reliability and fault recovery in distributed processing. OCEANS, 9:425–429, 1977.
13. Skeen D. Non-blocking commit protocols. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 133–142.
14. Stonebraker M. and Neuhold E.J. A distributed database version of ingres. In Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, 1977, pp. 19–36.
15. The Tandem Performance Group. Tandem database group – nonstop sql: a distributed, high-performance, high-availability implementation of sql. In Proc. Second Int. Workshop on High Performance Transaction Systems (HPTS), 1987, pp. 60–104.

Divergence Control

► Replica Freshness

Divergence from Randomness Models

ANABAPTISTS AMAIN
GIAMBATTISTA AMATI Ugo Bordon Foundation, Rome, Italy

Synonyms

Deviation from randomness

Definition

Divergence From Randomness (DFR) Information Retrieval models are term-document matching functions that are obtained by the product of two divergence functions. An example of DFR function is that related to Jensen's information of two probability distributions [9, pp. 26–28]:

$$\sum_i I_1(\hat{p}_i^+||\hat{p}_i).\, I_2(\hat{p}_i^+||\hat{p}_i)$$

where $I_1(\hat{p}_i^+||\hat{p}_i) = \hat{p}_i^+ - \hat{p}_i = \Delta\hat{p}_i$ and $I_2(\hat{p}_i^+||\hat{p}_i) = \log_2 \frac{\hat{p}_i+\Delta\hat{p}_i}{\hat{p}_i}$

The DFR generalizes the Jensen's information as follows:

$$\sum_i I_1(\hat{p}_i^+||\hat{p}_i).\, I_2(\hat{p}_i^+||p_i)$$

where

- p is a prior probability density function of terms (or documents) in the collection.
- $\hat{p}$ is the frequency of the term in a document (or in a subset of documents).
- $\hat{p}^+$ is the neighboring frequency of the term in a document (or in a subset of documents).
- $I_1(\hat{p}^+||\hat{p}_i) = \sum_i I_1(\hat{p}_i^+||\hat{p}_i) = 0$ if and only if $\hat{p}^+ = \hat{p}$.
- $I_2(\hat{p}||p) = \sum_i I_2(\hat{p}||p)$ is minimum when $\hat{p} = p$.

In a DFR model a term occurs randomly when $\hat{p} = p$, whereas a term is informative when $\hat{p} \gg p$.

Historical Background

Divergence From Randomness models were inspired by Harter's 2-Poisson model. Harter's model assumes that a word randomly distributedaccording to a Poisson distribution is not informative, whereas a word that does not follow a Poisson distribution indicates that it conveys information [7]. The Okapi retrieval function, BM25, was also inspired by Harter's 2-Poisson model, and indeed it can be derived from a DFR model [3].

Foundations

The Divergence From Randomness models have their roots in information theory. Following Shannon's theory of information a document can be seen as *a message to transmit*, where information is measured by the cost of transmission. For example, if a message $m(k)$ of length k is generated by a set $\mathbf{V}$ of n symbols $\mathbf{t}_i$, each symbol occurring with a frequency (*prior probability*) p_i, with $0 \leq i \leq n$, then the information is:

$$I_2 = -\log_2 p(m(k)) = -\log_2 \prod_{i=1}^{n} p_i^{k.pi}$$
$$= -k \sum_{i=1}^{n} .\ p_i \log_2 p_i$$

The entropy H of a the system generated by $\mathbf{V}$ is defined as the average information transmitted by its symbols

$$H = \frac{-\log_2 p(m(k))}{k} = -\sum_{0 \leq i \leq n} p_i \log_2 p_i$$

The average information transmitted by an arbitrary message $m(k)$ of length k is approximated by

$$I_2(k) = -\log_2 p(m(k)) \sim k.H \tag{1}$$

A collection **D** of **N** documents can be also conceived as a set of messages that are generated by a vocabulary **V** of n words $\mathbf{t}_i$. There are two types of DFR models:

1. *The document model.* Each document **d** of length k generates a partition of its terms over n cells (terms). The partition satisfies the constraint $\sum_{i=1}^{n} \mathbf{tf}_i = k$, where $\mathbf{tf}_i$ is the term frequency of the i-th term in the document, and each term $\mathbf{t}_i$ has a prior probability p_i of occurrence in an arbitrary document of the collection.
2. *The term model.* All occurrences of a term **t** in a collection **D** of **N** documents generates a partition over **N** cells (documents). Each document has a prior probability p_i of being selected. The partition satisfies the constraint $\sum_{i=1}^{\mathrm{N}} \mathbf{tf}_i = \mathbf{TF}$, where $\mathbf{tf}_i$ is the term frequency in the i-th document and **TF** is the term frequency in the collection.

Document Models

Equation (1) provides the mean information carried by an arbitrary document **d** of length k with expected term frequencies $k \cdot p_i$. However, if the term frequency X_i in the observed document is $\mathbf{tf}_i$ and not $k \cdot p_i$, then the information of the document is given by the multinomial distribution:

$$p\left(\mathrm{X}_1 = \mathbf{tf}_1, \ldots, X_n = \mathbf{tf}_n\right) = \binom{k}{\mathbf{tf_1 tf}_2 \ldots \mathrm{tf}_\mathrm{n}} p_1^{\mathbf{tf}_1} p_2^{\mathbf{tf}_2} \cdots p_\mathbf{n}^{\mathbf{tf}_\mathrm{n}}$$

Here, the term independence is assumed, according to which a document is treated as an ensemble and is said to be a *bag of words*. Setting $\hat{p}_i$ to the term frequency in the document $\frac{\mathbf{tf}_\mathrm{i}}{k}$, it can be shown that [9, Chap. 1, Ex. 5.12] that:

$$\lim_{k \to \infty} \frac{I_2(\hat{p}||p)}{k} = D(\hat{p}||p) \tag{2}$$

where $I_2(\hat{p}||p) = -\log_2 p(X_1 = \mathbf{tf}_1, \ldots, X_n = \mathbf{tf}_n)$ and $D(\hat{p}||p) \sim \sum_{\mathrm{t}_i \in \, V} \hat{p}_i \log_2 \frac{\hat{p}_i}{p_i}$ is the *Kullback-Leibler divergence.*

The prior probability p_i can be set to $\frac{\mathbf{n}_\mathrm{i}}{\mathbf{N}}$, the ratio of the number $\mathbf{n}_\mathrm{i}$ of documents in which the term occurs and the number **N** of documents of the collection (The contribution of a term not occurring in a document is null, being $\mathbf{tf} \cdot \log_2 \mathbf{tf} \to 0$ for $\mathbf{tf} \to 0$.):

$$I_2(\hat{p}||p) \sim k.D(\hat{p}||p) = k. \sum_{\mathbf{t}_i \in \mathbf{v}} \hat{p}_i . \log_2 \frac{\hat{p}_i}{p_i} \sim \sum_{\mathbf{t}_i \in \mathbf{v}} \mathbf{tf}_i \log_2 \frac{\mathbf{tf}_i . \mathbf{N}}{k . \mathbf{n}_\mathrm{i}} \tag{3}$$

The approximation of (3) is additive on terms, and additivity is a useful property to extract the

Divergence from Randomness Models. Table 1. The notations

Notation		Document model	Term model
tf	The term frequency in the document		
$X_i = \mathbf{tf}_i$	The frequency of the i-th term in the observed document		
$Y_j = \mathbf{tf}_j$	The frequency of the observed term in the jTtH document		
X	The vector of the X_i random variables		
Y	The vector of the Y_i random variables		
TF	The term frequency in the collection		
$\mathbf{n_t}$	The document frequency of the term in the collection		
N	The number of documents		
k	The document length		
$\hat{p}$	The estimate of probability density from observations (likelihood)	$\frac{\mathbf{tf}_i}{k}$	$\frac{\mathbf{tf}_i}{\mathbf{TF}}$
$\hat{p}^+$	The neighboring value of the probability density from observations	$\frac{\mathbf{tf}_i+1}{k+1}$	$\frac{\mathbf{tf}_i+1}{\mathbf{TF+1}}$
p	The prior probability density	$\frac{\mathbf{TF}_i}{\sum_i^n \mathbf{TF}_i}$ or $\frac{\mathbf{n_t}}{\mathbf{N}}$	$\frac{1}{\mathbf{N}}$
$D(\hat{p}\|\|p)$	The Kullback-Leibler divergence of $\hat{p}$ from p		

contribution of single terms to information. For example, given the query **q** the contribution of the query terms to the document is:

$$I_2(\hat{p}||p) = -\log_2 p(X_1 = \mathbf{tf_1}, ..., X_n = \mathbf{tf}_n|\mathbf{q}) = \sum_{\mathbf{t}_i \in \mathbf{q}} \mathbf{tf}_i \log_2 \frac{\mathbf{tf}_i.\mathbf{N}}{k.\mathbf{n}_\mathrm{i}} \quad (4)$$

Term Models

DFR term models are developed similarly as the DFR document models. The analogous formula of (3) is

$$\begin{aligned} I_2(\hat{p}||p) &= -\log_2(\mathrm{p}(\mathrm{Y}_1 = \mathbf{tf}_1, \ldots, \mathrm{Y}_\mathrm{n} = \mathbf{tf}_\mathrm{N})) \\ &\sim \mathbf{TF} \cdot D(\hat{p}||p) = \mathbf{TF} \cdot \sum_{i=1}^{\mathbf{N}} \hat{p}_i \cdot \log_2 \frac{\hat{p}_i}{p_i} \quad (5) \\ &= \sum_{i=1}^{\mathbf{N}} \mathbf{tf}_i \log_2 \frac{\mathbf{tf}_\mathrm{i} \cdot \mathbf{N}}{\mathbf{TF}} \end{aligned}$$

where $\hat{p}_i = \frac{\mathbf{tf}_i}{\mathbf{TF}}$ and $p_i = \frac{1}{\mathbf{N}}$ is the prior probability of a document.

Document Length Normalization

Similarly to TF-IDF used in the vector space model, also Formula 4 can be used for retrieval (see *vector space model*). However, information (see (3)) grows approximately linearly with the document length. If Formula 4 were used as retrieval function, then long documents would be preferred to short ones, while retrieval evaluation has shown that users prefer more distilled information. Formula 4 needs to be normalized with respect to the document length. Length normalization is a critical issue in information retrieval models: for example the vector space model normalizes weights by dividing them by the norms of the query and documents vectors, whilst language models normalizes by learning the value of a parameter that combines the two term frequencies p and $\hat{p}$.

It is indeed possible to average the information-$\log(p(X_1 = \mathbf{tf}_1, ..., X_n = \mathbf{tf}_n))$ by the length of the document, obtaining the Kullback-Leibler divergence, but in such a case shorter documents would be preferred to longer ones, because $\hat{p}_i$ is higher in very short documents than in long documents. A way to normalize the information carried by the term is to compute the *information gain*, which corresponds to the increase of information provided by the addition of one or more occurrences of the term, for example when the term frequency increases from $\hat{p}_i = \frac{\mathbf{tf}}{k}$ to its neighboring point $\hat{p}_i^+ = \frac{\mathbf{tf}+1}{k+1}$. The increment rate of information is

$$\sum_i I_1(\hat{p}_i^+||\hat{p}_i).I_2(\hat{p}_i||p_i) \sim \sum_{\mathbf{t}_i \in \mathbf{q}} \left(1 - \frac{\hat{p}_i}{\hat{p}_i^+}\right) k.\hat{p}_i. \log_2 \frac{\hat{p}_i}{p_i} \quad (6)$$

where $I_1(\hat{p}_i^+||\hat{p}_i) = 1 - \frac{\hat{p}_i}{\hat{p}_i^+}$.

The matching Formula 6 considers only the contribution of the query terms and exploits the additivity property of information over independent terms.

Examples of Document Models

The following two models are examples of DFR models generated by (6):

- *Example of a DFR document model.* Let $\hat{p}_i = \frac{\mathbf{tf}_i}{k}$, $\hat{p}_i^+ = \frac{\mathbf{tf}_i+1}{k+1}$. Equation (6) becomes

$$\sum_{\mathbf{t}_i \in \mathbf{q}} \frac{1 - \hat{p}_i}{k \cdot \hat{p}_i + 1} (-\log_2 p(x_1 = \mathbf{tf}_1, ..., x_n = \mathbf{tf}_n|\mathbf{q})) \sim k \cdot \sum_{\mathbf{t}_i \in \mathbf{q}} \frac{1 - \hat{p}_i}{k \cdot \hat{p}_i + 1} \cdot \hat{p}_i \cdot \log_2 \frac{\hat{p}_i}{p_i}$$

An approximation of $-\log_2 p(X_1 = \mathbf{tf}_1, \ldots, X_n = \mathbf{tf}_\mathrm{n}|\mathbf{q})$, different from the Kullback-Leibler divergence can be obtained exploiting the Stirling formula that is used for the computation of the factorials [1]. This is a parameter free model of IR and an analogous formula is implemented in the Terrier search engine [10] (The increment rate is computed by the χ-square divergence between the two neighboring probabilities $\hat{p}_i$ and $\hat{p}_i^+$.).

- *Example of a DFR term model.* Let $\hat{p}_i = B(\mathbf{tf}_i, \mathbf{TF}, \frac{\mathbf{n_t}}{\mathbf{N}})$, $\hat{p}_i^+ = B\big(\mathbf{tf}_i + 1, \mathbf{TF} + 1, \frac{\mathbf{n_t}}{\mathbf{N}}\big)$ where

$$B(\mathbf{tf}_i, \mathbf{TF}, p_i) = \binom{\mathbf{TF}}{\mathbf{tf}_i} \left(\frac{\mathbf{n}_\mathrm{t}}{\mathbf{N}}\right)^{\mathbf{tf}_i} \left(1 - \frac{\mathbf{n_t}}{\mathbf{N}}\right)^{\mathbf{TF}-\mathbf{tf}_i}$$

Different from document model, term models do not include the document length among its observable random variables. Therefore, the term frequency **tf** is not normalized with respect to the length of the document. The following term frequency normalization was shown to be very effective

$$\mathbf{tfn} = \mathbf{tf} \cdot \ln\left(1 + \frac{c \cdot \bar{k}}{k}\right)$$

where $\bar{k}$ is the average document length and c is a parameter [8]. Equation (6) becomes

$$\sum_{\mathbf{t}_i \in q} \frac{\mathbf{TF} + 1}{\mathbf{n_t} \cdot (\mathbf{tfn}_i + 1)} (-\log_2 p(Y_1 = \mathbf{tf}_1, ..., Y_\mathrm{N} = \mathbf{tf}_\mathrm{N}, Z = k|\mathbf{q})) \sim \sum_{\mathbf{t}_i \in \mathbf{q}} \frac{\mathbf{TF} + 1}{\mathbf{n}_t \cdot (\mathbf{tfn}_i + 1)} \cdot \mathbf{tfn}_i \cdot \log_2 \frac{\mathbf{tfn}_i \cdot \mathbf{N}}{\mathbf{TF}}$$

As for the previous document model, the Stripling formula can be used for the computation of the factorials to obtain a better approximation of $-\log_2 p(Y_1 = \mathbf{tf}_1, \ldots, Y_\mathbf{N} = \mathbf{tf}_\mathbf{N}, Z = k|\mathbf{q})$ [3]. Other DFR models can be built by varying the way both information and the information gain are defined [3].

Key Applications

DFR models can be also applied to query expansion, and to predict query performance [2].

Future Directions

The notion of information gain of the DFR models are strictly connected to the theory of causation and of aftereffect in future sampling (word burstiness) [4,5] [6, pp. 399–402]. The notion of neighboring points used in information gain is related to Fisher's information [9, pp. 26–28]. A deeper analysis of the relation of information gain to these concepts can lead to the discover of more performing models of IR.

URL to Code

DFR models are implemented in the search engine Terrier http://ir.dcs.gla.ac.uk/terrier/.

Cross-references

► 2-Poisson Model
► BM25
► Length Normalization
► Query Expansion Models

Recommended Reading

1. Amati G. Frequentist and Bayesian approach to Information Retrieval. In Proc. 28th European Conf. on IR Research (ECIR 2006). Volume 3936 of Lecture Notes in Computer Science, Springer, 2005, pp. 13–24.
2. Amati G., Carpineto C., and Romano G. Query difficulty, robustness, and selective application of query expansion. In S. McDonald and J. Tait (eds.). ECIR, volume 2997 of Lecture Notes in Computer Science, Springer, 2004, pp. 127–137.
3. Amati G. and Van Rijsbergen C.J. Probabilistic models of information retrieval based on measuring the divergence from randomness. ACM Trans. Inform. Syst. (TOIS), 20(4):357–389, 2002.
4. Gärdenfors P. Knowledge in Flux. MIT, 1988.
5. Gaussier E. and Clinchant S. The BNB distribution for text modeling. In ECIR, Lecture Notes in Computer Science. Springer, 2008.
6. Good I.J. A casual calculus i. Br. J. Phil. Sci., 11:305–318, 1961.
7. Harter S.P. A probabilistic approach to automatic keyword indexing. PhD thesis, Graduate Library, The University of Chicago, Thesis No. T25146, 1974.
8. He I. and Ounis B. On setting the hyper-parameters of the term frequency normalisation for information retrieval. ACM Trans. Inform. Syst., 2007.
9. Kullback S. Information Theory and Statistics. Wiley, New York, 1959.
10. Ounis I., Amati G., Plachouras V., He B., Macdonald C., and Johnson D. Terrier Information Retrieval Platform. In Proc. 27th European Conf. on IR Research (ECIR 2005), volume 3408 of Lecture Notes in Computer Science, Springer, 2005, pp. 517–519.

DNA Sequences

► Biological Sequence

Document

ETHAN V. MUNSON
University of Wisconsin-Milwaukee, Milwaukee, WI, USA

Definition

A document is a representation of information designed for consumption by people. A document may contain information in any medium or in multiple media, though text is generally the dominant medium. A document may be persistent and suitable for archival uses or it may be ephemeral, lasting only for one viewing.

Key Points

Dictionary definitions of the concept of a document generally emphasize documents that have a physical form. Even when computers are considered, the definitions assume that a document is a file on a storage device.

Modern computing, thanks especially to the Web, has greatly expanded the scope of the term. Computers make documents more diverse because computers allow users to create, store and manage material from a variety of media using similar metaphors and interfaces. Computers may also create documents on the fly from fragments stored in a database, as is common in e-commerce and other Web applications. These documents may only exist long enough to be transmitted from a server to a Web browser for a single viewing by a single user.

Cross-references

► Document Representations

Document Clustering

Ying Zhao[1], George Karypis[2]
[1]Tsinghua University, Beijing, China
[2]University of Minnesota, Minneapolis, MN, USA

Synonyms

Text clustering; High-dimensional clustering; Unsupervised learning on document datasets

Definition

At a high-level the problem of document clustering is defined as follows. Given a set S of n documents, we would like to partition them into a pre-determined number of k subsets S_1, S_2,...,S_k, such that the documents assigned to each subset are more similar to each other than the documents assigned to different subsets. Document clustering is an essential part of text mining and has many applications in information retrieval and knowledge management. Document clustering faces two big challenges: the dimensionality of the feature space tends to be high (i.e., a document collection often consists of thousands or tens of thousands unique words); the size of a document collection tends to be large.

Historical Background

Fast and high-quality document clustering algorithms play an important role in providing intuitive navigation and browsing mechanisms as well as in facilitating knowledge management. The tremendous growth in the volume of text documents available on the Internet, digital libraries, news sources, and company-wide intranets has led an increased interest in developing methods that can help users to effectively navigate, summarize, and organize this information with the ultimate goal of helping them to find what they are looking for. Fast and high-quality document clustering algorithms play an important role towards this goal as they provide both an intuitive navigation/browsing mechanism by organizing large amounts of information into a small number of meaningful clusters as well as to greatly improve the retrieval performance either via cluster-driven dimensionality reduction, term-weighting, or query expansion.

Foundations

Figure 1 shows the commonly used three-step process of transferring a document collection into clustering results that are of value to a user. Original documents are often plain text files, html files, xml files, or a mixture of them. However, most clustering algorithms cannot operate on such textual files directly. Hence, a *document representation* is needed to transform the original documents into the data model on which clustering algorithms can operate. Depending on the characteristics of the document collection and the application requirements, the actual clustering process can be performed using various types of clustering algorithms including partitional clustering, agglomerative clustering, model-based clustering, etc. Finally, the quality of the clustering results need to be properly assessed and presented to the users. Details on some of the most commonly used methods in this three-step process follows.

Document Representation

Documents are often represented using the *term frequency-inverse document frequency* (tf-idf) vector-space model [12]. In this model, each document d is considered to be a vector in the term-space and is represented by the vector

$$d_{tfidf} = (tf_1 \log(n/df_1), tf_2 \log(n/df_2), \ldots, tf_m \log(n/df_m)),$$

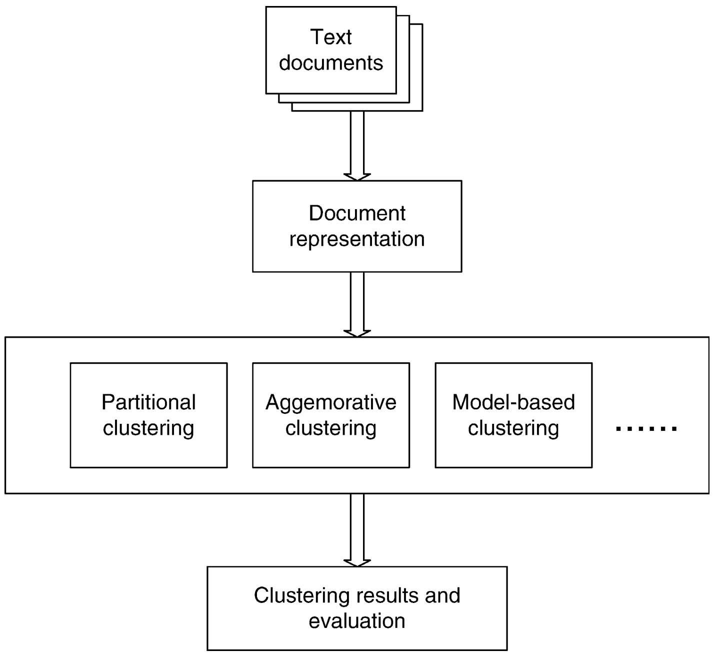
where tf_i is the frequency of the ith term (i.e., term frequency), n is the total number of documents, and df_i is the number of documents that contain the ith term (i.e., document frequency). To account for documents of different lengths, the length of each document vector is normalized so that it is of unit length.

Similarity Measures

Two prominent ways have been proposed to measure the similarity between two documents d_i and d_j when represented via their tf-idf representation. The first method is based on the commonly used [12] cosine function

$$\cos(d_i, d_j) = d_i^t d_j / (||d_i||\ ||d_j||),$$

and since the document vectors are of unit length, it simplifies to $d_i^t d_j$. The second method computes the similarity between the documents using the Euclidean distance $\text{dis}(d_i, d_j) = ||d_i - d_j||$. Note that besides the fact that one measures similarity and the other measures distance, these measures are quite similar to each other because the document vectors are of unit length.



Document Clustering. Figure 1. Structure of document clustering learning system.

Partitional Document Clustering

Partitional algorithms, such as *K*-means [11], *K*-medoids [8], probabilistic [3], graph-partitioning-based [14], or spectral based [1], find the clusters by partitioning the entire dataset into either a predetermined or an automatically derived number of clusters. A key characteristic of many partitional clustering algorithms is that they use a global criterion function whose optimization drives the entire clustering process. For some of these algorithms the criterion function is implicit (e.g., PDDP [1]), whereas for other algorithms (e.g., *K*-means [11]) the criterion function is explicit and can be easily stated. This latter class of algorithms can be thought of as consisting of two key components. First is the criterion function that the clustering solution optimizes, and second is the actual algorithm that achieves this optimization.

Criterion Function Criterion functions used in the partitional clustering reflect the underlying definition of the "goodness" of clusters. The partitional clustering can be considered as an optimization procedure that tries to create high quality clusters according to a particular criterion function. Many criterion functions have been proposed and analyzed [8,6,16]. Table 1 lists a total of seven different clustering criterion functions. These functions optimize various aspects of intra-cluster similarity, inter-cluster dissimilarity, and their combinations, and represent some of the most widely used criterion functions for document clustering. These criterion functions utilize different views of the underlying collection, by either modeling the objects as vectors in a high-dimensional space, or by modeling the collection as a graph.

The $\mathcal{I}_1$ criterion function (1) maximizes the sum of the average pairwise similarities (as measured by the cosine function) between the documents assigned to each cluster weighted according to the size of each cluster. The $\mathcal{I}_2$ criterion function (2) is used by the popular vector-space variant of the *K*-means algorithm [2]. In this algorithm each cluster is represented by its centroid vector and the goal is to find the solution that maximizes the similarity between each document and the centroid of the cluster that is assigned to. Comparing $\mathcal{I}_1$ and $\mathcal{I}_2$ we see that the essential difference between them is that $\mathcal{I}_2$ scales the within-cluster similarity by the $||D_r||$ term as opposed to the n_r term used by $\mathcal{I}_1$. $||D_r||$ is the square-root of the pairwise similarity between all the document in S_r and will tend to emphasize clusters whose documents have smaller pairwise similarities compared to clusters with higher pairwise similarities.

The $\mathcal{E}_1$ criterion function (3) computes the clustering by finding a solution that separates the documents

Document Clustering. Table 1. The mathematical definition of various clustering criterion functions

Criterion function	Optimization function
$\mathcal{I}_1$	(1) maximize $\sum_{i=1}^{k} \frac{1}{n_i}\left(\sum_{v,u\in S_i} \mathrm{sim}(v,u)\right)$
$\mathcal{I}_2$	(2) maximize $\sum_{i=1}^{k} \sqrt{\sum_{v,u\in S_i} \mathrm{sim}(v,u)}$
$\mathcal{E}_1$	(3) minimize $\sum_{i=1}^{k} n_i \frac{\sum_{v\in S_i,u\in S} \mathrm{sim}(v,u)}{\sqrt{\sum_{v,u\in S_i} \mathrm{sim}(v,u)}}$
$\mathcal{G}_1$	(4) minimize $\sum_{i=1}^{k} \frac{\sum_{v\in S_i,u\in S} \mathrm{sim}(v,u)}{\sum_{v,u\in S_i} \mathrm{sim}(v,u)}$
$\mathcal{G}_2$	(5) minimize $\sum_{r=1}^{k} \frac{\mathrm{cut}(V_r, V-V_r)}{W(V_r)}$
$\mathcal{H}_1$	(6) maximize $\frac{\mathcal{I}_1}{\mathcal{E}_1}$
$\mathcal{H}_2$	(7) maximize $\frac{\mathcal{I}_2}{\mathcal{E}_1}$

The notation in these equations are as follows: k is the total number of clusters, S is the total objects to be clustered, S_i is the set of objects assigned to the ith cluster, n_i is the number of objects in the ith cluster, v and u represent two objects, and sim (v,u) is the similarity between two objects.

of each cluster from the entire collection. Specifically, it tries to minimize the cosine between the centroid vector of each cluster and the centroid vector of the entire collection. The contribution of each cluster is weighted proportionally to its size so that larger clusters will be weighted higher in the overall clustering solution. $\mathcal{E}_1$ was motivated by multiple discriminant analysis and is similar to minimizing the trace of the between-cluster scatter matrix [6].

The $\mathcal{H}_1$ and $\mathcal{H}_2$ criterion functions (6) and (7) are obtained by combining criterion $\mathcal{I}_1$ with $\mathcal{E}_1$, and $\mathcal{I}_2$ with $\mathcal{E}_1$, respectively. Since $\mathcal{E}_1$ is minimized, both $\mathcal{H}_1$ and $\mathcal{H}_2$ need to be maximized as they are inversely related to $\mathcal{E}_1$.

The criterion functions that we described so far, view each document as a multidimensional vector. An alternate way of modeling the relations between documents is to use graphs. Two types of graphs are commonly used in the context of clustering. The first corresponds to the document-to-document similarity graph G_s and the second to the document-to-term bipartite graph G_b [15,4]. G_s is obtained by treating the pairwise similarity matrix of the dataset as the adjacency matrix of G_s, whereas G_b is obtained by viewing the documents and the terms as the two sets of vertices (V_d and V_t) of a bipartite graph. In this bipartite graph, if the ith document contains the jth term, then there is an edge connecting the corresponding ith vertex of V_d to the jth vertex of V_t. The weights of these edges are set using the *tf-idf* model.

Viewing the documents in this fashion, edge-cut-based criterion functions can be used to cluster document datasets. $\mathcal{G}_1$ and $\mathcal{G}_2$ (4) and (5) are two such criterion functions that are defined on the similarity and bipartite graphs, respectively. The $\mathcal{G}_1$ function [5] views the clustering process as that of partitioning the documents into groups that minimize the edge-cut of each partition. However, because this edge-cut-based criterion function may have trivial solutions the edge-cut of each cluster is scaled by the sum of the cluster's internal edges [5]. Note that cut(S_r, $S - S_r$) in (4) is the edge-cut between the vertices in S_r and the rest of the vertices $S - S_r$, and can be re-written as $D_r^t(D - D_r)$ since the similarity between documents is measured using the cosine function. The $\mathcal{G}_2$ criterion function [15,4] views the clustering problem as a simultaneous partitioning of the documents and the terms so that it minimizes the normalized edge-cut of the partitioning. Note that V_r is the set of vertices assigned to the rth cluster and $W(V_r)$ is the sum of the weights of the adjacency lists of the vertices assigned to the rth cluster.

Optimization Method There are many techniques that can be used to optimize the criterion functions described above. They include relatively simple greedy schemes, iterative schemes with varying degree of hill-climbing capabilities, and powerful but computationally expensive spectral-based optimizers [11,1,15,4,7]. Here is a simple yet very powerful greedy strategy that has been shown to produce comparable results to those produced by more sophisticated optimization algorithms. In this greedy straggly, a k-way clustering of a set of documents can be computed either directly or via a sequence of repeated bisections. A direct k-way clustering is computed as follows. Initially, a set of k objects is selected from the datasets to act as the *seeds* of the k clusters. Then, for each object, its similarity to these k seeds is computed, and it is assigned to the cluster corresponding to its most similar seed. This forms the initial k-way clustering. This clustering is then repeatedly refined so that it optimizes a desired clustering criterion function. A k-way partitioning via repeated bisections is obtained by recursively applying the above algorithm to compute two-way clustering (i.e., bisections). Initially, the objects are partitioned into two clusters, then one of these clusters is selected and is further bisected, and so on. This process continues $k - 1$ times, leading to k clusters. Each of these bisections is performed so that the resulting two-way clustering solution optimizes a particular criterion function.

Agglomerative Document Clustering

Hierarchical agglomerative algorithms find the clusters by initially assigning each object to its own cluster and then repeatedly merging pairs of clusters until a certain stopping criterion is met. Consider an n-object dataset and the clustering solution that has been computed after performing l merging steps. This solution will contain exactly $n - l$ clusters, as each merging step reduces the number of clusters by one. Now, given this $(n - l)$-way clustering solution, the pair of clusters that is selected to be merged next, is the one that leads to an $(n - l - 1)$-way solution that optimizes a particular criterion function. That is, each one of the $(n - l) \times (n - l - 1)/$two pairs of possible merges is evaluated, and the one that leads to a clustering solution that has the maximum (or minimum) value of the particular criterion function is selected. Thus, the criterion function is *locally* optimized within each particular stage of agglomerative algorithms. Depending on the desired solution, this process continues until either there are only k clusters left, or when the entire agglomerative tree has been obtained.

The three basic criteria to determine which pair of clusters to be merged next are single-link [13], complete-link [10] and group average (i.e., unweighted Pair Group Method with Arithmetic mean (UPGMA)) [8]. The single-link criterion function measures the similarity of two clusters by the maximum similarity between any pair of objects from each cluster, whereas the complete-link uses the minimum similarity. In general, both the single- and the complete-link approaches do not work very well because they either base their decisions to a limited amount of information (single-link), or assume that all the objects in the cluster are very similar to each other (complete-link). On the other hand, the group average approach measures the similarity of two clusters by the average of the pairwise similarity of the objects from each cluster and does not suffer from the problems arising with single- and complete-link.

Evaluation of Document Clustering

Clustering results are difficult to evaluate, especially for high dimensional data and without a priori knowledge of the objects' distribution, which is quite common in practical cases. However, assessing the quality of the resulting clusters is as important as generating the clusters. Given the same dataset, different clustering algorithms with various parameters or initial conditions will give very different clusters. It is essential to know whether the resulting clusters are valid and how to compare the quality of the clustering results, so that the right clustering algorithm can be chosen and the best clustering results can be used for further analysis.

In general, there are two types of metrics for assessing clustering results: metrics that only utilize the information provided to the clustering algorithms (i.e., *internal metrics*) and metrics that utilize a priori knowledge of the classification information of the dataset (i.e., *external metrics*).

The basic idea behind internal quality measures is rooted from the definition of clusters. A meaningful clustering solution should group objects into various clusters, so that the objects within each cluster are more similar to each other than the objects from different clusters. Therefore, most of the internal quality measures evaluate the clustering solution by looking at how similar the objects are within each cluster and how well the objects of different clusters are separated. In particular, the *internal similarity* measure, *ISim*, is defined as the average similarity between the objects of each cluster, and the *external similarity* measure, *ESim*, is defined as the average similarity of the objects of each cluster and the rest of the objects in the data set. The ratio between the internal and external similarity measure is also a good indicator of the quality of the resultant clusters. The higher the ratio values, the better the clustering solution is. One of the limitations of the internal quality measures is that they often use the same information both in discovering and in evaluating the clusters.

The approaches based on external quality measures require a priori knowledge of the natural clusters that exist in the dataset, and validate a clustering result by measuring the agreement between the discovered clusters and the known information. For instance, when clustering document datasets, the known categorization of the documents can be treated as the natural clusters, and the resulting clustering solution will be considered correct, if it leads to clusters that preserve this categorization. A key aspect of the external quality measures is that they utilize information other than that used by the clustering algorithms. The *entropy* measure is one such metric that looks are how the various classes of documents are distributed within each cluster.

Given a particular cluster, S_r, of size n_r, the entropy of this cluster is defined to be

$$E(S_r) = -\frac{1}{\log q}\sum_{i=1}^{q}\frac{n_r^i}{n_r}\log\frac{n_r^i}{n_r} \qquad (8)$$

where q is the number of classes in the data set, and $n_r{}^i$ is the number of documents of the ith class that were assigned to the rth cluster. The entropy of the entire clustering solution is then defined to be the sum of the individual cluster entropy weighted according to the cluster size. That is,

$$Entropy = \sum_{r=1}^{k}\frac{n_r}{n}E(S_r). \qquad (9)$$

A perfect clustering solution will be the one that leads to clusters that contain documents from only a single class, in which case the entropy will be zero. In general, the smaller the entropy values, the better the clustering solution is.

Key Applications

Document clustering is used to organize large collections of documents into meaningful groups in order to provide intuitive navigation aids, information summarization, data compression, and dimensionality reduction.

URL to Code

An illustrative example of a software package for clustering low- and high-dimensional datasets and for analyzing the characteristics of the various clusters is Cluto [9]. Cluto has implementations of the various clustering algorithms and evaluation metrics described in previous sections. It was designed by the University of Minnesota's data mining's group and is available at http://www.cs.umn.edu/∼karypis/cluto.

Data Sets

Utility tools for pre-processing documents into vector matrices and some sample document datasets are also available at http://www.cs.umn.edu/∼karypis/cluto.

Cross-references

► Clustering Assessment
► Clustering for post-hoc information retrieval
► Information Retrieval
► Text Mining
► Unsupervised Learning

Recommended Reading

1. Boley D. Principal direction divisive partitioning. Data Mining Knowl. Discov., 2(4), 1998.
2. Cutting D.R., Pedersen J.O., Karger D.R., and Tukey J.W. Scatter/gather: A cluster-based approach to browsing large document collections. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 318–329, Copenhagen.
3. Dempster A.P., Laird N.M., and Rubin D.B. Maximum likelihood from incomplete data via the em algorithm. J. R. Stat. Soc., 39, 1977.
4. Dhillon I.S. Co-clustering documents and words using bipartite spectral graph partitioning. In Knowledge Discovery and Data Mining, 2001, pp. 269–274.
5. Ding C., He X., Zha H., Gu M., and Simon H. 1Spectral min-max cut for graph partitioning and data clustering. Technical Report TR-2001-XX, Lawrence Berkeley National Laboratory, University of California, Berkeley, CA, 2001.
6. Duda R.O., Hart P.E., and Stork D.G. Pattern Classification. Wiley, New York, 2001.
7. Fisher D. Iterative optimization and simplification of hierarchical clusterings. J. Artif. Intell. Res., 4:147–180, 1996.
8. Jain A.K. and Dubes R.C. Algorithms for Clustering Data. Prentice Hall, New York, 1988.
9. Karypis G. Cluto: A clustering toolkit. Technical Report 02-017, Department of Computer Science, University of Minnesota, 2002.
10. King B. Step-wise clustering procedures. J. Am. Stat. Assoc., 69:86–101, 1967.
11. MacQueen J. Some methods for classification and analysis of multivariate observations. In Proc. Fifth Symp. Math. Stat. Prob., 1967, pp. 281–297.
12. Salton G. Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley, Reading, MA, 1989.
13. Sneath P.H. and Sokal R.R. Numerical Taxonomy. Freeman, London, UK, 1973.
14. Zahn K. Graph-tehoretical methods for detecting and describing gestalt clusters. IEEE Trans. Comput., (C-20):68–86, 1971.
15. Zha H., He X., Ding C., Simon H., and Gu M. Bipartite graph partitioning and data clustering. In Proc. Int. Conf. on Information and Knowledge Management, 2001.
16. Zhao Y. and Karypis G. Criterion functions for document clustering: Experiments and analysis. Mach. Learn. 55:311–331, 2004.

Document Databases

Frank Wm. Tompa
University of Waterloo, Waterloo, ON, Canada

Synonyms

Document repositories; Text databases; Corpora

Definition

A document database is a collection of stored texts managed by a system that provides query and update facilities. Usually the database includes many documents related by their subject matter, origin, or applicability to an enterprise. The content of each document may be free text, semi-structured text including a few well-identified fields (e.g., title, author, date), or highly structured tagged text such as might be encoded using XML. Occasionally documents may also contain multimedia components.

In contrast, the term *corpus* (plural *corpora*) typically refers to a static collection of texts that have been assembled by experts to study linguistic phenomena (e.g., the Brown Corpus, created in 1964 to study American English, and the Swedish Language Bank) or to provide a rich source of text for lexicographic needs (e.g., the Dictionary of Old English Corpus, including all extant texts written in Old English in the period 600–1150 AD, and the British National Corpus). Such corpora are often distributed or licensed in the form of data only, independently of any document management system.

Historical Background

Electronic documents have been stored on computers almost as long as numeric data. Early document systems supported text editing and formatting and evolved into sophisticated document creation and publishing systems. Electronic document management became an integral part of the move towards office automation that grew substantially during the 1970s and 1980s. Holding documents in computers also made possible the growth of *hypertexts* and hypermedia more generally, starting in the 1960s and continuing today. This, in turn, formed the core of the World Wide Web.

Simultaneously the field of information retrieval developed in response to the recognition that libraries hold a substantial volume of data that is often difficult to access effectively without the intervention of professional librarians. Initially, small document collections were amassed to test the performance of various algorithms designed to locate relevant sources of data in response to users' information needs. Information retrieval has since advanced substantially to deal efficiently and effectively with multi-gigabyte collections of texts, whether the objective is to find relevant documents or to find answers to very specific factual questions.

An outgrowth of office automation was the recognition that corporate documents form a business resource that deserves management commensurate with the effort put into managing capital, human resources, and more traditionally recognized forms of data. Thus document management may be viewed as an extension of database management to handle documents and document fragments with the same care as is given to tabular and other forms of business data.

Document management systems have evolved from each of three technologies: information retrieval engines, as special applications of object management systems, and as extensions to relational database management systems. They provide facilities to define sub-collections, to load new documents and delete old ones, to update existing documents, to retrieve documents that match precise criteria exactly, and to rank documents against a set of keywords or against criteria that specify users' needs in a less precise manner. Document databases form the core of *Enterprise Content Management* systems.

Foundations

Because document database systems arise from traditional database systems and traditional information retrieval systems, the scientific fundamentals of those technologies underlie document databases as well. When the databases include semi-structured or structured documents, they often include constraints in the form of regular expressions or context-free grammars; thus the principles and practices of regular and context-free languages also underlie document databases.

Key Applications

Document databases are typically created by corporations and other enterprises to subject various documents to database management protocols. Publishers and other organizations (or organizational sub-units) for which printed or electronic documents are their products use document databases to maintain drafts and other variants of their products as well as historical materials. As consumers of such products, organizations' digital libraries use document databases to hold and manage their collections. Thus document databases form a core technology for publishers, digital libraries, e-government, and e-business more generally.

In addition, some document databases maintain materials that are internal to the enterprise. These might comprise policies, procedures, advertising, blogs, email messages, customers' comments, confidential reports, etc. The content for other document databases might be collected from external sources and may include annual reports, legal documents, suppliers' product descriptions, financial reviews, etc. Document databases may also be created to support benchmarking studies or to meet specific application needs, such as source code repositories.

Cross-references

► Digital Libraries
► Information Retrieval
► Semi-structured Data Model
► Semi-Structured Database Design
► XML Retrieval
► XML Storage

Recommended Reading

1. Bertino E., Ooi B., Sacks-Davis R., Tan K.-L., and Zobel J. Text databases. In Indexing Techniques for Advanced Database Systems. Kluwer Academic, Norwell, MA, 1997, pp. 151–184.
2. Chin A.G. (ed.). Text Databases and Document Management: Theory and Practice. Idea Group, Hershey, PA, 2001.
3. Christophides V., Abiteboul S., Cluet S., and Scholl M. From structured documents to novel query facilities. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 313–324.
4. Kilpeläinen P., Lindén G., Mannila H., and Nikunen E. 1A structured text database system. In Proc. Int. Conf. on Electronic Publishing, Document Manipulation and Typography (EP 90), 1990, pp. 139–151.
5. Loeffen A. Text databases: a survey of text models and systems. SIGMOD Rec., 23(1):97–106, 1994.
6. Lowe B., Zobel J., and Sacks-Davis R. A formal model for databases of structured text, In Proc. Fourth Int. Conf. on Database Systems for Advanced Applications (DASFAA'95). 1995, pp. 449–456.
7. Macleod A. A data base management system for document retrieval applications. Inf. Syst., 6(2):131–137, 1981.
8. Sacks-Davis R., Arnold-Moore T., and Zobel J. Database systems for structured documents. In Proc. Int. Symp. on Advanced Database Technologies and Their Integration (ADTI'94), 1994, pp. 272–283.
9. Salminen A. and Tompa F.W. Requirements for XML document database systems. In Proc. ACM Symp. on Document Engineering (DocEng'01). 2001, pp. 85–94.
10. Stonebraker M., Stettner H., Lynn N., Kalash J., and Guttman A. Document processing in a relational database system. ACM Trans. Inf. Syst., 1(2):143–158, 1983.

Document Field

Vassilis Plachouras
Yahoo! Research, Barcelona, Spain

Definition

A document field is a part of a document or of the document metadata in which the text has a particular function. A document field can contain free or preformatted text. Each field, according to its function, has different characteristics, length, and term distributions.

Key Points

Textual documents have implicit structure, which aids the understanding of the text. Long textual documents are usually organized in chapters, sections, paragraphs, and each of those can have a concise description in the form of a title. In the case of hypertext documents, explicit links between documents in the form of hyperlinks are often associated with anchor text. News wire documents also have metadata such as date, or the name of the author. Efforts to standardize metadata about documents have resulted in projects such as the Dublin Core Metadata Initiative [1].

Fields are also being used to represent the annotations of text with semantic and syntactic information. For example, the semantic information may correspond to entities, or locations. The syntactic information may correspond to the part of speech of tokens or to syntactic relationships between tokens. Such information can be used to perform search tasks such as entity ranking [3].

The text and the distribution of terms in a particular field depend on the function of that field. For example, a term may occur many times in a document, because of the document's verbosity. On the other hand, the title of a document is a short and concise description of the document. Hence, terms are expected to appear only once or twice in the title of a document, and the resulting term frequency distribution is almost uniform [2]. Similarly, the anchor text of incoming hyperlinks of Web documents serves the purpose of providing a concise description of a document, and from this point of view, it is similar to the title of documents. However, a document is likely to have one title, while it is not unusual to have documents with several million incoming hyperlinks and associated anchor texts.

Cross-references

► Anchor Text
► Dublin Core
► Field-based Information Retrieval Models

Recommended Reading

1. Dublin Core Metadata Initiative. Retrieved April 15, 2008, http://dublincore.org/.
2. Jin R., Hauptmann A., and Zhai C. Title language model for information retrieval. In Proc. 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2002, pp. 42–48.
3. Zaragoza H., Rode H., Mika P., Atserias J., Ciaramita M., and Attardi G. Ranking Very Many Typed Entities on Wikipedia. In Proc. Int. Conf. on Information and Knowledge Management, 2007, pp. 1015–1018.

Document Formats

► Document representations (incl. native and relational)

Document Identifier

► Resource Identifier

Document Index and Retrieval

► Text Indexing and Retrieval

Document Length Normalization

Ben He
University of Glasgow, Glasgow, UK

Synonyms

Term frequency normalization; Length normalization

Definition

Document length normalization adjusts the term frequency or the relevance score in order to normalize the effect of document length on the document ranking.

Key Points

The reasons for employing a document length normalization method in an IR system are quite subtle. In general, the effect observed on the ranking by the presence of many lengthy documents in a collection is to favor their retrieval with respect to shorter documents.

Singhal, Buckley and Mitra gave the following two reasons for adopting a length normalization in the vector space model [4]:

1. The same term usually occurs repeatedly in long documents.
2. The vocabulary of a long document is usually large.

In 1994, Robertson and Walker also studied the effect of document length in the context of the probabilistic model. They observed that:

► Some documents may simply cover more material than others, [. . .], a long document covers a similar scope to a short document, but simply uses more words.

According to Robertson and Walker [2], term frequencies may also depend on author's writing style, that may describe concepts and facts either in details or concisely. Robertson and Walker called this phenomenon as the *verbosity hypothesis*.

According to the language modeling approach, the normalization of the document length is instead related to the *sparse data problem*. The sparse data problem is also the core problem in natural language processing for the estimation of the probability of string occurrences. The *smoothing technique* is usually applied to cope with the sparse data problem in the language modeling approach for IR [5].

In the context of Vector Space model, cosine normalization adjusts the effect of document length on document weights by computing the cosine similarity between the query and the document weight vectors.

Singhal et al. proposed an improvement of cosine normalization for the vector space model, called the *pivoted normalization* [4]. The basic idea of the pivoted normalization is to introduce a tunable hyper-parameter to empirically adjust the normalization factor of a given normalization method, by fitting the probability of retrieval to the probability of relevance. The probability of retrieval is computed from returned documents for each given query, and the probability of relevance is computed from the relevance information given by the human assessors.

In the context of probabilistic model, the BM25 weighting model employs a saturation function to normalize term frequency [3]. This normalization function is derived from the study of the document length effect in the 2-Poisson model.

Some of Divergence from Randomness (DFR) weighting models employ the *Normalization 2* for adjusting the relationship between term frequency and document length, that assumes a decreasing term frequency density function of document length [1].

Cross-references

► Probabilistic Models and Binary Independence Model
► Probability Smoothing

Recommended Reading

1. Amati G. Probabilistic models for information retrieval based on divergence from randomness. Ph.D. Thesis, Department of Computing Science, University of Glasgow, 2003.
2. Robertson S. E. and Walker S. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 232–241.
3. Robertson S.E., Walker S., Jones S., and Hancock-Beaulieu M. Okapi at trec-3. In Proc. The 3rd Text Retrieval Conference, 1994.
4. Singhal A., Buckley C., and Mitra M. Pivoted document length normalization. In Proc. 19th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1996, pp. 21–29.
5. Zhai C. and Lafferty J. A study of smoothing methods for language models applied to ad hoc information retrieval. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 334–342.

Document Links and Hyperlinks

Vassilis Plachouras
Yahoo! Research, Barcelona, Spain

Definition

Document links and hyperlinks are cross-references between different documents or between different parts of the same document. They facilitate the navigation of users in the document space. However, information seeking by only following hyperlinks is possible only for relatively small collections of hyperlinked documents. Hyperlinks can have explicit or implicit types. Two common types of links are organizational or navigational links and informational links.

Key Points

Textual documents are rich in structure, one aspect of which is the cross-references or links to different parts of the same document or to other documents. Bibliographic references is one form of links between documents for example. Bush [3] envisioned Hypertext as a natural way to organize, store and search for information, similar to the associative way in which the humans organize information.

Document links and hyperlinks are explicit cross-references between parts of the same document or different documents. Such links alter the information search process by allowing a user to navigate the document space by following hyperlinks. Navigation may be sufficient for small collections of documents. As the number of documents increases, or when navigation is allowed across heterogeneous sets of hypertext documents, however, users may not be able to locate information by merely following links. IR techniques address one aspect of this problem by allowing search for information, or locating starting points for browsing hypertext collection.

The links in hypertext systems can have explicit or implicit types. Baron et al. [2] identified two main types of links, namely the organizational and the content-based links. The former type of links was used to organize and help navigation among hypertext documents, while the latter type was used for pointing to documents on similar topics. However, as with bibliographic references in scientific publications, some hypertext systems do not provide typed links. The automatic inference of the link type is a difficult task, because it requires understanding the context of both the source and destination documents. Differently from identifying the type of hyperlinks [1], investigated the automatic typed linking of related documents. After linking all pairs of documents, the similarity of which exceeds a threshold, the resulting graph is simplified by iteratively merging links. A type is assigned to the resulting links, according to a predefined taxonomy.

Cross-references

► Anchor text

Recommended Reading

1. Allan J. Automatic hypertext link typing. In Proc. Seventh ACM Conf. on Hypertext, 1996, pp. 42–52.
2. Baron L., Tague-Sutcliffe J., Kinnucan M.T., and Carey T. Labeled, typed links as cues when reading hypertext documents. J. Am. Soc. Inform. Sci., 47(12):896–908, 1996.
3. Bush V. As we may think. The Atlantic Monthly, July, 1945.

Document Management

► Enterprise Content Management

Document Path Query

► Path Query

Document Repositories

► Document Databases

Document Representations (Inclusive Native and Relational)

Ethan V. Munson
University of Wisconsin-Milwaukee, Milwaukee, WI, USA

Synonyms

Documents; Document formats; Markup languages; Semi-structured data; Page representations

Definition

Native document representations are file formats designed for *documents*. They can be roughly divided into three types: page-oriented, stream-oriented, and tree-structured. Hybrid types can also be found. Within each type, document representations range from the simple to the complex. All native representations assume an implicit order of the document's information, reflecting the linear reading order of conventional documents. The most important document representation is the Extensible Markup Language (XML), which is tree-structured and can have any level of complexity. It is seeing widespread use on the Web and in business and is also popular for non-document applications.

Relational databases use a variety of document representations that map to a native representation. Page-oriented and stream-oriented documents are best stored in a coarse-grained manner and do not appear to have stimulated much research. In contrast, tree-structured documents are well-suited to fine-grained decomposition for storage in relational databases. As a result, XML databases are a very active research topic. The challenge for relational systems is to maintain the implicit order of the documents' elements while providing efficient access and updates.

Historical Background

Furuta et al. [6] survey document formatting systems up to 1982. The earliest document representations appear to have been created by programmers who wanted to be able to create their own documents without the aid of support staff, using readily available devices. All of the representations described in the survey are *markup languages*. The earliest markup languages, such as RUNOFF and PUB, were stream-oriented. Their markup was highly procedural, specifying changes to parameters of a simple formatter or line breaker. Later markup languages, such as Scribe and GML, supported higher levels of abstraction,were at least partially tree-structured, and were used in systems with higher-quality formatters. For the TeX system, Knuth developed advanced formatting algorithms [9] whose quality has yet to be surpassed. All markup language systems assumed that their users would edit language files with a text editor and then invoke a formatter on the command line to produce output for a printer.

The personal computer revolution in the 1980s spawned the creation of various word processing systems. These systems had user interfaces that were more accessible to non-technical workers, but their document representations were much simpler than those of the later markup-based systems. Early word processors used stream-based representations that were entirely procedural, with no facility for abstract concepts like figures or section headings. As these systems matured, they gained more abstract structural features, such as named styles for paragraphs, but their representations have remained essentially stream-oriented. In general, word processing document representations are not human-readable and are proprietary, though conversion tools between representations are widely available.

The simultaneous development of the laser printer required a means to transmit a page from computer to printer over the low-bandwidth connections then available. In response, various companies designed proprietary page description languages (PDLs) that described pages at a higher level, thus requiring substantially less bandwidth. The most important were Adobe's PostScript, used in the first personal laser printers, and Hewlett-Packard's Printer Command Language (PCL). Both are still widely used in printers, but today the most important PDL is Adobe's Portable Document Format (PDF) [1] because it is printer-independent, compact, and because Adobe distributes free viewing and printing software for all widely-used platforms.

By the mid-1980's, the diversity of incompatible markup languages and word-processing representations was making collaboration between authors quite difficult. In response, two competing document interchange formats were developed, the Standardized Generalized Markup Language (SGML) [7] and the Open Document Architecture (ODA). Only SGML was a success and its success was limited. However, SGML was the basis for the Hypertext Markup Language (HTML) used on the World Wide Web. As HTML came to be used more as a page description language than as a high-level tree-structured specification, the Web community sought a more structured solution. The result was the Extensible Markup Language (XML) [3], which is designed to allow Web documents to convey stronger semantics and to better support sophisticated, even intelligent, applications.

Foundations

Native Representations

Page-Oriented Representations There are two principal page-oriented representations: page images and page description languages (PDLs).

The simplest page-oriented document representation is a sequence of page images, usually created by scanning paper documents. While this representation may seem primitive, it is quite important because of the substantial number of documents that predate electronic representations of any kind or for which the electronic version has been lost. Often, in digital libraries, the page images will have been processed by a document analysis system in order to generate a searchable text stream or to produce an electronic version of the page that can be scaled or reformatted without producing image artifacts. The result is a hybrid representation mixing pages with a stream or tree structure. The development of efficient workflows for this analysis process has been an interesting area of research [13].

PDLs are considerably more complex. The core of any PDL is a two-dimensional vector graphics language with strong support for high-quality text rendering. This implies full support for scientific floating point computation, for conversion between various units of measure, and for specifying character fonts. PDLs must also have commands to control paper handling and common printing features like screening and halftoning. The PDLs used in printers (principally PostScript and PCL) are not suited to database applications because their documents are specific to particular printers and cannot be guaranteed to print or display correctly on all devices. In contrast, the PDF [1] representation is a generalization of PostScript that is device-independent and has evolved over time to have many of the best qualities of stream-oriented and tree-structured representations. Documents encoded by modern PDF generators typically include a complete text stream that can be indexed and searched. Both commercial and open-source tools can be found to generate and manipulate PDF. Finally, it worth mentioning that the PostScript PDL is a fully human-readable language that can be created in a standard text editor, though it also supports binary data formats.

Stream-Oriented Representations Stream-oriented representations organize documents as a sequence of characters or paragraphs. They may contain substantial amounts of formatting information, but unlike the page-oriented representations, generally do not encode the exact appearance of the document on the page or screen. The principal stream-oriented representations are raw text, the Rich Text Format, and various word processor formats.

A raw text document contains a sequence of characters. Any organization of the characters into lines, paragraphs, or pages is specified by the use of specialized characters such as the ASCII line feed and form feed characters. The most common character coding scheme is ASCII, but the more general *Unicode* format is also seen and may grow in importance over time. Raw text has the advantages of simplicity, compactness, portability, and ease of processing. Its primary disadvantage is the inability to represent almost any useful typographic, hypertext, or multimedia effect. The raw text representation is remarkably robust and remains in widespread use, especially in the software development community, where the ubiquity of programming tools makes raw text an attractive representation. It is also a common representation for e-mail.

Rich Text Format (RTF) [10] is a proprietary representation that is widely used for interchange among word processors. Its canonical form is a human-readable ASCII markup language that describes a document as a stream of paragraphs that may be divided into sections. RTF's sections and paragraphs embody regions of content with common formatting characteristics. Document content appears inside the paragraphs along with other markup.

Word processor representations resemble RTF in that they describe a sequence of paragraphs but until recently most have been proprietary, binary representations. Recently, human-readable non-proprietary formats for word processing have begun to be accepted, with the most important being the Open Document Format [11]. This format uses the tree-structured XML markup language, but its underlying structure is still a stream of paragraphs.

Tree-Structured Representations For databases, the most interesting native document representations are tree-structured markup languages. The most important such language is the Extensible Markup Language (XML) [3], which is essentially a simplification of the earlier SGML standard. Because XML is simple, general, and human-readable, it has become a standard representation for data interchange.

XML is really two languages: a markup syntax for documents and a context-free grammar meta-language for defining classes of documents that can be encoded in the markup syntax. The markup syntax primarily defines how a tree of elements with embedded content is specified by marking up the content with *tags*. The following example shows a trivial, but complete, "bookdata" document. The bookdata element is the root of the tree and contains title and editor elements. The bookdata element also has two attributes, which record the topic and year of the book. In general, elements are designed to hold content that will be shown to people and attributes are designed to hold metadata that could be processed by automated tools.

```
<? xml version="1.0" ?>
<bookdata topic="Databases" year ="2008">
   <title>Encyclopedia of Database
   Systems</title>
   <editor>Ling Liu</editor>
   <editor>Tamer \:{O}zsu</editor>
</bookdata>
```

XML has several important technical and philosophical differences from the page- and stream-oriented representations.

- Unlike the PDLs, XML is almost purely declarative. It is not a programming language and has no computational features. An XML document describes only a hierarchical organization of content, possibly with metadata.
- XML is designed to represent the logical organization of a document rather than its appearance. It has no predefined formatting features and does not make any assumptions about media or devices.
- While designed for representing documents, XML is not limited to this application. In fact, XML's simplicity and clean syntax have resulted in many unanticipated uses.
- XML is supported by a rich ecosystem of related languages that support tasks including document transformation (XSLT [8]) and alternative grammar systems (or *schemas*) for defining document classes (XSchema [5]). Especially important for databases is the XQuery document query language [2].

XML documents are often categorized into three classes: structured, semi-structured, and marked-up text. In a structured XML document class, all documents have the same tree structure and every element has a unique name. In semi-structured document classes, there may be variations in the tree structure at certain locations, such as alternate element types or variable repetition of one element or a group of elements. In both semi-structured and structured documents, document content is only found in the leaf elements of the tree. In contrast, marked-up text can have content at any level of the tree and may permit huge variations in tree structure. Marked-up text may have important elements of logical structure, such as sentences, that are not explicitly marked-up by elements and span multiple elements. Most database research has focused on structured and semi-structured XML.

Hybrid Representations Hybrid representations can deliver the advantages of multiple representations at the cost of increased complexity. They are most commonly seen as extensions that address the limitations of page-oriented representations.

The combination of page images with a parallel text stream has already been mentioned. This representation can be used to create document interfaces that show the scanned image, but allow indexing and searching of the content, including highlighting those portions of the original page image that match a search string.

Considerably more elaborate is Tagged PDF [1], which extends the page description core of PDF with a structural tagging system to encode the roles of text fragments (e.g. body text, footnote, etc.), adds explicit word breaks, and maps all fonts to Unicode. Used properly, Tagged PDF ensures that the content of a PDF document can be scanned in the same order that a human reader would scan it and clearly identifies elements like marginal notes and headers that are not part of the main text flow. It also supports search and indexing, as well as being able to encode some of the semantics of XML.

Relational Representations

In relational databases, documents can be represented either as atomic entities, using large objects (LOB), or decomposed into their component parts. The large object approach can be used with all native representations. Decomposition is usually called "shredding" and is only used with XML documents.

Large Object Representation LOB representation stores an entire document or medium-sized parts of an entire document as a large object in a relational table. This is the natural representation for documents whose native representation is page-oriented or stream-oriented and has some real advantages for XML documents as well. Long documents may be divided into a sequence of smaller LOBs, such as individual pages or sections.

LOB representation is useful for documents that do not need to be updated frequently and for which interesting metadata can be computed at the time of insertion into the database. In this case, the relational system provides an efficient way to find documents based on queries against the metadata. For page- and stream-oriented documents, LOB representation is a natural choice, because the internal structure of the documents (i.e. pages or sections/paragraphs) principally conveys presentation and has little semantics useful for queries and updates. In contrast, LOB representation is unlikely to be used for XML documents unless they are quite unstructured or if a description of the document class is not available.

LOB representation has the disadvantage that standard relational operations cannot be used to search or update the internal structure and content of the documents. Instead, access and update operations must be performed by other tools. While these tools may be useful and efficient for single documents, the performance and scalability benefits of the relational approach for large-scale collections are lost when using the LOB representation.

Shredded Representation *Shredding* is the process of tearing apart an XML document into its component elements for storage in database relations. There are many trade-offs in designing both relations and queries for the shredded elements. Draper [12] discusses the full range of choices. A key issue is whether the schema for the XML document class is known.

When a schema is not available for an XML document class, the *edge table* representation is used. An edge table has one tuple for each element or attribute in a document. The tuple has the following form:

`Edge(`*ID*`, parentID, name, value)`

The root element has a null parent ID and internal nodes of the tree have null values. A useful optimization is to replace the name with a `pathID` that points to another table holding the full path names of the nodes. Using pathIDs can reduce both table size and the number of joins required for common queries.

When a schema is available for the documents, *inlining* is a more efficient representation. Under inlining, elements are only placed in separate relations when they can appear multiple times. Elements that only appear once become columns in the relation for their parents. In the earlier "bookdata" example, there would be two relations: one for the bookdata element that would have columns for the two attributes and for the title; and another to hold the list of authors that would be connected to the bookdata element via a foreign key. The design of efficient queries over inlined databases is challenging. Shanmugasundaram et al.

[12] showed that a complex query structure called Sorted Outer Union provides the best combination of efficiency and generality.

A key problem when working with shredded XML documents is correctly maintaining the order of the elements. This problem arises because the order of the content in documents is usually quite important, but it is only encoded implicitly. In the earlier "bookdata" example, the order of the author's names should be preserved, but it is only apparent from the order in which the names appear in the XML source code. Relational databases do not represent order automatically, so additional information must be added to the tables. Tatarinov et al. [14] showed that the best choice of order information depends on the type of query load. When updates are rare, it is best to store a global order number (an integer representing the node's position in a pre-order tree traversal). For loads that mix updates and accesses, a variable-length numbering system related to the Dewey Decimal Classification system is superior.

Key Applications

Documents are pervasive in human society, so there are many applications for document representations. The most important application is the Web, which can be viewed narrowly as a document-sharing system. Every Web page is a document written in HTML or XHTML (an adaptation of HTML to the rules of XML). A growing number of Web documents are derived from information represented in XML or from XML fragments taken from a database. Because Web browsers have only limited support for XML itself, it is primarily used as a back-end representation.

Other important applications include:

- Scanned document images are widely used to represent for historical, legal, and financial documents. Systems that support scholars typically have rich metadata attached to the page images.
- Page description languages (especially PDF) are widely used as electronic representations of the final form of documents, especially business and official documents that are also distributed in print form.
- The pervasive use of word-processing software makes stream-based representations ubiquitous for business documents. The lack of widely-adopted open standards presents a real challenge for systems that try to support them.

Cross-references

► Dewey Decimal System
► Digital Libraries
► Document
► Document Databases
► Indexing Semi-Structured Data
► Markup Language
► Meta Data
► Semantic Web
► XML
► XPath/Xquery
► XSL/XSLT

Recommended Reading

1. Adobe Systems Incorporated, PDF reference. Sixth edn., 2006.
2. Boag S., Chamberlin D., Fernández M.F., Florescu D., Robie J., and Siméon J. XQuery 1.0: an XML query language. World Wide Web Consortium (W3C), 2007.
3. Bray T., Paoli J., Sperberg-McQueen C.M., Maler E., and Yergeau F., Extensible Markup Language (XML) 1.0. World Wide Web Consortium (W3C), fourth edn., 2006.
4. Draper D. Mapping between XML and Relational Data. In XQuery from the experts: a guide to the W3C XML query language. chap. 6, Addison Wesley, 2003.
5. Fallside D.C. and Walmsley P. XML Schema Part 0: Primer. World Wide Web Consortium (W3C), second edn., 2004.
6. Furuta R., Scofield J., and Shaw A. Document formatting systems: survey, concepts, and issues. Comput. Surv., 14 (3):417–472, 1982.
7. Goldfarb C.F. (ed.) Information processing – text and office systems – Standard Generalized Markup Language (SGML). International Organization for Standardization, Geneva, Switzerland, 1986, International Standard ISO 8879.
8. Kay M. XSL transformations (XSLT) version 2.0. World Wide Web Consortium (W3C), 2007.
9. Knuth D.E. and Plass M.F. Breaking paragraphs into lines. Software Prac. Exper., 11(11):1119–1184, 1982.
10. Microsoft Office Word 2007 Rich Text Format (RTF) Specification. 2007, version 1.9. Downloaded from microsoft.com, November 2007.
11. OASIS, Open Document Format for Office Applications (OpenDocument) v1.1. 2007, http://docs.oasis-open.org/office/v1.1/OS/, 2007.
12. Shanmugasundaram J., Shekita E., Barr R., Carey M., Lindsay B., Pirahesh H., and Reinwald B. Efficiently publishing relational data as XML documents. VLDB J., 10(2–3), 2001.
13. Simske S.J. and Baggs S.C. Digital capture for automated scanner workflows. In Proc. 2004 ACM Symp. on Document Engineering, 2004, pp. 171–177.
14. Tatarinov I., Viglas S.D., Beyer K., Shanmugasundaram J., Shekita E., and Zhang C. Storing and querying ordered XML using a relational database system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 204–215.

Document Retrieval

► Information Retrieval

Document Segmentation

► Text Segmentation

Document Summarization

► Text Summarization

Document Term Weighting

► Information Retrieval Models

Document Visualization

► Text Visualization

Documents

► Document representations (inclusive native and relational)

Domain Relational Calculus

► Relational Calculus

Downward Closure Property

► Apriori Property and Breadth-First Search Algorithms

DR

► Dimensionality Reduction

DRM

► Digital Rights Management

DT

► Decision Trees

Dublin Core

James Caverlee[1], Prasenjit Mitra[2], Mary Laarsgard[3]
[1]Texas A&M University, College Station, TX, USA
[2]Pennsylvania State University, University Park, PA, USA
[3]University of California-Santa Barbara, Santa Barbara, CA, USA

Definition

Dublin Core is a standardized metadata set for describing information resources like documents, videos, images, services, and other digital artifacts. Dublin Core is intended to provide a simple metadata model that can be adopted across a wide range of communities in an effort to enhance semantic interoperability. The Dublin Core Metadata Element Set has been formally endorsed by a number of standards bodies including ISO [5], NISO [7], and IETF [6].

Historical Background

In 1995, the Online Computer Library Center (OCLC) and the National Center for Supercomputing Applications (NCSA) co-sponsored a workshop to address the challenge of developing a common metadata set for describing networked resources [10]. The workshop was motivated in part by the explosive growth of the Web in the early 1990s and the inherent difficulty in finding Web resources. The name "Dublin" in Dublin Core derives from the location of this first workshop in Dublin, Ohio; the term "Core" refers to the basic importance of the elements defined in the metadata standard that can be applied broadly across a wide range of resources.

In the years since the first Dublin Core workshop, the community has hosted annual workshops and conferences devoted to Dublin Core and metadata applications. The continued development and organization of the metadata standard is overseen by the cross-disciplinary Dublin Core Metadata Initiative [3].

Foundations

Supporting information access, organization, and management functionalities in a massively distributed medium like the Web is a serious challenge. In an effort to provide support for these operations, Dublin Core advocates the use of metadata to provide descriptive information about information resources found on the Web, on digital libraries, in enterprize settings, and in other networked domains. In contrast to content-based features of information resources (like the text indexing of a Web document for use in a search engine), the metadata approach can rely on features that describe a resource (and that are not necessarily contained in the resource) to support information discovery, categorization, and other information management functionalities [1].

The Dublin Core metadata set provides a standardized set of metadata elements for describing a wide variety of resources – be they audio files, videos, documents, services, software packages, images, etc. By design, Dublin Core is simple so that metadata may be generated by experts and non-experts alike. The basic Dublin Core standard supports 15 different metadata elements that can be applied to a resource (as shown in Fig. 1). Each element may be used multiple times to describe a single resource, and only the necessary elements need be used in a description of a resource.

The 15 elements are intended to be core descriptors that could be applied regardless of the particular domain of interest. To illustrate, the element "Creator" could refer to the painter of a picture, the author of a book, or to an organization that publishes a software tool. Similarly, the "Description" element could refer to a free text description of a resource, the abstract of an article, a table of contents, or some other descriptive image or text. Although Dublin Core supports great flexibility in the use of these metadata elements, it is good practice to rely on some standard vocabularies for certain elements, e.g., to use standard MIME media types for the element "Format."

As a concrete example, consider a resource like an academic research paper. Figure 2 illustrates some relevant Dublin Core metadata for a sample paper.

Qualified Dublin Core

Dublin Core additionally supports optional *qualifiers* that may be used to extend and refine the 15 basic Dublin Core elements. Qualifiers can be used for either (i) element refinement; or (ii) declaring an encoding scheme [2,9].

Element refinement narrows the meaning of an element. For example, the element "Date" can be refined to "Created," meaning that the metadata associated with the element "Date" refers to a creation date of the resource, and not to the date it was modified. Alternatively, "Date" could be refined to "Modified" if the semantics of "Date" are meant to convey the date the resource was modified, but not created.

Declaring an encoding scheme provides additional information about the element that can be used for interpreting the meaning of the element value. For

	DC Element Name	Definition
1.	Title	A name given to the resource.
2.	Creator	An entity primarily responsible for making the resource.
3.	Subject	The topic of the resource.
4.	Description	An account of the resource.
5.	Publisher	An entity responsible for making the resource available.
6.	Contributor	An entity responsible for making contributions to the resource.
7.	Date	A point or period of time associated with an event in the lifecycle of the resource.
8.	Type	The nature or genre of the resource.
9.	Format	The file format, physical medium, or dimensions of the resource.
10.	Identifier	An unambiguous reference to the resource within a given context.
11.	Source	A related resource from which the described resource is derived.
12.	Language	A language of the resource.
13.	Relation	A related resource.
14.	Coverage	The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant.
15.	Rights	Information about rights held in and over the resource.

Dublin Core. Figure 1. The 15 Simple Dublin Core elements [4].

DC element name	Value
Title	Mining association rules between sets of items in large databases
Creator	Rakesh agrawal and tomasz imielinski and arun swami
Subject	Databases, data mining, association rules
Description	Seminal research paper that describes an efficient algorithm for extracting association rules from a database of customer transactions
Publisher	ACM SIGMOD International Conference on Management of Data
Date	2003
Language	English
Format	Application/pdf

Dublin Core. Figure 2. Sample Dublin Core metadata for a research paper.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">

   <rdf:Description rdf:about="http://rakesh.agrawal-family.com/papers/sigmod93assoc.pdf">

      <dc:title>Mining Association Rules Between Sets of Items in Large Databases </dc:title>
      <dc:creator>Rakesh Agrawal and Tomasz Imielinski and Arun Swami</dc:creator>
      <dc:subject>Databases, data mining, association rules</dc:subject>
      <dc:description>Seminal research paper that describes an efficient algorithm
                     for extracting association rules from a database of customer
                     transactions.</dc:description>
      <dc:publisher>ACM SIGMOD International Conferenceon Management of Data</dc:publisher>
      <dc:date>2003</dc:date>
      <dc:language>English</dc:language>
      <dc:format>application/pdf</dc:format>
   </rdf:Description>
</rdf:RDF>
```

Dublin Core. Figure 3. Example RDF/XML markup using Dublin Core.

example, the "Subject" element may be qualified with an encoding scheme for the Library of Congress Subject Headings (LCSH), a standard set of subject headings that are widely adopted in libraries. By relying on a controlled vocabulary instead of free text, the "Subject" element may provide clearer meaning to applications relying on Dublin Core.

Since Dublin Core is intended to be simple and easy-to-use, applications built to work with Dublin Core metadata should be able to ignore qualifiers entirely and still function in a useful way, albeit with some loss of expressiveness.

Encoding Dublin Core

Dublin Core metadata can be represented in a number of different formats, including plain text, HTML, XML, and RDF. With the rise in interest in the Semantic Web and other knowledge management activities, there has been a push to see Dublin Core widely adopted using RDF [8]. As an illustration of encoding Dublin Core in RDF, Fig. 3 shows the RDF-encoded metadata for the same resource described in Fig. 2.

Key Applications

Web, Semantic Web, digital libraries, business-to-business exchange.

Cross-references

► Metadata
► Metadata Registry

Recommended Reading

1. Cathro W. Metadata: An Overview. Standards Australia Seminar, 1997.
2. Dublin Core Metadata Initiative. Dublin Core Qualifiers, 2000.
3. Dublin Core Metadata Initiative. 2008, http://dublincore.org.
4. Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, Version 1.1, 2008.
5. International Organization for Standardization. ISO 15836-2003 Information and Documentation – The Dublin Core Metadata Element Set, 2003.
6. Internet Engineering Task Force. IETF RFC 5013 – The Dublin Core Metadata Element Set, 2007.
7. National Information Standards Organization. ANSI/NISO Z39.85-2007 The Dublin Core Metadata Element Set, 2007.
8. Nilsson M., Powell A., Johnston P., and Naeve A. Expressing Dublin Core Metadata Using the Resource Description Framework (RDF), 2008.

9. Weibel S. The State of the Dublin Core Metadata Initiative. Bull. Am. Soc. Inf. Sci., 25(5):18–22, 1999.
10. Weibel S., Godby J., Miller E., and Daniel R. OCLC/NCSA Metadata Workshop Report, 1995.

Dump

► Logging and Recovery

Duplicate Detection

► Record Matching
► Semantic Data Integration for Life Science Entities

Duplicate Semantics

► Bag Semantics

Duplication

► Replication

Durability

► ACID Properties

Duration

► Time Interval

DVDs

► Storage Devices

DW

► Data Warehouse

Dynamic Graphics

DIANNE COOK
Iowa State University, Ames, IA, USA

Synonyms

Multivariate data visualization; Multiple linked plots; Motion graphics; Rotation; Tour; Animation

Definition

Dynamic graphics for data, means simulating motion or movement using the computer. It may also be thought of as multiple plots linked by time. Two main examples of dynamic graphics are animations, and tours. An animation, very generally defined, may be produced for time-indexed data by showing the plots in time order, for example as generated by an optimization algorithm.

A tour is designed to study the joint distribution of multivariate data, in search of relationships that may involve several variables. It is created by generating a sequence of low-dimensional projections of high-dimensional data – typically 1D or 2D – so that many different aspects of high-dimensional data can be observed. Tours are thus used to find interesting lower-dimensional projections of the data, ideally for data which contains real-valued variables. The data $\mathbf{X}_{n \times p}$ is projected into $\mathbf{A}_{p \times d}$ to produce a data projection $\mathbf{Y}_{n \times d} = \mathbf{X}_{n \times p} \mathbf{A}_{p \times d}$. The projection matrix $\mathbf{A}_{p \times d}$ is orthonormal. The coefficients in $\mathbf{A}_{p \times d}$ are generated so that all values have some given probability of being chosen and consecutive projections are close to the previous, to provide apparently smooth motion.

Historical Background

The grand tour was defined and named by Asimov [2]. It computes the projections uniformly from a $(p-1)$-D sphere. All possible projections are equally likely. To provide a smooth path, he generated sequential projections by following a path on a high-dimensional torus. Buja and Asimov [3] further developed the grand tour by using an interpolated geodesic random walk, between randomly generated basis planes. This improvement ensures that the grand tour efficiently covers the space of all projections (Grassmann manifold), and that within-plane spin is absent. The mathematics and algorithms for this approach is described in detail by Buja et al. [4]. A simpler approach to generating a grand tour is used by Tierney [13].

The grand tour was first implemented by Buja et al. [5] and the work developed to include a guided tour, where projections are chosen according to a measure of interestingness (e.g. projection pursuit) and a correlation tour, where two sets of variables are toured using 1D projections in the horizontal and vertical directions. A correlation tour is related to regression analysis, modeling one or more response variables against many explanatory variables. The guided tour was developed further by Cook et al. [8].

Cook and Buja [7] developed manual controls for the tour, enabling the user to manually change the projection matrix coefficients, to assess the impact of selected variables on the visible data structure. Wegman [14] developed the full dimensional grand tour, where the projection is displayed in parallel coordinates, and also as a scatterplot matrix [6]. He further developed the image grand tour to study remote sensing data [15]. Scott [12] developed the density grand tour, where each 1D projection is shown as a density. Huh [11] developed a grand tour with a tail. Locations of points in previous projections are plotted for a given time period in future projections, giving a fuller sense of the motion of points.

Andrews curves [1] are a pre-cursor to the grand tour. A general explanation of tours suitable for a reasonably untechnical audience can be found in a book chapter by Cook et al. [9].

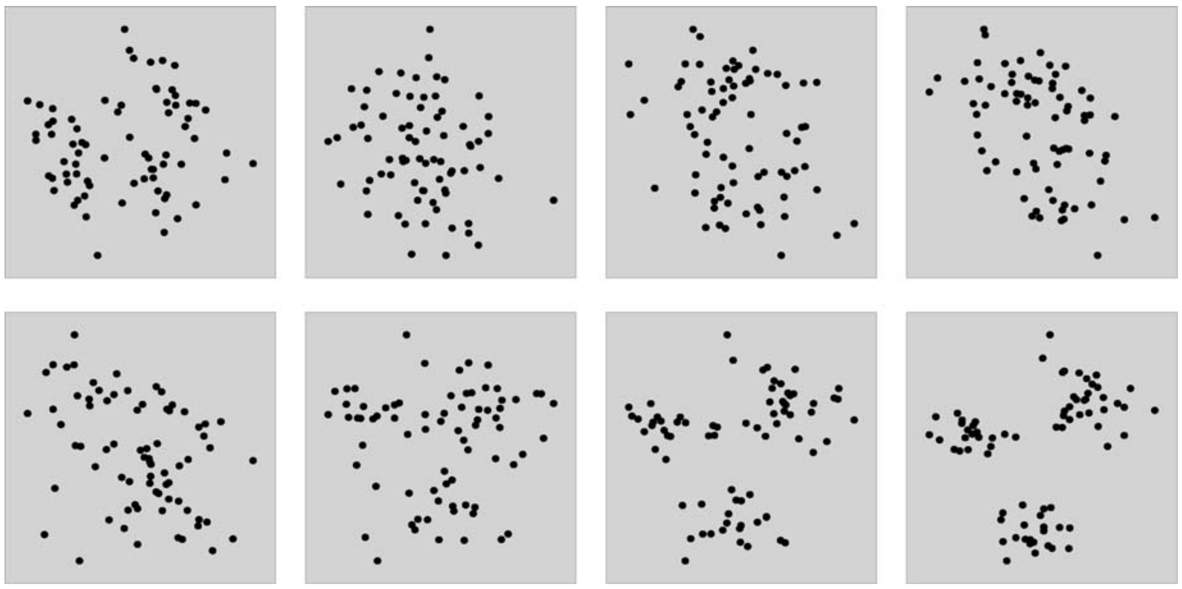
Foundations

The tour is ideally suited for examining the joint distribution of multivariate data, for relatively small p (on the order of small tens rather than hundreds), where the variables take on real-values. Figures 1–3 compare structure detection in a tour with that in a parallel coordinate plot.

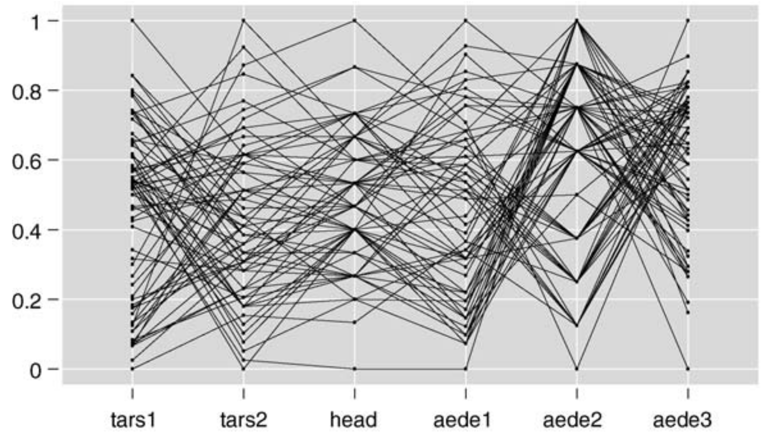
In Fig. 1 a set of eight 2D projections of 6D data are taken from the movie of projections shown in a grand tour. The data has three very well-separated clusters, that are elliptically shaped in the six dimensions. It should be noted that in this data, the three clusters are not perfectly visible in any pair of the six variables. Within seconds of viewing the tour, this is obvious to even the most novice audience. Viewers clue to the clusters by separations between points in certain projections and also the motion patterns of the points.

In a parallel coordinate plot (Fig. 2) the three clusters are not readily detectable. A trained eye can easily see two clusters, by recognizing two groups of similar trends in the lines. The third cluster of lines is a little more difficult to discern. If the three clusters are identified using color, then they become much more visible in the parallel coordinate plot (Fig. 3).

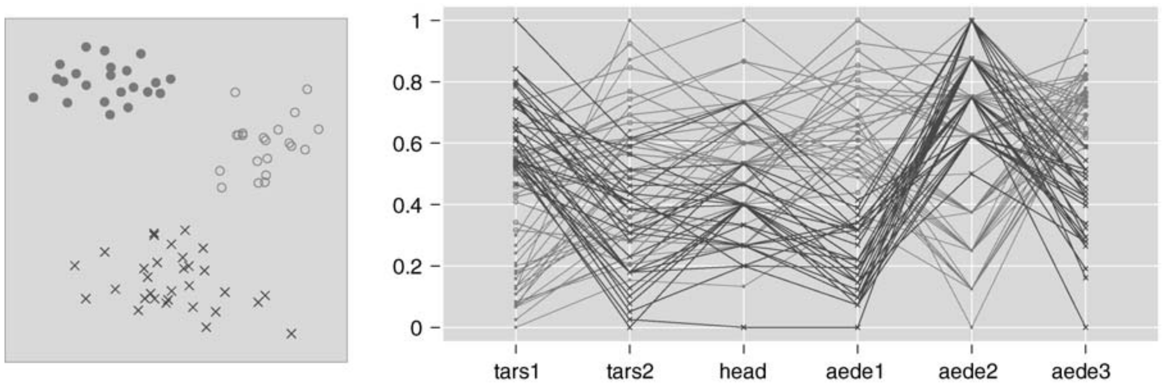
Generally tours are better than parallel coordinate plots for this type of data and structure such as this, clusters, outliers, linear or nonlinear dependencies,



Dynamic Graphics. Figure 1. A sequence of projections from a tour using 2D projections of 6D data. Within seconds of watching the tour clustering of the observations can be seen.



Dynamic Graphics. Figure 2. A parallel coordinate plot of the same 6D data as shown in Fig. 1. Two clusters of different line traces are readily seen but the three clusters are not so obviously recognized.



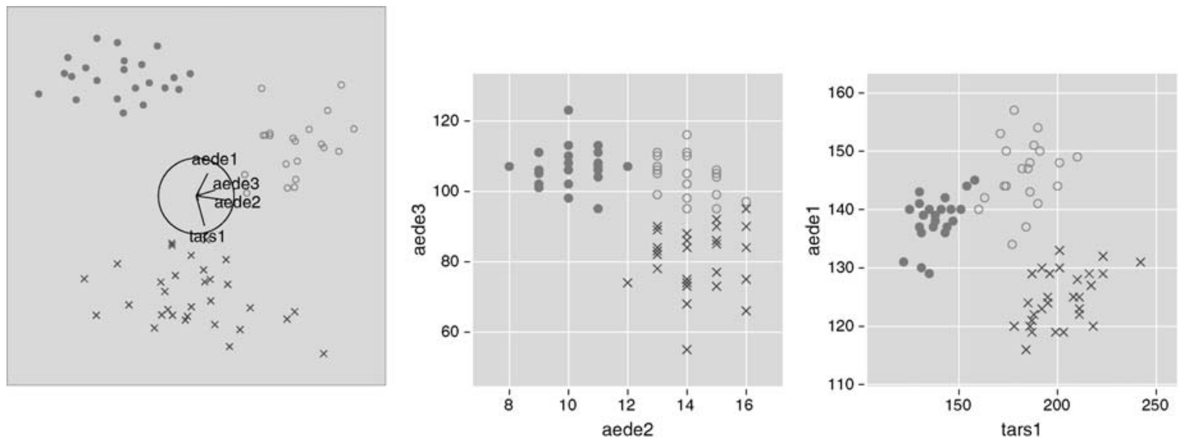
Dynamic Graphics. Figure 3. When the three clusters are colored they are more recognizably clusters in the parallel coordinates plot.

and relatively small dimension. Parallel coordinates are more appropriate when there is a mix of categorical and continuous variables, or when there are a large number of variables.

Tours are usually implemented with the inclusion of some navigation support. Figure 4 provides an example. The circle with radial line segments in the center of the plot represents the projection matrix, **A**, which for this projection is:

$$\mathbf{A} = \left[\begin{array}{cc|c} 0.215 & -0.775 & tars1 \\ 0.065 & 0.081 & tars2 \\ 0.053 & -0.102 & head \\ 0.298 & 0.589 & aede1 \\ 0.786 & -0.108 & aede2 \\ 0.490 & 0.155 & aede3 \end{array}\right]$$

Look at the magnitude and sign of these values to interpret the plot structure. The variables aede2 and aede3 contribute the most to the horizontal direction (first column). In this direction the orange (open circle) cluster is separated from the green (solid circle) cluster and to some extent the purple (cross) cluster. The variables tars1 and aede1 contribute the most to the vertical direction (column 2), with the contribution of tars1 being negative. In this direction the green (solid circle) cluster is separated from the purple (cross) cluster and to some extent the orange (open circle) cluster. This suggests that variables aede2, aede3 contribute to distinguishing between the orange (open circle) cluster from the other two, and variables tars1, aede1 contribute to distinguishing between green (solid circle) and purple (cross) cluster. These interpretations would be checked using pairwise scatterplots (middle, right), univariate plots, or parallel coordinates (Fig. 3). Using these additional aids one would decide that tars1



Dynamic Graphics. Figure 4. Circle inside the tour projection (*left*) provides navigation support for the tour. Scatterplots (*middle, right*) help to confirm the interpretation of structure.

separates green (solid circle) from purple (cross), and that aede2 separates green (solid circle) from orange (open circle). It is really a combination of these variables which makes the difference between the clusters marked, but individual variables contribute in partial ways to the separation.

Key Applications

Tours are a critical part of many different types of multivariate analyses: clustering, classification, multivariate tests, principal components analysis and multidimensional scaling. These types of methods are used in many, many different applications, including astronomy, biology, physics, social science, education, geology, agronomy, ecology, credit risk, defense (Wegman et al. [15] use the image grand tour to detect land mines in satellite images). The tour also can be used to study the geometry of high-dimensional spaces. Tours are an integral part of exploratory data analysis.

Cook and Swayne [10] provide many more examples where the tour might be used to gain insight into multivariate structure in data, or in the performance of multivariate methods and algorithms.

Future Directions

There is a lot of scope for research in dynamic graphics. For the tour, one might investigate different probability distributions for choosing projections, particularly for handling much larger numbers of variables, and making use of sparseness. Studies might be done to suggest the optimal viewing times for watching a tour in order to recognize different types of structure. Someone might study tours on something other than Euclidean space, to enable the study of more complex data. For example, a tour on the space of all permutations could be used to explore categorical data. Other methods for guiding the tour would be useful. Large data poses a problem, because points get over-plotted. The projections might be represented as density plots or convex hulls, if these can be computed sufficiently fast. There are interesting connections with statistical theory that might be explored. For example, most random projections of multivariate data look approximately Gaussian, is related to the Central Limit Theorem.

URL to Code

http://www.ggobi.org

Cross-references

- ▶ Business Intelligence
- ▶ Classification
- ▶ Cluster
- ▶ Cluster Visualization
- ▶ Curse of Dimensionality
- ▶ Data Mining
- ▶ Dimension
- ▶ Dimensionality Reduction
- ▶ Exploratory Data Analysis (EDA)
- ▶ Feature Selection
- ▶ Geographic Information Systems
- ▶ Information Extraction
- ▶ Machine Learning in Computational Biology
- ▶ Mining of Chemical Data
- ▶ Multidimensional Scaling (MDS)
- ▶ Multivariate Visualization Methods
- ▶ Parallel Coordinates

► Principal Components Analysis (PCA)
► Spatial Data Mining
► Visual Analytics
► Visual Classification
► Visual Clustering
► Visual Data Mining

Recommended Reading

1. Andrews D.F. Plots of high-dimensional data, Biometrics, 28:125–136, 1972.
2. Asimov D. The grand tour: a tool for viewing multidimensional data, SIAM J. Sci. Stat. Comput., 6(1):128–143, 1985.
3. Buja A. and Asimov D. Grand tour methods: an outline, Comput. Sci. Stat., 17:63–67, 1986.
4. Buja A., Cook D., Asimov D., and Hurley C. Computational Methods for High-Dimensional Rotations in Data Visualization, In Handbook of Statistics: Data Mining and Visualization, C.R. Rao, E.J. Wegman, J.L. Solka (eds.). Elsevier/North-Holland, 2005, pp. 391–414.
5. Buja A., Hurley C., and McDonald J.A. A data viewer for multivariate data, Comput. Sci. Stat., 17(1):171–174, 1986.
6. Carr D.B., Wegman E.J. and Luo Q. ExplorN: Design Considerations Past and Present, Technical Report 129, Center for Computational Statistics, George Mason University, Fairfax, VA, 1996.
7. Cook D. and Buja A. Manual controls for high-dimensional data projections, J. Comput. Graph. Stat., 6(4):464–480, 1997.
8. Cook D., Buja A., Cabrera J., and Hurley C. Grand tour and projection pursuit, J. Comput. Graph. Stat., 4(3):155–172, 1995.
9. Cook D., Lee E.-K., Buja A., and Wickham H. Grand tours, projection pursuit guided tours and manual controls, In Handbook of Data Visualization, C.-H. Chen, W. Härdle A. Unwin (eds.). Springer, Berlin, Germany, 2006.
10. Cook D. and Swayne D.F. Interactive and Dynamic Graphics for Data Analysis: With R and GGobi, Springer, New York, 2007.
11. Huh M.Y. and Kim K. Visualization of Multidimensional Data Using Modifications of the Grand Tour. J. Appl. Stat., 29 (5):721–728, 2002.
12. Scott D. Incorporating density estimation into other exploratory tools, In Proc. of the Section on Statistical Graphics, 1995, pp. 28–35.
13. Tierney L. LispStat: An Object-Oriented Environment for Statistical Computing and Dynamic Graphics, Wiley, New York, 1991.
14. Wegman E.J. The Grand Tour in *k*-Dimensions, Technical Report 68, Center for Computational Statistics, George Mason University, 1991.
15. Wegman E.J., Poston W.L., and Solka J.L. Image Grand Tour, In Automatic Target Recognition VIII – Proceedings of SPIE, 3371, SPIE, Bellingham, WA, 1998, pp. 286–294.

Dynamic Integrity Constraints

► Temporal Integrity Constraints

Dynamic Taxonomies

► Faceted Search

Dynamic Web Pages

Maristella Matera
Polytechnic University of Milan, Milan, Italy

Definition

They are Web pages that are composed at run-time, by dynamically extracting contents from a data source and composing them into pre-defined page templates.

Key Points

Historically, hypertext navigation was meant as a way to move among "static" documents, i.e., Web pages whose HTML code includes both the content to be presented, as well as the mark-up tags determining content rendition. Real-life Web applications however require the capability of serving to the users pages that dynamically publish content coming from one or more data sources. For example, the content of the home page of a news magazine is refreshed daily, by extracting the latest news from the news repository. This requirement goes beyond the original capabilities of the HTTP protocol, which is designed to exchange requests and resources between the browser and the server, and not to govern the process by which the desired resource is built.

Some server-side technologies have therefore been introduced to overcome these limitations and to enable the construction of Web pages "on the fly." The most common solution is to adopt server-side scripting technologies (such as JSP or PHP), which enable inserting into an HTML page template some programming instructions that a server-side program execute to compute the contents to be extracted dynamically from the application data source. The result sent back to the client is then a properly formatted HTML page, including the extracted contents.

Other solutions imply the extension of the Web server, through execution engines (for example Java Servlet API) able to serve the requests for the dynamic construction of Web pages.

Cross-references

► Web Characteristics and Evolution