Search Engine Architecture

11. Big Data Processing Part Two
Today’s Agenda

• Making Hadoop more efficient
• Dataflow languages
• What’s next?
Hadoop is slow...
A Major Step Backwards?

• MapReduce is a step backward in database access:
  • Schemas are good
  • Separation of the schema from the application is good
  • High-level access languages are good
• MapReduce is poor implementation
  • Brute force and only brute force (no indexes, for example)
• MapReduce is not novel
• MapReduce is missing features
  • Bulk loader, indexing, updates, transactions...

Hadoop vs. Databases: Grep

**Figure 4:** Grep Task Results – 535MB/node Data Set

**Figure 5:** Grep Task Results – 1TB/cluster Data Set

```
SELECT * FROM Data WHERE field LIKE '%XYZ%';
```

Hadoop vs. Databases: Select

Figure 6: Selection Task Results

```
SELECT pageURL, pageRank
FROM Rankings WHERE pageRank > X;
```

Hadoop vs. Databases: Aggregation

**Figure 7:** Aggregation Task Results (2.5 million Groups)

**Figure 8:** Aggregation Task Results (2,000 Groups)

```
SELECT sourceIP, SUM(adRevenue) 
FROM UserVisits 
GROUP BY sourceIP;
```

Hadoop vs. Databases: Join

Figure 9: Join Task Results

“On the first day of logging the Facebook clickstream, more than 400 gigabytes of data was collected. The load, index, and aggregation processes for this data set really taxed the Oracle data warehouse. Even after significant tuning, we were unable to aggregate a day of clickstream data in less than 24 hours.”

Why?

`Integer.parseInt`  
`String.substring`
Schemas are a good idea!

- Parsing fields out of flat text files is slow
- Schemas define a contract, decoupling logical from physical

Thrift

- Originally developed by Facebook, now an Apache project
- Provides a DDL with numerous language bindings
  - Compact binary encoding of typed structs
  - Fields can be marked as optional or required
  - Compiler automatically generates code for manipulating messages
- Provides RPC mechanisms for service definitions
- Alternatives include protobufs, Avro, Parquet

Thrift

struct Tweet {
  1: required i32 userId;
  2: required string userName;
  3: required string text;
  4: optional Location loc;
}

struct Location {
  1: required double latitude;
  2: required double longitude;
}
Dataflow Languages
Need for High-Level Languages

• Hadoop is great for large-data processing!
  • But writing Java programs for everything is verbose and slow
  • Data scientists don’t want to write Java
• Solution: develop higher-level data processing languages
  • Hive: HQL is like SQL
  • Pig: Pig Latin is a bit like Perl

Hive and Pig

• Hive: data warehousing application in Hadoop
  • Query language is HQL, variant of SQL
  • Tables stored on HDFS with different encodings
  • Developed by Facebook, now open source
• Pig: large-scale data processing system
  • Scripts are written in Pig Latin, a dataflow language
  • Programmer focuses on data transformations
  • Developed by Yahoo!, now open source
• Common idea:
  • Provide higher-level language to facilitate large-data processing
  • Higher-level language “compiles down” to Hadoop jobs

Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

```sql
SELECT s.word, s.freq, k.freq FROM shakespeare s
JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
ORDER BY s.freq DESC LIMIT 10;
```

<table>
<thead>
<tr>
<th></th>
<th>Word 1</th>
<th>Word 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>25848</td>
<td>62394</td>
</tr>
<tr>
<td>l</td>
<td>23031</td>
<td>8854</td>
</tr>
<tr>
<td>and</td>
<td>19671</td>
<td>38985</td>
</tr>
<tr>
<td>to</td>
<td>18038</td>
<td>13526</td>
</tr>
<tr>
<td>of</td>
<td>16700</td>
<td>34654</td>
</tr>
<tr>
<td>a</td>
<td>14170</td>
<td>8057</td>
</tr>
<tr>
<td>you</td>
<td>12702</td>
<td>2720</td>
</tr>
<tr>
<td>my</td>
<td>11297</td>
<td>4135</td>
</tr>
<tr>
<td>in</td>
<td>10797</td>
<td>12445</td>
</tr>
<tr>
<td>is</td>
<td>8882</td>
<td>6884</td>
</tr>
</tbody>
</table>

Hive: Behind the Scenes

SELECT s.word, s.freq, k.freq FROM shakespeare s
JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
ORDER BY s.freq DESC LIMIT 10;

(Abstract Syntax Tree)

(one or more of MapReduce jobs)

Hive: Behind the Scenes

Hive Architecture

Hive Implementation

- Metastore holds metadata
  - Databases, tables
  - Schemas (field names, field types, etc.)
  - Permission information (roles and users)
- Hive data stored in HDFS
  - Tables in directories
  - Partitions of tables in sub-directories
  - Actual data in files

Pig!

Pig: Example

Task: Find the top 10 most visited pages in each category

<table>
<thead>
<tr>
<th>User</th>
<th>Url</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>cnn.com</td>
<td>8:00</td>
</tr>
<tr>
<td>Amy</td>
<td>bbc.com</td>
<td>10:00</td>
</tr>
<tr>
<td>Amy</td>
<td>flickr.com</td>
<td>10:05</td>
</tr>
<tr>
<td>Fred</td>
<td>cnn.com</td>
<td>12:00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Url</th>
<th>Category</th>
<th>PageRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnn.com</td>
<td>News</td>
<td>0.9</td>
</tr>
<tr>
<td>bbc.com</td>
<td>News</td>
<td>0.8</td>
</tr>
<tr>
<td>flickr.com</td>
<td>Photos</td>
<td>0.7</td>
</tr>
<tr>
<td>espn.com</td>
<td>Sports</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.
visits = load ‘/data/visits’ as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);
urlInfo = load ‘/data/urlInfo’ as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts, 10);

store topUrls into ‘/data/topUrls’;

Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.
Load Visits

Group by url

Foreach url generate count

Load Url Info

Join on url

Group by category

Foreach category generate top10(urls)

Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.
Pig Script in Hadoop

Load Visits

Group by url

Foreach url generate count

Join on url

Group by category

Foreach category generate top10(urls)

Map

Reduce

Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.
What’s next?

The datacenter *is* the computer!

What’s the instruction set?
Shuffle and Sort: aggregate values by keys

Load Visits

Group by url

Foreach url
generate count

Join on url

Load Url Info

Group by category

Foreach category
generate top10(urls)

Map 1

Reduce 1

Map 2

Reduce 2

Map 3

Reduce 3

Answer?

Pig Slides adapted from Olston et al. (SIGMOD 2008) via Lin et al. Big Data Infrastructure, UMD Spring 2015.
Generically, what is this?

Collections of tuples

Transformations on those collections

Load Visits

Group by url

Foreach url generate count

Load Url Info

Join on url

Group by category

Foreach category generate top10(urls)
Dataflows

- Comprised of:
  - Collections of records
  - Transformations on those collections
- Two important questions:
  - What are the logical operators?
  - What are the physical operators?

Spark

• One popular answer to “What’s beyond MapReduce?”
• Open-source engine for large-scale batch processing
  • Supports generalized dataflows
  • Written in Scala, with bindings in Java, Python, R
• Brief history:
  • Developed at UC Berkeley AMPLab in 2009
  • Open-sourced in 2010
  • Became top-level Apache project in February 2014
  • Commercial support provided by DataBricks

Resilient Distributed Datasets

- RDD: Spark “primitive” representing a collection of records
  - Immutable
  - Partitioned (the $D$ in RDD)
- Transformations operate on an RDD to create another RDD
  - Coarse-grained manipulations only
  - RDDs keep track of lineage
- Persistence
  - RDDs can be materialized in memory or on disk
  - OOM or machine failures: What happens?
- Fault tolerance (the $R$ in RDD):
  - RDDs can always be recomputed from stable storage (disk)

Operations on RDDs

- Transformations (lazy):
  - map
  - flatMap
  - filter
  - union/intersection
  - join
  - reduceByKey
  - groupByKey
  - ...

- Actions (actually trigger computations)
  - collect
  - saveAsTextFile/saveAsSequenceFile
  - ...

Spark Architecture

Spark Physical Operators

Narrow Dependencies:
- map, filter
- union
- join with inputs co-partitioned

Wide Dependencies:
- groupByKey
- join with inputs not co-partitioned

Spark DataFrames

- The hot new RDD (built on RDDs)
  - Column-oriented, schemas, ...
  - Datasets – efficient ORM
- Spark SQL
  - The hot new Shark
  - Tight integration between procedural and relational processing
  - Catalyst optimizer – “don’t bet against the compiler”
  - IndexedRDD – you can see where this is going
- GraphFrames
  - The hot new GraphX
- MLlib
  - Machine learning/fast vector math over DFs
- SparkNet ...
Spark MLlib

- Machine learning/fast vector math over dataframes
- “We observe that often a simple idea is enough: separating matrix operations from vector operations and shipping the matrix operations to be run on the cluster, while keeping vector operations local to the driver.” (Zadeh et al. 2016)

![Figure 2: (a) Benchmarking results for ALS.](image)
Questions?