Search Engine Architecture

12. Stream Processing
Today’s Agenda

• Basics of stream processing
• Sampling and hashing
• Architectures for stream processing
What is a data stream?

- Sequence of items:
  - Structured (e.g., tuples)
  - Ordered (implicitly or timestamped)
  - Arriving continuously at high volumes
  - Not possible to store entirely
  - Sometimes not possible to even examine all items

What to do with data streams?

• Network traffic monitoring
• Datacenter telemetry monitoring
• Sensor networks monitoring
• Credit card fraud detection
• Stock market analysis
• Online mining of click streams
• Monitoring social media streams
What’s the scale? Packet data streams

- Single 2 Gb/sec link; say avg. packet size is 50 bytes
  - Number of packets/sec = 5 million
  - Time per packet = 0.2 microseconds
- If we only capture header information per packet: source/destination IP, time, no. of bytes, etc. – at least 10 bytes
  - 50 MB per second
  - 4+ TB per day
  - Per link!

What if you wanted to do deep-packet inspection?

Source: Minos Garofalakis, Berkeley CS 286
What are the top (most frequent) 1000 (source, dest) pairs seen by R1 over the last month?

SELECT COUNT (R1.source, R1.dest)
FROM R1, R2
WHERE R1.source = R2.source

How many distinct (source, dest) pairs have been seen by both R1 and R2 but not R3?

SET-EXPRESSION QUERY

SELECT COUNT (R1.source, R1.dest)
FROM R1, R2
WHERE R1.source = R2.source

SQL JOIN QUERY

Off-line analysis – Data access is slow, expensive

Source: Minos Garofalakis, Berkeley CS 286
Common Architecture

- Data stream management system (DSMS) at observation points
  - Voluminous streams-in, reduced streams-out
- Database management system (DBMS)
  - Outputs of DSMS can be treated as data feeds to databases

Source: Peter Bonz
DBMS vs. DSMS

**DBMS**
- Model: persistent relations
- Relation: tuple set/bag
- Data update: modifications
- Query: transient
- Query answer: exact
- Query evaluation: arbitrary
- Query plan: fixed

**DSMS**
- Model: (mostly) transient relations
- Relation: tuple sequence
- Data update: appends
- Query: persistent
- Query answer: approximate
- Query evaluation: one pass
- Query plan: adaptive

Source: Peter Bonz
What makes it hard?

• Intrinsic challenges:
  • Volume
  • Velocity
  • Limited storage
  • Strict latency requirements
  • Out-of-order delivery

• System challenges:
  • Load balancing
  • Unreliable message delivery
  • Fault-tolerance
  • Consistency semantics (lossy, exactly once, at least once, etc.)

What exactly do you do?

• “Standard” relational operations:
  • Select
  • Project
  • Transform (i.e., apply custom UDF)
  • Group by
  • Join
  • Aggregations

Issues of Semantics

• Group by... aggregate
  • When do you stop grouping and start aggregating?
• Joining a stream and a static source
  • Simple lookup
• Joining two streams
  • How long do you wait for the join key in the other stream?
• Joining two streams, group by and aggregation
  • When do you stop joining?

What’s the solution?

Windows

- Mechanism for extracting finite relations from an infinite stream
- Windows restrict processing scope:
  - Windows based on ordering attributes (e.g., time)
  - Windows based on item (record) counts
  - Windows based on explicit markers (e.g., punctuations)
  - Variants (e.g., some semantic partitioning constraint)

Windows on Ordering Attributes

- Assumes the existence of an attribute that defines the order of stream elements (e.g., time)
- Let $T$ be the window size in units of the ordering attribute.

\[ t_i' - t_i = T \]

\[ t_{i+1} - t_i = T \]

Source: Peter Bonz
Windows on Counts

- Window of size $N$ elements (sliding, tumbling) over the stream

- Challenges:
  - Problematic with non-unique timestamps: non-deterministic output
  - Unpredictable window size (and storage requirements)

Source: Peter Bonz
Windows from “Punctuations”

- Application-inserted “end-of-processing”
  - Example: stream of actions... “end of user session”
- Properties
  - Advantage: application-controlled semantics
  - Disadvantage: unpredictable window size (too large or too small)

Common Techniques
“Hello World” Stream Processing

- Problem:
  - Count the frequency of items in the stream
- Why?
  - Take some action when frequency exceeds a threshold
  - Data mining:
    raw counts $\rightarrow$ co-occurring counts $\rightarrow$ association rules

The Raw Stream...
Divide Into Windows…

Source: Peter Bonz
First Window

Source: Peter Bonz
Second Window

Source: Peter Bonz
Window Counting

• What’s the issue?
• What’s the solution?

Lessons learned?
Solutions are approximate (or lossy)

General Strategies

- Sampling
- Hashing

Reservoir Sampling

- Task: select \( s \) elements from a stream of size \( N \) with uniform probability
  - \( N \) can be very very large
  - We might not even know what \( N \) is! (infinite stream)
- Solution: Reservoir sampling
  - Store first \( s \) elements
  - For the \( k \)-th element thereafter, keep with probability \( s/k \) (randomly discard an existing element)
- Example: \( s = 10 \)
  - Keep first 10 elements
  - 11th element: keep with 10/11
  - 12th element: keep with 10/12
  - ...

Reservoir Sampling: How does it work?

- Example: $s = 10$
  - Keep first 10 elements
  - 11th element: keep with $10/11$
    
    If we decide to keep it: sampled uniformly by definition
    probability existing item discarded: $10/11 \times 1/10 = 1/11$
    probability existing item survives: $10/11$

- General case: at the $(k + 1)$th element
  - Probability of selecting each item up until now is $s/k$
  - Probability existing element is replaced: $s/(k+1) \times 1/s = 1/(k + 1)$
  - Probability existing element is not replaced: $k/(k + 1)$
  - Probability each element survives to $(k + 1)$th round:
    $(s/k) \times k/(k + 1) = s/(k + 1)$

Hashing for Three Common Tasks

• Cardinality estimation
  • What’s the cardinality of set $S$?
  • How many unique visitors to this page?

• Set membership
  • Is $x$ a member of set $S$?
  • Has this user seen this ad before?

• Frequency estimation
  • How many times have we observed $x$?
  • How many queries has this user issued?

HyperLogLog Log Counter

- Task: cardinality estimation of set
  - `size()` → number of unique elements in the set
- Observation: hash each item and examine the hash code
  - On expectation, 1/2 of the hash codes will start with 1
  - On expectation, 1/4 of the hash codes will start with 01
  - On expectation, 1/8 of the hash codes will start with 001
  - On expectation, 1/16 of the hash codes will start with 0001
  - ...

How do we take advantage of this observation?

Bloom Filters

• Task: keep track of set membership
  • put(x) → insert x into the set
  • contains(x) → yes if x is a member of the set

• Components
  • $m$-bit bit vector

  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

  • $k$ hash functions: $h_1 \ldots h_k$

Bloom Filters: put

\[
\begin{align*}
  h_1(x) &= 2 \\
  h_2(x) &= 5 \\
  h_3(x) &= 11
\end{align*}
\]

Bloom Filters: put

Bloom Filters: contains

$contains \ x$

$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$

Bloom Filters: contains

contains $x$

$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$

AND $\{ A[h_1(x)] , A[h_2(x)] , A[h_3(x)] \} = YES$

Bloom Filters: contains

contains $y$

$h_1(y) = 2$
$h_2(y) = 6$
$h_3(y) = 9$

Bloom Filters: contains

contains \( y \)

\[ h_1(y) = 2 \]
\[ h_2(y) = 6 \]
\[ h_3(y) = 9 \]

\[
\text{AND} \left\{ \begin{array}{l}
A[h_1(y)] \\
A[h_2(y)] \\
A[h_3(y)] \\
\end{array} \right\} = \text{NO}
\]

What's going on here?

Bloom Filters

• Error properties: contains(x)
  • False positives possible
  • No false negatives

• Usage:
  • Constraints: capacity, error probability
  • Tunable parameters: size of bit vector $m$, number of hash functions $k$

Count-Min Sketches

- **Task:** frequency estimation
  - \( \text{put}(x) \rightarrow \text{increment count of } x \text{ by one} \)
  - \( \text{get}(x) \rightarrow \text{returns the frequency of } x \)
- **Components**
  - \( k \) hash functions: \( h_1 \ldots h_k \)
  - \( m \) by \( k \) array of counters

Count-Min Sketches: put

Hat: put $x$

$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$
$h_4(x) = 4$

Count-Min Sketches: put

Count-Min Sketches: put

\[ h_1(x) = 2 \]
\[ h_2(x) = 5 \]
\[ h_3(x) = 11 \]
\[ h_4(x) = 4 \]

Count-Min Sketches: put

Count-Min Sketches: put

$\text{put } y$

$h_1(y) = 6$
$h_2(y) = 5$
$h_3(y) = 12$
$h_4(y) = 2$

Count-Min Sketches: put

put \( y \)

Count-Min Sketches: get

\[ h_1(x) = 2 \]
\[ h_2(x) = 5 \]
\[ h_3(x) = 11 \]
\[ h_4(x) = 4 \]

Count-Min Sketches: get

$h_1(x) = 2$
$h_2(x) = 5$
$h_3(x) = 11$
$h_4(x) = 4$

$\text{MIN}\left\{ A[h_1(x)], A[h_2(x)], A[h_3(x)], A[h_4(x)] \right\} = 2$

Count-Min Sketches: get

$y$

$\begin{align*}
  h_1(y) &= 6 \\
  h_2(y) &= 5 \\
  h_3(y) &= 12 \\
  h_4(y) &= 2
\end{align*}$

Count-Min Sketches: get

\[ h_1(y) = 6 \]
\[ h_2(y) = 5 \]
\[ h_3(y) = 12 \]
\[ h_4(y) = 2 \]

\[ \text{MIN} \left\{ A[h_1(y)], A[h_2(y)], A[h_3(y)], A[h_4(y)] \right\} = 1 \]

Count-Min Sketches

• Error properties:
  • Reasonable estimation of heavy-hitters
  • Frequent over-estimation of tail

• Usage:
  • Constraints: number of distinct events, distribution of events, error bounds
  • Tunable parameters: number of counters \( m \), number of hash functions \( k \), size of counters

Stream Processing Architectures
Storm

- Open-source real-time distributed stream processing system
  - Started at BackType
  - BackType acquired by Twitter in 2011
  - Now an Apache project
- Storm aspires to be the Hadoop of real-time processing!

Storm Topologies

- Storm topologies = “job”
  - Once started, runs continuously until killed
- A Storm topology is a computation graph
  - Graph contains nodes and edges
  - Nodes hold processing logic (i.e., transformation over its input)
  - Directed edges indicate communication between nodes

Streams, Spouts, and Bolts

• Streams
  - The basic collection abstraction: an unbounded sequence of tuples
  - Streams are transformed by the processing elements of a topology

• Spouts
  - Stream generators
  - May propagate a single stream to multiple consumers

• Bolts
  - Subscribe to streams
  - Streams transformers
  - Process incoming streams and produce new ones

Stream Groupings

- Bolts are executed by multiple workers in parallel
- When a bolt emits a tuple, where should it go?
- Stream groupings:
  - Shuffle grouping: round-robin
  - Field grouping: based on data value

Storm: Example

// instantiate a new topology
TopologyBuilder builder = new TopologyBuilder();

// set up a new spout with five tasks
builder.setSpout("spout", new RandomSentenceSpout(), 5);

// the sentence splitter bolt with eight tasks
builder.setBolt("split", new SplitSentence(), 8)
    .shuffleGrouping("spout"); // shuffle grouping for the output

// word counter with twelve tasks
builder.setBolt("count", new WordCount(), 12)
    .fieldsGrouping("split", new Fields("word")); // field grouping

// new configuration
Config conf = new Config();

// set the number of workers for the topology; the 5x8x12=480 tasks
// will be allocated round-robin to the three workers, each task
// running as a separate thread
conf.setNumWorkers(3);

// submit the topology to the cluster
StormSubmitter.submitTopology("word-count", conf, builder.createTopology());

Spark Streaming

Discretized stream processing:
Run a streaming computation as a series of very small, deterministic batch jobs

Continuous Operator Model
Discretized Streams

Source: Zaharia et al. (SOSP 2013)
Spark and Spark Streaming

Source: Zaharia et al. (SOSP 2013)