Systems Infrastructure for Data Science

Web Science Group
Uni Freiburg
WS 2012/13
Data Stream Processing
Topics

• Model Issues
• System Issues
• Distributed Processing
• Web-Scale Streaming
System Issues

• Architecture and Run-time operation

• Resource limitations
  – CPU
  – Memory
  – Bandwidth (distributed case)

• Performance goals
  – Low latency
  – High throughput
  – Maximum QoS utility
  – Minimum error
General Concerns

• In principle, same architecture choices as in databases

• Different tradeoffs:
  – Latency bounds more important than throughput
  – Processing driven by data arrival, not query optimization

• Architecture changes:
  – Push-based execution more popular (why?)
  – Decoupling using queues
  – Adaptive processing
System Issues

• Two systems as case studies:
  – Aurora [Brandeis-Brown-MIT]
  – STREAM [Stanford]
System Issues in Aurora
Aurora System Model
Aurora Quality of Service (QoS)

- Latency QoS

- Loss-tolerance QoS

- Value-based QoS
Aurora Architecture
Operator Scheduling

- **Goal**: To allocate the CPU among multiple queries with multiple operators so as to optimize a metric, such as:
  - minimize total average latency
  - maximize total average latency QoS utility
  - maximize total average throughput
  - minimize total memory consumption

- Deciding which operator should run next, for how long or with how much input.
- Must be low overhead.
Why should the DSMS worry about scheduling?
Thread-based vs. State-based Execution
Batching

• Exploit inter-box and intra-box non-linearities in execution overhead

• Train scheduling
  – batching and executing multiple tuples together

• Superbox scheduling
  – batching and executing multiple boxes together
Batching reduces execution costs
Distribution of Execution Overhead

Total running time:
- 410 secs
- 15 secs
- 8.5 secs

Relative Overhead

- Worker Thread
- Storage Manager
- Scheduler

Uni Freiburg, WS2012/13 Systems Infrastructure for Data Science
The Overload Problem

• If Load > Capacity during the spikes, then queues form and latency proliferates.

• Given a query network $N$, a set of input streams $I$, and a CPU with processing capacity $C$; when $Load(N(I)) > C$, transform $N$ into $N'$ such that:
  
  – $Load(N'(I)) < C$, and
  
  – $Utility(N(I)) - Utility(N'(I))$ is minimized.
Load Shedding in Aurora

Aurora Query Network

- Key questions:
  - when to shed load?
  - where to shed load?
  - how much load to shed?
  - which tuples to drop?

- Problem: When load > capacity, latency QoS degrades.
- Solution: Insert drop operators into the query plan.
- Result: Deliver “approximate answers” with low latency.
The Drop Operator

• is an abstraction for load reduction
• can be added, removed, updated, moved
• reduces load by a factor
• produces a “subset” of its input
• picks its victims
  – probabilistically
  – semantically (i.e., based on tuple content)
When to Shed Load?

- **Load coefficients**

\[
R_i \to \text{cost}_1 \text{ sel}_1 \to \text{cost}_2 \text{ sel}_2 \to \cdots \to \text{cost}_n \text{ sel}_n
\]

\[
L_i = \sum_{j=1}^{n} \left( \prod_{k=1}^{j-1} \text{sel}_k \right) \times \text{cost}_j
\]

(CPU cycles per tuple)

- **Total load**

\[
\sum_{i=1}^{m} L_i \times R_i
\]

(CPU cycles per time unit)
Aurora Load Shedding
Three Basic Principles

1. Minimize run-time overhead.
2. Minimize loss in query answer accuracy.
3. Deliver subset results.
Principle 1: Plan in advance.

<table>
<thead>
<tr>
<th>Excess Load</th>
<th>Drop Insertion Plan</th>
<th>QoS Cursors</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td><img src="#" alt="Diagram for 10%" /></td>
<td><img src="#" alt="Diagram for QoS Cursors 10%" /></td>
</tr>
<tr>
<td>20%</td>
<td><img src="#" alt="Diagram for 20%" /></td>
<td><img src="#" alt="Diagram for QoS Cursors 20%" /></td>
</tr>
<tr>
<td>300%</td>
<td><img src="#" alt="Diagram for 300%" /></td>
<td><img src="#" alt="Diagram for QoS Cursors 300%" /></td>
</tr>
</tbody>
</table>

- shed less!
- shed more!
Principle 2: Minimize error.

- Early drops save more processing cycles.
- Drops before sharing points can cause more accuracy loss.
- We rank possible drop locations by their loss/gain ratios.
Principle 3: Keep sliding windows intact.

- Two parameters: size and slide
- Example: Trades(time, symbol, price, volume)
Windowed Aggregation

• Apply an aggregate function on the window
  – Average, Sum, Count, Min, Max
  – User-defined

• Can be nested

• Example:

```
Filter symbol="IBM"
Slider ω = 5 min
δ = 5 min
diff = high-low
Aggregate diff > 5
Filter ω = 60 min
δ = 60 min
count > 0
Aggregate count > 0
Filter
```
Dropping from an Aggregation Query

Tuple-based Approach

• Drop before: non-subset result of nearly the same size

• Drop after: subset result of smaller size

\[
\bar{\omega} = 3 \\
\bar{\delta} = 3
\]

\[
.. 30 15 30 20 10 30 \rightarrow \text{Average} \\
\omega = 3 \\
\delta = 3 \rightarrow .. 25 20 \rightarrow ....
\]

\[
\bar{\omega} = 3 \\
\bar{\delta} = 3
\]

\[
.. 30 15 30 20 10 30 \rightarrow \text{Drop} \\
p = 0.5 \rightarrow .. 30 15 30 20 10 30 \rightarrow \text{Average} \\
\omega = 3 \\
\delta = 3 \rightarrow .. 15 15 \rightarrow ....
\]

\[
\bar{\omega} = 3 \\
\bar{\delta} = 3
\]

\[
.. 30 15 30 20 10 30 \rightarrow \text{Average} \\
\omega = 3 \\
\delta = 3 \rightarrow .. 25 20 \rightarrow \text{Drop} \\
p = 0.5 \rightarrow .. 25 20 \rightarrow ....
\]
Dropping from an Aggregation Query

Window-based Approach

• Drop before: subset result of smaller size

• Window-aware load shedding
  – works with any aggregate function
  – delivers correct results
  – keeps error propagation under control
  – can handle nesting
  – can drop load early
System Issues in STREAM
STREAM Query Plans

• Query in CQL -> Physical query plan tree

```
SELECT *
FROM S1 [ROWS 1000], S2 [RANGE 2 MINUTES]
WHERE S1.A = S2.A AND S1.A > 10
```
# STREAM Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Operator Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>select</td>
<td>relation-to-relation</td>
<td>Filters elements based on predicate(s)</td>
</tr>
<tr>
<td>project</td>
<td>relation-to-relation</td>
<td>Duplicate-preserving projection</td>
</tr>
<tr>
<td>binary-join</td>
<td>relation-to-relation</td>
<td>Joins two input relations</td>
</tr>
<tr>
<td>mjoin</td>
<td>relation-to-relation</td>
<td>Multiway join from [22]</td>
</tr>
<tr>
<td>union</td>
<td>relation-to-relation</td>
<td>Bag union</td>
</tr>
<tr>
<td>except</td>
<td>relation-to-relation</td>
<td>Bag difference</td>
</tr>
<tr>
<td>intersect</td>
<td>relation-to-relation</td>
<td>Bag intersection</td>
</tr>
<tr>
<td>antisemijoin</td>
<td>relation-to-relation</td>
<td>Antisemijoin of two input relations</td>
</tr>
<tr>
<td>aggregate</td>
<td>relation-to-relation</td>
<td>Performs grouping and aggregation</td>
</tr>
<tr>
<td>duplicate-eliminate</td>
<td>relation-to-relation</td>
<td>Performs duplicate elimination</td>
</tr>
<tr>
<td>seq-window</td>
<td>stream-to-relation</td>
<td>Implements time-based, tuple-based, and partitioned windows</td>
</tr>
<tr>
<td>i-stream</td>
<td>relation-to-stream</td>
<td>Implements $Istream$ semantics</td>
</tr>
<tr>
<td>d-stream</td>
<td>relation-to-stream</td>
<td>Implements $Dstream$ semantics</td>
</tr>
<tr>
<td>r-stream</td>
<td>relation-to-stream</td>
<td>Implements $Rstream$ semantics</td>
</tr>
</tbody>
</table>
STREAM Queues

- Queues encapsulate the typical producer-consumer relationship between the operators.
- They act as in-memory buffers.
- They enforce that tuple timestamps are non-decreasing.

- Why is this necessary?
  - Heartbeat mechanism for time management
STREAM Heartbeats in a Nutshell

- **Problem:** Out of order data arrival
  - Unsynchronized application clocks at the sources
  - Different network latencies from different sources to the DSMS
  - Data transmission over a non-order-preserving channel
- **Solution:** Order tuples at the input manager by generating heartbeats based on application-specified parameters
  - Heartbeat value $T$ at a given time instant means that all tuples after that instant will have a timestamp greater than $T$. 
STREAM Synopses

• A synopsis stores the internal state of an operator needed for its evaluation.
  – Example: A windowed join maintains a hash table for each of its inputs as a synopsis.

  ➢ Do we need synopses for all types of operators?

• Like queues, synopses are also kept in memory.

• Synopses can also be used in more advanced ways:
  – shared among multiple operators (for space optimization)
  – store summary of stream tuples (for approximate processing)
STREAM Performance Issues
Synopsis Sharing for Eliminating Data Redundancy

- Replace identical synopses with “stubs” and store the actual tuples in a single store.
- Also for multiple query plans.

```
SELECT * 
FROM S1 [ROWS 1000], S2 [RANGE 2 MINUTES]
WHERE S1.A = S2.A AND S1.A > 10
```

```
SELECT A, MAX(B) 
FROM S1 [ROWS 200]
GROUP BY A
```
STREAM Performance Issues

Exploiting Constraints for Reducing Synopsis Sizes

• Constraints on data and arrival patterns to reduce, bound, eliminate memory state

• Schema-level constraints
  – Clustering (e.g., contiguous duplicates)
  – Ordering (e.g., slack parameter in SQuAl)
  – Referential integrity (e.g., timestamp synchronization)
  – In relaxed form: k-constraints (k: adherence parameter)

• Simple example:
  – Orders (orderID, customer, cost)
  – Fulfillments (orderID, portion, clerk)
  – If Fulfillments is k-clustered on orderID, can infer when to discard Orders.
STREAM Performance Issues
Exploiting Constraints for Reducing Synopsis Sizes

• Data-level constraints: “Punctuations”
• Punctuations are special annotations embedded in data streams to specify the end of a subset of data.
  – No more tuples will follow that match the punctuation.
• A punctuation is represented as an ordered set of patterns, where each pattern corresponds to an attribute of a tuple.
  – Patterns: *, constants, ranges [a, b] or (a b), lists {a, b, ..}, Ø
  – Example: < item_id, buyer_id, bid >
    < {10, 20}, *, * > => all bids on items 10 and 20.
STREAM Performance Issues
Operator Scheduling for Reducing Intermediate State

• A global scheduler decides on the order of operator execution.

• Changing the execution order of the operators does not affect their semantic correctness, but may affect system’s total memory utilization.
Example Query Plan:

- OP1: cost = 1, selectivity = 0.2
- OP2: cost = 1, selectivity = 0

Input Arrival Pattern:

- Greedy always prioritizes OP1.
- FIFO schedules OP1-OP2 in sequence.
  - Greedy has smaller max. queue size.
- (Chain Scheduling Algorithm)

Total Queue Sizes for two alternative scheduling policies:

<table>
<thead>
<tr>
<th>Time</th>
<th>Greedy scheduling</th>
<th>FIFO scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>1.4</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td>1.6</td>
<td>2.2</td>
</tr>
<tr>
<td>4</td>
<td>1.8</td>
<td>3.0</td>
</tr>
<tr>
<td>5</td>
<td>2.0</td>
<td>3.2</td>
</tr>
<tr>
<td>6</td>
<td>2.2</td>
<td>4.0</td>
</tr>
</tbody>
</table>