The Design of the Borealis Stream Processing Engine

CIDR 2005
Brandeis University, Brown University, MIT

Presented by Mengmeng Liu (mengmeng@seas.upenn.edu)
Several slides in courtesy of Magdalena Balazinska, et al.

Distributed Stream Processing

- Data sources push tuples continuously
- Operators process windows of tuples

Others work
- Data models, operators, query languages
- Efficient single-site processing
  [STREAM, TelegraphCQ, NiagaraCQ, Gigascope, Aurora]
- Basic distributed systems
  - Servers [TelegraphCQ, Medusa/Aurora]
  - or sensor networks [TinyDB, Cougar]
  - Tolerance to node failures
  - Simple load management

Outline
- Three fundamental requirements
  - Dynamic revision of query results
    - Revision on input streams
    - Fault-tolerance
  - Dynamic query modification
    - Control lines
    - Time travel
  - Dynamic and distributed query optimization

Causes for Tuple Revisions
- Tuple revisions: insertion, deletion and replacement
- Data sources revise input streams
- Temporary overloads cause tuple drops
- Temporary failures cause tuple drops
- Counterparts: Turnstile model
  - Insertions and deletions, model replacement using delta

Aurora Data Model
- header data
  (time, a1, ..., an)
- time: tuple timestamp
New Data Model for Revisions

- **header**: \( (\text{time}, \text{type}, \text{id}, a_1, \ldots, a_n) \)
- **time**: tuple timestamp
- **type**: revision type
  - insertion, deletion, replacement
- **id**: unique identifier of tuple on stream

Revision Processing in Borealis

- Closed model: revisions produce revisions

Discussion of revision processing

- Stateless vs. stateful operators
- Processing cost vs. storage
- Revision proliferation: messages can increase exponentially in the number of stateful boxes.
- Tolerate imprecision (a rich field)
  - Whether app is interested in revision msg or not
  - Application’s QoS requirement
  - Semantic load shedding

Fault-Tolerance through Replication

- **Goal**: Tolerate node and network failures

Fault-Tolerance Approach

- If an input stream fails, find another replica
- No replica available, produce tentative tuples
- Correct tentative results after failures
Outline

Three fundamental requirements
- Dynamic revision of query results
  - Revision on input streams
  - Fault-tolerance
- Dynamic query modification
  - Control lines
  - Time travel
- Dynamic and distributed query optimization

Dynamic query modification
- Problem
  - Change certain attributes of the query at runtime
- Goal
  - Low overhead, fast, and automatic modifications
- Dynamic query modification
  - Control lines
    - Provided to each operator box
    - Indicate when the change in box semantics should take effect
  - Timing problem of data and control lines
    - Specify precisely what data is to be processed according to what control parameters
    - Data is ready for processing too late or too early
      - new data vs. old parameter -> buffered old parameters
      - old data vs. new parameter -> revision message and time travel

Time travel
- Motivation
  - Rewind history and then repeat it
  - Move forward into the future
- Connection Point (CP)
  - Store messages for specific query
  - Support ad-hoc query
- CP View
  - Independent view of CP
  - View range
    - Start time
    - Max time
  - Operations to enable time travel
    - Replay: replay a specified set of message within the view’s range
    - Undo: produce deletion revision for a specified set of message
- Time travel and Revision Records
  - Roll back: Undo and Replay
  - Travel future: prediction function

Optimization in a Distributed SPE
- Goal: Optimized resource allocation
- Challenges:
  - Wide variation in resources
    - High-end servers vs. tiny sensors
  - Multiple resources involved
    - CPU, memory, I/O, bandwidth, power
  - Dynamic environment
    - Changing input load and resource availability
  - Scalability
    - Query network size, number of nodes

Quality of Service
- A mechanism to drive resource allocation
- Aurora model
  - QoS functions at query end-points
  - Problem: need to infer QoS at upstream nodes
- An alternative model
  - Vector of Metrics (VM) carried in tuples
    - Content: tuple’s semantic importance, or its age
    - Performance: arrival time, total resource consumed, ...
  - Operators can change VM
  - A Score Function to rank tuples based on VM
    - Optimizers can keep and use statistics on VM
Borealis Optimizer Hierarchy

End-point Monitor(s)  Global Optimizer
Local Monitor      Neighborhood Optimizer
Local Optimizer

Optimization Tactics
- Priority scheduling - Local
- Modification of query plans - Local
  - Changing order of commuting operators
  - Using alternate operator implementations
- Allocation of query fragments to nodes - Neighborhood, Global
- Load shedding - Local, Neighborhood, Global

Correlation-based Load Distribution
- Under the situation that network BW is abundant and network transfer delays are negligible
- Goal: Minimize end-to-end latency
- Key ideas:
  - Balance load across nodes to avoid overload
  - Group boxes with small load correlation together
  - Maximize load correlation among nodes

Load Shedding
- Goal: Remove excess load at all nodes and links
- Shedding at node A relieves its descendants
- Distributed load shedding
  - Neighbors exchange load statistics
  - Parent nodes shed load on behalf of children
- Uniform treatment of CPU and bandwidth problem
- Load balancing or Load shedding?

Local Load Shedding

<table>
<thead>
<tr>
<th>Node A</th>
<th>Node B</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>f2</td>
</tr>
<tr>
<td>4C</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>No overload</td>
<td>No overload</td>
</tr>
</tbody>
</table>

Plan  Loss
Local  App. 25%  App. 58%

Distributed Load Shedding

<table>
<thead>
<tr>
<th>Node A</th>
<th>Node B</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>f2</td>
</tr>
<tr>
<td>4C</td>
<td>3C</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>No overload</td>
<td>No overload</td>
</tr>
</tbody>
</table>

Plan  Loss
Local  App. 25%  App. 58%
Distributed  App. 9%  App. 64%
Extending Optimization to Sensor Nets
- Sensor proxy as an interface
- Moving operators in and out of the sensor net
- Adjusting sensor sampling rates

Conclusions
- Second generation streaming applications require a flexible processing model
  - Distributed operation
  - Dynamic result and query modification
  - Dynamic and scalable optimization
  - Server and sensor network integration
  - Tolerance to node and network failures
- http://www.cs.brown.edu/research/borealis

Discussion
- Homogeneous input stream
- Measurements: maximum average QoS at system outputs.
- Handling missing and out-of-order data
- Integrate stored and streaming data
- Security
- One fits all solution?

Questions?