Operator Scheduling in a Data Stream Manager

Don Carney
Uğur Çetintemel
Alex Rasin
Stan Zdonik
Mitch Cherniack
Michael Stonebraker

Brown University
Brown University
Brown University
Brown University
Brandeis University
MIT
Stream-based Applications

- **Examples**
  - Traffic analysis
    - Streams of automobile locations
  - Market analysis
    - Streams of stock ticker data
  - Sensor monitoring
    - Streams of soldier locations

- **Characteristics**
  - Lots of data sources
  - Unpredictable and high rates of input
  - Latency expectations / deadlines
  - Timely & Sophisticated processing
Each **Application** Provides:

- A **Query** over input data streams
- A Quality-Of-Service Specification (**QoS**)
  (specifies utility of results)
A Look Inside
Scheduling in Action

Diagram showing operations such as \( \sigma \) (selection), \( \mu \) (set union), \( \text{Agg} \) (aggregation), and \( \text{App} \) (application). Arrows indicate the flow of data or operations.
Traditional Thread-driven Execution

- Thread per query or operator
- Resource management done by OS
  - Easy to program
  - Problems
    - No Application specific QoS
    - Scalability
Basic Architecture

“How to make this light-weight enough to meet QoS constraints under heavy load”
Aurora vs. Thread-Based

Average Latency (seconds)

Number of Boxes

Thread-per-box
Aurora
Scheduler Specifics

- Overhead reduction
- Box execution order
- Scalability
Minimizing Per Tuple Overhead

Tuple at a time:

... z y x

A (z) A (y) A (x)

B (A (z)) B (A (y)) B (A (x))

Train Scheduling:

Tuple Trains:

... z y x

A (z, y, x)

B (A (z), A (y), A (x))

Box & Tuple Trains (superbox):

... z y x

AB

B (A (z,y,x))

= Scheduler Action

VLDB 2003 Berlin 10
Scheduling Superboxes incurs lowest overhead

![Bar chart showing scheduling types and overhead time]

- **Tuple-at-a-Time**
  - Worker Thread: 15 seconds
  - Storage Manager: 8.5 seconds
  - Scheduling: 410 seconds
- **Tuple-Trains**
- **Superbox**

Overhead (seconds spent in CPU)
Superboxes provide best performance

- 5 Apps
- 500 boxes
Traversal Matters

Min-Cost Traversal
B1 → B2 → B3

Processing Cost
• Execution of Box

Call Overhead
• Context Switch

Average Latency
• Measured as average of above 2

<table>
<thead>
<tr>
<th></th>
<th>Processing (p)</th>
<th>Call Overhead (o)</th>
<th>Avg Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min-Cost</td>
<td>4p</td>
<td>3o</td>
<td>3p5o</td>
</tr>
</tbody>
</table>

VLDB 2003 Berlin
Traversals Matter

Min-Latency Traversal

Processing Cost
- Execution of Box
- Context Switch

Call Overhead

Average Latency
- Measured as average of above 2

<table>
<thead>
<tr>
<th></th>
<th>Processing (p)</th>
<th>Call Overhead (o)</th>
<th>Avg Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min-Cost</td>
<td>6p</td>
<td>3o</td>
<td>4.5p+3o</td>
</tr>
<tr>
<td>Min-Latency</td>
<td>6p</td>
<td>5o</td>
<td>3.25p+2.56o</td>
</tr>
</tbody>
</table>
Superbox Traversal

- **Box execution order to improve**
  - Throughput (Min-Cost)
    - Minimizes number of box calls
  - Latency (Min-Latency)
    - Produces tuples fastest
  - Memory Usage (Min-Memory)
    - Maximizes consumption of data per unit time

- **Traversal selection based on**
  - Targeted overhead
  - Achieving best QoS
Priority Assignment

\[ \text{p-tuple ordering:} \]
- Slope
- Slack

Critical Points

\[ \text{Current Tuple Latency} \quad \text{Expected Output Latency} \quad \text{Latency} \]

\[ \text{Slope} \]

\[ \text{Cost} \quad \text{Slack} \]

\[ \text{QoS} \]

\[ \text{1} \]

\[ \text{0} \]
P-tuple Ordering

- At each scheduling event
  1. Compute **p-tuple** for each box
  2. Sort

- Example:

<table>
<thead>
<tr>
<th>p-tuple</th>
<th>Slope</th>
<th>Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>b2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>b3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p-tuple</th>
<th>Slope</th>
<th>Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>b2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>b3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>b1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Priority Assignment Matters

20 applications
100 boxes
2 QoS graphs
1. Loose
2. Tight

Graph showing the relationship between load and average QoS for P-tuple and Round-Robin scheduling strategies.
Approximation for Scalability

- P-tuple method is slow
  - Compute for each box
  - Sort costly for large numbers of boxes

- Approximation to trade off quality for overhead
  - Bucketing
  - Pre-computation
Bucketing

Approach:

- Partition slope/slack space into buckets
- At Scheduling event
  - Assign boxes to buckets
  - Traverse buckets in p-tuple order
- # buckets controls approximation

But we still have to compute slope and slack
Pre-Computing Bucket Assignments

Given Latency, finding bucket in $O(1)$
Bucketing Works

200 Apps
1000 Boxes

2 QoS graph types
- Loose
- Tight

Better Approximation
with low overhead

Poor Approximations

Scheduling Overhead / Running Time

Average QoS

Num Partitions

BUCKETING
P-TUPLE
Related Work

- **Operating Systems**
  - [HLC91], [JRR97], [L88], [RS94]

- **Real-time Databases**
  - [AG93], [HCL-VLDB93], [KG94], [OS95], [R93]

- **DSMS**
  - Chain [BBDM-SIGMOD03]
    - Focuses on minimizing run-time memory usage
  - Eddies [AH-SIGMOD00]
    - Adaptability
Conclusions

- Overhead matters
  - Algorithms to reduce overhead
- Addressed QoS issues
  - Approximation technique trades scheduling quality for overhead
- Experimental investigation of scheduling algorithms
  - Run on Aurora prototype