Query Processing

Jon Frankel, Noi Jencharat, Ened Ketri, Anurag Maskey, Andy See, Larissa Smelkov

3/25/03

Opening Game - Who am I?

- Professor at the University of Wisconsin – Madison
- Specializing in database performance issues (i.e. joins)
- **Bonus**: What stream system have I worked on?

Query Processing – Papers

- Stratis Viglas and Jeffrey F. Naughton. *Rate-based query optimization for streaming information sources*. SIGMOD Conference 2002

Query Processing – Today’s Agenda

- 1:40 Motivation & Setup Examples
- 2:20 Rate Based Query Paper
- 2:50 Break
- 3:00 Window Joins Paper
- 3:30 K-Constraints Paper
- 4:00 Discussion
127 Flashback

- Optimizer - cost based
- Select * from students where major = ‘cosi’ and birthday = ‘0325’

Stream Challenges

- Final Answer?
- Block Reads?
- Cardinality?

Stream Challenges

- Final Answer?
- Block Reads?
- Cardinality?

Rate Based Analysis

<table>
<thead>
<tr>
<th>Day</th>
<th>New</th>
<th>MFC</th>
<th>Left</th>
<th>JDF</th>
<th>Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>6–3</td>
<td>0</td>
<td>3–1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>9–4</td>
<td>0</td>
<td>3–1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>10–5</td>
<td>2</td>
<td>3–1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2–1</td>
<td>0</td>
<td>3–1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rate Based Analysis
MFC – 10/day; JDF 3/day

<table>
<thead>
<tr>
<th>Day</th>
<th>New</th>
<th>MFC</th>
<th>Left</th>
<th>JDF</th>
<th>Left</th>
<th>JDF</th>
<th>Left</th>
<th>MFC</th>
<th>Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>12</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

127 Flashback – Joins

- Predicate Pushdown
- Select * from students as s, courses as c where s.major = 'cosi'
  and c.dept = 'cosi'
  and s.sid = c.sid

Cost Optimization - Speed??

Coming Up….
- Different ways to measure rates
- SPJ applicability
Stream Challenges II

- Blocking Query Operators
  - (option: pipelined join)
- Lost/Delayed/Unordered Data
- And yet, benefits are huge…

Stock Market – Econ 2A
Stock prices are based on?

Data is Out there!
(http://biz.yahoo.com/cc/)

Thu Mar 20 Times are U.S. Eastern
8:30 am  CYCL Centennial Communications Earnings (Q3 2003)
8:30 am  DV DeVry Inc. Acquires Ross University
8:30 am  ENTG Entegris, Inc. Earnings (Q2 2003)
8:30 am  PLUG Please Announcement
9:00 am  HOLL Hollywood Media Corp. Fourth Quarter and Year-End 2002
9:00 am  LEH Lehman Brothers Holdings First Quarter 2003 Earnings
10:00 am  FNLY Finlay Enterprises, Inc. Earnings (Q4 2002)
10:00 am  GIII G-III Apparel Group Earnings (Q4 2003)
10:00 am  GLY NY Galen's Trading Company, Inc. Fourth Quarter 2002
10:00 am  MDX Morgan Stanley Earnings (Q1 2003)
10:00 am  TRMS TrimTabs, Inc. Earnings (Q4 2002)
10:30 am  GPN Global Payments Inc. Earnings (Q3 2003)
11:00 am  GDTH BioSex's Agreement/Drug Eluding Stent Update
11:00 am  CRAI Charles River Associates Earnings (Q1 2003)
11:00 am  CHKX Checkers Drive-In Restaurants Earnings (Q4 2002)
11:00 am  CPWM Cost Plus Earnings (Q4 2002)
11:00 am  JCREW J. Crew Group, Inc. Earnings (Q4 2003)
**Query:** Short-term Downward Momentum:
Find all NASDAQ stocks between $20 and $200 that have moved down more than 2% in the last 20 minutes and there has been significant buying pressure (70% or more of the volume has traded toward the ask price) in the last 2 minutes.

Or by:
Earnings, News, Industry

---

**Aurora Example**

<table>
<thead>
<tr>
<th>Soldiers (time, ID, pos)</th>
<th>Tanks (time, ID, pos)</th>
<th>Problem?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, S1, A</td>
<td>1, T1, C</td>
<td>A S1, S2</td>
</tr>
<tr>
<td>1, S2, A</td>
<td></td>
<td>T2, S3, C T1</td>
</tr>
<tr>
<td>2, S1, A</td>
<td>2, T1, C</td>
<td>A S1</td>
</tr>
<tr>
<td>1, S3, B</td>
<td>1, T2, B</td>
<td>B S3, C T1, T2, S3</td>
</tr>
<tr>
<td>2, S3, B</td>
<td>2, T2, C</td>
<td>A T1, S1</td>
</tr>
<tr>
<td>3, S1, A</td>
<td>3, T1, A</td>
<td>B S3, C T2, S3</td>
</tr>
<tr>
<td>3, S3, B</td>
<td>3, T2, C</td>
<td></td>
</tr>
<tr>
<td>2, S2, C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3, S2, B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Join Challenges – Window Options**

- Aurora Option (by individual tuple)
- Stream Option (slide)

Tuple vs Timestamp

Order!
Join Challenges – Window Options

- Aurora Option (by individual tuple)
- Stream Option (slide)

Tuple vs Timestamp

<table>
<thead>
<tr>
<th>A</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>D1</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>D2</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>B2</td>
<td>C2</td>
</tr>
</tbody>
</table>

Order!

Join Challenges – Window Options

- Aurora Option (by individual tuple)
- Stream Option (slide)

Tuple vs Timestamp

<table>
<thead>
<tr>
<th>A</th>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>D1</td>
</tr>
<tr>
<td>F</td>
<td>D2</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>B2</td>
<td>C2</td>
</tr>
</tbody>
</table>

Order!

Accuracy – How to window?

Coming Up….
- Joining algorithms
- Lots of cool graphs
Motivations

- Traditional Optimizers requires cardinality of the input...
- In streams, cardinality is not known and inputs come at different rate...
- RATE-BASED optimization

What is Rate?

- Number of records per a unit of time.
- Output Rate = \frac{\# \text{ output transmitted}}{\text{time needed for transmission}}

Output Rate Estimation

- For Projections
- For Selections
- For Joins

Output Rate for Projections

- case 1: Mitch
  - Time to read papers is shorter than time between getting the papers
  - So the output rate = the input rate
Output Rate for Projections

- case 2: Jon
  Time to read papers is longer than time between getting the papers

So the output rate = 1/(time to do projection)

In general, time to do projection is low.
So

Output Rate = Input Rate

Output Rate for Selections

- Selectivity (f) = percentage of papers that will be selected

Output Rate for Selection

- case 1: Mitch
takes 1/2 hour to read 1 paper, with selectivity = 0.5

output rate = 1/2 paper/hour
So the output rate = f * the input rate
Output Rate for Selection

- case 2: John takes 1.5 hour to read 1 paper, with selectivity = 0.5
  
  output rate = 1/3 paper/hour (= 1/2 * 1/1.5)

  So the output rate = f * (1/time to select)

Output Rate for Selections

- In general, time to perform selection is less than interval between inputs.
- So

  Output Rate = Selectivity * Input Rate

  \[ r_o \times f \times r_i \]

Output Rate for Joins

- What are the papers by same author Mitch and Jon gives the same grading to?
- \[ r_M = \text{No. of papers Mitch reads per hour} \]
- \[ r_J = \text{No. of papers Jon reads per hour} \]
- \[ f = \text{Selectivity of join} \]
- \[ C_M = \text{Time to handle reviews from Mitch} \]
- \[ C_J = \text{Time to handle reviews from Jon} \]

Recall

- Output Rate =

  \[ \frac{\text{# output transmitted}}{\text{time needed for transmission}} \]

  Total #’s of papers in output

  Total time to do the Join
Output Tuples

- time interval = t:
  We have:
  \( r_M^t \) paper reviews from Mitch
  \( r_J^t \) paper reviews from Jon

\( f^* r_M^t r_J^t \) tuples that can be in the output.

---

after 1 hour

Number of output tuples:
\( f^* r_M^t r_J^t \)

---

after 2 hours

Number of output tuples:
\( f^* r_M^t r_J^t + f^* r_M^t 2r_J^t + f^* 2r_M^t r_J^t - f^* r_M^t r_J^t \)

---

after 2 hours

Number of output tuples:
\( f^* r_M^t r_J^t + 3^* f^* r_M^t r_J^t \)
after 3 hours

Number of output tuples:
\[ f \cdot r_M \cdot r_J + 3 \cdot f \cdot r_M \cdot r_J + f \cdot r_M \cdot 3r_J + f^3 r_M \cdot r_J - f^3 r_M \cdot r_J \]

Number of output tuples:
\[ 5 \cdot f \cdot r_M \cdot r_J \]

after time \( t \)

- There will be \( (2t-1) f \cdot r_M \cdot r_J \) new tuples in the output.

- Total number of outputs at time \( t \):
  \[ \int ((2t-1) f \cdot r_M \cdot r_J) \, dt = t^2 \cdot f \cdot r_M \cdot r_J - t^3 \cdot f \cdot r_M \cdot r_J = f \cdot r_M \cdot r_J \cdot t \cdot (t-1) \]

Time to Process Join

- at time \( t \)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Time</th>
<th>Processing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mitch</td>
<td>( r_M \cdot t )</td>
<td>( C_M )</td>
</tr>
<tr>
<td>Jon</td>
<td>( r_J \cdot t )</td>
<td>( C_J )</td>
</tr>
</tbody>
</table>

Total time = \( r_M \cdot t \cdot C_M + r_J \cdot t \cdot C_J \)

= \( t \cdot (r_M \cdot C_M + r_J \cdot C_J) \)
Output Rate for Joins

\[ \text{Output Rate} = \frac{\text{# output transmitted}}{\text{time needed for transmission}} \]

\[ = f^* r_M^* r_J^* t^* (t-1) \]

\[ = \frac{f^* r_M^* r_J^* (t-1)}{t (r_M^* C_M^* r_J^* C_J)} \cdot (r_M^* C_M^* + r_J^* C_J) \]

\[ \approx \frac{f^* r_M^* r_J^* t}{r_M^* C_M^* + r_J^* C_J} \]

Optimizing Queries

\[ \text{# outputs} = \int_0^t r(t) dt \]

- Optimize for a specific time point
  - which plan will produce the most results by \( t_0 \)?
- Optimize for output production size
  - which plan is the first one to reach \( N \) results?

Local Rate Maximization

- first, maximize output rate here
- then, maximize rate for this join

Local Time Minimization

- first, minimize time to produce \( n \) tuples
- finally, minimize time to get the desired tuples in result
- top-down time minimization
**Experiment I**

*the plans*

A ✗ B ✗ C

(5K, 20ms) (A ✗ B) ✗ C

(10K, 2ms) (A ✗ C) ✗ B

(20K, 10ms)

A ✗ B ✗ C

B ✗ C (20K, 10ms)

A ✗ B (10K, 2ms)

A ✗ C (5K, 20ms)

**Experiment I**

*performance until last tuple*

A ✗ B ✗ C

(5K, 20ms) (10K, 2ms) (20K, 10ms)

(10K, 2ms)

(5K, 20ms)

(20K, 10ms)

**Experiment I**

*performance for the first few thousand tuples*

A ✗ B ✗ C

(5K, 20ms) (10K, 2ms) (20K, 10ms)

(10K, 2ms)

(5K, 20ms)

(20K, 10ms)

**Complex Plans**

(1) Left Deep

(2) Fast Leaves

(3) Evenly Spread
**Experiment Result**

<table>
<thead>
<tr>
<th></th>
<th>Left Deep</th>
<th>Fast Leaves</th>
<th>Evenly Spread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output size (tuples)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100,000</td>
<td>200,000</td>
<td>300,000</td>
<td>400,000</td>
</tr>
<tr>
<td>200,000</td>
<td>300,000</td>
<td>400,000</td>
<td>500,000</td>
</tr>
<tr>
<td>300,000</td>
<td>400,000</td>
<td>500,000</td>
<td>600,000</td>
</tr>
</tbody>
</table>

Time (s)

**Comparison to Traditional Model**

<table>
<thead>
<tr>
<th>Plan</th>
<th>Traditional Est.</th>
<th>Rate-Based Est.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Deep</td>
<td>$10^4$</td>
<td>$1.3 \times 10^3$</td>
</tr>
<tr>
<td>Fast Leaves</td>
<td>$2 \times 10^3$</td>
<td>$9.7 \times 10^2$</td>
</tr>
<tr>
<td>Evenly Spread</td>
<td>$5 \times 10^3$</td>
<td>$8.8 \times 10^2$</td>
</tr>
</tbody>
</table>

**Evaluating Window Joins over Unbounded Streams**

- Jaewoo Kang
- Jeffrey F. Naughton
- Stratis D. Viglas

University of Wisconsin-Madison
Computer Sciences Department

**Moving Window Join**

$T_a \ 
\lambda_a \ 
A \ 
(\lambda_a T_a) \ 
\ 
B \ 
(\lambda_b T_b) \ 
\ 
T_b \ 
\lambda_b \ 
\lambda_b \ 
\ 
T_b$ - stream B window size
$\lambda_b$ - stream B arrival rate
Types of Join

- Nested loops join
- Hash join

Nested Loops Join

Hash Join
Open Questions

- How to measure the efficiency of a moving window join?
- Can the join of streams with different rates be more efficient?
- How to deal with fast input streams when system cannot manage them?
- How to share limited memory between the two windows for the two inputs?

Idea!

- Streaming join algorithms can be asymmetric
  - Hash – Nested Loops join
  - Nested Loops – Hash join
  - ... or symmetric
  - Nested Loops – Nested Loops join
  - Hash – Hash join

Cost of Moving Window Joins

(\text{unit time basis model})

\begin{align*}
C_{AB|B} &= \lambda_a\left(\text{probe}(b) + \text{insert}(a) + \text{invalidate}(a)\right) \\
&\quad + \lambda_b\left(\text{probe}(a) + \text{insert}(b) + \text{invalidate}(b)\right) \\
\end{align*}

\begin{align*}
C_{AB|\emptyset} &= \lambda_a\left(\text{probe}(b)\right) + \lambda_b\left(\text{insert}(b) + \text{invalidate}(b)\right) \\
&\quad + \lambda_b\left(\text{probe}(a)\right) + \lambda_a\left(\text{insert}(a) + \text{invalidate}(a)\right)
\end{align*}

Cost of Join

- Nested loops join

\begin{align*}
C_{AB|B}(NLJ) &= \left(\lambda_aT_b\lambda_b + 2\lambda_b\right)\times C_n \\
&\quad + \left(\lambda_bT_a\lambda_a + 2\lambda_a\right)\times C_n
\end{align*}

- Hash join

\begin{align*}
C_{AB|\emptyset}(HJ) &= \left(\lambda_aT_b\lambda_b\sigma_b + \lambda_bT_b\lambda_b\sigma_b + \lambda_b\right)\times C_h \\
&\quad + \left(\lambda_bT_a\lambda_a\sigma_a + \lambda_aT_a\lambda_a\sigma_a + \lambda_a\right)\times C_h
\end{align*}
Comparison of Joins

\[ T_A = 60, \lambda_A = 10, T_B = 60, \sigma_A = 0.1, \sigma_B = 0.1, C_A = 0.5, C_B = 0.65 \]

Full Joins

\[ T_A = 60, \lambda_A = 10, T_B = 60, \sigma_A = 0.1, \sigma_B = 0.1, C_A = 0.5, C_B = 0.65 \]

Full Joins: different selectivity

\[ \sigma_A = 0.05, \sigma_B = 0.05 \]

1-way Join: System/Model cost
Overhead Costs

- $C_h/C_n$
  - Ratio of overhead cost of Hash Join to Nested Loop join
  - Model: ratio = 1.3
- $|B|$
  - Number of hash buckets in window B, assumed same as number of unique keys in window B
  - Variable that can be changed in the model

Crossover Points

Output rates

CPU time
Insufficient Resources for handling the Stream Input Rates

- Problem
  - Very Expensive Predicates
  - Input rate > Join operator service rate

- Solution
  - Drop tuples from input

Resource Allocation Strategies

- Problem: Very Expensive Predicates
- Solution: Drop tuples from input

Limited Memory

- Variable Time Window
- Allocate Memory depending on Stream Rate

Memory Allocation Strategies

- Problem: Limited Memory
- Solution: Allocate Memory depending on Stream Rate

Figure 9. Performance of memory allocation strategies w/ fixed arrival rates ($\lambda_a=10$, $\lambda_b=50$, $M=1000$, $\sigma_a=0.005$, $\sigma_b=0.01$)
Memory Allocation Strategies

- Give all memory (biggest window size) to slowest input stream
  - Fast stream probes slow stream, skips insertion/invalidation
  - Full Join reduces to One Way Join on the direction of slow → fast
  - Choose Join Algorithm after memory allocation

Memory Allocation Implications

Conclusions

- A Full Join can be seen as two separate independent Single Joins
  - Exploit asymmetrical stream input rates
    - NLJ/HJ algorithms Combination
      - HNJ/NHJ best candidate
    - Resource allocation
      - Devote most resources to slowest stream

K-Constraints

Exploiting k-Constraints to Reduce Memory Overhead in Continuous Queries over Data Streams
Shivnath Babu and Jennifer Widom, Stanford University
Introduction

- Already saw:
  - Use Rate information to optimize.
- Now we’ll see
  - Use properties of streamed data.
  - In order to reduce memory usage.

Outline

- Constraints for streams
- K-constraints
- Synopsis
- Algorithm using k-constraints

Constraints

- Properties that data streams satisfy.
- Examples:
  - Many-one join constraints between two streams.
  - Referential-integrity constraints for streams
    - Between two streams in many-one join
    - “One” side arrives before “Many” side
  - Clustered-arrival constraints on an attribute
    - Duplicate values arrive together
  - Ordered-arrival constraints on an attribute
    - Values are clustered and ordered.

Constraints (visual)

- Referential Integrity
- Clustered Arrival
- Ordered Arrival
Constraints?

- How practical are these constraints for streams?
- Tuples may come out of order.
  - Clustered? Ordered?
- Data rate may vary.
  - Referential Integrity?

K Constraints

- Idea: allow some disorder.
- K-Constraints are:
  - Constraints that are almost met.
  - K is the adherence parameter
    - Lower K means streams come closer to the constraint.
  - Like “slack” in Aurora
    - Set amount of disorder can be tolerated by system.
  - Examples:

Referential Integrity

- Many-one join from S1 to S2.
- S2 tuple will arrive before joining S1 tuple, or within K tuples on S2.

Clumped arrival

- On attribute S.A:
- At most k tuples with different S.A values arrive between tuples with the same value for S.A.

Ordered arrival

- On stream attribute S.A:
- Tuples that arrive at least k+1 tuples after tuple s have a value greater than or equal to s.

| S.A | 4 | 3 | 2 | 2 | 1 | 3 |

K=3

s

The Idea

- Joins over streams take infinite memory.
- Idea is to use k-constraints to reduce memory usage
  - Slower increase in memory usage.
  - Constant memory usage in some cases.
- K-constraints can decide which tuples to keep around.

Terminology

- **Synopsis**: stream history
- Each Synopsis for a stream involved with a query:
  - Has 3 components of seen tuples:
    - Yes: may contribute to a result tuple
    - No: cannot contribute to a result tuple
    - Unknown: cannot be put in Yes or No.

- **Join Graph**: directed graph with arcs from “Many” (parent) to the “One” (child) of many-one join.

Synopsis example

Query: Students that have GPA < 3.0 in Kalman when fire alarm is on.

Stream Student gets tuple:

- (“Edison”, 12:00)
- (id1234, Kalman, 12:05)

Stream Fire gets tuple:

- (“Edison”, 2.9)
- (id1234, Kalman, 12:00)

Output:

- Student (stID, location, time)
  - GPA (stID, gpa)
  - Fire (location, time)

<table>
<thead>
<tr>
<th>Student</th>
<th>Fire</th>
</tr>
</thead>
<tbody>
<tr>
<td>(id1234, Kalman, 12:00)</td>
<td>(id1234, 2.9)</td>
</tr>
</tbody>
</table>

Unknown
Synopsis

- Why not just keep those tuples that are in the Yes or Unknown synopsis?
- Might cause tuples in other streams to be kept in Unknown rather than being discarded.

Referential Integrity

Join heart rates greater than 35 with soldiers in sector 3 on id and time.
Constraints:
- location gets transmitted first
- always arrives within 2 tuples of heart rate.

<table>
<thead>
<tr>
<th>id</th>
<th>time</th>
<th>loc</th>
<th>heart rate</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>1</td>
<td>sec3</td>
<td>&gt; 35</td>
<td>(s3,1,38)</td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>sec2</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>2</td>
<td>sec3</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>1</td>
<td>sec3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The No synopsis on left is never needed!!!

Referential Integrity

- If Referential Integrity with parameter K holds on many-one join S1 to S2
  □ Eliminate S2’s No component
  □ Keep S1’s Unknown component for only k tuples on S2.

 Synopsis example 2
Soldiers with heartrate = 0 where more than 2 missiles were seen.
Stream Heart (SoldierID,Rate) gets tuple: (s1,1) (s2,0) (s3,1)
Output
Stream Missile(Sector) gets tuple: (Sec3, 1) (Sec5, 4) (s2,0,Sec5)
Yes No Unknown
Heart(soldID, Rate)
Where(soldID, Sector)
Missiles(Sector)
**Ordered-Arrival Constraints (OA(k))**

**Two algorithms:**
- On child stream ("one" in many-one join)
  - OAC(k)
- On parent stream ("many" in many-one join)
  - OAP(k)

---

**Ordered Arrival (on "one")**

Soldiers in sector 3 while a soldier had heart rate of 0. (Join on time. Assume one location tuple per time.)

**Constraint:**
- Location comes in ordered, with at most 1 tuples out of order.

**Output:**

<table>
<thead>
<tr>
<th>id</th>
<th>time</th>
<th>loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>s4</td>
<td>1</td>
<td>sec2</td>
</tr>
<tr>
<td>s5</td>
<td>2</td>
<td>sec2</td>
</tr>
<tr>
<td>s7</td>
<td>4</td>
<td>sec3</td>
</tr>
</tbody>
</table>

**Ordered Arrival (on "many")**

Soldiers in sector 3 while a soldier had heart rate of 0. (Join on time. Assume one location tuple per time.)

**Constraint:**
- HeartRate comes in ordered, with at most 1 tuples out of order.

**Output:**

<table>
<thead>
<tr>
<th>id</th>
<th>time</th>
<th>loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>s4</td>
<td>0</td>
<td>sec3</td>
</tr>
<tr>
<td>s5</td>
<td>5</td>
<td>sec2</td>
</tr>
<tr>
<td>s7</td>
<td>2</td>
<td>sec3</td>
</tr>
</tbody>
</table>

**OAC(k)**

- Similar to Referential Integrity
- Eliminate No synopsis without filling parent Unknown synopsis:
- Maintain the minimum value L that will be seen on stream S.
- Tuples in parent Stream less than L that do not match S’s Unknown or Yes, must have no matching tuple in S – no need to put into Unknown.
OAP(k)

- **Idea:**
  - Given a child stream’s tuple \( s \),
  - If no future parent tuples can join with \( s \),
  - Then, Don’t store \( s \).
- If Ordered Arrival constraint on parent stream’s attribute \( A \) OAP(k)
  - Can drop child’s tuples after \( k \) tuples with larger \( A \) values.

### Clustered Arrival (CA(k))

- **Idea:**
  - Similar to Ordered arrival on parent stream.
- If parent streams have CA(k) on attribute \( A \):
  - After a joining tuple in parent, store \( s \) for only \( k \) more parent tuples.

---

**RIDS(k) Results**

Larger \( k \) means tuples are kept in Unknown synopsis longer, using more memory.

**CA(k) Results**

Smaller \( K \) means store fewer tuples in child streams. Yes synopsis

- \( k = 20000 \)
- \( k = 10000 \)
- \( k = 5000 \)
- \( k = 1000 \)
- \( k = 0 \)
**OAP(k)**

**Results**

Smaller K means store less in child Yes synopsis

**OAC(k)**

**Results**

Smaller K means tuples are kept in parent stream synopsis less time.

**CA(k) and OAC(k)**

Combining CA(k) and OAC(k) does better than either alone, especially at high values for K.

**CA(k) vs. combined CA(k) and RIDS(k)**

Note that at low K for RIDS(k), CA(k) does better. Some tuples are kept around longer than in pure CA(k).
Summary

- Speed
  - Cost Optimization
  - Cardinality -> Rate
  - Pin Slow Streams
- Accuracy
  - Windows for approximation
  - Memory issues
  - Join algorithms

Discussion

- When join by timestamp with a range, what is timestamp of output tuple?
- How are punctuation and K-constraints similar?
- Rate based paper didn’t account for windows – what is the effect?

Discussion

- What are the pros/cons of windows vs K-Constraints?
- The join paper assumed finite streams – do their conclusions work for infinite streams?
- Can you think of other cost measuring methods for the optimizer?
Discussion

- How would a stream system optimize across multiple, concurrent persistent queries? Does what we studied today apply?
- How would a stream system handle non-equijoins? Does what we studied today apply?

Open Questions

- Could this approach be used on systems like Aurora/Stream etc.?
- Can this model be modified so that it can be applied to other operators, and if so, would it have good benefits?
- How much asymmetry actually exists in practice?