Leveraging Run Time Knowledge about Event Rates to Improve Memory Utilization in Data Streams

Beth Plale
Computer Science Dept.
Indiana University

25 July 2002
• Data streams are flows of timestamped events that can be manipulated through continuously executing queries
  – Fjords (Franklin), NiagraCQ (DeWitt), STREAM (Widom), dQUOB (Plale)

• In this talk, we discuss performance optimizations identified through use of dQUOB continuous query processing for reducing overall memory utilization.
Outline

• Overview of dQUOB
• Existing approach to join handling
• Optimization to join handling for asynchronous streams
• Results
• Conclusion
Visualizing Doppler Radar Flows

- Streamed sweep data
- Archived sweep data
- raw Level 2 data
- Transform, Filter, Aggregate
- discards

Peachtree City, GA
Grier, SC
Hytop, AL

25 July 2002
dQUOB system: continuously executing queries over data streams

- SQL queries coupled with user-defined functions (e.g., FFT, data reduction).
- Assumes data stream is timestamped sequence of events.
- Event == tuple, data stream == relation
- Supports time-based stream join
  - Two events satisfy a join if they ‘happen at the same time’
- Applied to: large scale scientific instruments, scientific visualization, hazard detection

25 July 2002
dQUOB Architecture

Converts script of SQL query into C++ DAG of selects, projects, joins
Outline

• Overview of dQUOB
• Existing approach to join handling
• Optimization for asynchronous streams
• Results
• Conclusion

25 July 2002
Basic Join Query Cases

(a) No join

(b) Self join

(c) Stream join

flow of control

select, project, join operator

input stream

join window

25 July 2002
• **Join window** - a sliding window over an event stream; determines size of event sequence retained to participate in join
Problem 1: when R and S are asynchronous streams and stream S is slow and erratic, unneeded memory consumption

<table>
<thead>
<tr>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:</td>
</tr>
<tr>
<td>S:</td>
</tr>
</tbody>
</table>

Problem 2: difficult for user to pick right join window size. Cost of error is great: too large, consumes memory; too small, increases false negatives

25 July 2002
If we can get better use of memory space, we get better scaling in number of queries supported.
Outline

• Overview of dQUOB
• Existing approach to join handling
• Optimization for asynchronous streams
• Results
• Conclusion

25 July 2002
Approach to problem of asynchronous streams: express join window as interval of time

<table>
<thead>
<tr>
<th>time</th>
<th>( \rightarrow )</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:</td>
<td>... l k j i h g f e d c b a</td>
</tr>
<tr>
<td>S:</td>
<td>... 5 4 3 2 1</td>
</tr>
</tbody>
</table>

\[ \text{join window: count} \]

<table>
<thead>
<tr>
<th>time</th>
<th>( \rightarrow )</th>
</tr>
</thead>
<tbody>
<tr>
<td>R:</td>
<td>... l k j i h g f e d c b a</td>
</tr>
<tr>
<td>S:</td>
<td>... 5 4 3 2 1</td>
</tr>
</tbody>
</table>

join window: 10 sec.

25 July 2002
Step 1: user specifies interval in wallclock time. Why? Only interval known for certain at startup.

Step 2: sample during runtime to figure out stream rates. Map wallclock interval into timestamp interval

Step 3: adjust join window sizes

Step 4: use introspection technique to monitor and adapt to changes in event rates

25 July 2002
Rate sensitive join window algorithm

Sample event stream for sampling_interval

Continuous sampling of event stream

Calculate join window size

Exponential averaging

Effect change to join windows

introspection

Compute – observe – optimize cycle

25 July 2002
Join Algorithm Pseudo Code

At_startup (sampling_interval: integer) { 
    for all i concurrently {
        sample event stream[i] for duration of sampling_interval;
        barrier();

        max_timestamp_interval = last_event[i].timestamp – first_event[i].timestamp;

        join_window_size[i] = (events_received[i] * sampling_interval) / max_timestamp_interval;
    }
    effect_change[i];
}

25 July 2002
### event arrival times

<table>
<thead>
<tr>
<th></th>
<th>α</th>
<th>β</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>:03</td>
<td>:03</td>
</tr>
<tr>
<td>01:10:22</td>
<td>:12:20:28:30</td>
<td></td>
</tr>
<tr>
<td>01:10:22</td>
<td>:12:20:28:30</td>
<td></td>
</tr>
<tr>
<td>01:10:22</td>
<td>:12:20:28:30</td>
<td></td>
</tr>
<tr>
<td>01:10:22</td>
<td>:12:20:28:30</td>
<td></td>
</tr>
<tr>
<td>01:10:22</td>
<td>:12:20:28:30</td>
<td></td>
</tr>
<tr>
<td>01:10:22</td>
<td>:12:20:28:30</td>
<td></td>
</tr>
<tr>
<td>01:10:22</td>
<td>:12:20:28:30</td>
<td></td>
</tr>
<tr>
<td>01:10:22</td>
<td>:12:20:28:30</td>
<td></td>
</tr>
<tr>
<td>01:10:22</td>
<td>:12:20:28:30</td>
<td></td>
</tr>
</tbody>
</table>

- **sampling_interval**: 30 seconds
- **timestamp_interval(α)** = 10 seconds
- **timestamp_interval(β)** = 2 seconds
- **max_timestamp_interval** = 10 seconds
- **join_window(α)** = \((11 \times 30) / 10 = 33\) events
- **join_window(β)** = \((3 \times 30) / 10 = 9\) events

33 events == 30 sec. in wallclock time and 10 seconds in timestamp time

---

25 July 2002
Implementation of Sampling and Window Calculation in Quoblet
Outline

• Overview of dQUOB
• Existing approach to join handling
• Optimization for asynchronous streams
• Results
• Conclusion
Measuring Wide-area Performance

Workload: 540 events generated by global atmospheric transport model

Environment:
- Georgia Tech: Sun Ultra 30 cluster, Solaris 7
- Albuquerque High Performance Computing Center (AHPCC):
  Onyx 2 8 processor, Irix64 6.5
- NCSA, Urbana-Champaign Illinois: Origin 2000 Array,
  48 processor, Irix

Network: Abilene (2.4 Gbits per second)

25 July 2002
-- Baseline case (LAN communication)
-- Query's filtering capability progressively strengthens in response to changes in network environment

25 July 2002
Pushing data transformation closer to source yields total duration times that are closer to baseline (LAN) time.
- variance due to traffic flow control in TCP
Outline

• Overview of dQUOB
• Existing approach to join handling
• Optimization to join handling: rate sensitive, dynamic adjustment of join window.
• Results
• Conclusion
Related Research

Continuous Queries

• Eddies (Avnur and Hellerstein 2000)
• Fjords (Madden and Franklin 2002)
• NiagraCQ (DeWitt 2000)
• Naughton (2000)
• STREAM (Widom and Motwani 2001)
• Tapestry (Terry 1992)

Data Streaming

• Active Data Repository (Ferreira, Kurc et. al. 1999)
Future Work

• Quantify impact of optimizations through performance evaluation.
• Extending join window size work to include probability assessment of likelihood of false negatives.
  - When user sets window size of .01 second, we can return with a warning “probability of false negatives is 90%”
• Work out ownership
• Public release dQUOB compiler, server, and runtime (slated for Fall 2002).

http://www.cs.indiana.edu/~plale/projects/dQUOB