Window-aware Load Shedding for Aggregation Queries over Data Streams

Nesime Tatbul    Stan Zdonik
Talk Outline

- Background
  - Load shedding in Aurora
  - Windowed aggregation queries
- Window-aware load shedding
- Experimental results
- Related work
- Conclusions and Future directions
Problem: When load > capacity, latency QoS degrades.

Solution: Insert drop operators into the query plan.

Result: Deliver “approximate answers” with low latency.
Subset-based Approximation

- For all queries, the delivered tuple set must be a subset of the original query answer set.
  - Maximum subset measure (e.g., [SIGMOD’03, VLDB’04])
  - Loss-tolerance QoS of Aurora [VLDB’03]

- For each query, the number of consecutive result tuples missed (i.e., gap tolerance) must be below a certain threshold.
Why Subset Results?

- The application may expect to get correct values.
- Missing tuples get updated with more recent ones.
- Preserving subset guarantee anywhere in the query plan helps understand how the error propagates.
- It depends on the application.
Windows

- Finite excerpts of a data stream
- Two parameters: size ($\omega$) and slide ($\delta$)
- Example: \texttt{StockQuote(symbol, time, price)}

\begin{itemize}
  \item size = 10 min
  \begin{itemize}
    \item ("IBM", 10:00, 20)
    \item ("INTC", 10:00, 15)
    \item ("MSFT", 10:00, 22)
    \item ("IBM", 10:05, 18)
    \item ("MSFT", 10:05, 21)
    \item ("IBM", 10:10, 18)
    \item ("MSFT", 10:10, 20)
    \item ("IBM", 10:15, 20)
    \item ("INTC", 10:15, 20)
    \item ("MSFT", 10:15, 20)
  \end{itemize}
\end{itemize}

slide by 5 min
Windowed Aggregation

- **Apply an aggregate function on the window**
  - Average, Sum, Count, Min, Max
  - User-defined function
- **Can have grouping, nesting, and sharing**
- **Example:**

![Diagram](image)

- **Filter**
  - symbol = “IBM”
- **Aggregate**
  - $\omega = 5$ min
  - $\delta = 5$ min
  - $\text{diff} = \text{high-low}$
- **Filter**
  - $\text{diff} > 5$
- **Aggregate**
  - $\omega = 60$ min
  - $\delta = 60$ min
  - count
- **Filter**
  - count $> 0$
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
Dropping from an Aggregation Query
Window-based Approach

- Drop before: subset result of smaller size

“window-aware load shedding”
Window Drop Operator

**Functionality:**
- Attach *window keep/drop specifications* into tuples.
- Discard tuples that can be *early-dropped*.

**Key parameters:**
- $\omega$: window size
- $\delta$: window slide
- $p$: drop probability (one per group)
- $B$: batch size (one per group)

<table>
<thead>
<tr>
<th>Window Specification</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Don't care.</td>
</tr>
<tr>
<td>0</td>
<td>Drop the window.</td>
</tr>
<tr>
<td>$T$ ((T &gt; 0))</td>
<td>Keep the window.</td>
</tr>
<tr>
<td></td>
<td>Keep all tuples with timestamp $&lt; T$.</td>
</tr>
</tbody>
</table>
Window Drop for Nested Aggregation
(Pipeline Arrangements)

- Example:

\[
\omega = \sum_{i=1}^{n} \omega_i - (n-1)
\]

\[
\delta = \delta_n
\]

\[
B = B
\]
Window Drop for Shared Aggregation
(Fan-out Arrangements)

Example:

\[ \omega = lcm(\delta_1, \ldots, \delta_n) \]
\[ + \max_{i=1}^n \{ extent(A_i) \} \]
where \( extent(A_i) = \omega_i - \delta_i \)

\[ \delta = lcm(\delta_1, \ldots, \delta_n) \]

\[ B = \min_{i=1}^n \left\{ \frac{B_i}{lcm(\delta_1, \ldots, \delta_n) / \delta_i} \right\} \]
Window Drop for Nesting + Sharing
(Composite Arrangements)

- Apply the two basic rules in any order.

- Fan-out - Pipeline

- Pipeline - Fan-out

\[ \omega_0 = 4, \delta_0 = 1 \]
\[ \omega = 7, \delta = 6 \]
\[ \omega_1 = 3, \delta_1 = 3 \]
\[ \omega_2 = 3, \delta_2 = 2 \]

\[ \omega_0 = 4, \delta_0 = 1 \]
\[ \omega = 6, \delta = 3 \]
\[ \omega_1 = 3, \delta_1 = 3 \]
\[ \omega_2 = 3, \delta_2 = 2 \]

\[ \omega_0 = 4, \delta_0 = 1 \]
\[ \omega = 10, \delta = 6 \]
\[ \omega_1 = 3, \delta_1 = 3 \]
\[ \omega_2 = 3, \delta_2 = 2 \]

\[ \omega_0 = 4, \delta_0 = 1 \]
\[ \omega = 6, \delta = 3 \]
\[ \omega_0 = 4, \delta_0 = 1 \]
\[ \omega_2 = 3, \delta_2 = 2 \]
Window Drop in Action

- Mark tuples & apply early drops at the Window Drop.

- Decode the marks at the Aggregate.

Window Drop
\[ \omega = 3, \delta = 3 \]
\[ p = 0.5 \]

Decode the marks at the Aggregate.
Fake Tuples

- Fake tuples carry no data, but only window specs.
- They arise when:
  - the tuple content is not needed as the tuple only indicates a window drop
  - the tuple to mark is
    - originally missing, or
    - filtered out during query processing
- Overhead is low.
Early Drops

- Sliding windows may overlap.
- Window drop can discard a tuple if and only if all of its windows are going to be dropped.
- At steady state, each tuple belongs to $\sim \omega/\delta$ windows (i.e., for early drops, $B \geq \omega/\delta$).
- If $B < \omega/\delta$, then no need to push window drop further upstream than the first aggregate in the pipeline.
Talk Outline

- Background
  - Load shedding in Aurora
  - Windowed aggregation queries
- Window-aware load shedding
- Experimental results
- Related work
- Conclusions and Future directions
Experiments

- **Setup**
  - Single-node Borealis
  - Streams: (time, value) pairs
  - Aggregation queries with “delays”
  - Basic measure: % total loss

- **Goals**
  - Performance on nested and shared query plans
  - Effect of window parameters
  - Processing overhead
Performance for Nested Plans

- Window Drop
  - Delay 5ms
  - Aggregate 10, 10
  - Delay 50ms
  - Aggregate 100, 100
  - Delay 500ms

- Random Drop

Graph:
- X-axis: % Excess Rate
- Y-axis: % Loss
- Bars for Window Drop and Random Drop
Performance for Shared Plans

- Window Drop
- Delay 5ms

Aggregate 10, 10
- Delay 50ms

Aggregate 20, 20
- Delay 100ms

Random Drop

% Excess Rate

% Total Loss

- Window Drop
- Random Drop

% Excess Rate:

- 20
- 40
- 60
- 80
- 100

% Total Loss:

- 0
- 10
- 20
- 30
- 40
- 50
- 60
- 70
- 80
- 90
- 100
Effect of Window Slide

- Window Drop \( \omega = 100, \delta \)
- Delay 5ms
- Aggregate \( \omega = 100, \delta \)
- Delay 5ms

Excess Rate = 25%
Excess Rate = 50%
Excess Rate = 66%

Delay 5ms

Aggregate \( \omega = 100, \delta \)
## Processing Overhead

### Throughput Ratio (Window-Drop (p=0) / No Window-Drop)

<table>
<thead>
<tr>
<th>Window size (ω)</th>
<th>Selectivity = 1.0</th>
<th>Selectivity = 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>0.99</td>
<td>0.96</td>
</tr>
<tr>
<td>50</td>
<td>0.99</td>
<td>0.98</td>
</tr>
<tr>
<td>75</td>
<td>1.0</td>
<td>0.98</td>
</tr>
<tr>
<td>100</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Related Work

- Load shedding for aggregation queries
  - Statistical approach of Babcock et al [ICDE’04]
- Punctuation-based stream processing
  - Tucker et al [TKDE’03], Li et al [SIGMOD’05]
- Other load shedding work (examples)
  - General: Aurora [VLDB’03], Data Triage [ICDE’05]
  - On joins: Das et al [SIGMOD’03], STREAM [VLDB’04]
  - With optimization: NiagaraCQ [ICDE’03, SIGMOD’04]
- Approximate query processing on aggregates
  - Synopsis-based approaches, online aggregation [SIGMOD’97]
Conclusions

- We provide a **Window-aware Load Shedding** technique for a general class of aggregation queries over data streams with:
  - sliding windows
  - user-defined aggregates
  - nesting, sharing, grouping

- **Window Drop** preserves window integrity, enabling:
  - easy control of error propagation
  - subset guarantee for query results
  - early load reduction that minimizes error
Future Directions

- Prediction-based load shedding
  - Can we guess the missing values?
  - Statistical approaches vs. Subset-based approaches
- Window-awareness on joins
- Memory-constrained environments
- Distributed environments
Questions?
## Decoding Window Specifications

<table>
<thead>
<tr>
<th>Window Start</th>
<th>Window Spec</th>
<th>Keep Boundary</th>
<th>To Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>T</td>
<td>*</td>
<td>Open window. Set boundary = T - (ω - 1) Set spec = T - (ω - 1)</td>
</tr>
<tr>
<td>✓</td>
<td>0</td>
<td></td>
<td>Open window. Set boundary = t + ω (if &gt;)</td>
</tr>
<tr>
<td>✓</td>
<td>-1</td>
<td>Beyond</td>
<td>Skip window.</td>
</tr>
<tr>
<td>✗</td>
<td>T</td>
<td>*</td>
<td>Set boundary = T - (ω - 1) Set spec = T - (ω - 1) Mark as “fake tuple”.</td>
</tr>
<tr>
<td>✗</td>
<td>0</td>
<td>*</td>
<td>Mark as “fake tuple”.</td>
</tr>
<tr>
<td>✗</td>
<td>-1</td>
<td>*</td>
<td>Ignore.</td>
</tr>
</tbody>
</table>