Semantics and Evaluation Techniques for Window Aggregates in Data Streams

J. Li, D. Maier, K. Tufte, V. Papadimos, and P. Tucker

Presentation by: Kevin Quan (kquan@uwaterloo.ca)

Overview

- Introduction
- Window semantics
- WID approach
- Experimental Results
- Conclusions

Definitions

- Data Streams
  - Possibly unbounded sequence of tuples
  - e.g. sensor data, call records, auction bids
- Window
  - Moving view of a subset of the sequence of tuples
- Window extents
  - A specific windowed view
  - Cookie metaphor
- Punctuation
  - Embedded information in tuple telling system that no more tuples with certain attribute values will be seen.

Problems & Challenges

- Goals
  - Near real-time results
  - Accuracy
  - Minimize memory usage
- Obstacles
  - High data arrival rates
  - Bursty traffic
  - Out of order data
  - Large amount of data

What's New in this Paper?

- Define semantically what a window is.
- Avoids intra-operator buffering of tuples and tuple re-processing
- Avoid keeping all active input tuples in-memory
- No assumption that stream data is ordered

Window Semantics

- Window specification
  - RANGE: length of window
  - SLIDE: step at which window moves
  - WATTR: tuple attribute over which RANGE and SLIDE are specified
    - Timestamps
    - Rows number
  - Optionally
    - SATTR
    - RATTR
    - PATTR

Examples in CQL

```
CREATE TABLE Traffic
{
   row-num integer,
   seg-id integer,
   speed integer,
   ts datetime
};
```

```
SELECT seg-id, MAX(speed),
FROM   Traffic [
       RANGE 300 seconds
       SLIDE 60 seconds
       WATTR ts ]
GROUP BY seg-id

SELECT seg-id, COUNT(*)
FROM   Traffic [
       RANGE 300 seconds
       RATTR ts
       SLIDE 1 row
       SATTR row-num ]
```

Framework to Define Window Semantics

- window-ids
  - Names a specific window extent
- windows(T,S)
  - T is set of tuples that compose stream
  - S is windows specification
  - Returns all window-ids of extents composed of tuples in T.
- extent(w,T,S)
  - w is a window extent
  - Returns all tuples in w.
- wids(t,T,S)
  - Inverse of extent function
  - t is a specific tuple.
  - Returns all window-ids which t belongs to.

An Example

$$windows(T,S[RANGE,SLIDE,WATTR]) = \{0,1,2...\}$$

$$extent(w,T,S[RANGE,SLIDE,WATTR]) = \{t \in T \mid$$

$$\max\begin{pmatrix} \min_{WATTR}(T) \\ \min_{WATTR}(T) + (w+1) * SLIDE - RANGE \end{pmatrix}$$

$$\leq t.WATTR < \min_{WATTR}(T) + (w+1) * SLIDE\}$$

$$wids(t,T,S[RANGE,SLIDE,WATTR]) =$$

$$\{w \in W \mid (t.WATTR - \min_{WATTR}(T))/(SLIDE - 1) < w$$

$$\leq (t.WATTR + RANGE - \min_{WATTR}(T))/(SLIDE - 1)\}$$

WID Approach to Aggregation

- Based on Niagara Query Engine
- Uses defined window semantics
- Uses punctuation to mark the end of each window extents.
- First step: Bucket Operator
  - Tags a tuple with its associated window-ids given a window specification
  - Bucket does not need to maintain any state (for context free windows) -- I think it does.

WID Approach 2

- Aggregation
  - Takes a tuple from bucket operator and updates intermediate aggregate values for all its wids().
  - Using punctuation, aggregate operator detects when each extent is completed and outputs result.

Forward-Context Aware Windows

- Forward context means that wids() requires future tuples to return result.
- Slide-by-tuple window
  - Each new tuple forms a new extent
  - RANGE is specified over an attribute that is not necessarily related to number of tuples.
  - e.g. `[RANGE 300 seconds, RATTR ts, SLIDE 1 row, SATTR row-num ]`
- Previous WID approach won't work

WID Approach for FCA Windows

- Bucket operator
  - Tags tuple with range spanning from its RATTR value to the RATTR+RANGE
- Aggregation
  - Creates or updates partial aggregate values of "bins" between (RATTR, RATTR+RANGE)
  - Bins are bounded by every tuple's RATTR and RATTR+RANGE values.
  - Results returned when punctuation arrives.

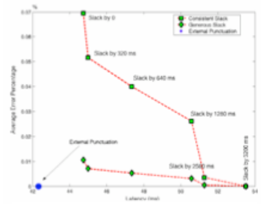
WID for FCA 2

More Thoughts on Punctuation

- Punctuation is only one way to counteract disorder.
- Alternatives:
  - Slack: tuples not more than N positions out of order
  - Heartbeats: punctuations on timestamp
- May have tardy policy to ignore tuples more than S seconds late
- Delays in punctuation arrive affect latency of results

Effectiveness of Punctuation

- Punctuation produces results with minimum latency and maximum accuracy.



Effectiveness of WID

- WID is more effective on streams with many window extents

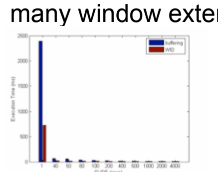


Figure 8 (a): Execution Time: WID versus Buffering – Overview

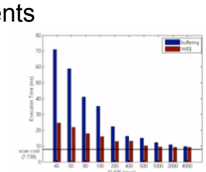


Figure 8 (b): Execution Time: WID versus Buffering – Zoom-in

Conclusions

- WID separates window definition from operator implementation
- WID uses punctuation to produce more accurate results with smaller latency
- WID reduces buffer space by maintaining partial aggregates instead of each window extent.

Open Issues for Discussion

- Experimental results show perfect accuracy using punctuation in disordered data but real-life punctuation has tardy policy that introduces error?
- Why is WID still faster than buffering at maximum slide (only one window)?
- Is WID faster than buffering for out of order streams?
- How complex is `wids()` and `extent()` in implementation? This may make the aggregation algorithm complex.
- Bucket operator needs to store state to map window-ids to WATTR.

Any Questions?