Resource Sharing in Continuous Sliding-Window Aggregation

A. Arasu
J. Widom

Presented by: Hossein S. Attar

Motivation
- Large number of concurrent continuous queries
  - Publish-subscribe systems
  - Handling each query separately
    - Inefficient
- Resource sharing among queries
  - Computation
  - Memory
  - Disk bandwidth

Motivation
- Operator level sharing
  - A single generic operator handling several queries
  - Previous work
    - Filters (stateless)
  - This paper
    - Aggregation over sliding windows
      - Each operator maintains state
Background
- Sliding windows
  - Used to solve problems with unbounded streams
  - Window size:
    - Time interval (time-based)
    - Number of tuples (tuple-based)
- Suffix windows
  - More recent data
- Non-suffix (historical) windows

Background
- Aggregation over a sliding window (ASW)
- Aggregations on substreams
  - Similar to GROUP BY
- Substream filters (SSF)
  - Similar to HAVING

Output Model
- ASW output updated
  - Actively
    - As window slides
    - Using some interval
  - Upon request
    - Lookup Model
    - Considered in this paper
Cost Parameters

- 3 parameters
  - Space (memory) for maintaining state
  - Time to compute answer (lookup time)
  - Update time when a new tuple arrives

Space-Update-Lookup Tradeoff
- Partial answer computation at update time
- Compute final answers using partial results at lookup time

Resource Sharing

- In this work sharing is possible among operators
  - Over the same streams
  - Of the same type
  - Only different in their sliding window specification
- QUANTILE is an exception
  - Quantile parameter can also be different

ASW Operators
Agg. Functions

- Aggregation Functions
  - Distributive
    - \( f(X_1) \) and \( f(X_2) \Rightarrow f(X_1 \cup X_2) \)
    - \text{SUM, COUNT, MAX, MIN}
  - Algebraic
    - There exists a synoptic function \( g \) such that
      - \( g(X) \Rightarrow f(X) \)
      - \( g \) is distributive
      - \text{AVG}
  - Holistic
    - Not algebraic
    - \text{QUANTILE}

Algorithms

- Distributive or algebraic
  - Base Intervals (B-INT)
  - Landmark Intervals (L-INT)
- Results are precomputed and stored for some intervals
- Final answer for other intervals calculated using precomputed results

Base Intervals (B-INT)

- Base intervals
  - \((2^{i+1}, 2^{(i+1)})\) for some \(i\)
- Active intervals
  - Intervals to the right of the beginning of the earliest window
B-INT

B-INT Costs
- Space
  - $O(N_{\text{max}})$
- Update (amortized)
  - $O(1)$
- Lookup (worst case)
  - $O(\log W)$

Landmark Intervals (L-INT)
- $W_{\text{min}}$ and $W_{\text{max}}$
- Landmark intervals
  - $(\alpha W_{\text{min}}, \alpha W_{\text{min}} + d)$
  - $(\alpha W_{\text{min}} - d, \alpha W_{\text{min}} - 1)$
  - $d < W_{\text{max}}$
L-INT Costs

- $W_{min}$ and $W_{max}$ should be close to equal
- Space
  - $O(N_{max})$
- Update (amortized)
  - $O(1)$
- Lookup (worst case)
  - $O(1)$
PSoup
- Proposed by Chandrasekharan and Franklin
- Uses an augmented $n$-ary search tree
  - Leaves are tuples
  - Internal nodes store the value of agg. Function for its descendents
- Update
  - $O(\log N_{\text{max}})$
- Lookup
  - $O(\log N_{\text{max}})$

QUANTILES
- Consider a bag of $N$ elements
- QUANTILE $(\phi) =$ element at position $\phi N$ in the sorted sequence of elements
- B-INT-QNT
  - Based on B-INT
  - Store a sorted array of elements for each base interval
  - Compute QUANTILE using the sorted arrays

SSF Operators
Sharing in SSF

- sharing among SSF operators that differ only in
  - Window specification
  - Range predicate

Simple Approach

- Simple Approach
  - For each substream
    - process ASW suboperations using one of the ASW sharing algorithms
    - Lookup cost depends on the size of key attribute domain |K|
  - More efficient approaches
    - For certain comb. of window type and functions

CI-COUNT

- Used for function COUNT on substreams when range conditions are one sided
- Produces approximate answers
CI COUNT

- The following are equivalent
  - Look for substreams that have received more than \( v \) elements in the last \( T \) time units
  - Look for substreams for which the \( v \)th element from the end has timestamp greater than \( \tau - T \)

CI_COUNT

- Maintain an index over the timestamps of the \( v \)th element from the end of all substreams
- Instead of maintaining an index for each \( v \) value, maintain an index for each level
- All \( v \) values such that \( \log v = l \) share the same index

CI COUNT Example
CI-COUNT Costs

- Space
  - $O(|K| \log N_{\text{max}})$
- Update (amortized)
  - $O(\log |K|)$
- Lookup
  - $O(|K_o|)$

Comments

- The space and execution costs of algorithms are good
- Algorithms are easy to implement
- But no proof of optimality