Trading Timeliness and Accuracy in Geo-Distributed Streaming Analytics

Benjamin Heintz  Abhishek Chandra  Ramesh K. Sitaraman
What does this paper want to solve?

- In this paper, it focuses on windowed grouped aggregation, an important and widely used primitive in streaming analytics and it studies the tradeoff between staleness and error.
Windowed grouped aggregation

- “Windowed” means a time window.
- “grouped aggregation” means grouping values for each key in one time window. For example, sum all values for one key.

<table>
<thead>
<tr>
<th>The time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate value for key k(i)</td>
</tr>
</tbody>
</table>

A window
Tradeoff

Staleness
- Network congestion
- Delay transmission until end of the time window

Error
- Edge sends only partial aggregates
- Edge omits some aggregates altogether
Ideal methods to balance the tradeoff

1. Minimizing staleness under an error constraint (error $\leq E$)

2. Minimizing error under a staleness constraint (Staleness $\leq S$)
Minimizing Staleness (Error-Bound)

• Important concepts:
  • $V_k(t)$ is the aggregate value of all arrivals for key $k$ during the current window prior to time $t$.
  • Prefix error $e_k(t) = \text{error}(V_k(t), V_k(T+W))$. $e_k(T)$ is the initial prefix error.

• **Definition 1**: Given an error constraint $E$, the Eager Prefix Error algorithm flushes each key $k$ at first time $t$ such that $e_k(t) \leq E$. If $e_k(T) \leq E$, EPE avoids flushing key $k$ altogether.
Minimizing Error (Staleness-Bound)

When talking about this method, the author defines a model to abstract the network.

**Definition 2:** The Smallest Potential Error First algorithm iterates over the n slots beginning with the earliest, assigning to each slot a key with the smallest potential error that has not yet been assigned to a slot.

**Definition 3:** Let m be the number of keys that the SPEF algorithm omits. Then the SPEF with Early Omissions algorithm first omits the m keys with the smallest initial prefix errors, then assigns the remaining n-m keys in SPEF order.
Practical methods

1. Error-Bound Algorithm

(1) Cache partitioning policy – Primary cache and secondary cache.
(2) Cache eviction policy

When aggregate value is prompted from secondary cache to primary cache?

**Accumulated error** -- A simple understanding of the original explanation is that if the aggregate value change enough big, this aggregation value should be prompted from secondary cache to primary cache.
Practical methods

2. Staleness-Bound Algorithms
   (1) Predict initial prefix error – $e_k(T)$ – use accumulated error.

   (2) Separate primary and secondary caches. The boundary is decided by the equation:
   \[ B(t) = \sigma(t) + f(t) \]
   $B(t)$ is the total number of networks slots remaining until the staleness constraint.
   $\sigma(t)$ primary cache size.
   $f(t)$ donates the number of future arrivals of these large-prefix-error keys.

   (3) Evict keys from primary cache.
   Use LRU to evict flushed key from primary cache during time window.
   When reaching the end of the window, there is no need to predict, because all final aggregate values are known.
Implementation

Edge:
1. The Aggregator defines window boundaries, maintains the tow-level cache from on-line algorithm.
2. Each Aggregator aggregate a hash-partitioned subset of the key space.
3. The reorderer is responsible for delaying records as necessary in order to maintain punctuation semantics.

Center:
1. Once completing aggregation for a window, the Aggregator emits summary metrics including error and staleness, and these metrics are summarized by the final StatsCollector.
2. Track queuing delays within both the SocketSender and the SocketReceiver. These delays are communicated upstream, and the edge Aggregators use them as congestion signals to dynamically adjust the rate at which they push records downstream.
Evaluation

• Streaming
  ➢ send every record immediately.

• Batching
  ➢ send all keys whose $e_k(T)$ is larger than $E$ at time $T+W$

• Batching with Random Early Update
  ➢ Working as batching, but flush keys randomly during the window. When time is $T+W$, work as batching.

• Optimal
  ➢ Implement *Eager Prefix Error* and *smallest Potential Error First with Early Omissions* algorithms.

• Chaching
  ➢ Emulate the above two algorithms.
Evaluation

**Figure 3.** Normalized staleness for Sum aggregation with various error bounds. Note the logarithmic y-scale.

**Figure 4.** Normalized error for Sum aggregation with various staleness bounds. Note the logarithmic axis scales.
Evaluation

(a) Staleness under an error constraint.

(b) Error under a staleness constraint.

**Figure 5.** Comparison of Random and Caching algorithms for various aggregation functions on a PlanetLab testbed.
Evaluation

![Evaluation Diagrams]

(a) Staleness under an error constraint.
(b) Error under a staleness constraint.

**Figure 6.** Comparison of Random and Caching algorithms for Sum aggregation under dynamic workload and WAN bandwidth on a local testbed. Arrival rate increases during window 6 (dashed vertical line), and available bandwidth decreases during window 20 (dotted vertical line).
Questions

1. Which kinds of applications do not need grouped aggregation data?
2. For one key, how many times it is flushed in one window?