Time-Decaying Aggregates in Out-of-order Streams

Graham Cormode

Flip Korn

AT&T Labs - Research

{graham,flip}@research.att.com

Srikanta Tirthapura

Iowa State

snt@iastate.edu

Outline

- Aggregate Computation under Time Decay
- Sliding Window Approach
  - Approximating Counts on out-of-order streams
  - Ranges and application for aggregates
- Value Division Approach
- Experimental Comparison

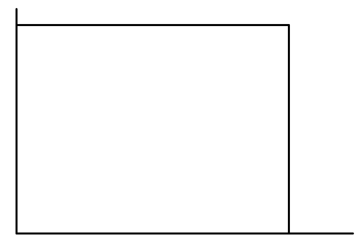
Data Stream Computations

- Streams of updates must be processed in single pass
  - IP traffic measurements, stock feeds, sensor readings
- Need to mine holistic aggregates
  - Medians (quantiles), frequent items
- Recent updates more important than older data
  - Weight updates based on a function of age
  - Quality issues: Data may not be seen in timestamp order
- Need to keep small summaries, give accurate answers
  - Much work on sketches, summaries without decay

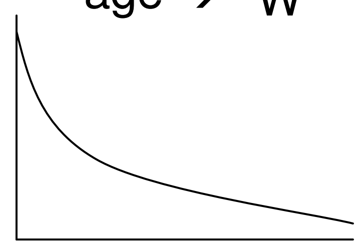
Decay Functions

- Given age a, g returns *decayed weight* of the item
  - Require $g(0) = 1$ and $0 \le g(a) \le 1$ for $a>0$

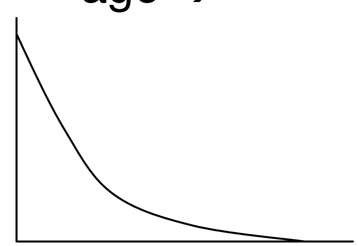
- Sliding window, size W:
  $g(a) = 1$ for $a < W$, else $g(a)=0$



- Polynomial decay, parameter α:
  $g(a) = (a+1)^{-\alpha}$

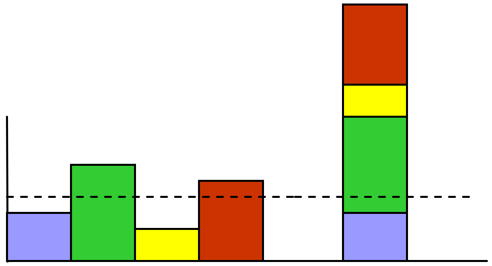


- Exponential decay, parameter λ:
  $g(a) = \exp(-\lambda a)$



Aggregates of Interest

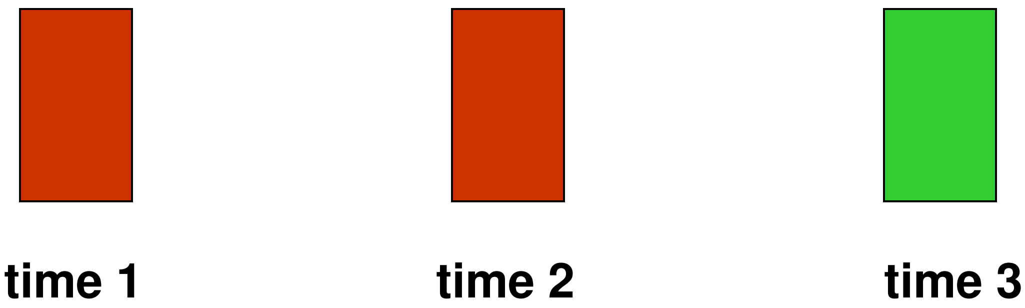
- Streaming model: Given stream of $\langle t_i, v_i \rangle$ tuples
  - v_i is an item, t_i its timestamp
  - E.g. IP flow, start time
  - Total weight at time t under g is $D(t) = \sum_i g(t - t_i)$
- ϕ-Heavy Hitters
  - Find items v so that $\sum_{vi = v} g(t-t_i) > \phi\ D(t)$



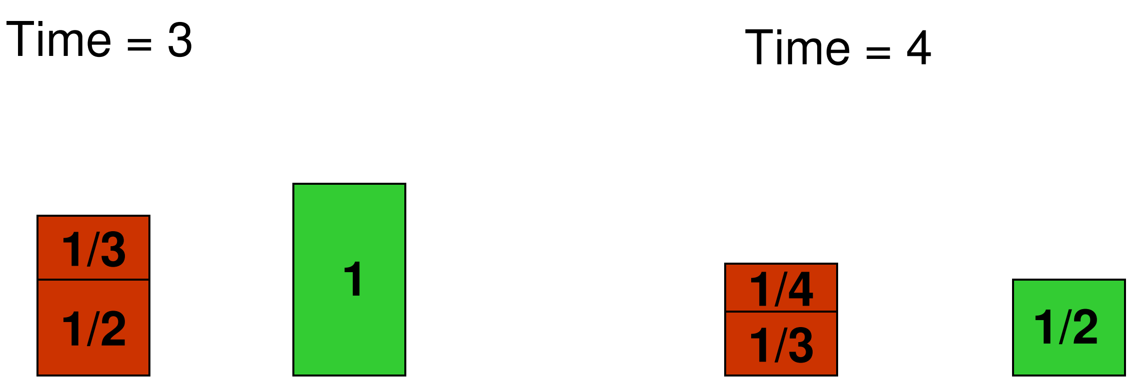
- ϕ-Quantiles
  - Find q so that $(\phi - \varepsilon)D(t) \leq \sum_{vi \leq q} g(t-t_i) \leq (\phi + \varepsilon)D(t)$

- $g(x)=1$: same as standard approx heavy hitters/quantiles

Example



Decay fn. $g(a) = 1/(1+a)$ Heavy hitters with $\phi = 1/2$



Prior Work

- Much prior work has focused on aggregate computation under sliding window
  - Exponential histograms [Datar,Gionis,Indyk,Motwani'02]
  - Deterministic waves [Gibbons,Tirthapura'02]
  - Quantiles and heavy hitters [Arasu,Manku'04]
  - Tighter bounds for heavy hitters [Lee, Ting'06]
- But this work *critically* assumes in-order arrivals
  - Some study of counts and samples for arrivals not ordered by timestamp [Busch Tirthapura '07, Cormode, Tirthapura, Xu '07]
- Little work on aggregates under other decay functions
  - Counts and sums under general decay [Cohen, Strauss'03]

Our Results

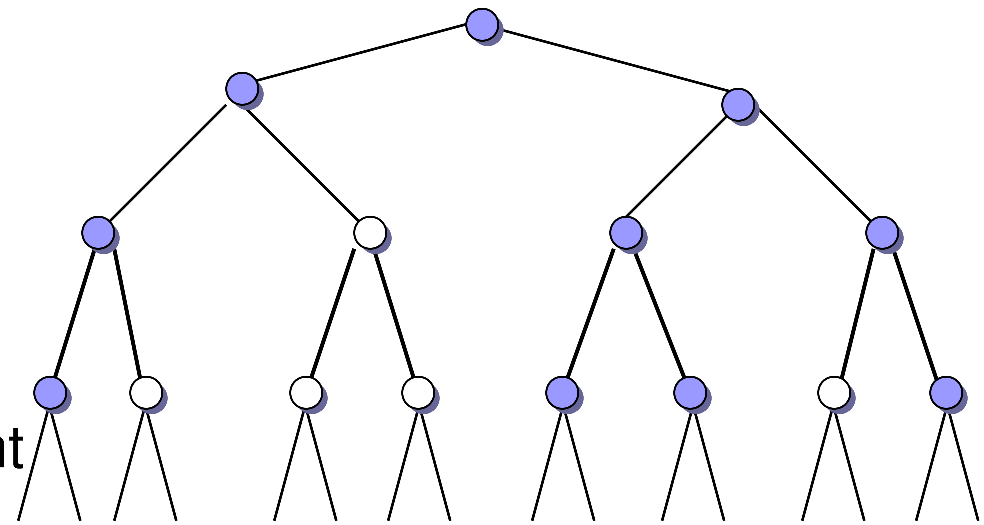
- First results for quantiles and heavy hitters under arbitrary decay, out of order arrivals.
- Two approaches yield poly(log N,1/ε,log W) solutions:
  1. Solve sliding window problem, then reduce other decay functions to multiple instances of sliding window
  2. Use decay-function specific division of time domain and bound number of mergable summaries kept
- Both methods give deterministic guarantees, independent of the amount of disorder in stream
- Better method depends on desired decay function

Sliding Window Count

- First analyze count under sliding window
  - Needed for other computations
  - Technique is generalized for more complex aggregates
- Given w at query time t, compute how many items arrives between $w-t$ and t with relative error ε
- N = upper bound on # arrivals, W = upper bound on w
- Keep $J = \log (\varepsilon N / \log W)$ summaries Q_j
- Q_j summarizes the $9 \log W 2^j/\varepsilon$ most recent arrivals
- Q_0 simply buffers the $9 \log W/\varepsilon$ most recent items

Summary Structures

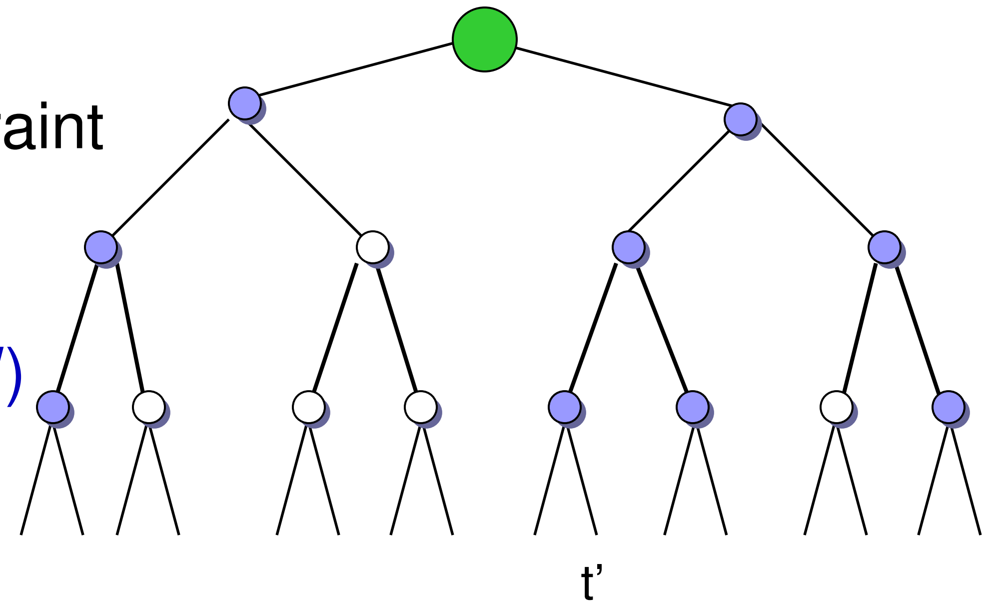
- Q_j based on Q-digest [Hershberger, Shrivastava, Suri, Toth '04]
- Impose binary tree on top of time domain
- Track counts satisfying
  - If node has non-zero count, so does its parent
  - Each non-leaf range has count $\leq 2^j$
  - Each node, sibling, parent triple has count $\geq 2^j$



- Retain at most α=9log W/ε ranges, discard old ranges

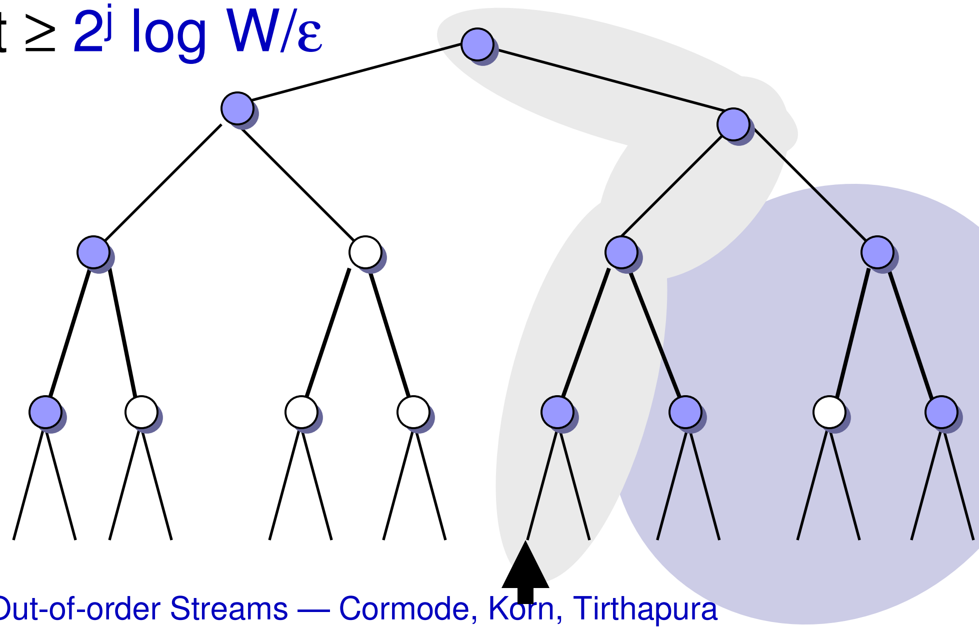
Summary Maintenance

- Given arrival of new item with timestamp t' (possibly out of order), find smallest range containing it
- Add there, or in child if it would violate count constraint
- Simple searching takes O(log W), binary search on path takes O(log log W)
- The data structure gives additive error 2^j on window count queries



Window Count Estimation

- Given window size w at time t
  - Find smallest j>0 so that Q_j contains t-w
  - Estimate by sum of counts of ranges later than t-w
- Error: only ranges that are ancestors of t-w
  - By contstraint, these contribute at most $2^j \log W$
  - By choice of α, true count $\geq 2^j \log W/\varepsilon$
  - Error / true count $\leq$
    $2^j \log W/(2^j \log W/\varepsilon) = \varepsilon$

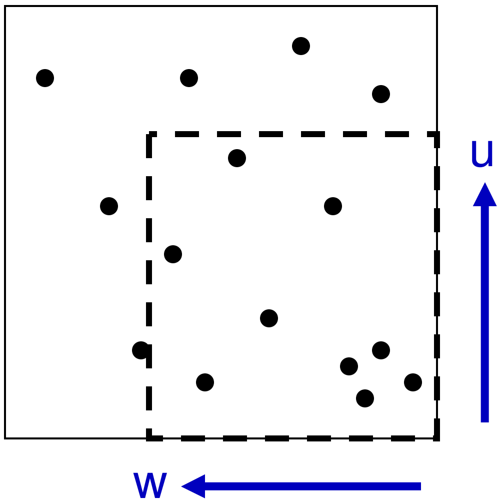


Space and Time Cost

- Bound size based on simple counting argument
  - Each summary Q_j has total count $\approx \alpha\, 2^j$
  - Each triple (parent and children) has count $\geq 2^j$
  - So size is $O(\alpha) = O(\log W/\varepsilon)$
  - Over $J = \log (\varepsilon N/\log W)$ summaries
  - Total space = $O(\log W/\varepsilon \log (\varepsilon N/\log W))$
- Time cost:
  - Periodic pruning of summaries takes linear time
  - Amortized $O(\log (\varepsilon N / W))$ per update

Sliding Window Range Queries

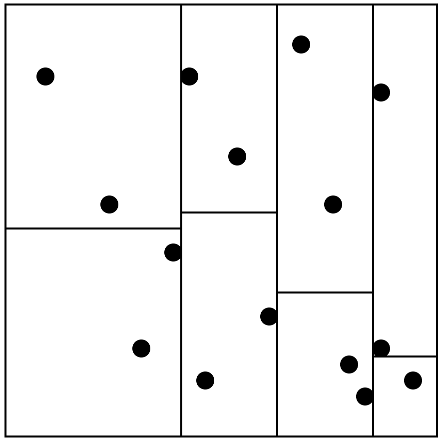
- Range queries are a stepping stone to other aggregates
- 2D Ranges: updates are a sequence of (t,v) pairs
- Queries: time window w, value window u
  - R(w,u) = count of points that fall in this range



- Require error in count to be ε D(t)
  - Chosen to match requirements for quantiles and heavy hitters
  - Cheaper than guaranteeing ε R(w,u)

Supporting Range Queries

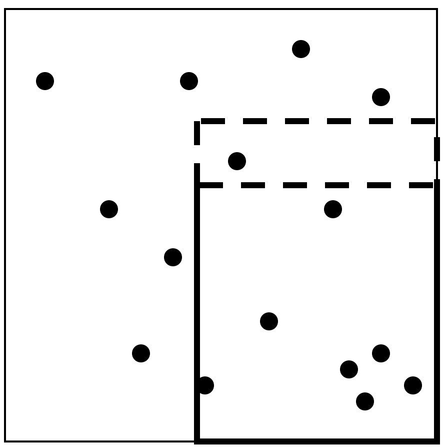
- Solution: keep Q_j structures as before on the time dimension (ignoring value dimension)
- Within each node in Q_j structure, keep a second summary on values of items summarized by that node



- Various choices of exactly how to implement, details in paper
- Space: $O(1/\varepsilon \log^2 W \log \varepsilon N)$
- Time: $O(\log \log U \log W \log \varepsilon N)$

Reductions to Range Queries

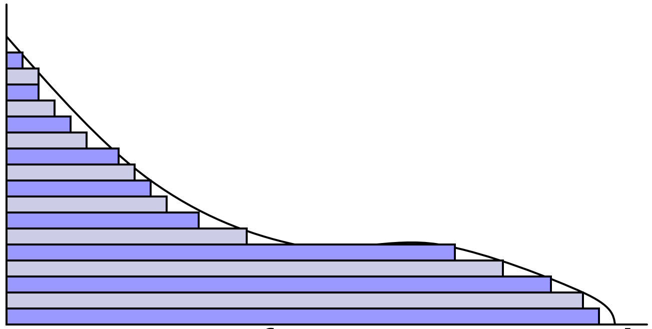
- Both quantiles and heavy hitters in sliding windows can be answered by range queries
- Quantiles: find u so that $R(w,u) \approx \phi\ D(t)$
- Heavy hitters: find u so that $R(w,u) - R(w,u\text{-}1) \geq \phi\ D(t)$



So bounds on previous slide immediately apply for these problems

Arbitrary Decay Functions

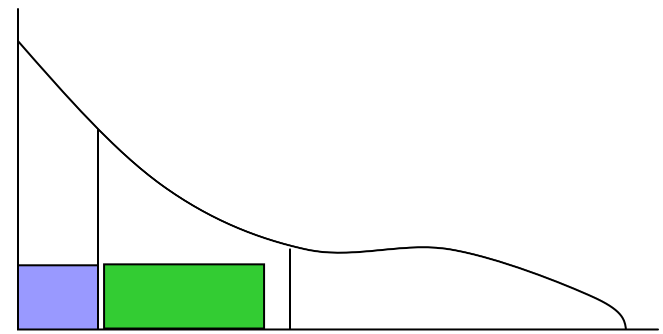
- [Cohen-Strauss'03]: sum under arbitrary decay functions can be reduced to scaled sums of sliding window queries
- Same observation holds for quantiles and HHs: these aggregates are composed of counts



- In particular, can approximate count of a range under arbitrary decay function specified at query time
- Make efficient by evaluating at specific time windows
- Space and time cost same as for window decay

Value Division

- Alternate solution for certain “smooth” decay functions
- Divide time into regions where decay function varies by at most $(1+\varepsilon)$ factor



- Keep at most one summary of items falling between two divisions
- Merge pairs of summaries that fall between two divisions
- Number of summaries is $O(\log_{1+\varepsilon} g(W)) = O(1/\varepsilon\ g(W))$

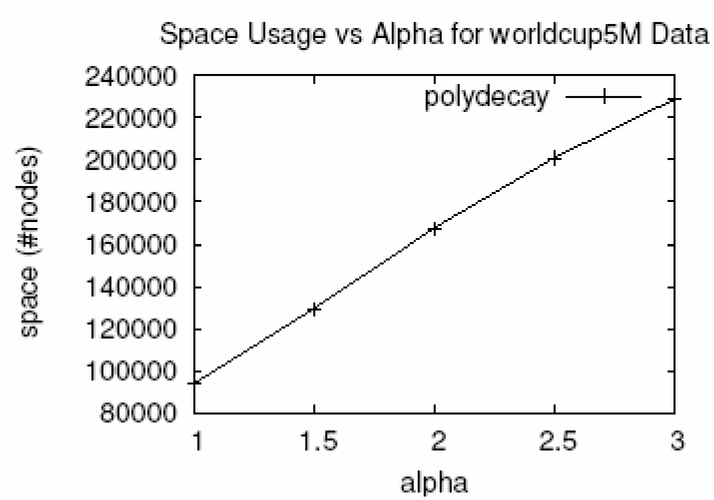
Value Division Analysis

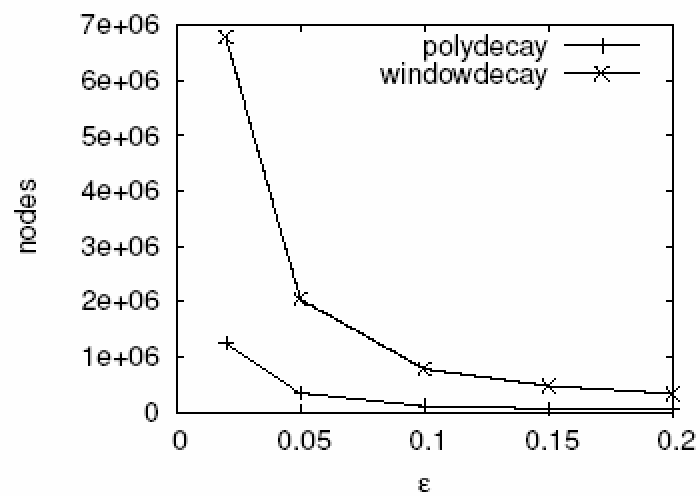
- Requires that a decay function be fixed in advance to determine boundaries
  - But: can still choose a different decay function at query time, provided it is “dominated” by the default function
- Naturally accommodates out-of-order arrivals
- For polynomial decay and quantiles, space cost is $O(1/\varepsilon^2 \log U \log t)$, depends linearly on poly exponent

Experimental Set Up

- Implemented these algorithms in C, compared to the undecayed case
- Evaluated on data sets of 5M requests to WorldCup'98 webserver, and on 5M flows from large ISP network
  - WorldCup'98: Introduced large disorder by dropping date information from timestamps
  - Network data: Contains some moderate disorder by using **`begin_time`** as timestamp on data sorted by **`end_time`**
- Compared no decay to
  - sliding window
  - polynomial (via value division)

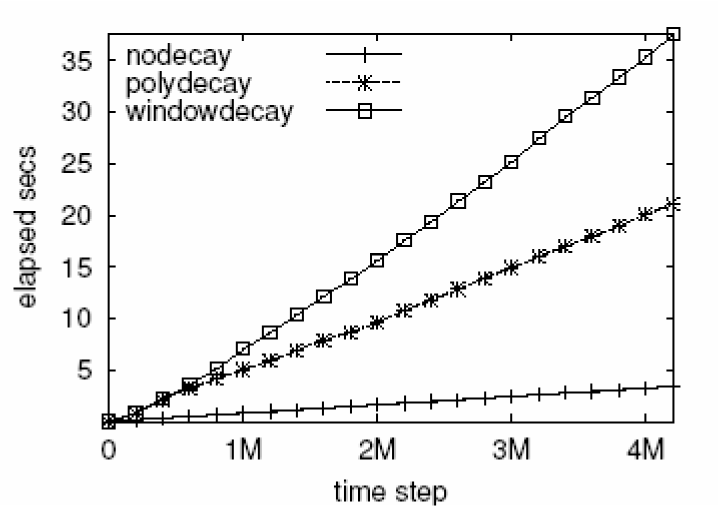
Space Cost

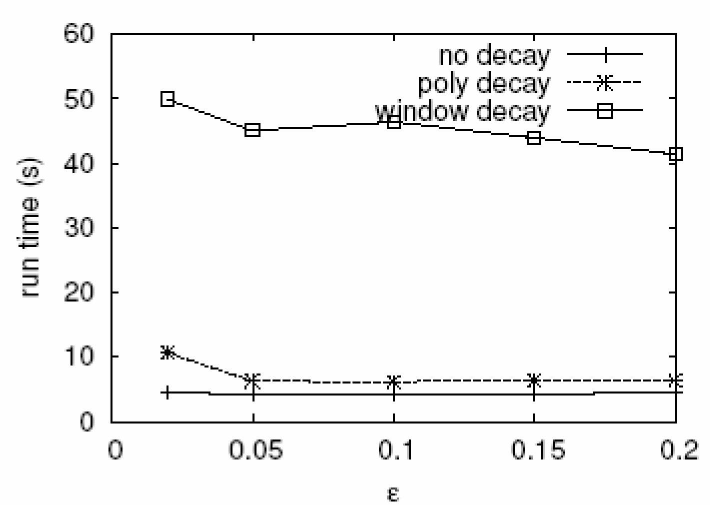




- Space cost scales linearly with polynomial exponent for smooth decay, as predicted
- Space of window-based approach several times larger than for smooth decay approach

Time Cost





- Time scales near linearly with the input size
- Window decay approximately 10x slower than no decay
- Smooth decay can be close to cost of no decay

Conclusions

- Novel algorithmic techniques required to compute aggregates with time decay on out-of-order streams
- Results come at a cost compared to no decay, but still practical
- Always some limitations (assumptions on time domain, or on smoothness of decay function)
- Natural questions: other aggregates, improved bounds