Exponentially Decayed Aggregates on Data Streams

Graham Cormode
Flip Korn
AT&T Labs - Research
{graham,flip}@research.att.com

Srikanta Tirthapura
Iowa State
snt@iastate.edu
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
- Need to keep small summaries, give accurate answers
  - Much work on sketches, summaries without decay
Exponential Decay

- Give items a weight that depends exponentially on age
  - Age \( a \) has weight \( \exp(-\lambda a) \), for parameter \( \lambda \)
- Given a stream of timestamps \( t_i \), easy to compute decayed count
  - Track current decayed count \( C \), last update \( t \)
  - Given \( t_i \), \( C \leftarrow C \times \exp(-\lambda(t_i - t)) + 1 \)
    \( t \leftarrow t_i \)
  - Generalizes to exponentially decayed sum
- We study more complex (holistic) aggregates, show that there are fast, small solutions for these as well.
Aggregates of Interest

- **Streaming model**: Given stream of \( \langle x_i, w_i, t_i \rangle \) tuples
  - \( x_i \) is an item, \( w_i \) its weight, \( t_i \) its timestamp
  - E.g. IP flow, weight in bytes, start time
  - Total weight at time \( t \) is \( D(t) = \sum_i w_i \exp(\lambda(t - t_i)) \)

- **\( \phi \)-Heavy Hitters (under exponential decay)**
  - Find items \( x \) so that \( \sum_{x_i = x} w_i \exp(\lambda(t-t_i)) > (\phi \pm \epsilon) D(t) \)

- **\( \phi \)-Quantiles (under exponential decay)**
  - Find \( q \) so that \( (\phi - \epsilon)D \leq \sum_{x_i \leq q} w_i \exp(\lambda(t-t_i)) \leq (\phi + \epsilon)D \)

- \( \lambda = 0 \) \( \Rightarrow \) no decay
  - Same as standard approximate heavy hitters/quantiles
Decayed Heavy Hitters

Algorithm 1.1: HeavyHitterUpdate($x_i, w_i, t_i, \lambda$)

**Input:** item $x_i$, timestamp $t_i$, weight $w_i$, decay factor $\lambda$

**Output:** Current estimate of item weight

if $\exists j$. item[$j$] = $x_i$;
    then $j \leftarrow$ item$^{-1}$(x$_i$)
    else $j \leftarrow$ arg min$_k$(count[k]);

item[$j$] $\leftarrow$ $x_i$;

count[$j$] $\leftarrow$ count[$j$] + $w_i$ exp($\lambda t_i$)

return (count[$j$] * exp($-\lambda t_i$))

- We extend the “SpaceSaving” algorithm of [MAA05]
  - Keep $k = 1/\varepsilon$ items and counters
  - If next item is stored, update its count with $+ w_i \exp(\lambda t_i)$
  - Else, overwrite the stored item with smallest count
  - On output, scale stored counts by $\exp(-\lambda t)$
**SpaceSaving Analysis**

- Smallest counter value, $\min$, is at most $\varepsilon D \cdot \exp(t)$
  - Counters sum to $N = D(t) \exp(t)$ by induction
  - $1/\varepsilon$ counters, so avg is $\varepsilon N$: smallest cannot be bigger

- True count of an uncounted item is between 0 and $\min$
  - Proof by induction, true initially, $\min$ increases monotonically
  - Hence, the count of any item stored is off by at most $\varepsilon N$

- Any item $x$ whose true count $> \varepsilon N$ is stored
  - By contradiction: $x$ was evicted in past, with $\text{count} \leq \min_t$
  - Every count is an overestimate, using above observation
  - So est. count of $x > \varepsilon N \geq \min \geq \min_t$, and would not be evicted

So: Find all items with count $> \varepsilon N$, error in counts $\leq \varepsilon N$
Heavy Hitters Result

- Algorithm finds $\varepsilon$-approximate exponentially decayed heavy hitters in space $O(1/\varepsilon)$, update time $O(\log 1/\varepsilon)$.
  - Time cost: keep a standard heap to allow find minimum
  - Index items with efficient dynamic tree structure (deterministic) or hash table (randomized)

- Space and time costs asymptotically the same as the non-decayed version.
Decayed Quantiles

- **Q-digest** [SBAS05] summarizes distribution over fixed domain \( U \).
- Tracks nodes and counts in binary tree over domain.
- Ensures that non-leaf nodes have small counts \( \leq \varepsilon \frac{N}{\log U} \).
- Ensures that counts of parents + two children \( > \varepsilon \frac{N}{\log U} \).
- Guarantees any quantile can be found with \( \varepsilon N \) error.
- Space used is bounded by \( O(\log U/\varepsilon) \).
Extended Q-digest

- Replace counters by exponentially decayed counts
- Observe that for the Q-digest:
  - updates can be arbitrary fractional values
  - multiplying all counts by $\gamma$ gives a summary of input multiplied by $\gamma$
- Hence: sum of all counts is $D(t)$, no count $> \frac{\varepsilon D}{\log U}$ etc.
- Careful analysis shows that we can estimate the rank of any item using the stored counts
  - Error arises from counts of ancestor nodes from leaf to root
  - Total error is bounded by $\varepsilon D(t)$, as required
Quantiles Result

- Can find $\varepsilon$-approximate decayed quantiles in space $O(1/\varepsilon \log U)$
- Time per update $O(\log \log U)$, queries take $O(\log U/\varepsilon)$
  - Updates are fast based on a ‘lazy’ update procedure: Don’t decay all counts every update, only those affected by the update

- Space and time independent of $\lambda$.
  - Same space as non-decayed version
  - Slightly slower because of count maintenance
Experimental Evaluation

- Implemented q-digest in C, experimented on 5 million World Cup requests and 5 million IP flow records
- **Space cost**: almost identical to non-decayed space
Experimental Throughput

- **Throughput**: about 70-80% as fast as undecayed version
- **Processing**: about 800K updates per second
Conclusions

- Quantiles and frequent items under exponential decay
  - Cost is very close to that for no decay
  - Adapt existing algorithms to handle decayed counts
  - Extend analysis to show correctness and throughput

- Other decay functions:
  - Polynomial decay, logarithmic decay, sliding window
  - Harder, even for simple sums and counts [CS03]
  - New algorithms for aggregates given in [CKT08], in PODS