Biased Quantiles

Graham Cormode
cormode@bell-labs.com

S. Muthukrishnan
muthu@cs.rutgers.edu

Flip Korn
flip@research.att.com

Divesh Srivastava
divesh@research.att.com
Quantiles summarize data distribution concisely.

Given $N$ items, the $\phi$–quantile is the item with $\text{rank } \phi N$ in the sorted order.

Eg. The median is the 0.5-quantile, the minimum is the 0-quantile.

Equidepth histograms put bucket boundaries on regular quantile values, eg 0.1, 0.2...0.9

Quantiles are a robust and rich summary: median is less affected by outliers than mean.
**Quantiles over Data Streams**

Data stream consists of $N$ items in arbitrary order.

Models many data sources eg network traffic, each packet is one item.

Requires linear space to compute quantiles exactly in one pass, $\Omega(N^{1/p})$ in $p$ passes.

$\epsilon$-approximate computation in sub-linear space

- $\Phi$-quantile: item with rank between $(\Phi-\epsilon)N$ and $(\Phi+\epsilon)N$
- [GK01]: insertions only, space $O(1/\epsilon \log(\epsilon N))$
- [CM04]: insertions & deletions, space $O(1/\epsilon \log U \log 1/\delta)$
Why Biased Quantiles?

IP network traffic is very skewed
- Long tails of great interest
- Eg: 0.9, 0.95, 0.99-quantiles of TCP round trip times

Issue: uniform error guarantees
- $\varepsilon = 0.05$: okay for median, but not 0.99-quantile
- $\varepsilon = 0.001$: okay for both, but needs too much space

Goal: support relative error guarantees in small space
- Low-biased quantiles: $\phi$-quantiles in ranks $\phi(1 \pm \varepsilon)N$
- High-biased quantiles: $(1-\phi)$-quantiles in ranks $(1-(1 \pm \varepsilon)\phi)N$
Prior Work

- Sampling approach due to Gupta & Zane [GZ03]
  - Keep $O(1/\varepsilon \log N)$ samplers at different sample rates, each keeping a sample of $O(1/\varepsilon^2)$ items
  - Total space: $O(1/\varepsilon^3)$, probabilistic algorithm

- Deterministic alg [CKMS05]
  - Worst case input causes linear space usage
  - Showed lower bound of $\Omega(1/\varepsilon \log \varepsilon N)$

- Improved probabilistic alg of Zhang+ [ZLXKW05]
  - Needs $O(1/\varepsilon^2 \text{polylog } N)$ space and time
Our Approach

Domain-oriented approach: items drawn from $[1...U]$, want space to depend on $O(\log U)$

- Impose binary tree structure over domain
- Maintain counts $c_w$ on (subset of) nodes
- Count represents input items from that subtree

So counts to left of a leaf are from items strictly less; uncertainty in rank of item is from ancestors

Similar to [SBAS04] approach for uniform quantiles
Functions over the tree

We define some functions to measure counts over the tree.

- \( \text{lf}(x) = \) leftmost leaf in subtree \( x \)
- \( \text{anc}(x) = \) set of ancestors of node \( x \)
- \( L(v) = \sum_{\text{lf}(w) < \text{lf}(v)} c_w \) (Left count)
- \( A(x) = \sum_{w \in \text{anc}(x)} c_w \) (Ancestor count)
Accuracy Invariants

To ensure accurate answers, we maintain two invariants over the set of counts:

∀ x. L(x) − A(x) ≤ rank(x) ≤ L(x)  \hspace{1cm} ①

ensures we can deterministically bound ranks

∀ v. v ≠ lf(v) ⇒ (c_v \leq \alpha \cdot L(v))  \hspace{1cm} ②

ensures range of possible ranks is bounded

To guarantee ε-accurate ranks, will set \( \alpha = \frac{\varepsilon}{\log U} \) (since we use ② summed over log U ancestors)

Claim: any summary satisfying ① and ② allows us to find \( r'(x) \) so \( |r'(x) - \text{rank}(x)| \leq \varepsilon \cdot \text{rank}(x) \)
Maintenance

Need to show how to maintain the accuracy invariants, while guaranteeing space is bounded and updates are fast.

- Will Insert each update \( x \). Insert will be defined to maintain accuracy, but space may grow.

- Periodically will run a linear scan of data structure to Compress it.

- Will argue that these two together maintain space and time bounds.
Data Structure

Store subset of nodes and counts as “bq-summary”

Nodes with count 0 do not need to be stored

Split bq into two: bq-leaves (bql) and bq-tree (bqt).

This division is needed to get tightest space bounds.

- bq-leaves is a subset of leaf nodes only
- bq-tree is subset of nodes strictly to right of bq-leaves
Space Conditions

We will maintain four additional conditions to ensure space is bounded. Set \( z = \max_{u \in bql} u. \)

\[ z < \text{lf}(\text{par}(v \in bqt)) \Rightarrow c_{\text{par}(v)} \geq \alpha L(\text{par}(v)) \quad 3 \]
Ensures parents of the bqt nodes are full

\[ \frac{1}{\alpha} \log(\alpha N) \geq |bql| \geq \min(N, \frac{1}{\alpha}) \quad 4 \]
Ensures not too many or too few bql nodes

\[ z < \min_{v \in bqt} \text{lf}(v) \quad 5 \]
Ensures bq-leaves to left of bq-tree nodes

\[ \sum_{v \in bql \cup bqt} c_v = N \quad 6 \]
Sanity check on conservation of counts
Space Bound Outline

Will show that maintaining all six conditions ensures that space is tightly bounded.

Main effort is in proving size of $bqt$ is bounded.

Will divide $bqt$ into “equivalence classes” based on increasing $L()$ values.

Since each $L()$ value of class must increase by a multiplicative factor, can bound total space.
Equivalence Classes

Only consider “full” nodes $V$ in bqt (with at least one child present): by $\mathbb{3}$, for $v \in V$, $c_v = \alpha L(v)$

- Partition $V$ into equivalence classes based on $L(v)$

- $E_i$ is set of nodes in $i$’th equivalence class, with $L$ value = $L_i$

- $L_1$ is sum of bq-leaves: $L_1 = \sum_{v \in bql} c_v$

Example with $\alpha = \frac{1}{2}$
Space Bound

- By 4, we have $|bql| = L_1 \geq 1/\alpha$
- The $L_i$’s increase exponentially, can show $L_{i+1} \geq L_1 \prod_{j=1}^{i}(1+\alpha|E_j|)$
- Consider item $U+1$, so $\text{rank}(U+1)=N$.
- By 6, $N = L(U+1) \geq 1/\alpha \prod_{j=1}^{q}(1 + \alpha|E_j|)$
- Taking logs allows us to bound size of $|bqt|$
- So total space $= |bql| + |bqt|$
  $= O(1/\varepsilon \log (\varepsilon N) \log U)$
Insert Procedure

Must show we can maintain data structure quickly

Insert allows space constraints to lapse slightly by using old (pre-calculated and stored) \( L() \) values.

Given update item \( x \):

- Compare to \( z = \max_{u \in bql} u \)
- If \( x \leq z \), place \( x \) in \( bql \) in time \( \mathcal{O}(1) \)
- If \( x > z \) place \( x \) in \( bqt \) in time \( \mathcal{O}(\log \log U) \):
  - Find closest materialized ancestor \( y \) of \( x \) in \( bqt \)
  - Add 1 to \( c_y \) unless this would make \( c_y > \alpha L(y) \), if so then create child of \( y \) with count = 1
Accuracy of Insert

*Insert* procedure maintains \( \boldsymbol{1} \), \( \boldsymbol{2} \), \( \boldsymbol{5} \), and \( \boldsymbol{6} \)

Fairly easy to check each of these, e.g.

\[
\forall x. \, L(x) - A(x) \leq \text{rank}(x) \leq L(x) \quad \boldsymbol{1}
\]

- Inserting into *bq*, increases \( L(x) \) and \( \text{rank}(x) \) for everything to the right of inserted item.

- Other conditions preserved either by inspection, or by design of *Insert* routine (eg inserting into child node if inserting into \( y \) would break \( \boldsymbol{2} \)
Compress

- If we keep inserting, space can grow without limit, but in worst case, we add one new node per insert, so Compress when space doubles

- Need to periodically recompute $L()$ values for nodes, and merge together nodes when possible
  - First, resize bq-leaves so $|bql| = \min(N, 1/\alpha)$
  - Recompute $z = \max_{v \in bql} v$ in time linear in $|bql|$, Insert leaves removed from $|bql|$ into $bqt$.
  - Tricky part is compressing bq-tree...
**Compress Tree**

- “Compress Tree” operation takes a (sub)tree in `bqt`, ensures that each node becomes “full” (has count = $\alpha L(v)$) by “pulling up” weight from below
  - For node $v$ compute $L(v)$ and $wt(v) = \sum_{v \in \text{anc}(w)} c_w$
  - Set $c_v$ as big as possible by borrowing from $wt(v)$
  - If $c_v = \alpha L(v)$, then recurse on children in order
  - Else, we have accounted for all weight below, so delete all descendents

- With care, Compress Tree takes time $O(|bqt|)$ and computes $L(v)$ incrementally as a side effect

- Can show that Compress maintains conditions 1, 2, 5, and 6 and restores conditions 3 and 4
Final Result

- Can answer rank queries with error $\varepsilon \text{rank}(x)$, using space $O(1/\varepsilon \log \varepsilon N \log U)$, and amortized update time $O(\log \log U)$.
  - Lower bound on space = $O(1/\varepsilon \log (\varepsilon N))$

- To answer queries, need latest values of $L(v)$, so need time $O(1/\varepsilon \log \varepsilon N \log U)$ to preprocess
  - Can then answer queries in time $O(\log U)$ each
  - Alternatively, spend $O(\log U)$ time on updates and allow $L(v)$ values to be computed in time $O(\log U)$
  - Quantile queries can be answered by binary searching for item with desired rank
Extensions

- Partially biased algorithm
  - Sometimes only need accuracy down to some $\varepsilon'N$
  - Can reduce space slightly for this weaker guarantee
  - Space required is $O(1/\varepsilon \log (\varepsilon/\varepsilon') \log U)$

- Uniform algorithm
  - The Compress Tree idea can be applied to $\varepsilon N$ error
  - bq-leaves not needed, space used is $O(1/\varepsilon \log U)$
  - Time is $O(\log \log U)$ amortized as before
Experimental Results

- CKMS, MRC = prior work, SBQ = this work
- Outperforms prior work in both time and space
Commentary

- Took some amount of effort to get the invariants and conditions “just right”:
  - Small changes to conditions meant either space or time bounds would break
  - bq-leaves needed to ensure that space bounds are as tight as possible

- Easy to merge together summaries to get summary of union (for distributed computations)
  - Linearity of $L$ and $A$ means everything goes through
Conclusions

- Close to optimal space bounds
  - What about faster updates, less work for queries?
- Made crucial use of tree-structure over universe
  - Any way to drop U and work over arbitrary domains?