Biased Quantiles

Graham Cormode
cormode@bell-labs.com

Flip Korn
flip@research.att.com

S. Muthukrishnan
muthu@cs.rutgers.edu

Divesh Srivastava
divesh@research.att.com

Quantiles

Quantiles summarize data distribution concisely.

Given N items, the ϕ–quantile is the item with *rank* ϕN 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.

ε-approximate computation in sub-linear space

- Φ-quantile: item with rank between (Φ-ε)N and (Φ+ε)N
- [GK01]: insertions only, space O(1/ε log(εN))
- [CM04]: insertions & deletions, space O(1/ε log U log 1/δ)

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: ϕ-quantiles in ranks $\phi(1\pm\varepsilon)N$
- High-biased quantiles: $(1-\phi)$-quantiles in ranks $(1-(1\pm\varepsilon)\phi)N$

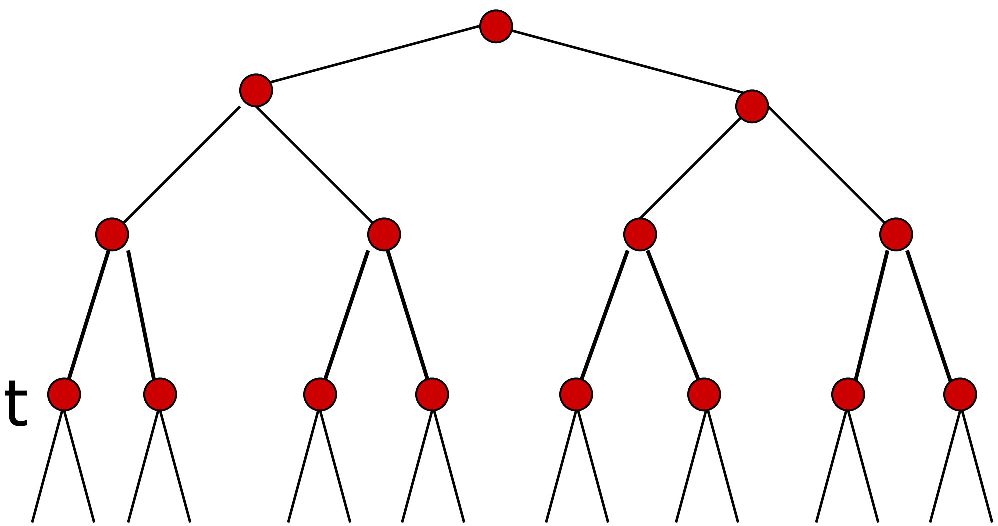
Prior Work

- 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



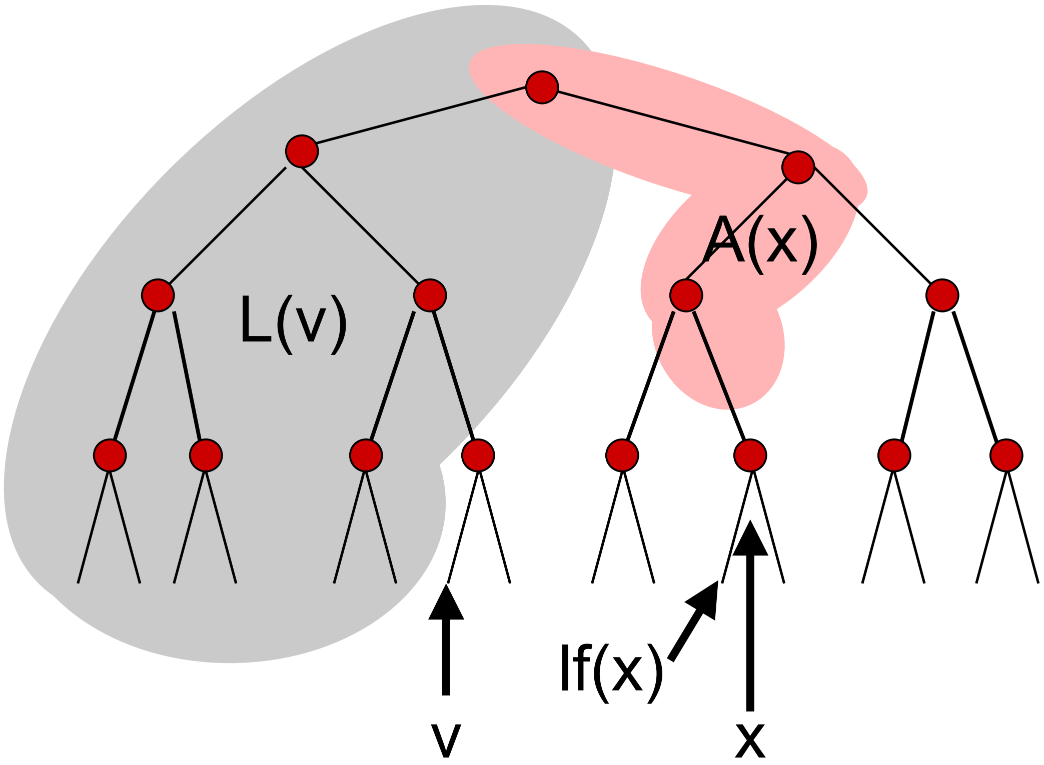
So counts to left of a leaf are from items strictly less; uncertainty in rank of item is from ancestors

Similar to [SBAS04] approach for uniform quantiles

Functions over the tree

We define some functions to measure counts over the tree.

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



Accuracy Invariants

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

$$\forall x.\ L(x) - A(x) \leq \text{rank}(x) \leq L(x) \qquad ❶$$

ensures we can deterministically bound ranks

$$\forall v.\ v \neq \text{lf}(v) \Rightarrow (c_v \leq \alpha\, L(v)) \qquad ❷$$

ensures range of possible ranks is bounded

To guarantee ε-accurate ranks, will set $\alpha = \varepsilon/\log U$ (since we use ❷ summed over log U ancestors)

Claim: any summary satisfying ❶ and ❷ allows us to find $r'(x)$ so $|r'(x) - \text{rank}(x)| \leq \varepsilon\, \text{rank}(x)$

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

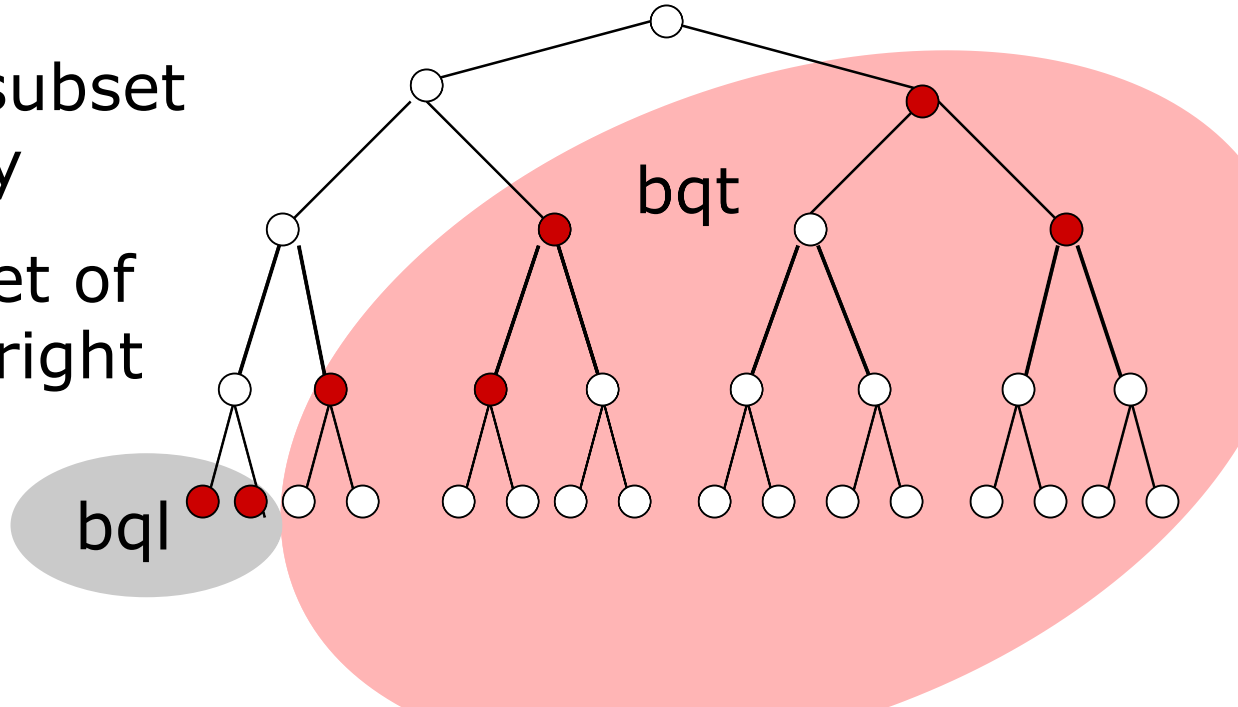
Store subset of nodes and counts as "bq-summary"

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 $z = \max_{u \in bql} u$.

$z < lf(par(v \in bqt)) \Rightarrow c_{par(v)} \geq \alpha L(par(v))$ ❸

Ensures parents of the bqt nodes are full

$1/\alpha \log(\alpha N) \geq |bql| \geq \min(N, 1/\alpha)$ ❹

Ensures not too many or too few bql nodes

$z < \min_{v \in bqt} lf(v)$ ❺

Ensures bq-leaves to left of bq-tree nodes

$\sum_{v \in bql \cup bqt} c_v = N$ ❻

Sanity check on conservation of counts

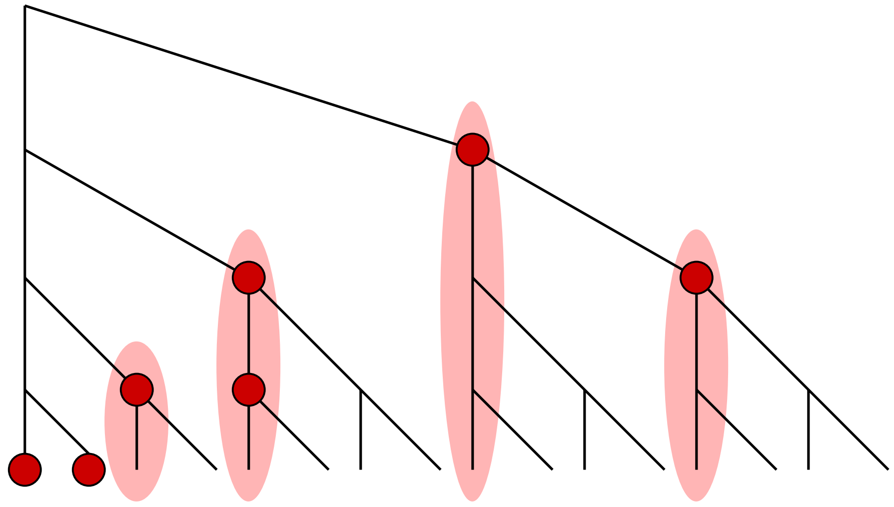
Space Bound Outline

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

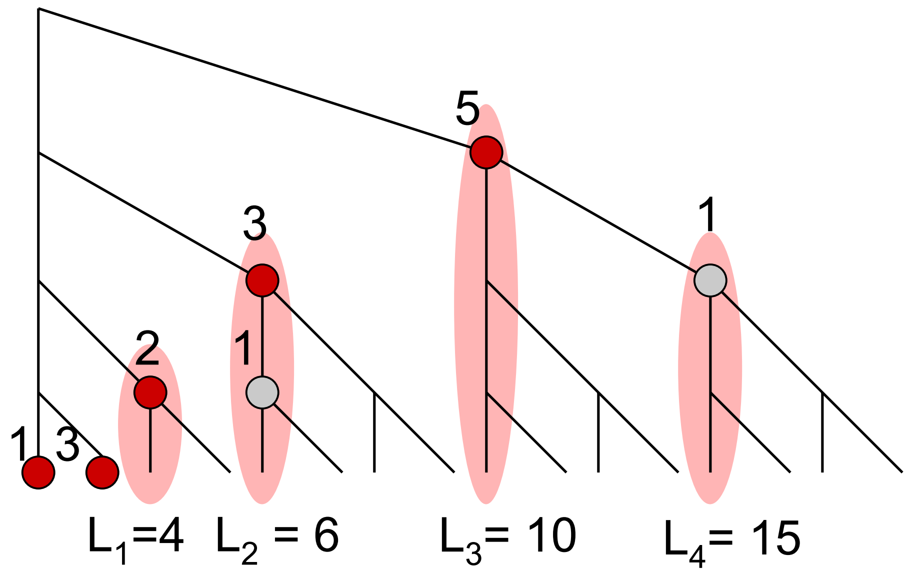


Equivalence classes

Equivalence Classes

Only consider "full" nodes V in bqt (with at least one child present): by ❸, 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 $\alpha = ½$

Space Bound

- By ❹ we have $|bql| = L_1 \geq 1/\alpha$
- The L_i's increase exponentially, can show $L_{i+1} \geq L_1 \Pi_{j=1}^{i}(1+\alpha|E_j|)$
- Consider item $U+1$, so $rank(U+1)=N$.
- By ❻, $N = L(U+1) \geq 1/\alpha \, \Pi_{j=1}^{q} (1 + \alpha|E_j|)$
- 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 $O(1)$
- If $x > z$ place x in bqt in time $O(\log \log U)$:
  - 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 ❶, ❷, ❺, and ❻

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

$$\forall x.\ L(x) - A(x) \leq rank(x) \leq L(x) \quad ❶$$

- Inserting into bq, increases L(x) and 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 ❷)

Compress

- If we keep `Insert`ing, 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 $wt(v) = \sum_{v \in anc(w)} c_w$
  - 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 ❶, ❷, ❺, and ❻ and restores conditions ❸ and ❹

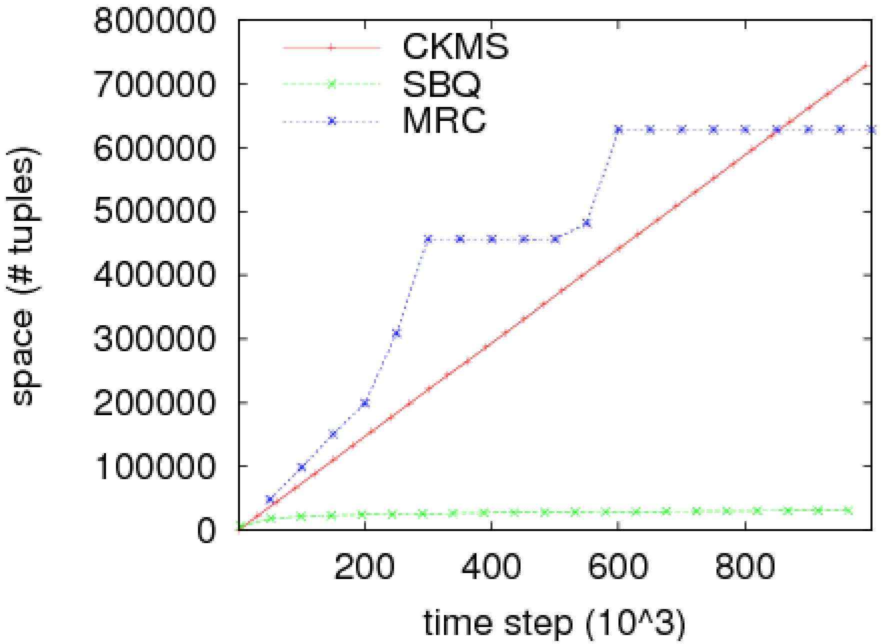
Final Result

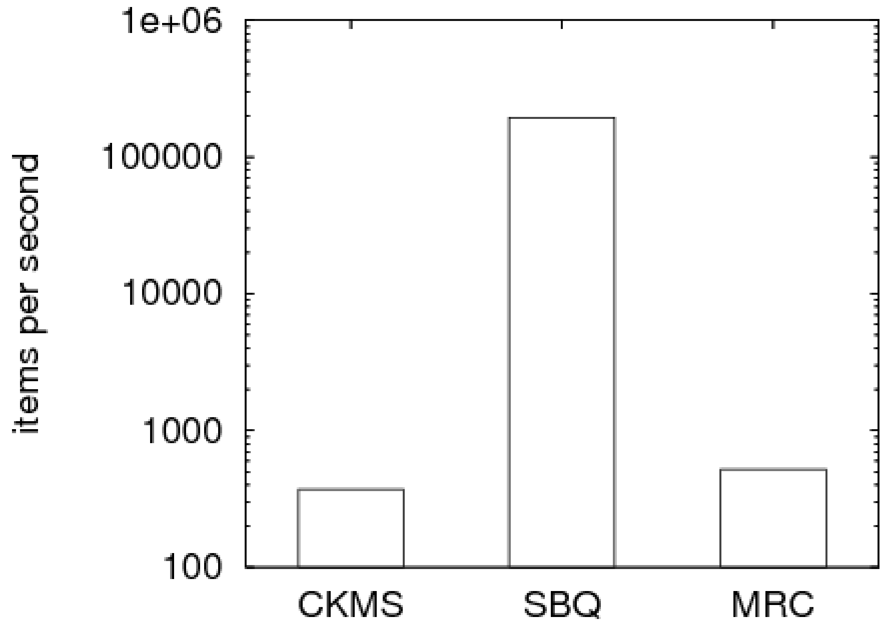
- Can answer rank queries with error ε rank(x), using space $O(1/\varepsilon \log \varepsilon N \log U)$, and amortized update time $O(\log \log U)$.
  - Lower bound on space = $O(1/\varepsilon \log (\varepsilon N))$
- To answer queries, need latest values of $L(v)$, so need time $O(1/\varepsilon \log \varepsilon N \log U)$ 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 εN 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?