Effective Computation of Biased Quantiles over Data Streams

Graham Cormode[*]
Rutgers University
graham@dimacs.rutgers.edu

Flip Korn
AT&T Labs–Research
flip@research.att.com

S. Muthukrishnan[†]
Rutgers University
muthu@cs.rutgers.edu

Divesh Srivastava
AT&T Labs–Research
divesh@research.att.com

Abstract

Skew is prevalent in many data sources such as IP traffic streams. To continually summarize the distribution of such data, a high-biased set of quantiles (e.g., 50th, 90th and 99th percentiles) with finer error guarantees at higher ranks (e.g., errors of 5, 1 and 0.1 percent, respectively) is more useful than uniformly distributed quantiles (e.g., 25th, 50th and 75th percentiles) with uniform error guarantees. In this paper, we address the following two problems. First, can we compute quantiles with finer error guarantees for the higher ranks of the data distribution effectively, using less space and computation time than computing all quantiles uniformly at the finest error? Second, if specific *quantiles and their error bounds are requested* a priori, *can the necessary space usage and computation time be reduced?*

We answer both questions in the affirmative by formalizing them as the "high-biased" quantiles and the "targeted" quantiles problems, respectively, and presenting algorithms with provable guarantees, that perform significantly better than previously known solutions for these problems. We implemented our algorithms in the Gigascope data stream management system, and evaluated alternate approaches for maintaining the relevant summary structures. Our experimental results on real and synthetic IP data streams complement our theoretical analyses, and highlight the importance of lightweight, non-blocking implementations when maintaining summary structures over high-speed data streams.

1 Introduction

Skew is prevalent in many data sources such as IP traffic streams. Distributions with skew typically have long tails which are of great interest. For example, in network management, it is important to understand what performance users experience. An important measure of performance perceived by the users is the round trip time (RTT) (which in turn affects dynamics of the network through mechanisms such as TCP flow control). RTTs display a large amount of skew: the tails of the distribution of round trip times can become very stretched. Hence, to gauge the performance

of the network in detail and the effect on all users (not just those experiencing the average performance), it is important to know not only the median RTT but also the 90%, 95% and 99% quantiles of TCP round trip times to each destination. In developing data stream management systems that interact with IP traffic data, there exists the facility for *posing* such queries [CKM+04]. However, the challenge is to develop approaches to answer such queries efficiently and accurately given that there may be many destinations to track. In such settings, the data rate is typically high and resources are limited in comparison to the amount of data that is observed. Hence it is often necessary to adopt the data stream methodology [BBD+02, GGR02, Mut03]: analyze IP packet headers in one pass over the data with storage space and per-packet processing time that is significantly sublinear in the size of the input.

Typically, IP traffic streams and other streams are summarized using quantiles: these are order statistics such as the minimum, maximum and median values. In a data set of size n, the ϕ-quantile is the item with rank $\lceil \phi n \rceil$[1]. The minimum and maximum are easy to calculate precisely in one pass but in general exact computation of certain quantiles requires space linear in n [MP80]. So the notion of ϵ-approximate quantiles relaxes the requirement to finding an item with rank between $(\phi - \epsilon)n$ and $(\phi + \epsilon)n$. Much attention has been given to the case of finding a set of *uniform quantiles*: given $0 < \phi < 1$, return the approximate $\phi, 2\phi, 3\phi, \ldots, \lfloor 1/\phi \rfloor \phi$ quantiles of a stream of values. Note that the error in the rank of each returned value is bounded by the same amount, ϵn; we call this the *uniform error* case.

Summarizing distributions which have high skew using uniform quantiles is not informative because having a uniformly spread-out summary of a stretched distribution does not describe the interesting tail region adequately. Motivated by this, we introduce the concept of *high-biased quantiles*: to find the $1 - \phi, 1 - \phi^2, 1 - \phi^3, \ldots, 1 - \phi^k$

---

[1]We use the rank of an item to refer to its position in the sorted order of items that have been observed.

quantiles of the distribution.[2] In order to give accurate and meaningful answers to these queries, we must also scale the approximation factor ϵ so the more biased the quantile, the more accurate the approximation should be. The approximate low-biased quantiles should now be in the range $(1 - (1 \pm \epsilon)\phi^j)n$: instead of additive error in the rank $\pm \epsilon n$, we now require *relative* error of factor $(1 \pm \epsilon)$.

Finding high- (or low-) biased quantiles can be seen as a special case of a more general problem of finding *targeted* quantiles. Rather than requesting the same ϵ for all quantiles (the uniform case) or ϵ scaled by ϕ (the biased case), one might specify in advance an *arbitrary* set of quantiles and their desired errors of ϵ for each in the form (ϕ_j, ϵ_j). For example, input to the targeted quantiles problem might be $\{(0.5, 0.1), (0.2, 0.05), (0.9, 0.01)\}$, meaning that the median should be returned with 10% error, the 20th percentile with 5% error, and the 90th percentile with 1%.

Both the biased and targeted quantiles problems could be solved trivially by running a uniform solution with $\epsilon = \min_j \epsilon_j$. But this is wasteful in resources since we do not need all of the quantiles with such fine accuracy. In other words, we would like solutions which are more efficient than this naive approach both in terms of memory used as well as in running time, thereby adopting to the precise quantile and error requirements of the problem.

Our contributions are as follows:

- We give the first-known deterministic algorithms for the problem of finding biased and targeted quantiles with a single pass over the input and prove that they are correct. These are simple to implement, yet give strong guarantees about the quality of their output.

- We consider the issues that arise when incorporating such algorithms into a high speed data stream management system (Gigascope), and develop a variety of implementations of them.

- We perform a set of experiments that show our algorithms are extremely space-efficient, and significantly outperform existing methods for finding quantiles when applied to our scenarios. We evaluate them on live and simulated IP traffic data streams and show that our methods are capable of processing high throughput network data.

2   Related Work

Computing concise summaries such as histograms and order statistics on large data sets is of great importance in particular for query planning. Histogram summaries such as equi-depth histograms and end-biased histograms have been studied extensively [Ioa03]. Note that our notion of high-biased quantiles is quite distinct from high-biased histograms [IC93] which find the most *frequent* items (i.e., the modes) of the distribution. There is a large body of work on ways to compute appropriate histograms and order-statistics in general; here, we describe prior work that computes quantiles with one pass over the input data since this is the focus of our work.

A lower bound of $\Omega(n)$ space was shown by Munro and Paterson [MP80] in order to exactly compute the median of n values. In the same paper, they showed an algorithm which finds any quantile in p passes and used memory $O(n^{1/p} \log n)$. Manku *et al* [MRL98] observed that after the first pass this algorithm finds bounds on the rank of items, and so it can be used to answer ϵ-approximate quantiles in space $O(\frac{1}{\epsilon} \log^2(\epsilon n))$. They also gave improvements with the same asymptotic space bounds but better constant factors. These algorithms require some knowledge of n in advance: they operate with $O(\log(\epsilon n))$ buffers of size $O(\frac{1}{\epsilon} \log(\epsilon n))$ each and so an upper bound on n must be known *a priori*.[3] Subsequent work removed this requirement. Manku *et al* [MRL99] gave a randomized algorithm, which fails with probability at most δ, in space $O(\frac{1}{\epsilon}(\log^2 \frac{1}{\epsilon} + \log^2 \log \frac{1}{\delta}))$.

Greenwald and Khanna made a significant contribution to computing quantiles in one pass by presenting a deterministic algorithm with space bounded by $O(\frac{1}{\epsilon} \log(\epsilon n))$ and no requirement that n be known in advance. This algorithm very carefully manages upper and lower bounds on ranks of items in its "sample" and intuitively is more informed in pruning items of non-interest than random-sampling based methods. As a result, the worst case space bound above is often pessimistic; on "random" data sets it has been observed to use $O(\frac{1}{\epsilon})$ space [GK01].

All these methods are designed to find uniform quantiles on insert-only data streams. When items can be deleted as well as inserted, Gilbert *et al* [GKMS02] gave a randomized algorithm to find quantiles in space $O(\frac{1}{\epsilon^2} \log^2 U \log \frac{\log U}{\delta})$, where U is the size of the universe, ie the number of possible values of items; this has since been improved to space $O(\frac{1}{\epsilon} \log^2 U \log \frac{\log U}{\delta})$ in [CM04]. In the sliding window model, where only the last W data elements are considered, there has been recent progress on finding quantiles [AM04] in space $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log W)$.

In this paper, our attention is on extensions of the basic quantile finding problem where some values are required to greater accuracy than others. In [MRL99], the authors study the "extreme values" quantile finding problem: for given values of ϕ close to zero (or, symmetrically, close to one), find an ϵ-approximation of the ϕ quantile using

---

[2]Symmetrically, the *low-biased quantiles* are the $\phi, \phi^2 \ldots \phi^k$ quantiles of the distribution.

[3]Although, if the estimate of n is too low, this will merely result in a decline in the quality of the results and a requirement for some extra buffers.

space significantly less than is required for maintaining ϵ-accuracy for the whole data set. The algorithm randomly samples at a rate $\frac{k}{n\phi}$ but only the k smallest elements of the sample are retained. The largest of these k items is returned as the approximate ϕ-quantile. Analysis sets k, the memory requirement, to be $O(\frac{\phi}{\epsilon}\log\frac{1}{\delta})$. In the case where $\epsilon = \epsilon'\phi$ (that is, where the "local" ϵ is simply the product of a global constant ϵ' and the value of ϕ), then this simplifies to $O(\frac{1}{\epsilon'}\log\frac{1}{\delta})$. The problem of finding biased quantiles was studied by Gupta and Zane [GZ03] in the context of approximating the number of inversions (disordered pairs of items) in a list. They presented a similar algorithm of retaining k smallest elements after sampling at an appropriate rate. This is then repeated for every quantile of the form $\frac{1}{\epsilon}(1 + \epsilon)^i$ up to n. This gives a total of $\frac{1}{\epsilon}\log\epsilon n$ parallel sampling routines. The $\frac{1}{\epsilon}$ smallest items can be stored exactly. Overall, the space requirement is $O(\frac{1}{\epsilon^2}\log\epsilon\log\frac{1}{\delta})$.[4]

There is significant opportunity for improvement here; for example, no information is shared between the different samplers which operate independently of each other. In this paper, we give a *deterministic* algorithm that also keeps "samples" from the input stream of values and gives ϵ-approximate biased or targeted quantiles. As in the result of Greenwald and Khanna [GK01], we also carefully maintain upper and lower bounds on ranks of items (to different levels of accuracy in different portions of the distribution which is in contrast to [GK01] where the levels are uniform over all the quantiles); as a result, we are able to make informed decisions on pruning items of non-interest. Interestingly, this improves the space needed greatly, and is also faster to process each new item.

3 Biased Quantiles Problem

We begin by formally defining the problem of biased quantiles. To simplify the notation, we present the material in terms of low-biased quantiles; high-biased quantiles can be obtained via symmetry, by reversing the ordering relation.

Definition 1 *Let a be a sequence of n items, and let A be the sorted version of a. Let ϕ be a parameter in the range $0 < \phi < 1$. The* low-biased quantiles *of a are the set of values $A[\lceil\phi^j n\rceil]$ for $j = 1, \ldots, \log_{1/\phi} n$.*

Sometimes we will not require the full set of biased-quantiles, and instead only search for the first k. Our algorithms will take k as a parameter.

It is well known that computing quantiles exactly requires space linear in n [MP80]. In our applications, we seek solutions that are significantly sublinear in n, preferably depending on $\log n$ or small polynomials in this quan-

---

tity. So we will allow *approximation* of the quantiles, by giving a small range of tolerance around the answer.

Definition 2 *The* approximate low-biased quantiles *of a sequence of n items, a, is a set of k items $q_1, \ldots, q_k$ which satisfy*

$$A[\lfloor(1-\epsilon)\phi^j n\rfloor] \leq q_j \leq A[\lceil(1+\epsilon)\phi^j n\rceil]$$

In fact, we shall solve a slightly more general problem: after processing the input then for any supplied value ϕ' we will be able to return an ϵ-approximate quantile q' that satisfies

$$A[\lfloor(1-\epsilon)\phi' n\rfloor] \leq q' \leq A[\lceil(1+\epsilon)\phi' n\rceil]$$

Any such solution clearly can be used to compute a set of approximate low-biased quantiles.

3.1 Algorithm for biased quantiles

Our algorithm draws inspiration from the algorithm proposed by Greenwald and Khanna [GK01], henceforth referred to as GK, for the uniform quantiles problem. The algorithm keeps information about particular items from the input, and also stores some additional tracking information. The intuition for this algorithm is as follows: suppose we have kept enough information so that the median can be estimated with an absolute error of ϵn in rank. Now suppose that there is a large number of insertions of items above the median, so that this item is pushed up to being the first quartile (the item which occurs $\frac{1}{4}$ through the sorted order). For this to happen, then the current number of items must be $2n$; hence if the same absolute uncertainty of ϵn is maintained, then this corresponds to a relative error of size $\frac{1}{2}\epsilon$. This shows that we will be able to support greater accuracy for the high-biased quantiles provided we manage the data structure correctly.

As in GK, the data structure at time n, $S(n)$, consists of a sequence of s tuples $\langle t_i = (v_i, g_i, \Delta_i)\rangle$, where each v_i is a sampled item from the data stream and two additional values are kept: (1) g_i is the difference between the lowest possible rank of item i and the lowest possible rank of item $i - 1$; and (2) Δ_i is the difference between the greatest possible rank of item i and the lowest possible rank of item i. The total space used is therefore $O(s)$. For each entry v_i, let $r_i = \sum_{j=1}^{i-1} g_j$. Hence, the true rank of v_i is bounded below by $r_i + g_i$ and above by $r_i + g_i + \Delta_i$.

Depending on the problem being solved (uniform, biased, or targeted quantiles), the algorithm will maintain an appropriate restriction on $g_i + \Delta_i$. We will denote this with a function $f(r_i, n)$, which for the current values of r_i and n gives an upper bound on the permitted value of $g_i + \Delta_i$. For biased quantiles, this invariant is:

Definition 3 (Biased quantiles invariant) *We set $f(r_i, n) = 2\epsilon r_i$. Consequently we ensure that $g_i + \Delta_i \leq 2\epsilon r_i$ for all i.*

---

[4]In [GZ03], the space bound is proportional to $\frac{1}{\epsilon^3}$, but using the sampler from [MRL99] improves this by a factor of $\frac{1}{\epsilon}$.

As each item is read, an entry is created in the data structure for it. Periodically, the data structure is "pruned" of unnecessary entries to limit its size. We ensure at all times that the invariant is maintained, which is necessary to show that the algorithm operates correctly. The operations are defined as follows:

Insert. To insert a new item, v, we find i such that $v_i < v \leq v_{i+1}$, we compute r_i and insert the tuple $(v, g = 1, \Delta = \lfloor f(r_i, n) \rfloor - 1)$. This gives the correct settings to g and Δ since the rank of v must be at least 1 more than the rank of v_i, and (assuming the invariant holds before the insertion), the uncertainty in the rank of v is at most one less than the uncertainty of v_i, Δ_i, which is itself bounded by $\lfloor f(r_i, n) \rfloor$ (since Δ_i is always an integer). We also ensure that min and max are kept exactly, so when $v < v_0$, we insert the tuple $(v, g = 1, \Delta = 0)$ before v_0. Similarly, when $v > v_{s-1}$, we insert $(v, g = 1, \Delta = 0)$ after v_{s-1}.

Compress. Periodically, the algorithm scans the data structure and merges adjacent nodes when this does not violate the invariant. That is, find nodes (v_i, g_i, Δ_i) and $(v_{i+1}, g_{i+1}, \Delta_{i+1})$, and replace them with $(v_{i+1}, (g_i + g_{i+1}), \Delta_{i+1})$ provided that $(g_i + g_{i+1} + \Delta_{i+1}) \leq f(r_i, n)$. This also maintains the semantics of g and Δ being the difference in rank between v_i and v_{i-1}, and the difference between the highest and lowest possible ranks of v_i, respectively.

Query. Given a value $0 \leq \phi \leq 1$, let i be the smallest index so that $r_i + g_i + \Delta_i > \phi n + \frac{1}{2} f(\phi n, n)$. Output v_{i-1} as the approximated quantile.

These three routines are the same for the different problems we consider, being parametrized by the setting of the invariant function f. It generalizes the GK algorithm, which is equivalent to the above routines with $f(r_i, n) = 2\epsilon n$. Figure 1 presents the pseudocode of the algorithm.

3.2 Correctness of the Algorithm

Theorem 1 *The algorithm correctly maintains ϵ-approximate biased quantiles.*

Proof: First, observe that Insert maintains the invariant since, for the inserted tuple, clearly $g + \Delta \leq \lfloor 2\epsilon r_i \rfloor$. All tuples below the inserted tuple are unaffected; for tuples above the inserted tuple, their $g_i + \Delta_i$ remains the same, but their r_i increases by 1, and so the invariant still holds. Compress checks that the invariant is not violated by its merge operations, and for tuples not merged, their r_i is unaffected, so the invariant must be preserved.

Next, we demonstrate that any algorithm which maintains the biased quantiles invariant means that the output function will correctly approximate biased quantiles. Because i is the smallest index so that $r_i + g_i + \Delta_i > \phi n + f(\phi n, n)/2 = \phi n + \epsilon \phi n$, then $r_{i-1} + g_{i-1} + \Delta_{i-1} \leq (1+\epsilon)\phi n$. Using the invariant, then $(1+2\epsilon)r_i > (1+\epsilon)\phi n$

and consequently $r_i > (1 - \epsilon)\phi n$. Hence $(1 - \epsilon)\phi n < r_{i-1} + g_{i-1} \leq r_{i-1} + g_{i-1} + \Delta_{i-1} \leq (1 + \epsilon)\phi n$. Recall that that the true rank of v_i is between $r_i + g_i$ and $r_i + g_i + \Delta_i$: so the derived inequality means that v_{i-1} is within the necessary error bounds for biased quantiles. ∎

This gives an error bound of $\pm \epsilon \phi n$ for every value of ϕ. In some cases we have a lower bound on how precisely we need to know the biased quantiles: this is when we only require the first k biased quantiles. It corresponds to a lower bound on the allowed error of $\epsilon \phi^k n$. Clearly we could use the above algorithm which gives stronger error bounds for some items, but this may be inefficient in terms of space. Instead, we modify the invariant as follows to avoid this slackness and so reduce the space needed. The algorithm is identical to before but we modify the invariant to be $f(r_i, n) = 2\epsilon \max\{r_i, \phi^k n\}$. This invariant is preserved by Insert and Compress. The Output function can be proved to correctly compute biased quantiles with this lower bound on the approximation error using straightforward modification of the above proof.

```
/* n = #items, k = asymptote */
/* S = data structure, s = #samples */
Insert(v):
01   r_0 := 0;
02   for i:= 1 to s do
03        if (v_i < v) break;
04        r_i := r_{i-1} + g_i;
05   add (v, 1, ⌊f(r_i, n)⌋ − 1) to S before v_i;
06   n++;


Compress():
01   for i := (s − 1) downto 1 do
02        if (g_i + g_{i+1} + Δ_{i+1} ≤ f(r_i, n)) then
03             merge t_i and t_{i+1};


Query(φ):
01   r_i := 0;
02   for i := 1 to s do
03        r_i := r_{i-1} + g_{i-1};
04        if (r_i + g_i + Δ_i > φn + f(φn, n)/2)
05             print(v_{i-1}); break;


Main():
01   for each item v do
02        Insert(v);
03        if (Compress_Condition()) then
04             Compress();
```

Figure 1: Approximate Quantiles Algorithm

3.3 Time and Space Bounds

The running time of the algorithm to process each new update v depends on (i) the data structures used to implement the sorted list of tuples, S, and (ii) the frequency with which Compress is run. This is governed by the function Compress_Condition(), which can be implemented in a variety of ways: it could always return true, or return true every $1/\epsilon$ tuples, or with some other frequency. Note that the frequency of compressing does not affect the correctness, just the aggressiveness with which we prune the data structure. The time for each Insert operation is that to find the position of the new data item v in the sorted list. With a sensible implementation (e.g., a balanced tree structure), this is $O(\log s)$. With such a data structure we can efficiently maintain r_i of each tuple in the same time bounds. Compress is much simpler to implement since it requires just a linear pass over the sorted elements in time $O(s)$ but we must decide when to run Compress to balance the time cost and the space used by the algorithm. We return to these issues later when we consider different implementation strategies in Section 5.

In [GK01] it is shown that the GK algorithm requires space $O(\frac{1}{\epsilon} \log \epsilon n)$ in the worst case. By analogy, the worst case space requirement for finding biased quantiles is expected to be $O(\frac{k \log 1/\phi}{\epsilon} \log \epsilon n)$. Consider the space used by the algorithm to maintain the biased quantiles for the values whose rank is between $n/2$ and n. Here we maintain a synopsis where the error is bounded below by ϵn and the algorithm operates in a similar fashion to the GK algorithm. So the space required to maintain this region of ranks should be bounded by $O(\frac{1}{\epsilon} \log \epsilon n)$. Similarly for the range of ranks $n/4$ to $n/2$, items are maintained to an error no less than $\epsilon/2$ but we are maintaining a range of at most half as many *ranks*. Thus the space for this should be bounded by the same amount $O(\frac{1}{\epsilon} \log \epsilon n)$. This argument can be repeated until we reach $n/2^x = \phi^k n$ where the same amount of space suffices to maintain information about ranks up to ϕ^k with error $\epsilon \phi^k$. The total amount of space is no more than $O(\frac{x}{\epsilon} \log \epsilon n) = O(\frac{k \log 1/\phi}{\epsilon} \log \epsilon n)$. If ϕ is not specified *a priori*, then this bound can be easily rewritten in terms of k and ϵ. Also, we never need $k \log 1/\phi$ to be greater than $\log \epsilon n$, which corresponds to an absolute error of less than 1, so the bound is equivalent to $O(\frac{1}{\epsilon} \log^2 \epsilon n)$.

We also note the following lower bound for *any* method that finds the biased quantiles.

Theorem 2 *Any algorithm that guarantees to find biased quantiles ϕ with error at most $\phi \epsilon n$ in rank must store $\Omega(\frac{1}{\epsilon} \min\{k \log 1/\phi, \log(\epsilon n)\})$ items.*

Proof: We show that if we query all possible values of ϕ, there must be at least this many different answers produced. Assume without loss of generality that every item in the input stream is distinct. Consider each item stored by the algorithm. Let the true rank of this item be R. This is a good approximate answer for items whose rank is between $R/(1+\epsilon)$ and $R/(1-\epsilon)$. The largest stored item must cover the greatest item from the input, which has rank n, meaning that the lowest rank input item covered by the same stored item has rank no lower than $n(1-\epsilon)/(1+\epsilon)$. We can iterate this argument, to show that the lth largest stored item covers input items no less than $n(1-\epsilon)/(1+\epsilon)^l$. This continues until we reach an input item of rank at most $m = n\phi^k$. Below this point, we need only guarantee an error of $\epsilon \phi^k$. By the same covering argument, this requires at least $p = (n\phi^k)/(\epsilon n\phi^k) = 1/\epsilon$ items. Thus we can bound the space for this algorithm as $p + l$, when $n(1-\epsilon)/(1+\epsilon)^l \leq m$. Then, since $\frac{1-\epsilon}{1+\epsilon} \leq (1-\epsilon)$, we have $\ln(m/n) \geq l \ln(1-\epsilon)$. Since $\ln(1-\epsilon) \leq -\epsilon$, we find $l \geq \frac{1}{\epsilon} \ln \frac{n}{m} = \frac{1}{\epsilon} \ln \frac{n}{n\phi^k}$. This bounds $l = \Omega(\frac{k \log 1/\phi}{\epsilon})$, and gives the stated space bounds.

Note that it is not meaningful to set k to be too large, since then the error in rank becomes less than 1, which corresponds to knowing the exact rank of the smallest items. That is, we never need to have $\epsilon n\phi^k < 1$; this bounds $k \log 1/\phi \leq \log(\epsilon n)$ and so the space lower bounds translates to $\Omega(\frac{1}{\epsilon} \min\{k \log 1/\phi, \log(\epsilon n)\})$. ∎

4 Targeted Quantiles Problem

The targeted quantiles problem considers the case that we are only concerned with a set of quantile values with associated error bounds that are supplied *a priori*. Formally, the problem is as follows:

Definition 4 (Targeted Quantiles Problem) *The input is a set of tuples $T = \{(\phi_j, \epsilon_j)\}$. Following a stream of input values, the goal is to return a set of $|T|$ values v_j such that*

$$A[\lceil(\phi_j - \epsilon_j)n\rceil] \leq v_j \leq A[\lceil(\phi_j + \epsilon_j)n\rceil].$$

As in the biased quantiles case, we will maintain a set of items drawn from the input as a data structure, $S(n)$. We will keep tuples $\langle t_i = (v_i, g_i, \Delta_i)\rangle$ as before, but will keep a different constraint on the values of g_i and Δ_i.

Definition 5 (Targeted Quantiles Invariant) *We define the invariant function $f(r_i, n)$ as*

$$(i) \quad f_j(r_i, n) = \frac{2\epsilon_j r_i}{\phi_j}, \qquad \phi_j n \leq r_i \leq n;$$

$$(ii) \quad f_j(r_i, n) = \frac{2\epsilon_j(n - r_i)}{(1 - \phi_j)}, \quad 0 \leq r_i \leq \phi_j n$$

and take $f(r_i) = \min_j f_j(r_i)$. As before we ensure that for all i, $g_i + \Delta_i \leq f(r_i, n)$.

An example invariant f is shown in Figure 2 where we plot $f(\phi n, n)$ as ϕ varies from 0 to 1. Dotted lines indicate
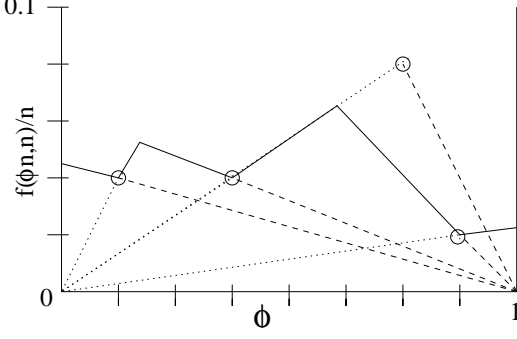
Figure 2: Error function $f(\phi n, n)/n$ computed for the input $\{(\frac{1}{8}, 0.02), (\frac{3}{8}, 0.02), (\frac{6}{8}, 0.04), (\frac{7}{8}, 0.01)\}$

the constraints of type (i) when $r_i \leq \phi_j n$ and constraints of type (ii) when $r_i \geq \phi_j n$, to illustrate how the function is formed. The function f itself is illustrated with a solid line seen as the lower envelope of the f_j's. Note that if we allow T to contain a large number of entries then setting

$$T = \{(\tfrac{1}{n}, \epsilon), (\tfrac{2}{n}, \epsilon), \ldots, (\tfrac{n-1}{n}, \epsilon), (1, \epsilon)\}$$

captures the uniform error approximate quantiles problem solved by GK. Similarly setting

$$T = \{(\tfrac{1}{n}, \tfrac{\epsilon}{n}), (\tfrac{2}{n}, \tfrac{2\epsilon}{n}) \ldots (\tfrac{n-1}{n}, \tfrac{(n-1)\epsilon}{n}), (1, \epsilon)\}$$

captures the biased quantiles problem. In both cases, the invariant function f computed by Definition 5 reduces to the function used by GK and in Section 3 above, respectively.

4.1 Algorithm and Correctness

`Insert`, `Compress` and `Query` operations are the same as in Figure 1 with the new invariant. Therefore, the running time is the same as before plus the time to compute the invariant f which takes time at most $O(\log |T|)$ per invocation. We will show that the operations maintain the invariant and that if the invariant holds then targeted quantiles are estimated to the required degree of accuracy.

Lemma 1 *The targeted quantiles invariant is preserved by* `Insert` *and* `Compress` *operations.*

Proof: We will show that assuming the invariant holds for each v_i before an insertion, it must hold afterwards. For the tuple that is inserted we the invariant trivially holds by the setting of the values in the inserted tuple. However, since the invariant depends on r_i and n, it is possible that the constraints on other tuples will change and we must ensure that these do not violate the invariant. Note that if the new value v is inserted before v_i then r_i increases by 1. Otherwise r_i stays the same and n increases by 1 in all cases. For each tuple v_i, consider the error function f

and the f_j that is tightest for the current value of v_i. First, suppose that before and after the insertion then the same f_j is tight. There are two cases:
(i) The invariant is $\frac{2\epsilon_j r_i}{\phi_j}$. Then whether r_i increases or stays the same, the invariant does not get tighter for v_i, and so the inequality is preserved.
(ii) The invariant is $\frac{2\epsilon_j(n-r_i)}{(1-\phi_j)}$. Then as n increases by one, and r_i either stays the same or increases, again this invariant does not get tighter, so the inequality is preserved.

Now suppose that before the insertion v_i was subject to some f_j and afterward is subject to some f_k. There are many cases to consider depending on the type of constraint of f_j and $f_{j'}$. For example, suppose that v_i was initially constrained by $\frac{2\epsilon_j r_i}{\phi_j}$ and is then constrained by $\frac{2\epsilon_k((n+1)-(r_i+1))}{1-\phi_k}$, with $j < k$. Then observe that there must be some values of $r' \geq r_i$ so that $\frac{2\epsilon_j r'}{\phi_j} = \frac{2\epsilon_k((n+1)-(r'+1))}{1-\phi_k}$, since the constraints are linear. Writing $\chi = \frac{\epsilon_k \phi_j}{\epsilon_j(1-\phi_k)}$, then the constraint is violated after the insertion if $\frac{r}{n-r_i} > \chi = \frac{r'}{n-r'}$. That is, if $r_i(n-r') > r'(n-r_i) \Rightarrow r_i > r'$. But this contradicts our assertion that $r' \geq r_i$ and so the constraint cannot be violated.

There are several similar cases to work through; we omit full details here. Finally note that compress operations trivially preserve the invariant since we only delete tuples when this does not violate the constraint; other tuples are unaffected by deletions since r_i and n are unchanged. ∎

Theorem 3 *If the biased quantiles invariant holds, then running* `Output` *for all ϕ_j will find the specified quantiles ϕ_j within the given error bounds ϵ_j.*

Proof: Consider the quantile and error pair (ϕ_j, ϵ_j). The `Query` function finds the smallest index i such that $r_i + g_i + \Delta_i > \phi_j n + \frac{1}{2} f(\phi_j n, n) \geq \phi_j n + \frac{\epsilon_j \phi_j n}{\phi_j} = (\phi_j + \epsilon_j)n$ Then $r_{i-1} + g_{i-1}\Delta_{i-1} \leq (\phi_j + \epsilon_j)n$ and $r_i + (g_i + \Delta_i) > (\phi_j + \epsilon_j)n$. Now, consider how $g_i + \Delta_i$ is bound by f_j: we know from Definition 5 that either case (i) or case (ii) holds. If case (i) holds, then we are done, since we have $\phi_j n \leq r_i \leq (\phi_j + \epsilon_j)n$. So suppose case (ii) holds. Then,

$$
\begin{aligned}
r_i + \tfrac{2\epsilon_j(n-r_i)}{1-\phi_j} &> (\phi_j + \epsilon)n \\
(1-\phi_j)r_i + 2\epsilon_j n - 2\epsilon r_i &> (\phi_j + \epsilon_j - \phi_j^2 - \epsilon_j\phi_j)n \\
(1-\phi_j)r_i - 2\epsilon_j r_i &> (1-\phi_j)(\phi_j - \epsilon_j)n \\
r_i &> (\phi_j - \epsilon_j)n
\end{aligned}
$$

In both cases, we can bound $(\phi_j - \epsilon_j)n < r_{i-1} + g_{i-1} \leq r_{i-1} + g_{i-1} + \Delta_{i-1} \leq \phi_j n$. So we know that the true rank of item v_{i-1} is in the range $(\phi_j \pm \epsilon_j)n$. ∎

Informally, we argue that the space used by this algorithm is bounded if we set $\epsilon' = \min_j \epsilon_j$ as $O(\frac{1}{\epsilon'} \log \epsilon' n)$, by applying the space bound argument of the Greenwald and Khanna argument, and by observing that our algorithm can prune more aggressively than the GK algorithm. In

practice, we would expect to see much tighter space bounds, as even small ϵ_js can be achieved in smaller space if the corresponding ϕ_js are sufficiently far from $\frac{1}{2}$. We expect to see a dependency on the greatest value of $\frac{\phi_j}{\epsilon_j}$ by analogy to the biased quantiles case.

Note that the algorithm is slightly more general than was claimed. In addition to the targeted quantiles, information about the whole distribution is kept. Given an arbitrary value of $0 \leq \phi \leq 1$, the algorithm will find a value whose rank is between $(\phi n - f(\phi n, n)/2)$ and $(\phi n + f(\phi n, n)/2)$. For example, in Figure 2 which plots f against ϵ as ϕ increases, we see that f is never more than $0.07n$. Here we have omitted formally discussing and proving such general claims, for brevity.

5 Implementation Issues

As described, the algorithms presented in Sections 3 and 4 allow for much freedom in implementing them. In this section, we present a few alternatives used to gain an understanding of which factors are important for achieving good performance over a data stream. The three alternatives we considered are natural choices and exhibit standard data structure trade-offs, but our list is by no means exhaustive.

5.1 Abstract Data Types

Recall that the (biased) quantile summary $S(n)$ is a sequence of tuples $\langle t_i = (v_i, g_i, \Delta_i) \rangle$. Operations on the quantile summary, such as Compress and Output, require access to adjacent tuples with respect to v_i-values. Hence, to give efficient access to the tuples we will maintain them in sorted order on v_i-values.

The periodic reduction in size of the quantile summary done by Compress is based on the invariant function f which determines tuples eligible for deletion (that is, merging the tuple into its adjacent tuple). Note that this objective function is rank-dependent: it depends not only on g_i and Δ_i, but also on r_i and n. Hence, it is not possible to efficiently maintain candidates for compression incrementally due to dynamically changing ranks. Instead, all of our implementations sequentially scan through (a portion of) the quantile summary and test tuples t_i to see if they can be safely merged. Instead of periodically running a full Compress, we amortize this work by scanning a fraction of the data structure for every insertion: we process $s/2\epsilon$ tuples per item. The goal is to lower the worst-case processing at any time step and thus keep up with the data stream.

5.2 Methods

We now describe three alternatives for maintaining the quantile summary tuples ordered on v_i-values in the presence of insertions and deletions:

- **Batch:** This method maintains the tuples of $S(n)$ in a linked list. Incoming items are buffered into blocks of size $1/2\epsilon$, sorted, and then batch-merged into $S(n)$. Insertions and deletions can be performed in constant time, but the periodic buffer sort, occurring every $1/2\epsilon$ items, costs $O((1/\epsilon) \log(1/\epsilon))$.

- **Cursor:** This method also maintains the tuples of $S(n)$ in a linked list. Incoming items are buffered in sorted order and are inserted with the aid of an insertion cursor which, like the compress cursor, sequentially scans a fraction of the tuples and inserts a buffered item whenever the cursor is at the appropriate position. Maintaining the buffer in sorted order costs $O(\log(1/\epsilon))$ per item.

- **Tree:** This method maintains $S(n)$ using a balanced binary tree. Hence, insertions and deletions cost $O(\log s)$. In the worst case, all ϵs tuples considered for compression can be deleted, so the cost per item is $O(\epsilon s \log s)$.

These methods were implemented in C++ and attempts were made to make the three implementations as uniform as possible for a fair comparison (for example, all methods use approximately the same amount of space). The C++ STL `list` container type was used for storing $S(n)$ in both Batch and Cursor whereas a `multiset` container was used for Tree.[5] Cursor uses the priority queue from `<queue>` to maintain the buffer of incoming items in sorted order. As a disclaimer, there were many optimizations we did not employ which would likely improve the performance of all the methods such as parallelism, pre-allocated memory, cache-locality, etc.

6 Experiments

In the first part of this section, we evaluate the accuracy/space trade-off for both the biased quantiles and targeted quantiles problems, in comparison to naively applying the GK algorithm [GK01]. In accordance with [GK01], the algorithms used here differ from that described in Section 3 in two ways: a new observation v is inserted as a tuple $(v, 1, g_i + \Delta_i - 1)$, where $v_{i-1} < v \leq v_i$, and Compress is run after every insertion into $S(n)$, to delete one tuple when possible. When no tuple could be deleted without violating the error constraint, the size of $S(n)$ grows by one. Space is measured by the number of tuples.

For biased quantiles, we consider the following two questions. First, with error requirements that are non-uniform over the ranks, can we achieve such accuracy in less space than pessimistically requiring all the quantiles at the finest error? We compare our proposed algorithm for finding the first k biased quantiles against GK run with error $\epsilon \phi^k$. Second, how does space depend on the trail-off parameter k?

---

[5] Our implementation of STL uses red-black trees for `<set>`.

For targeted quantiles, we consider the following two questions. First, if we know the desired quantiles and their errors *a priori*, then can we focus the algorithm to yield the required accuracy at only those quantiles to save space? We illustrate using the case when a single order statistic (e.g., $\phi = 0.5$, aka the median) is desired within error ϵ. Whereas GK allows all quantiles to be given at this accuracy, our approach only provides this guarantee for a specified ϕ-quantile and gives weaker guarantees for other values. Second, how does the space usage of our algorithm depend on the value of ϕ? It has been noted in [MRL99] that, if the desired quantile is an extreme value (e.g., within the top 1% of the elements), then the space requirements of existing algorithms are overly pessimistic. The authors showed that, when simply taking quantiles over a random sample, probabilistic guarantees can be obtained in less space for extreme values than for the median. Our algorithm exhibits this same phenomenon. Furthermore, we show that, for *any* quantile, if the desired error bounds are known in advance, existing algorithms are also overly pessimistic.

In the second part of this section, we evaluate and compare the performance of the different implementation alternatives described in Section 5 using the Gigascope data stream system [CJSS03]. These alternatives vary in terms of the different aspects of the algorithm they optimize, in terms of their simplicity, and with respect to blocking behavior. The goal is to shed some light on which factors are most important for performance.

6.1 Space Usage for Biased Quantiles

For these experiments, we compared the space usage of our proposed biased quantile algorithm with that of GK. In order to obtain $p\epsilon$ error at quantiles $p \in \{\phi, \phi^2, \ldots, \phi^k\}$, the GK algorithm must be run at the finest level of error, yielding $\epsilon\phi^k$-approximate quantiles. We set $\phi = 0.5$ and tried different parameter values for k and ϵ. We used a variety of different data streams: "hard", sorted, and "random" (the inputs used in [GK01]).[6]

Figure 3 reports space usage for different values of k and ϵ on the "hard" input. Clearly, the proposed method uses much less space with the gap increasing both with k and inversely with ϵ. At time step $n = 10^5$ with $\epsilon = 0.001$, the ratio is approximately 4 with $k = 4$ and 19.5 with $k = 6$. Figure 4 gives similar graphs for random input. Here we observed similar trends at different values of ϵ so we only present the graphs at $\epsilon = 0.001$. At time step $n = 10^5$, the ratio is approximately 4.4 with $k = 4$ and 11.8 with $k = 6$. If the space for GK is bounded by $O(\frac{1}{\epsilon\phi^k} \log \epsilon\phi^k n)$, and our algorithm for biased quantiles by $O(\frac{k \log \phi}{\epsilon} \log \epsilon n)$,

---

[6]The "hard" input is created by examining the current state of the data structure and inserting items in order to try to force the worst-case performance.

then for $\phi = \frac{1}{2}$ the ratio of their space usage should be roughly $2^k/k$. For random input we in fact see values that are similar to these: for $k = 4$ the theoretical ratio is 4 and for $k = 6$ it is 10.7; for the "hard" input, the ratios were even higher. Figure 5 plots space as a function of k, indicating an exponential dependence on k for GK and a linear dependence on k for the proposed algorithm, as predicted by the $O(\frac{2^k}{\epsilon} \log \epsilon n)$ and $O(\frac{k}{\epsilon} \log \epsilon n)$ bounds.

Figure 6(a) illustrates the space used by three competing methods: GK run at error ϵ (denoted "GK1"), GK run at error $\epsilon\phi^k$ (denoted "GK2"), and our proposed method. It uses the random input, with $\epsilon = 0.01$ and $k = 6$, and the results are given at time step $n = 10^6$. Figure 6(b) plots the bound on error as a function of p, that are required for biased quantiles. Note that while GK1 uses the least space, it does not satisfy the error bound. GK2, on the other hand, is overly pessimistic, achieving the smallest error at all p-values but requiring much more space than the other methods. The proposed method achieves the least amount of space while staying within the error bounds. In fact, its space usage is much closer to that of the algorithm with the weaker accuracy, GK1 (factor of 4 more) than that of GK2 (factor of 16.5 less).

6.2 Space Usage for Targeted Quantiles

Our targeted quantiles algorithm can find the $\lfloor \phi n \rfloor$th order statistic with a maximum error of ϵ; its precision guarantees are weaker for other ranks. We compared against GK, which is capable of finding *any* quantile within ϵ error. We also considered the random sampling approach analyzed in [MRL99], but this approach was unable to obtain reliable estimates for any of the data sets. Given the space used by our proposed algorithm, we considered the probabilistic accuracy guarantees that could be given by the sampling algorithm. For the random input, the bounds gave guarantees that held with 70% probability to find quantiles that our algorithm found with absolute certainty. For the "hard" input, which attempts to force the worst-case space usage, the probability for the randomized algorithm improved to around 95%, still far short of the low failure rates demanded by network managers. Hence, we do not report further on the results obtained by random sampling for the remainder of this section.

Figure 7 presents space usage as a function of time step, with a variety of ϕ-values from 0.5 to 0.99, for (a) hard and (b) random inputs; $\epsilon = 0.001$. The gap in space usage between the two methods grows with increasing ϕ-value, which is consistent with the observation in [MRL99] that extreme values require less space.

6.3 Performance Comparison

We compared the three implementation alternatives described in Section 5 (namely, Batch, Cursor and Tree) with respect
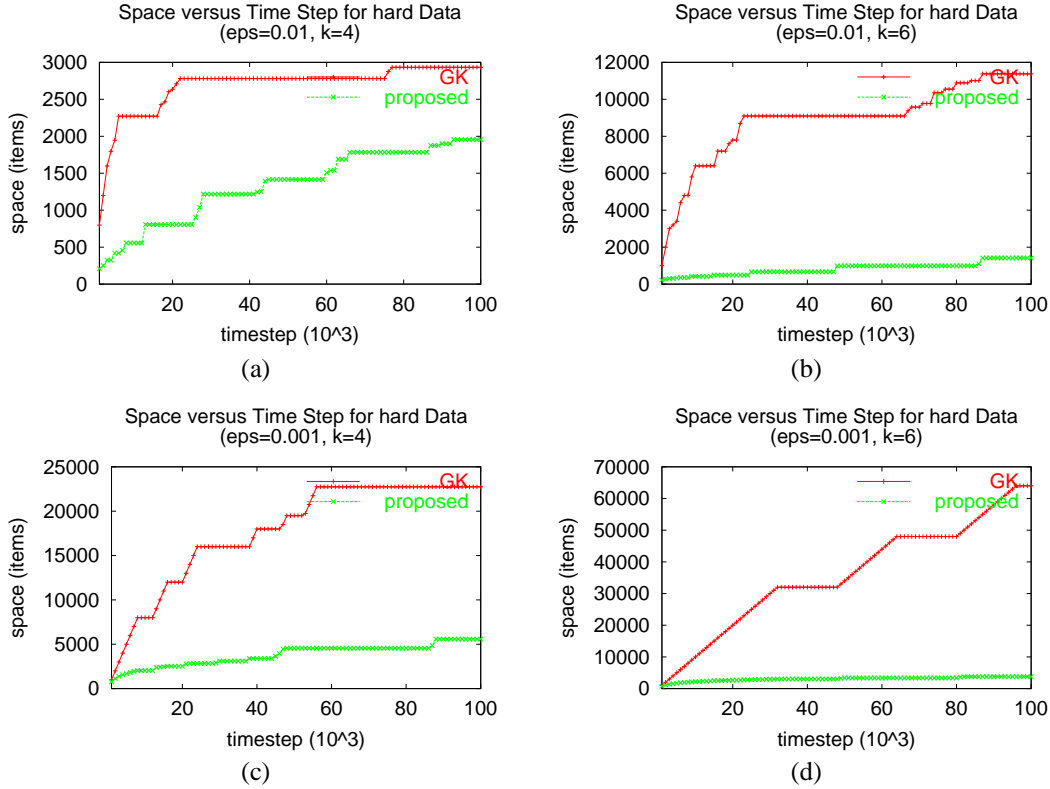
Figure 3: Comparison of GK and proposed approach on "hard" input: (a) with $k = 4$; (b) with $k = 6$. Here $\epsilon = 0.01$. (c) with $k = 4$; (d) with $k = 6$. Here $\epsilon = 0.001$.

to per-packet processing time and packet loss using the User-Defined Aggregation Function (UDAF) facility of the Gigascope DSMS, a highly optimized system for monitoring very high speed data streams [CJSS03]. Gigascope has a two-level query architecture: at the low level, data is taken from the Network Interface Card (NIC) and is placed in a ring buffer; queries at the high level then run over the data from the ring buffer. Gigascope *creates* queries from an SQL-like language (called GSQL) by generating C and C++ code, which is compiled and linked into executable queries. To integrate a UDAF into Gigascope, the UDAF functions are added to the Gigascope library and query generation is augmented to properly handle references to UDAFs; for more details, see [CKM+04].

For performance testing, we used two data sources. The first data source is an Agilent Technologies RouterTester 5.0 traffic generator [Tec]. Using it, one can generate 1 Gbps of traffic (GigEth speed). The traffic generator is not a sophisticated source of randomness; we could only vary the packet length and payload, both independently and uniformly random. The average packet length is always 782 bytes, which is equivalent to about 160,000 packets per second at GigEth speed. Queries were run over the

generated stream using a 2.8 Ghz Pentium processor and 4 GBs of RAM. The second data source is real IP traffic data obtained by monitoring the span port[7] of the router which connects AT&T Research Labs to the Internet via a 100 Mbit/sec link. Queries were run over this stream using a 733 Mhz Pentium with 128 Mbytes of RAM.

Biased quantile queries were run over a single attribute from these data sources and output at 1-minute intervals over a total duration of 30 minutes; the parameter ϵ was set to 0.01 and k was set to 4, unless indicated otherwise below. As a baseline, we also compared against the performance of a "null" UDAF which computes the max aggregate, to isolate out the processing overhead for UDAFs.

Table 1 reports the results from using the traffic generator at OC-3 speed (155.5 Mbps). The algorithms were run over the packet_length field of IPv4 packet headers (which were randomly generated). All methods were able to keep up with this rate without incurring packet loss, but were taxed at different levels. The Batch and Cursor methods operated at ten times slower than the "null" UDAF, with Cursor showing slightly better performance. The Tree

---

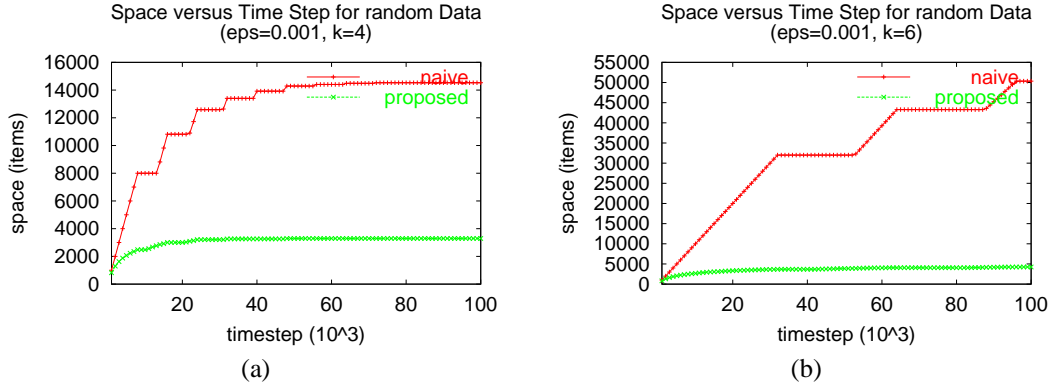[7]A span port mirrors all traffic for monitoring purposes.

Figure 4: Comparison of GK and proposed approach on random input: (a) with $k = 4$; (b) with $k = 6$. Here $\epsilon = 0.001$.
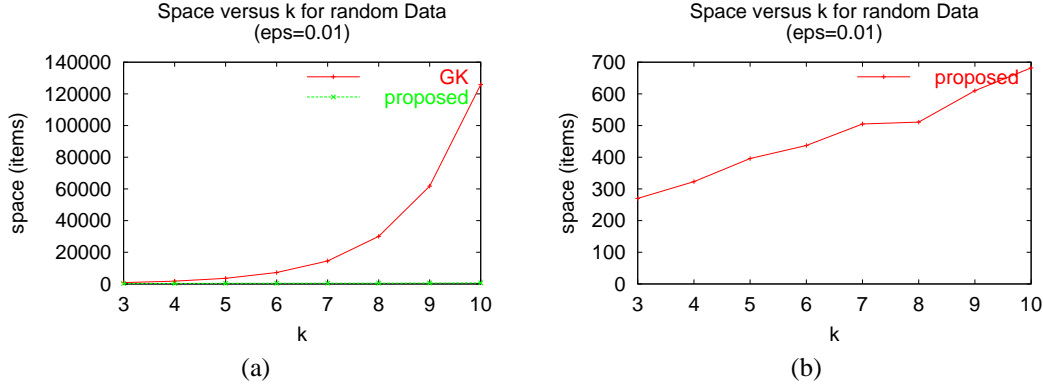

Figure 5: Space usage (at time step $n = 10^6$) versus k on random input with $\epsilon = 0.01$ for (a) GK and proposed algorithms; and (b) just the proposed algorithm.

method was yet four times slower than these, and had a very high CPU utilization.

Algorithm Implementation	CPU utilization	user time (μs) per packet
null	0.05%	0.481
Batch	11.99%	5.302
Cursor	11.88%	5.093
Tree	67.61%	21.969

Table 1: Per-packet processing time (μs) for the different implementations, over traffic generated at OC-3 speed.

At GigEth speed (1 Gbps), the Tree method has reached its limit and incurs so much packet loss that no useful statistics could be reported (see Table 2). Batch incurs more traffic loss than Cursor due to the periodic batch-sorting and merge that is required after every $1/2\epsilon$ items. Presumably, the lower average CPU time for Batch compared to Cur-

sor is due to not processing the packets that get dropped. To get a frame of reference, we also compared against uniform quantiles based on the GK algorithm run with errors ϵ and $\epsilon\phi^k$ (denoted "GK1" and "GK2", respectively). GK2 dropped so many packets that we could not compute a meaningful statistic. Note that GK1 does not achieve the desired error bound; it is presented merely as a baseline. To use the GK algorithm properly would requre the finer error bound of GK2.

Table 3 reports the results on the real IP network data and summarized by the average CPU utilization and user time (in microseconds) per packet; we were unable to measure packet loss. The algorithms were run over the header_checksum field of the packet headers. Although the overall traffic load, averaging 50-75 Mbps, was much less than that of the traffic generator, it is very bursty.

In summary, the choice of UDAF implementation is crucial to the performance of the quantile algorithm, confirming observations in [CKM+04]. Whereas the Batch and
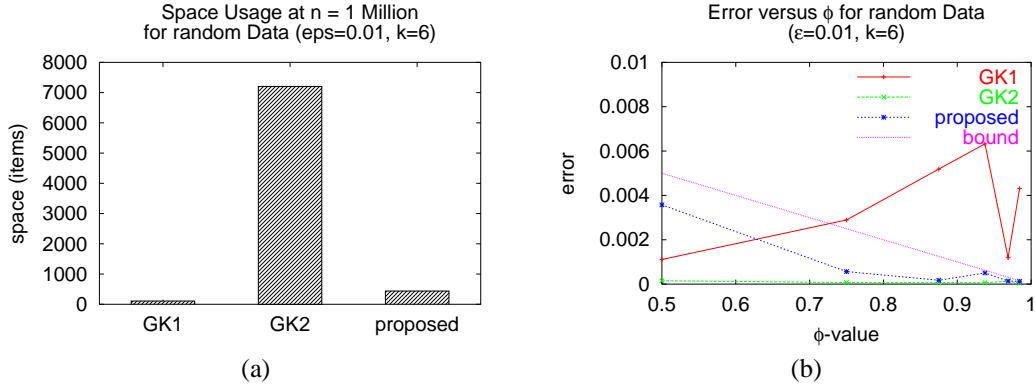
Figure 6: Comparison of GK, run with error ϵ ("GK1") and $\epsilon 2^{-k}$ ("GK2"), against the proposed approach on random input with $\epsilon = 0.01$ and $k = 6$ at time step $n = 10^6$: (a) space usage; and (b) *a posteriori* error of biased quantiles.
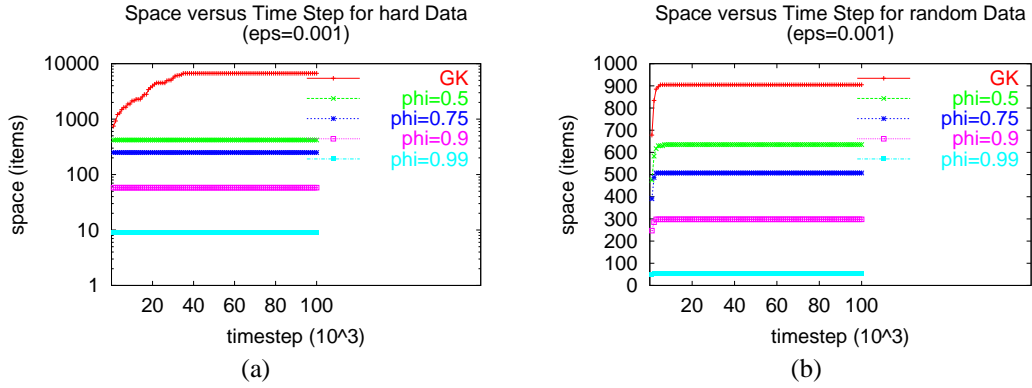

Figure 7: Comparison of GK and targeted approach with different ϕ-values: (a) for "hard" input; and (b) for random input. Here $\epsilon = 0.001$.

Algorithm Implementation	CPU utilization	user time (μs) per packet	packet loss
null	6%	0.5	0%
Batch	75%	5.522	1.82%
Cursor	81%	5.613	0.26%
Tree	—	—	—
GK1	18%	1.32	0%
GK2	—	—	—

Table 2: Per-packet processing time (μs) for the different implementations, over traffic generated at GigEth speed.

Algorithm Implementation	CPU utilization	user time (μs) per packet
null	3%	1.167
Batch	27%	11.514
Cursor	26%	11.164
Tree	33%	14.059

Table 3: Per-packet processing time (μs) for the different implementations, over real IP network traffic data.

Cursor approaches were able to process at GigEth speed, the Tree approach was not able to keep up, and even pushes its limit at OC-3 speed. Although keeping the quantile summary in a tree is good for maintaining sort order, it incurs a lot of overhead during `Compress` operations. Hence, approaches with the more lightweight list-based quantile summaries perform better. Batch is the simplest of these, but the blocking due to sorting results in more packet loss compared to Cursor. Therefore, Cursor seems to strike the right balance between simplicity and non-blocking behavior.

7 Conclusions and Future Work

We introduced the notion of biased and targeted quantiles and presented one-pass deterministic algorithms that approximate these values within user-specified accuracy. Our experimental work has shown that these algorithms are extremely effective in practice: the space needed is very small and is smaller than that needed by existing algorithms that give the same guarantees. We have shown they can be implemented within a database management system that deals processes high speed data streams resulting from IP network traffic. We also observed that in these high speed scenarios, implementation details can make significant differences in practicality and amortizing computation cost to avoid blocking but staying lightweight is vitally important.

We briefly discuss the feasibility of various extensions. Previous work has extended the work on finding ϵ-approximate quantiles (uniform error) to the sliding window model [AM04]. We claim that similar techniques based on keeping summaries for previously seen subsequences of items of various lengths can be applied to our algorithms. Other work has studied the problem of approximating quantiles when items can depart as well as arrive [GKMS02]. In this model, we claim that no algorithm can guarantee to find all biased-quantiles without keeping $\Omega(n)$ items. This is because the problem insists that we must be able to recover the minimum or maximum value exactly. If deletions are allowed, after processing n insertions we could repeatedly request and delete the minimum or maximum value, thus recovering the whole set of inserted values. Likewise, solving the k biased quantiles problems requires $\Omega(\epsilon/2^k)$ space, by a similar argument. Meanwhile, it remains open to formally characterize the space usage of the algorithms we have described for biased and targeted quantiles to the tightest possible estimate.

More generally, our work was motivated by developing appropriate statistics for summarizing skewed data. Data skew is highly prevalent in many applications. We believe that it is of interest to study further problems that do not treat all input uniformly, but rather require non-uniform guarantees dependent on the skew of the data.

Acknowledgements

References

[AM04]   A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the Twenty-Third ACM Symposium on Principles of Database Systems*, pages 286–296, 2004.

[BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of Symposium on Principles of Database Systems*, pages 1–16, 2002.

[CJSS03] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of ACM SIGMOD*, pages 647–651, 2003.

[CKM+04] G. Cormode, F. Korn, S. Muthukrishnan, T. Johnson, O. Spatscheck, and D. Srivastava. Holistic udafs at streaming speeds. In *Proceedings of ACM SIGMOD*, 2004.

[CM04]   G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 2004. in press.

[GGR02] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look. In *Proceedings of ACM SIGMOD*, 2002.

[GK01]   M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of ACM SIGMOD*, pages 58–66, 2001.

[GKMS02] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 454–465, 2002.

[GZ03]   A. Gupta and F. Zane. Counting inversions in lists. In *Proceedings of the thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 253–254, 2003.

[IC93]   Y. E. Ioannidis and S. Christodoulakis. Optimal histograms for limiting worst-case error propagation in the size of the join radius. *ACM Transactions on Database Systems*, 18(4):709–748, 1993.

[Ioa03] Yannis E. Ioannidis. The history of histograms (abridged). In *Proceedings of 29th International Conference on Very Large Data Bases*, pages 19–30, 2003.

[MP80]   J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.

[MRL98] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proceedings of ACM SIGMOD*, pages 426–435, 1998.

[MRL99] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *Proceedings of ACM SIGMOD*, volume 28(2) of *SIGMOD Record*, pages 251–262, 1999.

[Mut03] S. Muthukrishnan. Data streams: Algorithms and applications. In *ACM-SIAM Symposium on Discrete Algorithms*, http://athos.rutgers.edu/~muthu/stream-1-1.ps, 2003.

[Tec]   Agilent Technologies. Router tester. http://advanced.comms.agilent.com/n2x/index.htm.