Domain-Driven Data Synopses for Dynamic Quantiles

Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss

Abstract—In this paper, we present new algorithms for dynamically computing quantiles of a relation subject to insert as well as delete operations. At the core of our algorithms lies a small-space multiresolution representation of the underlying data distribution based on *random subset sums* or RSSs. These RSSs are updated with every insert and delete operation. When quantiles are demanded, we use these RSSs to estimate quickly, without having to access the data, all the quantiles, each guaranteed to be accurate to within user-specified precision. While quantiles have found many uses in databases, in this paper, our focus is primarily on network management applications that monitor the distribution of active *sessions* in the network. Our examples are drawn both from the telephony and the IP network, where the goal is to monitor the distribution of the length of active calls and IP flows, respectively, over time. For such applications, we propose a new type of histogram that uses RSSs for summarizing the dynamic parts of the distributions while other parts with small volume of sessions are approximated using simple counters.

Index Terms—Quantiles, database statistics, data streams.

1 Introduction

Most database management systems (DBMSs) maintain order statistics, i.e., quantiles, on the contents of their database relations. Medians (half-way points) and quartiles (quarter-way points) are elementary order statistics. In the general case, the ϕ-quantiles (for small real-valued $\phi > 0$) of an ordered sequence of N data items are the values with rank $k\phi N$, for $k = 1, 2, \dots \frac{1}{\phi} - 1$.

Quantiles find multiple uses in databases. Simple statistics such as the mean and variance are both insufficiently descriptive and highly sensitive to data anomalies in real-world data distributions. Quantiles can summarize massive database relations more robustly. Many commercial DBMSs use *equi-depth histograms* [30], [32], which are in fact, quantile summaries, during query optimization in order to estimate the size of intermediate results and pick competitive query execution plans. Quantiles can also be used for determining association rules for data mining applications [1], [2], [3]. Quantile distribution helps design well-suited user interfaces to visualize query result sizes. Also, quantiles provide a quick similarity check in coarsely comparing relations, which is useful in data cleaning [25]. Finally, they are used as splitters in parallel database systems that employ value range data partitioning [31] or for fine-tuning external sorting algorithms [14].

Computing quantiles on demand in many of the above applications is prohibitively expensive as it involves scanning large relations. Therefore, quantiles are precomputed within DBMSs. The central challenge then is to *maintain* them since database relations evolve via transactions. Updates, inserts, and deletes change the data distribution of the values stored in relations. As a result, quantiles have to be updated to faithfully reflect the changes in the underlying data distribution. Commercial database systems often hide this problem. Database administrators may periodically (say, every night) force the system to recompute the quantiles accurately. This has two well-known problems. Between recomputations, there are no guarantees on the accuracy of the quantiles: Significant updates to the data may result in quantiles being arbitrarily bad, resulting in unwise query plans during query optimization. Also, recomputing the quantiles by scanning the entire relation, even periodically, is still both computationally and I/O intensive.

In applications such as those described above, it often suffices to provide reasonable approximations to the quantiles and there is no need to obtain precise values. In fact, it suffices to get quantiles to within a few percentage points of the actual values. Here, we present one such algorithm for dynamically computing *approximate* quantiles of a relation subject to *both* insert and delete operations. Update operations of the form "change an attribute value of a specified record from its current value x to new value y" are thought of as a delete followed by an insert and are also supported. The algorithm monitors the operations and maintains a simple, small-space hierarchical representation of the underlying data distribution based on *random subset sums* or RSSs. Using these RSSs, we can estimate, *without having to access the data*, all the quantiles on demand, each guaranteed a priori to be accurate to within user-specified precision.

While RSSs can be used for tracking quantiles in all of the aforementioned applications, their value shines in applications that handle massive volumes of "streaming"

- *A.C. Gilbert and M.J. Strauss are with the Department of Mathematics, University of Michigan, 2074 East Hall, Ann Arbor, MI 48109-1109. E-mail: {annacg, martinjs}@umich.edu.*
- *Y. Kotidis is with AT&T Labs-Research, Room D272, 180 Park Ave., PO Box 971, Florham Park, NJ 07932-0971. E-mail: kotidis@research.att.com.*
- *S. Muthukrishnan is with the Department of Computer and Information Sciences, Rutgers University, 319 Core Bldg, 110 Frelinghuysen Rd., Piscataway, NJ 08854. E-mail: muthu@cs.rutgers.edu.*

Manuscript received 28 Aug. 2003; revised 9 June 2004; accepted 11 Oct. 2004; published online 18 May 2005.
For information on obtaining reprints of this article, please send e-mail to:

transactions for which the underlying relation is never materialized. In particular, in this paper, we focus on network management applications that monitor active network sessions. A session in the telephony network is a phone call and the distribution of interest that of the length of the active calls in the network. The notion of a session extends naturally to other domains (Web session, IP-flows). While our core RSS algorithm can be applied directly for monitoring the distribution of such calls, we have been able to adapt this algorithm to the particular characteristics of the application. Our solution is based on a histogram representation of the domain that allocates RSSs for the most voluminous parts of the distribution, while the rest of the domain is approximated using simpler statistics. The new algorithm has space requirements similar to those of the core RSS algorithm, but provides a lot more accurate estimates on real data sets, as our experiments demonstrate.

Despite the commercial use of quantiles, their popularity in database literature and their obvious fundamental importance in DBMSs, no comparable solutions were known previously for maintaining approximate quantiles efficiently with similar a priori guarantees. To our knowledge, this is the first streaming algorithm that provides a priori guarantees in the dynamic setting. Previously known one-pass quantile estimation algorithms that provide quality and performance guarantees either do not handle delete operations or they do not provide strong guarantees in the presence of deletes. Still, a comparison of our work with previous work must be kept in perspective since the insert-only and insert/delete settings are, strictly speaking, incomparable requirements, leading to different algorithmic goals and even to incomparable statements of cost—our cost depends on the size of the universe, whereas most previous work states cost in terms of the size of the relation. An application that has only inserts of data can demand better performance than an application facing inserts and deletes; previous results will be somewhat better than ours in the insert-only case. Our result is a general purpose technique that gives reasonable guarantees in the case of insert-only data and gives the first strong guarantees in the case of arbitrary inserts and deletes.

1.1 Two Applications for Dynamic Quantiles

The focus of our experimental analysis is on realistic data sets drawn from AT&T's voice network. Voice switches constantly generate flows of Call Detail Records (CDRs) that describe the activity of the network. Ad hoc analysis as part of network management focuses on *active* voice calls, that is, calls currently in progress at a switch. The goal is to get an accurate, but succinct representation of the length of all active calls and monitor the distribution over time.

Transactions in this context consist of the start and end of every call as observed by the switch. Each such action is monitored in order to maintain the distribution of all active calls in the network. Our algorithm fits nicely in this strongly dynamic context. In Section 6, we use archived call detail records to test our framework. Our experimental results show that it can accurately monitor the underlying distribution using a very small footprint (on the order of a few KBytes for call volume exceeding 2 million voice calls over a period of a day).

IP-networking applications also operate on session data. Network routers handle IP flows: a series of packets associated with a source/destination IP pair and specific start and end times. The "relation" at any point consists of active flows: A new flow that is initiated is considered an insert and any terminated flow is considered a delete. Our algorithm applies here ideally. CISCO routers generate one flow trace record per flow consisting of source/destination IP address, start and end time of the flow, type and size of the flow, and other ISP specific information [10]. In this context, our algorithm can be used for tracking *active* IP flows using flow data traces that are constantly generated from IP routers.

Voice and IP networks are challenging places for employing database infrastructure for network management. In fact, this need has already inspired several ongoing projects within the database community, e.g., the STREAM project at Stanford, the HIMALAYA project at Cornell, etc. Our work adds to this growing genre. An important aspect of network monitoring is that operations data is mostly distributed. As we discuss below, using our algorithm, one can process network data *where it is collected*, communicate only small RSS-summaries to a central location, then approximate quantiles for the combined data set *without any accuracy loss attributable to the distributed setting*. This is advantageous compared with techniques that require all the raw data to be sent to a centralized location.

2 Related Work

Since the early 1970s, there has been much focus on finding the median of a static data set. Following the breakthrough result of [7] that there is a comparison-based algorithm to find the median (and all the quantiles) in $O(N)$ worst-case time, more precise bounds have been derived on the precise number of comparisons needed in the worst case [29].

A series of algorithms have been developed for finding quantiles in the insert-only ("incremental") model. The idea presented in [28] leads to an algorithm that maintains an $O((\log^2 \epsilon N)/\epsilon)$ space data structure, which gets updated for each insert. Using this data structure, ϕ-quantiles can be produced that are a priori guaranteed to be ϵ-approximate. This algorithm was further extended in [6], [26] to be more efficient in practice and improved in [21] to use only $O((\log \epsilon N)/\epsilon)$ space and time. The approach in these papers is to maintain a "sample" of the values seen thus far, but the sample is chosen deterministically by enforcing various weighting conditions. Earlier, [28] had shown that any p-pass algorithm needs space $\Omega(N^{1/p})$ to compute quantiles exactly and, so, one needs to resort to approximation in order to get small space and fast algorithms.

A different line of research has been to use randomization so that the output is ϵ-approximate ϕ-quantiles with probability of at least $1-\delta$ for some prespecified probability δ of "failure." The intuition is that allowing the algorithms to only make probabilistic guarantees will potentially make them faster or use smaller space. In [26], [27], an $O(\frac{1}{\epsilon}\log^2\frac{1}{\epsilon}+\frac{1}{\epsilon}\log^2\log\frac{1}{\delta})$ space and time algorithm was given for this problem. Note that this is independent of N, dependent only on probability of failure and approximation factor. Other one-pass algorithms [3], [9], [24] do not provide a priori guarantees; however, performance of these algorithms on various data sets has been experimentally investigated. Both [27] and [21] also presented extensive experiments on the incremental model.

Despite the extensive literature above on probabilistic/deterministic, approximate/exact algorithms for finding quantiles in the incremental model, we do not know of a significant body of work that directly addresses the problem of dynamic maintenance of quantiles with a priori guarantees. Even though many of the algorithms that have been devised for the incremental setting can be extended to support a delete operation, such extensions do not provide strong guarantees in the computation of quantiles.

A novel algorithm that handles deletions has been proposed in [17] for dynamic maintenance of equi-depth histograms, which are ϕ-quantile summaries. Using a notion of error different from ours, the authors present an algorithm based on the technique of a "backing sample" (an appropriately maintained random sample [17], [34]) and provide a priori probabilistic guarantees on equi-depth histogram construction. This algorithm works well when deletions are infrequent, but, in general, it is forced to rescan the entire relation in presence of frequent deletes. In fact, the authors say "*based on the overheads, it is clear that the algorithm is best suited for insert-mostly databases or for data warehousing environments*."

Another related line of research is the maintenance of wavelet coefficients in presence of inserts and deletes. This was investigated in [18], [33], where the emphasis was on maintaining the largest (significant) coefficients. In [20], an algorithm was provided for dynamically maintaining V-Opt histograms. No a priori guarantees for finding quantiles can be obtained by using these algorithms.

Our techniques borrow some intuition from stream results [8], [11], [15], [20], [21], [23], where a "sketch"of a signal is used to summarize it. A sketch is any linear projection of the signal against a random vector. Thus, sketches include random samples as well as random subset sums. The basic sketching technique using linear projections was introduced in [4] (*AMS sketches*) for estimating the self-join sizes of relations in a dynamic context. The popularity of AMS sketches stems from their simplicity. As shown, such sketches can be generated with limited space using any family of four-wise independent random variables. More exotic random vectors have been employed in [8], [11], [18], [22] for tuning these sketches for different applications. In [8], the authors introduce the *CountSketch* for estimating the most frequent items in the data stream. CountSketches extend AMS sketches in that each entry (counter) in the AMS sketch is replaced with a hash table of multiple counters. This solves the problem of having high frequency items spoil estimates of lower frequency items when colliding in the same bucket. In our technique, we use a different method of decomposing the domain into dyadic intervals of different resolutions. Our technique is motivated by the requirement of the quantile estimation algorithm for obtaining accurate rangesum estimates over the distribution.

At an abstract level, the RSSs that our algorithm manipulates are hierarchical sketches over the data domain. Beyond this abstraction, algorithmic details of our work differ from all of the above results and, in particular, the time used by our algorithm to produce quantiles is much less than the time used by the previous results. Random subset sums require less randomness in their construction compared with AMS sketches. In fact, as we show in Section 4, any family of pairwise independent variables suffices, making construction and use of RSSs a lot simpler.

In a recent publication ([12]), the authors introduced the Count-Min sketches as an improvement over previous sketch-based data structures. Count-Min sketches improve previously known space bounds from $1/\epsilon^2$ to $1/\epsilon$ while requiring sublinear (in the size of the structure) update cost, unlike, for instance, the sketches of [4] whose update cost is proportional to the size of the sketch. By replacing RSSs with Count-Min sketches while keeping the hierarchical construction we describe in this paper, we can reduce the space requirements of our quantile estimation algorithm from $O(\log^2|U|\log(\log(|U|)/\delta)/\epsilon^2)$ to $O(\log^2|U|\log(\log(|U|)/\delta)/\epsilon)$, i.e., a $1/\epsilon$ improvement. A similar speedup is obtained in the computation of the ϕ-quantiles utilizing our technique of breaking each range-sum query into a set of point-estimates over dyadic intervals of different resolutions.

3 Approximate Quantile Estimation

3.1 Problem Definition

We consider a relational database and focus on some numerical attribute. The domain of the attribute is $U = \{0, \ldots, |U|-1\}$, also called the *Universe*.[1] In general, the domain may be a different discrete set or it may be real-valued and has to be appropriately discretized. Our results apply in either setting, but we omit those discussions. In what follows, we assume that the size of the domain $|U|$ is known a priori and is a power of two. In Section 4.4, we discuss the general case when U is unknown.

At any time, the database relation is a multiset of items drawn from the universe. We can alternately think of this as an array $\mathbf{A}[0\cdots|U|-1]$, where $\mathbf{A}[i]$ represents the number of tuples in the relation with value i in that attribute. Transactions consist of inserts and deletes. Insert(i) adds a tuple of value i, i.e., $\mathbf{A}[i] \leftarrow \mathbf{A}[i]+1$ and delete(i) removes an existing tuple with value i, i.e., $\mathbf{A}[i] \leftarrow \mathbf{A}[i]-1$. Let $\mathbf{A}_t$ be the array after t transactions and let $N_t = \sum_i \mathbf{A}_t[i]$; we will drop the subscript t whenever it is unambiguous from context.

Our goal is to estimate ϕ-quantiles on demand. In other words, we need to find the tuples with ranks $k\phi N$, for $1 \leq k < 1/\phi$. We will focus on computing ϵ-approximate ϕ-quantiles. That is, we need to find a j_k such that

$$(k\phi - \epsilon)N \leq \sum_{i \leq j_k} \mathbf{A}[i]$$

and

$$\sum_{i < j_k} \mathbf{A}[i] \leq (k\phi + \epsilon)N$$

1. We use $|U|$ to denote the size of the domain.

for $k = 1, \ldots, 1/\phi - 1$. The set of $j_1, \ldots, j_{\frac{1}{\phi-1}}$ will be the ϕ-quantiles approximate up to $\pm\epsilon N$. If $\epsilon = 0$, then we seek the exact quantiles.

Our goal is to solve this problem using sublinear resources. It would be ideal, of course, to use space no more than the $1/\phi$ that it takes to store the quantiles, but that appears to be unrealistic since the quantiles may change substantially under inserts and deletes. Therefore, in the spirit of prior work, our data structure will use space that is polylogarithmic in the universe size, which is typically much less than the size of the data set. Furthermore, we will only use time per operation nearly linear in the size our data structure, namely, polylogarithmic in the universe size.

If no transactions are allowed, we refer to the problem as *static*. If only insertions are allowed, we refer to it as *incremental* and, when both insertions and deletions are allowed, we refer to it as *dynamic*. It is obvious that, in a database system, $\mathbf{A}_t[i] \geq 0$ at any time t since one cannot delete a tuple that is not present in the relation. A sequence of transactions is called *well-formed* if it leads to $\mathbf{A}_t[i] \geq 0$; we will consider only well-formed sequences of transactions in this paper.

3.2 Our Main Algorithmic Result

We present a new technique for the problem of dynamically estimating quantiles. At the core of our algorithm lies an $O(\log^2 |U| \log(\log(|U|)/\delta)/\epsilon^2)$ space representation of $\mathbf{A}$. This gets updated with every insertion as well as deletion. When quantiles are demanded, we can estimate, *without having to access the data*, all the quantiles on demand, each guaranteed a priori to be accurate to within user-specified precision $\pm\epsilon N$ with user-specified probability $1 - \delta$ of success.

3.3 Challenges and Limitations

We now provide some more intuition into the problem of maintaining quantiles under inserts and deletes.

3.3.1 Recovering the Residual Distribution

In order to understand the challenge of maintaining approximate quantiles using small space, let us consider the following example: Our goal will be to maintain four quartiles to within moderate error of $\pm 0.1N$. Suppose a transaction stream consists of one million insertions followed by 999,996 deletions, leaving $N = 4$ items in the relation. Our problem specification essentially requires that, with high probability, we recover the database exactly.[2] A space-efficient algorithm knows very little about a data set of size one million and it does not know which 999,996 items will be deleted; yet, ultimately, it must recover the four surviving items. Although this is a contrived example, it illustrates the difficulty with maintaining order statistics in the face of deletes which dramatically change the data distribution.

Some incremental algorithms ([17], [21], [26]) work by sampling the data, either randomly and obliviously or with care to make sure the samples are spaced appropriately. Some of these techniques give provable results in the incremental setting. In the dynamic setting, however, a sample of the million items in the database at its peak size will be of little use at the time of the query in the example above since the sample is unlikely to contain any of the four eventual items. To apply known sampling algorithms, one needs to sample from the net data set at every point in time, which is difficult if there is severe cancellation. For example, in [16], the author addresses the problem of sampling from the net data set after inserts and deletes and states that "*if a substantial portion of the relation is deleted, it may be necessary to rescan the relation in order to preserve the accuracy of the guarantees.*"

3.3.2 Handling Distributed Session Data

Some of these issues with sampling arise when one wants to merge several data sets, even without deleting anything. That is, suppose there are two data sets for which we have samples, constructed either carefully or at random. It is not immediately clear how to combine the samples in a reasonable way. It is possible that the combined sample will be different from the sample that would have been produced directly from the combined data set.

Even with severe cancellation, quantiles of dynamic data can still be easy if the insertions and deletions come from, say, the same uniform distribution on a range and the relation always "covers" the range (even if, asymptotically, there are as many deletes as inserts). In that case, the net data set is also from the uniform distribution, so the quantiles of the net data set are approximately the same as the corresponding quantiles in the data set of insertions (ignoring deletions).

In general, however, severe cancellation is possible and the net distribution can be quite different from the distribution on insertions. Furthermore, all distributions can each change over time. In particular, we consider *session* data in Section 5. A session, such as a phone call, generates an insertion when the session starts and a deletion when the phone call ends. (The value i of the insertion and deletion is the start time.) Thus, the data set at any time consists of the active sessions and we can ask questions like "what is the median duration, start to present, of active sessions?" For session data, note that:

- The steady-state size of the data set may be constant or slowly growing so that there is severe cancellation of insertions with deletions.
- The distribution of insertions changes with time (by definition).
- If sessions typically have positive duration, then the distribution on insertions is different from the distribution on deletions.

4 The RSS Algorithm for Maintaining Quantiles

In this section, we will first present the high-level view of our algorithm with the main idea. Then, we will present specific details. In what follows, $E[X]$ and $var[X]$ denote the expected value and the variance of a random variable X,

2. For each pair i_1, i_2 of consecutive items in such a small relation, a quantiles algorithm gives us some j with $i_1 \leq j \leq i_2$. One can learn all four items exactly by making a few queries about ϕ-quantiles for ϕ slightly less than $1/4$ on a database consisting of the four original items and a few additional inserted items with strategic, known values.

respectively. At first, we will assume that $|U|$ is known to the algorithm; later, we will remove this assumption.

4.1 High-Level View of the Algorithm

In order to compute approximate ϕ-quantiles, we need a way to approximate $\mathbf{A}$ with a priori guarantees. In fact, our algorithm works by estimating *range-sums* of $\mathbf{A}$ over dyadic intervals. A *dyadic* interval $I_{j,k}$ is an interval of the form $[k2^{\log(|U|)-j}, (k+1)2^{\log(|U|)-j}-1]$, for j and k integers. The parameter j of a dyadic interval is its *resolution level* from *coarse*: $I_{0,0} = U$, to *fine*: $I_{\log(|U|),i} = \{i\}$. There are $\log(|U|)+1$ resolution levels and $2|U|-1$ dyadic intervals altogether in a tree-like structure. We will drop indexes j, k when referring to a dyadic interval unless it is required from the context.

We describe the main idea behind our algorithm here. The simplest form of a dyadic interval estimate is a *point* estimate, $\mathbf{A}[i]$. We proceed as follows: Let S be a (random) set of distinct values, each drawn from the universe with probability 1/2. Let $\mathbf{A}_S$ denote $\mathbf{A}$ projected onto the set S and let $\parallel \mathbf{A}_S \parallel = \sum_{i\in S}\mathbf{A}[i]$ denote the number of items with values in S. We keep $\parallel \mathbf{A}_S \parallel$ (a single number known as a *Random-Subset-Sum (RSS)*) for each of several random sets S. Observe that the expected value of $\parallel \mathbf{A}_S \parallel$ is $\frac{1}{2} \parallel \mathbf{A} \parallel$ since each point is in S with probability $\frac{1}{2}$.

For $\mathbf{A}[i]$, consider $E[\parallel \mathbf{A}_S \parallel | i \in S]$, which can be estimated by looking at counts $\parallel \mathbf{A}_S \parallel$ only for the Ss that contain i (close to half of all Ss, with high probability). One can show that this conditional expected value is $\mathbf{A}[i] + \frac{1}{2} \parallel \mathbf{A}_{U\setminus\{i\}} \parallel$ since the contribution of i is always counted but the contribution of each other point is counted only half the time. Since we also know $\parallel \mathbf{A} \parallel$, we can estimate $\mathbf{A}[i]$ as

$$2\left(\mathbf{A}[i] + \frac{1}{2} \parallel \mathbf{A}_{U\setminus\{i\}} \parallel\right) - \parallel \mathbf{A} \parallel .$$

It turns out that this procedure yields an estimate good to within ϵN, additively, if we take an average of $O(1/\epsilon^2)$ repetitions.

We can similarly be in position to estimate the number of data set items on any dyadic interval in U, up to $\pm\epsilon N$, by repeating the procedure for each dyadic resolution level up to $\log(|U|)$. A set S in this case is a collection of dyadic intervals from the same level, each taken with probability 1/2. A similar argument as above applies.

By writing any interval as a disjoint union of at most $\log(|U|)$ dyadic intervals, we can estimate the number of data set items in any interval. Now, we can perform repeated binary searches to find the quantiles left to right one at a time (i.e., first, second, etc.).

The entire algorithm relies on summarizing $\mathbf{A}$ using RSSs. Each item in $0, \dots, |U|-1$ participates in the construction of RSSs. In other words, we summarize the Universe using RSSs. This is a departure from previous algorithms for finding quantiles, which rely on keeping a sample of specific items in the input data set.

4.2 Algorithm Details

We will first describe our data structure and its maintenance before describing our algorithm for quantile estimation and presenting its analysis and properties.

4.2.1 *Our Data Structure and Its Maintenance*

For each resolution level j of dyadic intervals, we do the following: Pick a subset of the intervals $I_{j,k}$ at level j. Let S be the union of these intervals and let $\parallel \mathbf{A}_S \parallel$ be the count of values in the data sets that are projected onto S (formally, $\parallel \mathbf{A}_S \parallel = \sum_{i\in S}\mathbf{A}[i]$) . We repeat this process

$$num\_copies = 24\log(\log(|U|)/\delta)\log(|U|)/\epsilon^2$$

times and get sets $S_1, \dots, S_{num\_copies}$ (per level). The counts $\parallel \mathbf{A}_{S_l} \parallel$ for all sets that we have picked comprise our *Random Subset Sum* summary structure. In addition, we store (and maintain) $\parallel \mathbf{A} \parallel = N$ exactly.

We maintain these RSSs during inserts/deletes as follows: For insert(i), for each resolution level j, we quickly locate the single dyadic interval $I_{j,k}$ into which i falls (determined by the high order bits of i in binary). We then quickly determine those sets S_l that contain $I_{j,k}$. For each such set, we increase by one the counter for $\parallel \mathbf{A}_{S_l} \parallel$. For deletions, we simply decrease the counters. This process can be extended to handle *batch* insertions/deletion by increasing/decreasing the counters with appropriate weights.

An important technical detail is how to store and index various S_ls, which are random subsets. The straightforward way would be to store them explicitly, perhaps as a bitmap. But, this would use space $O(|U|)$, which we cannot afford. For our algorithm, we instead store certain random seeds of size $O(\log|U|)$ bits and compute a (pseudorandom) function that explicitly shows whether $i \in S_l$ or not. For this, we use the standard three-wise independent random variable construction shown below since it works well with our dyadic construction. We note that, for our algorithms, any pairwise random variable construction would suffice.

We need a generator $G(s,i) = S_i$ that quickly outputs the ith bit of the set S, given i and a short seed s for S. In particular, the generator takes a $O(\log|U|)$-bit seed and can be used to generate sets S of size $O(|U|)$. The generator G is the extended Hamming code, e.g.,

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix},$$

which consists of a row of 1s and then all the columns, in order. So, for each resolution level j, there is a G of size $(j+1)\times 2^j$. Then, $G(s,i)$ is the seed s of length $j+1$ dotted with the ith column of G modulo 2, which is efficient to compute—note that the ith column of G is a 1 followed by the binary expansion of i. This construction is known to provide 3-wise independent random variables [5]. We will use this property extensively when we analyze our algorithm.

4.2.2 *Estimating Quantiles*

Our algorithm for estimating quantiles relies on estimating sum of ranges, i.e., $\parallel \mathbf{A}_I \parallel$ for intervals I. First, we focus on dyadic intervals and then extend it to general intervals. Then, we show how to compute the quantiles:

1. *Computing* $\| \mathbf{A}_I \|$ *for dyadic intervals* I. Recall that $\| \mathbf{A}_I \|$ is simply the number of values in the data set that fall within the interval I. Given dyadic interval $I_{j,k}$, we want an estimate $\| \mathbf{A}_{I_{j,k}} \|_\sim$ of $\| \mathbf{A}_{I_{j,k}} \|$. We consider the random sets only in the resolution level j. Recall that there are *num_copies* such sets. Again, using the pseudorandom construction, quickly test each set to see whether it contains $I_{j,k}$ and ignore the remaining sets for this interval. An *atomic computation* for $\| \mathbf{A}_{I_{j,k}} \|$ is $2 \| \mathbf{A}_{S_l} - \| \mathbf{A} \|$ for $\mathbf{A}_{S_l}$ corresponding to a set S_l containing $I_{j,k}$.
2. *Computing* $\| \mathbf{A}_I \|$ *for arbitrary intervals.* Given an arbitrary interval I, write I as a disjoint union of at most $\log(|U|)$ dyadic intervals $I_{j,k}$. For each $I_{j,k}$, group the atomic computations into $3\log(\log(|U|)/\delta)$ groups of $8\log(|U|)/\epsilon^2$ each and take the average in each group. We can get an estimate for I by summing one such average for each of the dyadic intervals $I_{j,k}$. Since we have $3\log(\log(|U|)/\delta)$ groups, this creates $3\log(\log(|U|)/\delta)$ atomic estimates for I. Their median is the final estimate $\| \mathbf{A}_I \|_\sim$.

   In what follows, it is more convenient to regard our estimate $\| \mathbf{A}_I \|_\sim$ as an overestimate, $\| \mathbf{A}_I \| \leq \| \mathbf{A}_I \|_\sim \leq \| \mathbf{A}_I \| + \epsilon N$, by using a value of ϵ half as big as desired at top level and adding $\frac{\epsilon}{2}N$ to each estimate.
3. *Computing the quantiles.* We would like to estimate ϵ-approximate ϕ-quantiles. Recall that ϵ is fixed in advance. For $k = 1, \ldots, \frac{1}{\phi} - 1$, we want a j_k such that $\| \mathbf{A}_{[0,j_k)} \| = (k\phi \pm \epsilon)N$. Here, j_k is the value with rank $k\phi$, not to be confused with the resolution level j. For each prefix I, we can compute $\| \mathbf{A}_I \|_\sim$ as described above. Using binary search, find a prefix $[0, j_k)$ such that $\| \mathbf{A}_{[0,j_k)} \|_\sim < k\phi N \leq \| \mathbf{A}_{[0,j_k+1)} \|_\sim$ and return j_k. Repeat for all values of k.

We call the entire algorithm for the discovery and maintenance of quantiles the RSS algorithm.

4.2.3 Analysis of the RSS Algorithm

First, we consider the correctness of our algorithm in the lemma below and then summarize its performance in a theorem.

Lemma 1. *The* RSS *algorithm estimates each quantile to within* $\epsilon \| \mathbf{A} \| = \epsilon N$ *with probability at least* $1 - \delta$.

Proof. First, fix a resolution level j. Consider the set S formed by putting each dyadic interval $I_{j,k}$ at level j into S with probability $1/2$ as we did. In what follows, we drop the resolution level when indexing a dyadic interval, so $I_k = I_{j,k}$. Let X_k be a random variable defined by

$$X_k = \begin{cases} 2 \| \mathbf{A}_{I_k} \|, & I_k \in S; \\ 0, & \text{otherwise} \end{cases}$$

and let $X = \sum_k X_k$. Suppose we are presented with an interval I_{k_0}, dyadic at level j. We have, using 3-wise independence (pairwise will do),

$$\begin{aligned} E[X | I_{k_0} \in S] &= 2 \| \mathbf{A}_{I_{k_0}} \| + \sum_{k \neq k_0} E[X_k] \\ &= 2 \| \mathbf{A}_{I_{k_0}} \| + \sum_{k \neq k_0} \| \mathbf{A}_{I_k} \| \\ &= \| \mathbf{A}_{I_{k_0}} \| + \| \mathbf{A} \| . \end{aligned} \tag{1}$$

Also, since $\| \mathbf{A}_{I_{k_0}} \| \leq X \leq \| \mathbf{A}_{I_{k_0}} \| + 2 \| \mathbf{A} \|$,

$$\operatorname{var}[X | I_{k_0} \in S] \leq \| \mathbf{A} \|^2 .$$

Each prefix I is the disjoint union of $r \leq \log(|U|)$ dyadic intervals at different levels, $I = I_{k_1} \cup I_{k_2} \cup \cdots \cup I_{k_r}$. Let S_j be a random set of intervals at level j and let Y be the sum of corresponding X estimates. Then, summing (1) over j,

$$E[Y | \forall j \; I_{k_j} \in S_j] = \| \mathbf{A}_I \| + r \| \mathbf{A} \|,$$

so $E[Y | \forall j \; I_{k_j} \in S_j] - r \| \mathbf{A} = \| \mathbf{A}_I \|$, as desired. (Note that we have stored $\| \mathbf{A} \|$ exactly.) Also,

$$\operatorname{var}[Y | \forall j \; I_{k_j} \in S_j] \leq \log(|U|) \| \mathbf{A} \|^2 .$$

It follows that, if we let Z be the average of $8(\log(|U|)/\epsilon^2)$ repetitions of Y, the conditional expectation of $Z - r \| \mathbf{A} \|$ is $\| \mathbf{A}_I \|$ and the conditional variance of $Z - r \| \mathbf{A} \|$ is at most $\epsilon^2 N^2 / 8$. By the Chebyshev inequality, $|Z - r \| \mathbf{A} \| - \| \mathbf{A}_I \| | < \epsilon N$ with probability at least $7/8$. Finally, if we take $3\log(\log(|U|)/\delta) = 3(\log(1/\delta) + \log\log(|U|))$ copies of Z and take a median, by the Chernoff inequality, $|Z - r \| \mathbf{A} \| - \| \mathbf{A}_I \| | < \epsilon N$ with probability at least $1 - \delta / \log(|U|)$. Both Chebyshev and Chernoff inequalities can be found in [5] and averaging arguments similar to the above can be found, for example, in [4].

We performed binary search to find a j_k such that $\| \mathbf{A}_{[0,j_k)} \|_\sim < k\phi N \leq \| \mathbf{A}_{[0,j_k+1)} \|_\sim$. It follows that

$$\begin{aligned} \| \mathbf{A}_{[0,j_k)} \| &\leq \| \mathbf{A}_{[0,j_k)} \|_\sim \\ &< k\phi N \\ &\leq \| \mathbf{A}_{[0,j_k+1)} \|_\sim \\ &\leq \| \mathbf{A}_{[0,j_k+1)} \| + \epsilon N, \end{aligned}$$

as desired.

To estimate a single quantile, we will, $\log(|U|)$ times, estimate $\| \mathbf{A}_I \|$ on a prefix I in the course of binary search. Since *each* estimate fails with probability $\delta / \log(|U|)$, the probability that *any* estimate fails is at most $\log(|U|)$ times that, i.e., δ. □

Therefore, by summing up space used and the time taken for algorithms we have described, we can conclude the following.

Theorem 2. *The* RSS *algorithm uses*

$$O\left(\log^2(|U|) \log\left(\frac{t \log(|U|)}{\phi\delta}\right) / \epsilon^2\right)$$

space and provides ϵ-approximate ϕ-quantiles with probability at least $1 - \delta$ for t queries. The time for each insert or delete operation and the time to find each quantile on demand are proportional to the space.

Note that we can find a single quantile with cost $O((\log^2(|U|)\log(\log(|U|)/\delta))/\epsilon^2)$. If we make t queries, each of which requests $1/\phi$ quantiles, we need the probability of *each* failure to be less than $\delta\phi/t$ in order that the probability of *any* failure to be less than δ. This accounts for the cost factor $\log(t/\phi)$.

4.3 Some Observations on the RSS Algorithm

Our approach of summarizing the universe using RSSs has interesting implications for our algorithm which we summarize here:

- The RSS algorithm may return a quantile value which was not seen in the input. In general, in the face of severe cancellation, an algorithm with space less than N cannot keep track of which items are currently in the data set.
- The distribution on values returned by the RSS algorithm depends only on the data set active at the time of the query. Thus, one can change the order of insertions and deletions (e.g., batch them) without affecting results.
- Our RSSs are *composable*, that is, if updates are generated in two separate locations, each location can compute random subset sums on its data, using preagreed common random subsets (seeds). The subset sums for the combined data set is just the sum of the two subset sums. Hence, we can compute the quantiles of the combined data set very quickly from their RSSs alone. Notice that no loss in accuracy is generated from this process. The resulting ϕ-quantiles will be the same had we directed all transactions to a single RSS (of the same size) and computed the quantiles there. It is OK if either or both of the individual data sets are ill-formed, provided the combined data set is well-formed. For example, a car rental agency franchise can insert a record with the model year of a car as the car is rented out and a (possibly different) franchise can delete the record when the car is returned. Thus, the various franchises may compute on ill-formed data sets. To estimate the median age of cars on the road, the franchises need only share the subset sums, not the large data sets; since the combined data set is well-formed, the computation on the combined RSS is correct.
- Because RSSs are composable, our entire algorithm is easily parallelizable. If data is arriving quickly (for example, in the case of IP network operations data), the data can be sent to an array of parallel machines for processing. Of course, the cumulative space requirements of the RSSs will be multiplied by the number of parallel machines used. Results can be composed at the end.

4.4 Extension to When the Universe Size Is Unknown

In the previous section, we assumed that the universe size is known in advance. In practice, this may not be the case; fortunately, our algorithm can easily adapt to an increasing universe, with modest increased cost factor of at most $\log^2\log(|U|)$ compared with knowing the universe size in advance.

We start the algorithm as above, with a predicted range $[0, u-1]$ for U. Suppose we see an insertion of $i \geq u = |U|$, where, at first, we assume $i < u^2$. Then, before processing i, we update the RSS data structure in two ways. First, we construct random subsets at each of the new (coarser) resolution levels. We initialize the corresponding RSSs to zero or to the stored value $\parallel \mathbf{A} \parallel$, as appropriate. For each of the resolution levels already present, we extend each random subset to the new universe by picking a new seed to specify membership within $[u, u^2)$. (We keep both the old and the new seed for this random subset since we'll need both to decide membership on all of $[0, u^2)$.) Each new RSS at these resolution levels maintains itsr old value, which remains correct in the context of the new universe $[0, u^2)$ since no elements in $[u, u^2)$ have yet been seen. It remains to analyze the costs of the data structure.

Suppose the largest item seen is i_* and let u_* be the smallest power of 2 greater than i_*. Thus, if we knew i_* in advance, we would use a single instance of RSS on a universe of size u_*, with cost $f(u_*)$ for f given in Theorem 2. The multi-instance data structure we construct has the largest instance on a universe of size u_*^2 and $\log\log(u_*^2)$ instances altogether. Thus, the time and space costs of the multi-instance data structure are at most $O(f(u_*^2)\log\log(u_*))$. Since the dependence on u of f is polylogarithmic, the cost of the multi-instance data set is just the factor $\log\log(u_*)$ compared with knowing u_* in advance. An additional cost factor of $2\log j$ is needed for the jth instance, $j = 1, 2, \ldots, \log\log(u_*)$, to drive down the probability of failure to $1/j^2$ so that the overall probability $\sum_j \frac{1}{j^2}$ remains bounded. The claimed overhead factor $O(\log^2\log(|U|))$ follows.

4.5 Handling Infinite Resolution Domains

We can perform a similar process for items that fall *between* elements of the universe. For example, suppose the universe consists of all text strings of length 20. For ease of exposition, assume that the text strings are actually bit strings. If we see a string of length 21, it falls equally spaced between two strings of length 20. We can extend the universe to include finer resolution levels in a way analogous to the way we extended the universe to include coarser levels above, with only a moderate cost increase compared with knowing the universe in advance. Of course, there are additional heuristics that one can employ for certain cases. For instance, if the dictionary mainly consists of small strings, it will be more efficient to handle large strings as an exception and store them aside in a different data structure.

4.6 The $\mathrm{RSS}[\ell]$ Algorithm

For the coarsest few levels, say, to level ℓ, it is more efficient to store exact subset sums for *each* of the (few) dyadic intervals at that level. This immediately lets us get $\parallel \mathbf{A}_I \parallel$ for any I dyadic at that level, in time $O(1)$. Furthermore, if space is at a premium, we can store just the subset sums for the dyadic intervals *at* level ℓ itself since any coarser interval can be written as the disjoint union of dyadics at level ℓ. We refer to such an implementation as RSS$[\ell]$.

5 RSS-Histograms for Session Data

In general, the RSS algorithm is designed for arbitrary dynamic data. In practice, there may be empirical properties of the data that can be exploited to save space in the data structure. Previous algorithms considered dynamic data that is mostly insertions or close to uniform on a known range. In this section, we consider yet other common assumptions and show how the RSS algorithm and another simple algorithm (described below) can be combined to exploit these assumptions.

We assume that the domain of $\mathbf{A}$ is time, discretized in some sensible fashion (e.g., seconds, minutes) depending on the application. Our focus is on network monitoring applications that generate session data (i.e., call detail, IP flows). Our goal is to provide approximate quantile computations for the duration (start to present) of active sessions. We use the following paradigm: Each time a session starts at time t_i, we add 1 to $\mathbf{A}[t_i]$ and, when the session ends, we subtract 1 from $\mathbf{A}[t_i]$ (notice t_i is the start-time of the session in both cases). We call the first operation an "insertion" and the latter a "deletion." Our algorithm directly approximates quantiles for the start time; it is easy to see that the current time minus the kth $\phi\text{-quantile}$ start time is the $(1-k\phi)\text{th}$ quantile duration, and that approximate quantiles for start time are equivalent to approximate quantiles for duration.

Observe that our setup typically has several interesting characteristics:

- The range of possible values (start times) constantly changes as old sessions end and new sessions begin. This is markedly different from data drawn from an unknown distribution that fully covers a known range.
- The data is subject to severe cancellation. One expects that, in the long run, each phone call that is started is also finished.
- Insertions happen only at the right. That is, a new start time can only be the current time.[3] Deletions can occur anywhere.

In practice, sometimes one can assume a *minimum utilization level* on the observed network element N_{min} at time t, i.e.,

$$N_t = \sum_{t_i} \mathbf{A}_t[t_i] \geq N_{\min}.$$

Note that, even with large $N_{\min}$ in force, a stream may still be subject to severe cancellation and to a shifting range of values.

If there's a large $N_{\min}$ *and* insertions only occur at the right, then one can use a simpler algorithm, similar to [13]. The algorithm, at steady state,[4] maintains a histogram in which the bucket sizes are as follows:

- All buckets have size at most $\epsilon N_{\min}$.
- Any two consecutive buckets have combined size greater than $\epsilon N_{\min}$.

When the most recent bucket grows to size $\epsilon N_{\min}$ (due to a net surplus of inserts over deletions from this bucket), it is sealed and a new most recent bucket is started. When some pair of consecutive buckets get a combined size of at most $\epsilon N_{\min}$, the two are merged.

By assumption, $N_t \geq N_{\min}$, so $\epsilon N_{\min} \leq \epsilon N_t$. If we answer quantile queries from the boundaries of buckets, the error is at most $\epsilon N_{\min} \leq \epsilon N_t$, which is allowed. By assumption, insertions only occur at the right, so buckets other than the most recent will never grow, so they will never be bigger than $\epsilon N_{\min}$. The number of buckets at time t is bounded above, by $O(N_t/(\epsilon N_{\min}))$, which, depending on $N_t/N_{\min}$, may be much more efficient than RSSs.

If either of these assumptions fails, however, the above algorithm will be incorrect and something like RSSs are needed. For instance, $N_{\min}$ may be small, even zero. Then, the number of histogram buckets, $N_t/(\epsilon N_{\min})$, will be unacceptably large. Furthermore, although insertions only occur at the current time, there may be a delay in reporting them. For instance, we may have several network elements that periodically propagate their data to a local processing node for further processing and summarization. We may still want approximate quantiles with respect to the best-effort available data. If insertions can occur in buckets of size at least ϵN_t, then the histogram algorithm will fail.

We now address data where $N_{\min}$ is small or insertions are delayed but there is a bound to the delay of insertions. We propose the RSS_hist data structure, a hybrid between the RSS and histogram, that is provably correct, yet provides some of the space savings of the histogram structure. We partition the universe into intervals. Data in each interval is tracked either by a counter, as in the above histogram algorithm, or by an RSS. The interval for the most recent data is always tracked by an RSS because insertions (which may be delayed) may occur in the interior of that interval. Intervals where the count is greater than $\epsilon N_{\min}$ are also tracked by an RSS; other intervals are tracked by counters.

We need to say how to update this structure:

- By construction, all intervals where (possibly delayed) insertions are allowed are tracked by RSSs, which are updated in the usual way.
- When the time length of the most recent interval gets to some parameter ΔU, that interval is sealed and a new interval is started. At this point, both of the two most recent intervals are tracked by an RSS since insertions will be possible in both intervals.
- Two consecutive intervals tracked by counters with a combined sum of at most $\epsilon N_{\min}$ are merged into a combined interval tracked by a single counter.
- If a deletion occurs in an interval tracked by an RSS making the count drop to $\epsilon N_{\min}$ and the interval is

3. Below, we also consider sessions whose start is not reported to us immediately. The effect will be that insertions become restricted only to happen near the right.

4. At the outset, one may either preload the structure with at least $N_{\min}$ active sessions hypothesized to exist or, in practice, expect to miss the sessions active at the start of monitoring. In the latter case, we ignore deletions of start times prior to the start of monitoring since these refer to sessions never entered into the data structure.

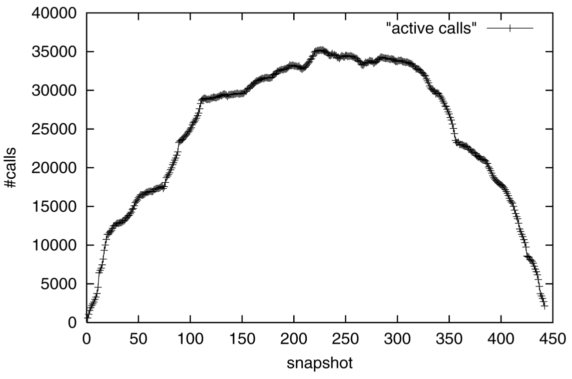


Fig. 1. Active phone calls over time.

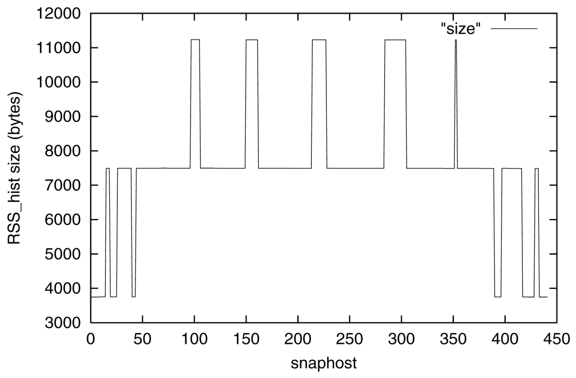


Fig. 2. Size of `RSS_hist` over time.

old enough that insertions are no longer possible, the RSS is demoted to a single counter for the interval.

By varying ΔU and our conservative estimate $N_{\min}$ for the true minimum utilization, we can tune the space used. Note that, as $N_{\min}$ gets small, intervals are likely to have more than $\epsilon N_{\min}$ elements in which case they are likely to be tracked by RSSs instead of counters (so the best $N_{\min}$ to use is the maximum true value). As ΔU gets smaller, there are likely to be more intervals in which delayed insertions are allowed and, so, RSSs are needed. But, if ΔU is too big, as an extreme, we get just a single RSS and no benefit of using counters. The ideal balance is a situation in which RSSs are used only where needed and counters are used for vast sparse portions of the universe.

6 Experiments

In [19], we have presented an experimental study of the performance of the `RSS` algorithm for synthetic and real data sets that contain mix of insert and delete operations. Here we focus on the performance of `RSS` and `RSS_hist` using archived Call Detail Records (CDRs) from AT&T's network. Switches constantly generate flows of CDRs that describe the activity of the network. Ad hoc analysis as part of network management focuses on *active* voice calls, that is, calls currently in progress at a switch. The goal is to get an accurate, but succinct representation of the length of all active calls and monitor the distribution over time.

The basic question we want to answer is how to compute the median length of *ongoing* calls at any given point in time, i.e., what is the median length of a call that is currently active in the network? We then focus on other quantiles.

Our data is presented here as a stream of transactions of the form

$$(time\_stamp, orig\_tel, start\_time, flag),$$

where $time\_stamp$ indicates the time an action has happened (start/end of a call), $orig\_tel$ is the telephone number that initiates the call, $start\_time$ indicates when a call was started, and flag has two values: +1 for indicating the beginning of the call and -1 for indicating the end of the call. The actual CDRs carry a lot of additional information that is not required for this discussion.

Given this data, we define a *virtual* array $\mathbf{A}[t_i]$ that counts the number of phone calls started at time t_i. This array can be maintained in the following manner: Each time a phone call starts at time t_i, we add 1 to $\mathbf{A}[t_i]$ and, when the call ends, we subtract 1 from $\mathbf{A}[t_i]$ (notice t_i is the $start\_time$ in both cases). For example, the following CDRs:

12:00	999-000-0000	12:00	$+1$
12:01	999-000-0001	12:01	$+1$
12:03	999-000-0001	12:01	-1
12:10	999-000-0000	12:00	-1

describe two phone calls. The first originates from number 999-000-0000, starts at 12:00, and ends at 12:10, while the second originates from number 999-000-0001 at 12:01 and lasts for two minutes.

We used a data set of 2.2 million CDRs (4.4 million records since each phone call generates two records in our schema) covering a period of 18 hours. Fig. 1 shows the number of active calls over time; we take one snapshot every 10,000 records. There are up to 35,000 active calls at peak times. We first executed the `RSS_hist` algorithm with parameters $\epsilon = 0.1$, $N_{min} = 20,000$, and $\Delta U = 2,048$ seconds (i.e., around 35 minutes). We further rounded all bucket boundaries at times later than the start of monitoring by some multiple of ΔU. We assumed that all insertions occur at the right. Fig. 2 plots the size of the histogram over time. (Note that the size of an RSS is about 3.7KB and there were either one, two, or three RSSs throughout the experiment.) The maximum memory footprint was slightly below 11KB with the number of buckets in the histogram varying between 1 and 6.

Fig. 3 plots the error in computing the median of the ongoing calls (in resolution of 1 second) over time (we probed for estimates every 10,000 records). We notice that the actual error is well below the ϵ value used. By cross-checking these errors with Fig. 1, we notice that the method is extremely accurate even during periods when the number of call is less than N_{min} (20,000). In Fig. 4, we repeated this experiment using RSSs and also an implementation of the algorithm presented in [17] for incremental maintenance of approximate histograms (denoted `Hist` in the figure). This algorithm uses a "backing sample," which

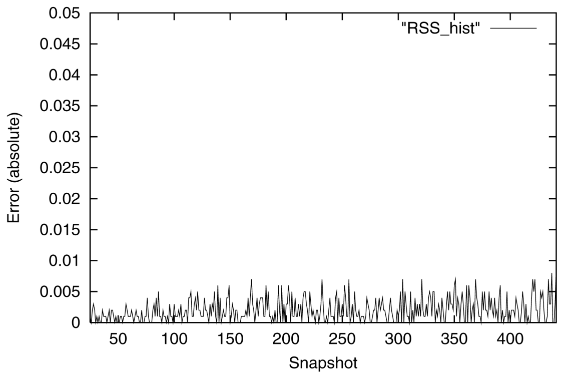


Fig. 3. Actual error in computation of median for $\epsilon = 0.1$.

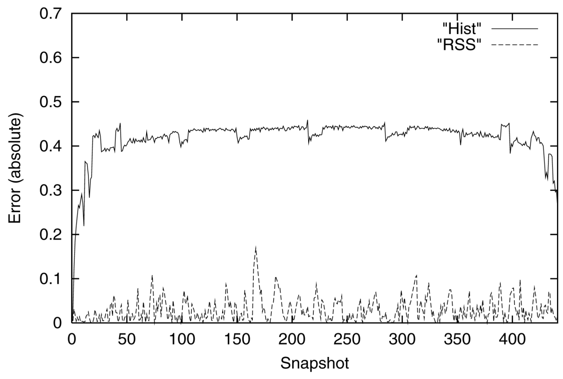


Fig. 4. Error in computation of median.

is an approximate random sample of the data set kept up-to-date in the presence of insert/delete operations. The backing sample is based on the *Reservoir Sampling* technique by Vitter [34]. No rescans were allowed as the focus of this study is on streaming data. Both algorithms were set up to use 11KB of space, i.e., the maximum footprint size of the RSS_hist histogram. Note that an error of η in computing the median indicates that the rank returned is $.5 \pm \eta$. So, an error of $\eta = 0.5$ indicates useless output. In Fig. 5, we further plot the average error for all ϕ-quantiles for $\phi = 0.10$ (deciles). For a more clear view, the y-axis (errors) is logarithmic. In this figure, note that the maximum error of the kth ϕ-quantile is $\max(k\phi, 1 - k\phi)$.

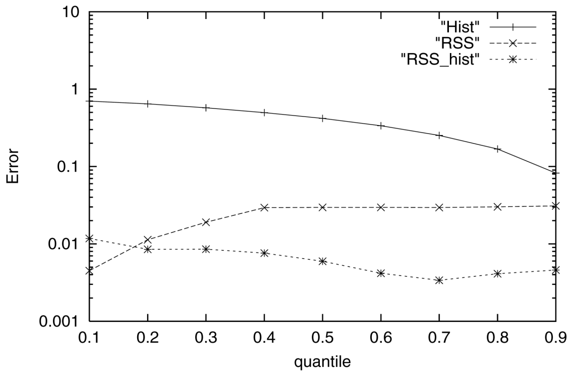


Fig. 5. Average error for all $\phi = 0.1$ quantiles (log y-axis).

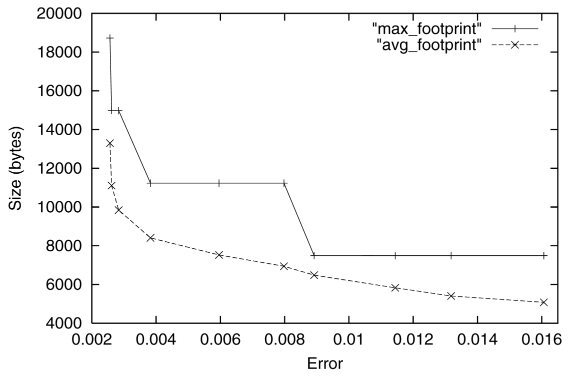


Fig. 6. Maximum and average footprint size of RSS_hist for different errors.

In Fig. 6, we summarize the results for several runs of the RSS_hist algorithm, varying N_{min} and fixing $\epsilon = 0.1$. The x-axis plots the average error over all deciles and all snapshots (4,110 values), while the y-axis shows the average and maximum memory size of the hybrid histogram for that error.

Up to this point, we concentrated on the absolute quantile error of the approximation, which is the focus of this paper. As noted earlier, the maximum error of the kth ϕ-quantile is $\max(k\phi, 1 - k\phi)$. An implication of the quantile computation is that, on sparse areas of the data distribution, quantile boundaries can be chosen loosely without sacrifice in the quality of the approximation. In fact, we exploit this observation in the RSS_hist algorithm. If the location of the quantile of interest is given by index i_1 (in $\{0 \dots |U| - 1\}$), a quantile approximation algorithm that outputs index i_2 with, e.g., $i_2 > i_1$, will produce an error proportional to $\sum_{j=i_1+1}^{i_2} \mathbf{A}[j]$. In sparse areas, the actual error of the quantile computation will be small or even zero when there are no data points between i_1 and i_2. For most applications, this does not seem to have severe implications. When monitoring session data, the actual index value (t_i) of the quantile computation is important in order to interpret the results. For a better view of the approximation obtained, we plot in Fig. 7 the error (in seconds) of the approximation of the median length of the active calls over time. The errors for the other deciles were similar. We notice that RSS_hist

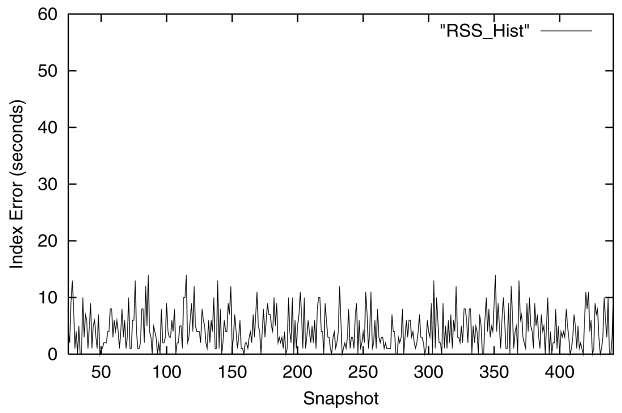


Fig. 7. Error in estimation of median-length (RSS_hist).

provides a very accurate estimate of the median length of the calls; the computed median is, on average, within 4.3 seconds of the real value (which was around 4 minutes on the average).

7 Conclusions

We have presented an algorithm for maintaining dynamic quantiles of a relation in the presence of both insert as well as delete operations. The algorithm maintains a small-space representation (RSSs) that summarizes the universe and the underlying distribution of the data within it. This algorithm is novel in that, without having to access the relation, it can estimate each quantile to within user-specified precision. Previously published algorithms provide no such guarantees under the presence of deletions.

Our guarantees are in terms of the universe size and not in the size of the relation, the number of transactions, or the ratio of insertions over deletions. The algorithm maintains a small-space representation (RSSs) that summarizes the universe and the underlying distribution of the data within it. Furthermore, it can be adapted for cases where the universe size or resolution change. In this paper, we focused on session data as part of network monitoring applications. We have been able to devise a hybrid histogram data structure that uses RSSs for approximating the distribution of active network sessions over time. To our knowledge, our algorithm is the first to provide strict guarantees in this setup.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining Associations between Sets of Items in Massive Databases," *Proc. ACM SIGMOD,* pp. 207-216, May 1993.

[2] R. Agrawal and R. Srikant, "Mining Quantitative Association Rules in Large Relational Tables," *Proc. ACM SIGMOD,* pp. 1-12, June 1996.

[3] R. Agrawal and A. Swami, "A One-Pass Space-Ecient Algorithm for Finding Quantiles," *Proc. Conf. Management of Data,* 1995.

[4] N. Alon, Y. Matias, and M. Szegedy, "The Space Complexity of Approximating the Frequency Moments," *Proc. ACM Symp. Theory of Computing,* pp. 20-29, 1996.

[5] N. Alon and J.H. Spencer, *The Probabilistic Method.* New York: Wiley and Sons, 1992.

[6] K. Alsabti, S. Ranka, and V. Singh, "A One-Pass Algorithm for Accurately Estimating Quantiles for Disk-Resident Data," *Proc. Very Large Data Bases Conf.,* pp. 346-355, 1997.

[7] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan, "Time Bounds for Selection," *J. Computer and System Sciences,* vol. 7, no. 4, pp. 448-461, 1973.

[8] M. Charikar, K. Chen, and M. Farach-Colton, "Finding Frequent Items in Data Streams," *Proc. 29th Int'l Colloquium Automata, Languages, and Programming,* 2002.

[9] F. Chen, D. Lambert, and J.C. Pinheiro, "Incremental Quantile Estimation for Massive Tracking," *Proc. Int'l Conf. Knowledge Discovery and Data Mining,* pp. 516-522, Aug. 2000.

[10] Cisco NetFlow, http://www.cisco.com/warp/public/732/net flow/, 1998.

[11] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan, "Comparing Data Streams Using Hamming Norms (How to Zero In)," *Proc. Very Large Data Bases Conf.,* pp. 335-345, 2002.

[12] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-Min Sketch and Its Applications," *LATIN,* pp. 29-38, 2004.

[13] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining Stream Statistics over Sliding Windows," *Proc. 13th ACM-SIAM Symp. Discrete Algorithms,* 2002.

[14] D.J. DeWitt, J.F. Naughton, and D.A. Schneider, "Parallel Sorting on a Shared-Nothing Architecture Using Probabilistic Splitting," *Proc. Conf. Parallel and Distributed Information Systems,* pp. 280-291, 1991.

[15] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi, "Processing Complex Aggregate Queries over Data Streams," *Proc. ACM SIMGOD,* pp. 61-72, June 2002.

[16] P. Gibbons, "Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports," *Proc. Very Large Data Bases Conf.,* pp. 541-550, 2001.

[17] P. Gibbons, Y. Matias, and V. Poosala, "Fast Incremental Maintenance of Approximate Histograms," *Proc. Very Large Data Bases Conf.,* pp. 466-475, 1997.

[18] A.C. Gilbert and Y. Kotidis, S. Muthukrishnan, and M.J. Strauss, "Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries," *Proc. Very Large Data Bases Conf.,* pp. 79-88, 2001.

[19] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M.J. Strauss, "How to Sumamtize the Universe: Dynamic Maintenance of Quantiles," *Proc. Very Large Data Bases Conf.,* pp. 454-465, 2002.

[20] A.C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M.J. Strauss, "Fast, Small-Space Algorithms for Approximate Histogram Maintenance," *Proc. 34th ACM Symp. Theory of Computing,* pp. 389-398, 2002.

[21] M. Greenwald and S. Khanna, "Space-Efficient Online Computation of Quantile Summaries," *Proc. ACM SIGMOD,* pp. 58-66, May 2001.

[22] P. Indyk, "Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computation," *Proc. 41st Symp. Foundations of Computer Science,* pp. 189-197, 2000.

[23] P. Indyk, N. Koudas, and S. Muthukrishnan, "Identifying Representative Trends in Massive Time Series Data Sets Using Sketches," *Proc. Very Large Data Bases,* pp. 363-372, 2000.

[24] R. Jain and I. Chlamtac, "The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms without Storing Observations," *Comm. ACM,* vol. 28, no. 10, 1985.

[25] T. Johnson, S. Muthukrishnan, P. Dasu, and V. Shkapenyuk, "Mining Database Structure; or, How to Build a Data Quality Browser," *Proc. ACM SIGMOD,* 2002.

[26] G.S. Manku, S. Rajagopalan, and B.G. Lindsay, "Approximate Medians and Other Quantiles in One Pass and with Limited Memory," *Proc. ACM SIGMOD,* 1998.

[27] G.S. Manku, S. Rajagopalan, and B.G. Lindsay, "Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Data Sets," *Proc. ACM SIGMOD,* pp. 251-262, 1999.

[28] J.I. Munro and M.S. Paterson, "Selection and Sorting with Limited Storage," *Theoretical Computer Science,* vol. 12, pp. 315-323, 1980.

[29] M.S. Paterson, "Progress in Selection," technical report, Univ. of Warwick, Coventry, U.K., 1997.

[30] V. Poosala, "Histogram-Based Estimation Techniques in Database Systems," PhD dissertation, Univ. of Wisconsin-Madison, 1997.

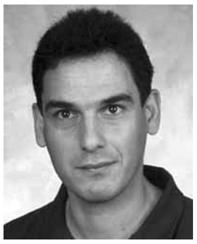
[31] V. Poosala and Y.E. Ioannidis, "Estimation of Query-Result Distribution and Its Application in Parallel-Join Load Balancing," *Proc. Very Large Data Bases Conf.,* pp. 448-459, 1996.

[32] V. Poosala, Y.E. Ioannidis, P.J. Haas, and E.J. Shekita, "Improved Histograms for Selectivity Estimation of Range Predicates," *Proc. ACM SIGMOD,* pp. 294-305, 1996.

[33] Y. Matias, J. Vitter, and M. Wang, "Dynamic Maintenance of Wavelet-Based Histograms," *Proc. Very Large Data Bases Conf.,* pp. 101-110, Sept. 2000.

[34] J.S. Vitter, "Random Sampling with a Reservoir," *ACM Trans. Math. Software,* vol. 11, no. 1, pp. 37-57, 1985.

Anna Gilbert received the SB degree from the University of Chicago and the PhD degree from Princeton University, both in mathematics. She is an assistant professor in the Mathematics Department at the University of Michigan. From 1998 to 2004, she was a member of technical staff at AT&T Labs-Research in Florham Park, New Jersey. From 1997 to 1998, she was a postdoctoral fellow at Yale University and AT&T Labs-Research. Her interests include analysis, probability, networking, and algorithms.



Yannis Kotidis received the BSc degree in electrical engineering and computer science from the National Technical University of Athens and the MSc and PhD degrees in computer science from the University of Maryland. He is a senior technical specialist at AT&T Labs-Research in Florham Park, New Jersey. His interests include data warehousing, approximate query processing, sensor/stream databases, and data integration.

S. Muthukrishnan graduated from the Courant Inst of Mathematical Sciences; his thesis work was on probabilistic games and pattern matching algorithms. He was a postdoctoral researchers working on computational biology for a year, on the faculty at the University of Warwick, United Kingdom, and has been at Bell Labs and then at AT&T Research and Rutgers University. His research explores fundamental algorithms as well as algorithms applied to databases, networking, compression, scheduling, etc.

Martin J. Strauss receives the AB degree from Columbia University and the PhD degree from Rutgers University, both in mathematics, and spent a year at Iowa State University and seven years at AT&T before joining the University of Michigan. He is an assistant professor in the Math and Electrical Engineering and Computer Science Departments at the University of Michigan. He has written several articles in algorithms, complexity theory, cryptography, and computer security, mathematical approximation theory, and other topics. Dr. Strauss is currently interested in algorithms for massive data sets.