How to Summarize the Universe: Dynamic Maintenance of Quantiles

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

AT&T Labs Research
Florham Park, NJ
07032 USA
{*agilbert,kotidis,muthu,mstrauss*}*@research.att.com*

Abstract

Order statistics, i.e., quantiles, are frequently used in databases both at the database server as well as the application level. For example, they are useful in selectivity estimation during query optimization, in partitioning large relations, in estimating query result sizes when building user interfaces, and in characterizing the data distribution of evolving datasets in the process of data mining.

We present a new algorithm for dynamically computing quantiles of a relation subject to insert as well as delete operations. The algorithm monitors the operations and maintains a simple, small-space representation (based on *random subset sums* or RSSs) of the underlying data distribution. Using these RSSs, we can quickly estimate, without having to access the data, all the quantiles, each guaranteed to be accurate to within user-specified precision. Previously-known one-pass quantile estimation algorithms that provide similar quality and performance guarantees can not handle deletions. Other algorithms that can handle delete operations cannot guarantee performance without rescanning the entire database.

We present the algorithm, its theoretical performance analysis and extensive experimental results with synthetic and real datasets. Independent of the rates of insertions and deletions, our algorithm is remarkably precise at estimating quantiles in small space, as our experiments demonstrate.

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

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 of an ordered sequence of N data items are the values with rank $k\phi N$, for k=$1, 2, \ldots 1/\phi$.

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* [21, 23], which are in fact quantiles, 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, 3, 2]. 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 [16]. Finally, they are used as splitters in parallel database systems that employ value range data partitioning [22] or for fine-tuning external sorting algorithms [9].

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 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.

We present a new algorithm for dynamically computing quantiles of a relation subject to *both* insert and delete operations.[1] The algorithm monitors the operations and maintains a simple, small-space representation (based on *random subset sums* or RSSs) of the underlying data distribution. 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. The algorithm is highly efficient, using space and time significantly sublinear in the size of the relation.

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. Previously known one-pass quantile estimation algorithms that provide similar a priori quality and performance guarantees can not handle delete operations; they are useful for refreshing statistics on an append-only relation but are unsuitable in presence of general transactions. Other algorithms that can handle modify or delete operations rely on a small "backing sample" or "distinct sample" of the database and cannot guarantee similar performance without rescanning the relation.

We perform an extensive experimental study of maintaining quantiles in presence of general transactions. We use synthetic data sets and transactions to study the performance of our algorithm (as well as a prior algorithm we extended to our dynamic setting) with varying mixes of inserts and deletes. We also use a real, massive data set from an AT&T warehouse of active telecommunication transactions. Our experiments show that our algorithm has a small footprint in space, is fast, and it performs with remarkable accuracy in all our experiments, even in presence of rapid inserts and deletes that change underlying data distribution substantially.

In the rest of this section, we state our problem formally, discuss prior work and describe our results more, before presenting the specifics. In Section 2, we describe the challenges in dynamically maintaining quantiles and present non-trivial adaptations of prior work. In Section 3 we present our algorithm in detail. In Section 4, we present experimental results. Finally, Section 5 has concluding remarks.

1.1 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*. 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.

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.[2] 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 $k\phi N$, $k = 1, \ldots, 1/\phi$. 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

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

for $k = 1, \ldots, 1/\phi$. The set of $j_1, \ldots, j_{1/\phi}$ will be the ϕ quantiles approximate up to $\pm\epsilon N$. (If $\epsilon = 0$, then we seek the exact quantiles.) Note that, for $\phi \neq 0$, the approximation above is in fact good to relative error $\left(1 \pm \frac{\epsilon}{\phi}\right)$; one can set $\epsilon = \phi\epsilon'$ to get the factor $(1 \pm \epsilon')$.

Our goal is to solve this problem using sublinear resources. It would be ideal of course to use space no more than $1/\phi$ 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

[1] Update operations of the form "change an attribute value of a specified record from its current value x to new value y" can be thought of as a delete followed by insert, for the purposes of our discussions here. Hence, we do not explicitly consider them hereafter.

[2] Formally, the attribute we focus on is the key for the relation.

dataset. 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 can not delete a tuple that is not present in the relation. A sequence of transactions is called *well-formed* if it leads to $\mathbf{A}_t \geq 0$; we will consider only well-formed sequences of transactions in this paper.

1.2 Previous Work

Since the early 70's, 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 [20].

A series of algorithms have been developed for finding quantiles in the incremental model. The idea presented in [19] leads to an algorithm that maintains an $O((\log^2 \epsilon N)/\epsilon)$ space data structure on $\mathbf{A}$ 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, 17] to be more efficient in practice, and improved in [14] 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, [19] had shown that any p-pass algorithm needs time $\Omega(N^{1/p})$ 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 at least $1-\delta$ for some pre-specified 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 [17, 18], $O(\epsilon^{-1}\log^2\epsilon^{-1} + \epsilon^{-1}\log^2\log\delta^{-1})$ 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, 8, 15] do not provide a priori guarantees; however, performance of their algorithms on various data sets has been experimentally investigated. Both [18] and [14] 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. An exception is the result of [11] on dynamic maintenance of equi-depth histograms which are ϕ-quantiles. 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) and provide a priori probabilistic guarantees on equi-depth histogram construction. Their algorithm works well for the case 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 [12, 24] where the emphasis was on maintaining the largest (significant) coefficients. In [13], an algorithm was provided for dynamically maintaining V-Opt histograms. No a priori guarantees for finding quantiles can be obtained by using these algorithms.

1.3 Our Main Algorithmic Result

We present a new algorithm for the problem of dynamically estimating quantiles. We maintain $O(\log^2|U|\log(\log(|U|)/\delta)/\epsilon^2)$ 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.

2 Challenges and Partial solutions

In this section, we provide some intuition into the problem of maintaining quantiles under inserts and deletes. We also adapt prior work on incremental quantile-finding algorithms to work in presence of deletes for comparison purposes.

2.1 Challenges

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.[3] A space-efficient algo-

[3] 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.

rithm knows very little about a dataset 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 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 dataset at every point in time, which is difficult if there is severe cancellation. For example, in [10], the author addresses the problem of sampling from the net data set after inserts and deletes and states that "*If substantial portion of the relation is deleted, it may be necessary to rescan the relation in order to preserve the accuracy of the guarantees.*"

2.2 Extending Previous Algorithms

Among the previously known algorithms, we consider the Greenwald-Khanna (GK) algorithm [14] in more detail since it provides the best known performance for the incremental case. It has other desirable properties: it uses small space and time, and does not rescan the database for approximate quantile generation. We will describe ways to modify the algorithm for the dynamic case. No a priori guarantees can be obtained in this manner; nevertheless, this provides a useful benchmark with which to compare our algorithm. Previously known algorithms that rely on backing or distinct samples can not be extended in this manner; as a result, rescanning of the database can not be avoided with those algorithms.

We consider the bounded-memory form of the GK algorithm. The algorithm first fills its memory with data points (values), as they arrive. When its memory is full and additional values arrive, the GK algorithm kicks out some point it has (possibly the newly-arrived point), and keeps a count of the number of data points between samples. It chooses the point to kick out carefully, to minimize the error in its answering procedure, which is to return the least sample point j such that $\sum_{i\leq j}\mathbf{A}[i]$ exceeds the desired rank. The algorithm further specifies a data structure to facilitate efficient updates and queries. The error of the algorithm depends on the memory available and the number of items in the dataset.

Although the GK algorithm is designed for incremental (insert-only) data, there are several ways to extend it to dynamic data.

Ignore deletions: One can simply ignore delete transactions. This will be a reasonable solution provided the rate of deletions is small compared with the rate of insertions.

Use two parallel GKs: One can use two instances of GK, one for insertions and one for deletions. For estimating each ϕ quantile, we search for two points i_1 and i_2 such that

- each of i_1 and i_2 appears as a sample point in one of the instances of GK.
- there is no sample point between i_1 and i_2 in either of the instances.
- Let i_1^{ins} be the greatest sample among insertions such that $i_1^{\mathrm{ins}}\leq i_1$, let i_1^{del} be the greatest sample among deletions such that $i_1^{\mathrm{del}}\leq i_1$, and similarly for i_2^{ins} and i_2^{del}. We have
$$\sum_{j<i_1^{\mathrm{ins}}}\mathbf{A}^{\mathrm{ins}}[j]-\sum_{j<i_1^{\mathrm{del}}}\mathbf{A}^{\mathrm{del}}[j]\leq\phi N\leq$$
$$\sum_{j<i_2^{\mathrm{ins}}}\mathbf{A}^{\mathrm{ins}}[j]-\sum_{j<i_2^{\mathrm{del}}}\mathbf{A}^{\mathrm{del}}[j],$$
where $\mathbf{A}^{\mathrm{ins}}$ and $\mathbf{A}^{\mathrm{del}}$ are incremental datasets of insertions and deletions, respectively.

Heuristically, one can then output i_1 or i_2 as an approximate answer.

There are several further ways to specify the above algorithm.

- One can *interpolate* between i_1 and i_2 by returning $i_1+(i_2-i_1)\Delta$, where
$$\begin{aligned}\Delta &= \left(\sum_{j<i_2^{\mathrm{ins}}}\mathbf{A}^{\mathrm{ins}}[j]-\sum_{j<i_2^{\mathrm{del}}}\mathbf{A}^{\mathrm{del}}[j]\right)\\ &- \left(\sum_{j<i_1^{\mathrm{ins}}}\mathbf{A}^{\mathrm{ins}}[j]-\sum_{j<i_1^{\mathrm{del}}}\mathbf{A}^{\mathrm{del}}[j]\right).\end{aligned}$$
- The pair (i_1,i_2) is not unique, in general. One can look for all such pairs, and combine the results for each, say, by taking a mean or median.

In our experimental section, we use the above-mentioned dual instances of the GK algorithm, called GK2 by us. We interpolated between i_1 and i_2 and took a mean over all pairs (i_1,i_2); this performed well.

3 Our 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 respectively. At the beginning, we will assume that U is known to the algorithm; later, we will remove this assumption.

3.1 High-Level View of Our Algorithm

In order to compute approximate ϕ quantiles we need of 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 I. (Dyadic intervals will be defined shortly.)

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 $\|\mathbf{A}_S\| = \sum_{i\in S}\mathbf{A}[i]$ denote the number of items with values in S. We keep $\|\mathbf{A}_S\|$ (a single number known as a *Random-Subset-Sum (RSS)*) for each of several random sets S. Observe that the expected value of $\|\mathbf{A}_S\|$ is $\frac{1}{2}\|\mathbf{A}\|$, since each point is in S with probability $\frac{1}{2}$.

For $\mathbf{A}[i]$, consider $E[\|\mathbf{A}_S\|\,|i \in S]$, which can be estimated by looking at counts $\|\mathbf{A}_S\|$, only for the S's that contain i (close to half of all S's, with high probability). One can show that this conditional expected value is $\mathbf{A}[i] + \frac{1}{2}\left\|\mathbf{A}_{U\setminus\{i\}}\right\|$, 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 $\|\mathbf{A}\|$, we can estimate $\mathbf{A}[i]$ as

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

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 dataset items on any dyadic interval in U (defined below), up to $\pm\epsilon N$, by repeating the procedure for each dyadic resolution level up to $\log|U|$. Of course, a set S in this case is a collection of dyadic intervals from the same level, each taken with probability 1/2. 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 dataset 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 $0,\ldots,|U|-1$ 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.

3.2 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. 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.

3.2.1 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 $\|\mathbf{A}_S\|$ be the count of values in the datasets that are projected onto S (formally, $\|\mathbf{A}_S\| = \sum_{i\in S}\mathbf{A}[i]$) . We repeat this process $num\_copies = 24\log(\log(|U|)/\delta)\log(|U|)/\epsilon^2$ times and get sets $S_1,\ldots,S_{num\_copies}$ (per level). The counts $\|\mathbf{A}_{S_l}\|$ for all sets that we have picked comprise our *Random Subset Sum* summary structure. In addition we store (and maintain) $\|\mathbf{A}\| = N$ 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). Then quickly determine those sets S_l that contain $I_{j,k}$. For each such set we increase count $\|\mathbf{A}_{S_l}\|$ by one. For deletions we simply decrease the counters. Notice that 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_l's, which are random subsets. The straightforward way would be to store them explicitly, perhaps as a bitmap. This would use space $O(|U|)$ which we can not 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 3-wise independent random variable construction shown below, since it works well with our dyadic construction.

We need a generator $G(s,i) = S_i$ that quickly outputs the i'th bit of the set S, given i and a short seed 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 1's and then all the columns, in order. So, for each resolution level j, there's a G of size $(j+1) \times 2^j$. Then $G(s,i)$ is the seed s of length $j+1$ dotted with the i'th column of G modulo 2, which is efficient to compute—note that the i'th 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.

3.2.2 Estimating Quantiles

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

Computing $\|\mathbf{A}_I\|$ for dyadic intervals I.

Recall that $\|\mathbf{A}_I\|$ is simply the number of values in the dataset that fall within the interval I.

Given dyadic $I_{j,k}$, we want an estimate $\left\|\mathbf{A}_{I_{j,k}}\right\|_\sim$ of $\left\|\mathbf{A}_{I_{j,k}}\right\|$. 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 $\left\|\mathbf{A}_{I_{j,k}}\right\|$ is $2\left\|\mathbf{A}_{S_l}\right\| - \|\mathbf{A}\|$ for $\mathbf{A}_{S_l}$ corresponding to a set S_l containing $I_{j,k}$.

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, $\|\mathbf{A}_I\| \le \|\mathbf{A}_I\|_\sim \le \|\mathbf{A}_I\| + \epsilon N$, by using a value of ϵ half as big as desired at top level, and adding $\frac{\epsilon}{2}N$ to each estimate.

Computing the quantiles.

We would like to estimate ϵ-approximate ϕ-quantiles. Recall that ϵ is fixed in advance. For $k = 1, \ldots, 1/\phi$ we want an j_k such that $\left\|\mathbf{A}_{[0,j_k)}\right\| = (k\phi \pm \epsilon)N$. (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 $\left\|\mathbf{A}_{[0,j_k)}\right\|_\sim < k\phi N \le \left\|\mathbf{A}_{[0,j_k+1)}\right\|_\sim$, and return j_k. Repeat for all values of k.

We call the entire algorithm for maintenance of quantiles the `RSS` algorithm.

3.2.3 Analysis of Our Algorithm

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

Lemma 1 *Our 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 $I_{j,k} = I_k$ 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. Let X_k be a random variable defined by

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

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),

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

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

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

Each prefix I is the disjoint union of $r \le \log|U|$ 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

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

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,

$$\operatorname{var}[Y|\forall j\ I_{k_j} \in S_j] \le \log|U|\,\|\mathbf{A}\|^2.$$

It follows that, if we let Z be the average of $8\log|U|/\epsilon^2)$ repetitions of X, 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, $|Z - \|\mathbf{A}_I\|\,| < \epsilon N$ 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, $|Z - \|\mathbf{A}_I\|\,| < \epsilon N$ with probability at least $1 - \delta/\log|U|$. Both Chebyshev and Chernoff inequalities can be found in [5], and averaging arguments similar to above can be found in for example [4].

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

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

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 *Our* 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 is 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 $\log(t/\phi)$.

3.2.4 Extension to when U is Unknown

Above we assumed that the universe U 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 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 $i < u^2$. We then keep statistics for the universe $[0, u-1]$ and $[0, u^2-1]$, directing all insertions and deletions with value in $[0, u-1]$ to the original instance of RSS and all insertions and deletions with value in $[u, u^2-1]$ to the new instance. In general, we may need to square the size of the universe repeatedly, upon seeing a sequence of insertions with growing values or even a single insertion with very large value. We thus get several instances of RSS; each but the first extends the previous ones. The number of items in each dyadic interval can be estimated by consulting a single instance of RSS. Thus we have specified how to perform updates and queries; 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 largest instance on a universe of size at 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 dataset 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 j'th 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.

3.3 Some Observations on Our Algorithm

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

- Previous (incremental) algorithms for quantiles can guarantee always to return a value in the input dataset, whereas our algorithm may return a quantile value which was not seen in the input. This does not appear to have severe implications in various applications of quantiles. In general, in the face of severe cancellation, an algorithm with less than N space cannot keep track of which items are currently in the dataset.

- The distribution on values returned by our algorithm depends only on the dataset active at the time of the query. Thus, one can change the order of insertions and deletions without affecting results. This contrasts with previously known algorithms for finding quantiles where the order of inserts impacts the approximate quantiles output by the algorithm.

- 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 pre-agreed common random subsets. The subset sums for the combined dataset 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. 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. Results can be composed at the end.

4 Experiments

4.1 Datasets, Algorithms Implementation

To ratify our performance claims, we present an extensive set of experiments, with synthetic and real data sets. The synthetic data that we used are described in Table 1.

4.2 Performance of Our Algorithm

Each dataset is used to generate a population of size N, drawn from the range $[0 \ldots U-1]$. We use this data to compare the following algorithms:

- `Naive[`ℓ`]`: This is a simple algorithm that maintains exact counts on all dyadic intervals $I_{\ell,0}, \ldots, I_{\ell,2^\ell-1}$ at level $j = \ell$ and uses them to compute quantiles. Estimates for intervals below level ℓ are zero. The purpose of presenting the performance of this algorithm is twofold. First, it allows as to verify the performance of the `RSS[`ℓ`]` (see below) implementation that maintains exact counts at level ℓ and random sums below that level. Second, the performance of `Naive` is an indication of the hardness of the data for computing quantiles. E.g. `Naive` will do well if the quantiles are fairly wide-spread.

- `RSS[`ℓ`]`: This is a implementation of a variation of our `RSS` 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 $\|\mathbf{A}_I\|$ for any I dyadic at that level, in time $O(1)$. In fact, 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[`ℓ`]`. In our implementation, dyadic sums at level ℓ are stored explicitly. A short cut that we implemented is that we store the random sets below level ℓ using bitmaps instead of using random seeds. This does not affect the quality of the results presented here. The space requirements are computed as if random seeds were used.

- `GK`: This is an implementation of the Greenwald-Khanna algorithm.

- `GK2`: This uses two GK instances, one for insertions and one for deletions and interpolates to estimate the quantiles as described in Section 2.2.

The rest of this section is organized as follows. Subsection 4.2 presents a study on the performance of `RSS` on synthetic data. Subsection 4.3 compares our algorithm against the other competitors for datasets that include both insertions and deletions. Finally, subsection 4.4 compares all algorithms using Call Detail Data from AT&Ts telephony network and demonstrates the effectiveness of `RSS` for computing quantiles on large, dynamic datasets.

For these experiments we evaluate `RSS` for computing quantiles for datasets of various sizes and distributions. `Naive` gives us a measure of the hardness of computing quantiles for these datasets. The universe size in all experiments was $U = 2^{20}$. In all cases we compute 15 quantiles at positions $k\frac{1}{16}$ for $k = 1, 2, \ldots, 15$ (e.g. median is for $k = 8$). The footprint of the `RSS` algorithm was 11K in all runs. All numbers are averages over four runs with different seeds/data values.

Table 2 summarizes our results for Zipf distributed data varying N. The large errors reported by `Naive` for small values of k are because most of the mass of the Zipf distribution is concentrated on the left size of the domain, with 0 being the most popular item. As a result, small errors in identifying the correct quantiles near zero result in large errors for these quantiles. As expected, the errors for `RSS` seem to be independent of the population size N for a fixed domain.

The results for a Normal distribution of the data are tabulated in Table 3. This time `Naive` is a more serious competitor and sometimes surpasses `RSS`, especially for the "easy" quantiles (14/16,15/16). Since `RSS` stores the same sets as `Naive` for level 7, the relative success of `Naive` is due to the variance introduced by the random sets below that level. A bigger footprint for `RSS` closes the gap for these cases (results are omitted due to space limitations).

4.3 Comparison for Mix of Inserts/Deletes

We now investigate the performance of the algorithms when both insertions and deletions are present. We model the following scenario: we insert $N = 104,858$ items drawn uniformly from distribution $D1 = \text{Uni}[1, U]$, $U = 2^{20}$. Then we super-impose a second compact distribution $D2 = \text{Uni}[U/2 - U/32, U/2 + U/32]$ with αN values. Finally, all values from the first distribution are deleted so that the remaining values all come from $D2$. Parameter α controls the mass of the second distribution with respect to the mass of $D1$. All algorithms were set up to use 11KB of memory for their data structures.

Table 4 summarizes the average error over all quantiles (for $k = 1, \ldots, 15$) and 4 repetitions of the experiment. For `GK` we simply ignore deletions. Performance of RSS does not depend of the volume of the data ($(1+\alpha)N$ after all insertions, $\alpha * N$ in the end). When the mass of $D2$ is much larger that the mass of the initial distribution, even GK performs well as insertions/deletions (from $D1$) do not significantly affect the final picture. However, when the values of $D2$ become indistinguishable within the mass of $D1 + D2$, `RSS` is the clear winner with average error about 10 times less the errors of `GK`, `GK2`. In these cases, even `Naive` is better that both of them.

Hence, under severe cancellations, summarizing the

Dataset	Description
Uni[A,B]	Uniform data within range [A...B]
Zipf	Zipf distributed values
Normal[m,v]	Normal distribution with *mean*=m and *variance*=v

Table 1: Synthetic Datasets

	N=10,485		N=1,048,576		N=10,485,760	
k	`Naive[7]`	`RSS[7]`	`Naive[7]`	`RSS[7]`	`Naive[7]`	`RSS[7]`
1	0.7526	0.0818	0.9753	0.0272	0.9812	0.0090
2	0.7526	0.0818	0.9753	0.0272	0.9812	0.0090
3	0.7526	0.0001	0.9753	0.0000	0.9812	0.0000
4	0.5281	0.1134	0.7031	0.1490	0.8731	0.0496
5	0.4154	0.0000	0.5545	0.2222	0.6014	0.0740
6	0.4154	0.0000	0.5545	0.2222	0.6014	0.0740
7	0.4154	0.0000	0.5545	0.0000	0.6014	0.0000
8	0.4153	0.0000	0.5545	0.0000	0.6014	0.0000
9	0.4154	0.0000	0.5545	0.0000	0.6014	0.0000
10	0.2932	0.0000	0.3888	0.0833	0.4211	0.0277
11	0.2932	0.0000	0.3888	0.0416	0.4211	0.0138
12	0.2265	0.0221	0.3001	0.0296	0.3249	0.0459
13	0.1754	0.0142	0.2413	0.0186	0.2637	0.0157
14	0.1154	0.0312	0.1562	0.0104	0.1700	0.0320
15	0.0595	0.0272	0.0803	0.0289	0.0873	0.0208

Table 2: Errors for Zipf data

Universe (as `Naive` and `RSS` do) seems to be the only viable approach.

4.4 Performance with Real Data Sets

For this experiments we used Call Detail Records (CDRs) that describe usage of a small population from AT&T's customers. 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.

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 dataset of 4.42 million CDRs covering a period of 18 hours. All algorithms were set up to use 4KB of space (ℓ was 6 for `Naive` and `RSS`). Figure 1 shows the number of active calls over time, while Figure 2 plots the error in computing the median of the ongoing calls (in resolution of 1 sec) over time (we probed for estimates every 10,000 records). We do not include `Naive` in this figure for clarity. `RSS` seems to introduce sporadic large variations in the error of the reported median, especially when lots of deletions are happening (e.g. near the end of the run). Our interpolation used for $GK2$ does not seem to work in this case. As more data is processed, `RSS` shows a clear advantage over both `GK` and `GK2`.

This dataset stretches performance on all algorithms. As phone calls end and new calls are initiated, the mass of the distribution is smoothly shifted

	N=10,485		N=1,048,576		N=10,485,760	
k	Naive[7]	RSS[7]	Naive[7]	RSS[7]	Naive[7]	RSS[7]
1	0.0579	0.0181	0.0775	0.0361	0.0839	0.0320
2	0.0920	0.0435	0.1231	0.0292	0.1333	0.0310
3	0.0295	0.0295	0.0397	0.0270	0.0430	0.0311
4	0.0973	0.0438	0.1306	0.0457	0.1415	0.0842
5	0.0349	0.0288	0.0473	0.0244	0.0512	0.0347
6	0.1219	0.0284	0.1655	0.0254	0.1801	0.0531
7	0.0594	0.0268	0.0821	0.0202	0.0898	0.0309
8	0.1510	0.0213	0.1518	0.0079	0.1518	0.0163
9	0.0885	0.0093	0.1190	0.0206	0.1291	0.0396
10	0.0259	0.0118	0.0357	0.0258	0.0388	0.0362
11	0.0946	0.0042	0.1265	0.0356	0.1373	0.0147
12	0.0322	0.0120	0.0432	0.0143	0.0471	0.0195
13	0.0660	0.0214	0.0886	0.0134	0.0964	0.0412
14	0.0035	0.0082	0.0053	0.0459	0.0061	0.0197
15	0.0027	0.0220	0.0042	0.0216	0.0048	0.0100

Table 3: Errors for Normal[U/2,U/50] distribution

α	Naive[7]	RSS[7]	GK	GK2
10.00	0.1196	0.0362	0.0211	0.0009
1.00	0.1145	0.0384	0.1203	0.0036
0.10	0.1018	0.0464	0.2194	0.0204
0.01	0.1097	0.0304	0.2203	0.2742

Table 4: Errors for insertions and deletions

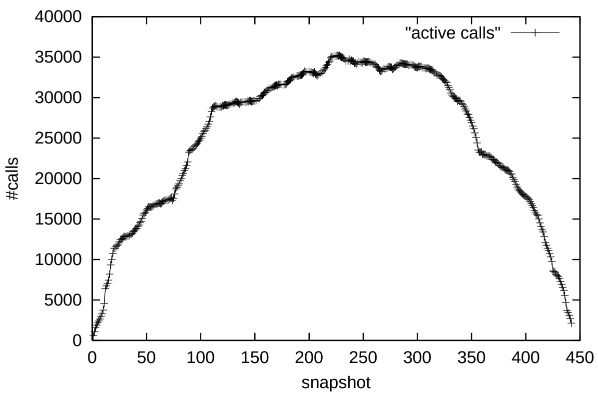

Figure 1: Number of active phone calls over time

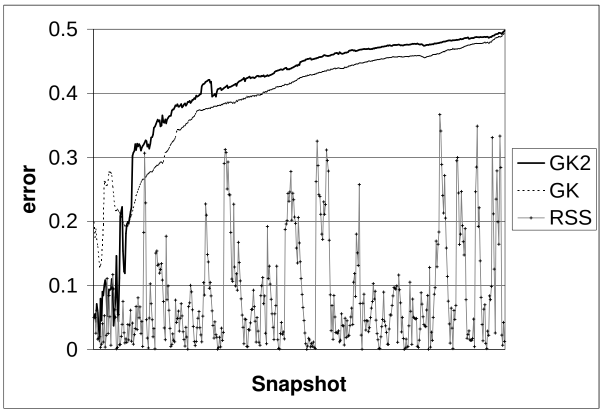

Figure 2: Error in computation of median over time

to larger values (t_is). We can think of the array as a queue with values inserted from the front (at $t_i = present()$) but deletions may happen anywhere depending on the length of the call.

The actual median length of the calls that were still active when the last median computation was done (last snapshot) was 4 minutes. RSS reported 3.7 minutes, while GK and GK2 168.8 and 206.7 minutes respectively. The error of these estimates is 0.0126 for RSS[6] (e.g. the reported quantile was 0.5 ± 0.0126), 0.4944 for GK and 0.4977 for GK2. All estimates but for RSS indicate a length that is beyond 99% of all active calls, thus they are completely inaccurate. This is to be attributed to the dynamics of the data.

In Figure 3, we plot the average error over the last 50 snapshots of the data for all $k\phi$ quantiles for $\phi = 0.10$ (deciles). For example the 0.90-quantile denotes a time period (from present going backwards) that includes 90% of all active calls (e.g., the sort order is reversed based on t_is.). Quantile computation is hard for small quantiles as the data is clustered around t_i=$present()$ and this is depicted in the errors of GK and GK2. The errors for Naive are increased around the median. This is an indication that most data calls are voice calls with length around 4 minutes. Since Naive uses a fixed resolution level it can not distinguish among them at places with high data density

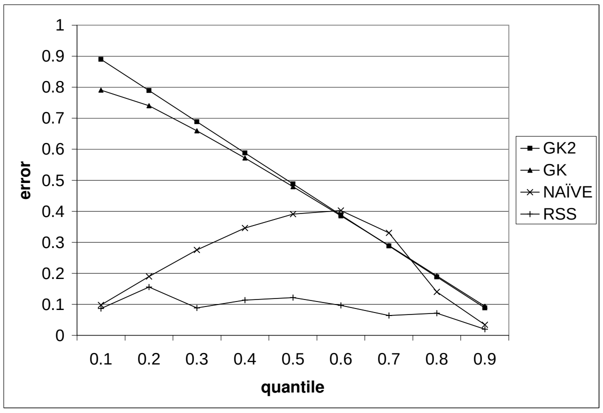

Figure 3: Average error over the last 50 snapshots for all ϕ=0.1 quantiles

(e.g. around the median). To our knowledge RSS seems the only viable choice for these computations.

5 Conclusions

We have presented an algorithm for maintaining dynamic quantiles of a relation in 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.

We believe our techniques are unique for handling massive dynamic datasets. Drawing from the property that we summarize the universe instead of a snapshot of the dataset, RSSs can handle dramatic changes or shifts in the data distribution as is demonstrated from our experiments with real datasets.

Acknowledgments

We would like to thank David Poole for providing us the call detail dataset and the anonymous reviewers for their helpful comments and suggestions.

References

[1] R. Agrawal, T. Imielinski and A. Swami. Mining Associations between Sets of Items in Massive Databases. In *Proc. of ACM SIGMOD*, pages 207–216, Washington D.C, May 1993.

[2] R. Agrawal and R. Srikant. Mining Quantitative Association Rules in Large Relational Tables. In *Proceedings of ACM SIGMOD*, pages 1–12, Montreal Canada, June 1996.

[3] R. Agrawal and A. Swami. A One-Pass Space-Ecient Algorithm for Finding Quantiles. In *Proceedings of COMAD*, Pune, India, 1995.

[4] N. Alon, Y. Matias, M. Szegedy. The Space Complexity of Approximating the Frequency Moments. JCSS 58(1): 137–147 (1999).

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

[6] K. Alsabti, S. Ranka and V. Singh. A One-Pass Algorithm for Accurately Estimating Quantiles for Disk-Resident Data. In *Proceedings of VLDB*, pages 346–355, Athens, Greece, 1997.

[7] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan. Time Bounds for Selection. *JCSS* 7(4): 448–461, 1973.

[8] F. Chen, D. Lambert and J. C. Pinheiro. Incremental Quantile Estimation for Massive Tracking. In *Proceedings of KDD*, pages 516–522, Boston, August 2000.

[9] D. J. DeWitt, J. F. Naughton and D. A. Schneider. Parallel Sorting on a Shared-Nothing Architecture using Probabilistic Splitting. In *PDIS*, pages 280–291, 1991.

[10] P. B. Gibbons. Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. In *Proc of VLDB*, pages 541–550, Rome, Italy, 2001

[11] P. Gibbons, Y. Matias and V.Poosala. Fast Incremental Maintenance of Approximate Histograms. In *Proceedings of VLDB*, pages 466–475, Athens, Greece, 1997.

[12] A. C. Gilbert and Y. Kotidis and S. Muthukrishnan and M. J. Strauss Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries In *Proc. of VLDB*, 2001.

[13] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, Small-Space Algorithms for Approximate Histogram Maintenance. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, Montréal, Québec, Canada, May 2002.

[14] M. Greenwald and Sanjeev Khanna. Space-Efficient Online Computation of Quantile Summaries. In *Proceedings of ACM SIGMOD*, pages 58–66, Santa Barbara, California, May 2001.

[15] R. Jain and I. Chlamtac. The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms Without Storing Observations. In *Communications of the ACM*, 28(10):1076–1085, October 1985.

[16] T. Johnson, S. Muthukrishnan, P. Dasu and V. Shkapenyuk. Mining Database Structure; Or, How to Build a Data Quality Browser. In *Proc. of ACM SIGMOD*, to appear, 2002.

[17] G.S. Manku, S. Rajagopalan, B.G. Lindsay. Approximate Medians and other Quantiles in One Pass and with Limited Memory. In *Proc of ACM SIGMOD*, pages 426–435, Seattle, WA, 1998.

[18] G.S. Manku, S. Rajagopalan, B.G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets In *Proc of ACM SIGMOD*, 1999.

[19] J. I. Munro and M. S. Paterson. Selection and Sorting with Limited Storage. In *TCS* 12, 1980.

[20] M. S. Paterson. Progress in Selection. Technical Report, University of Warwick, Coventry, UK, 1997.

[21] V. Poosala. *Histogram-Based Estimation Techniques in Database Systems.* Ph. D. dissertation, University of Wisconsin-Madion, 1997.

[22] V. Poosala and Y. Ioannidis. Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balancing In *Proceedings of VLDB*, pages 448–459, 1996.

[23] V. Poosala, Y. E. Ioannidis, P. J. Haas and E. J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. In *Proc of ACM SIGMOD*, pages 294–305, 1996.

[24] Y. Matias, J. Vitter and M. Wang. Dynamic Maintenance of Wavelet-based Histograms. In *Proceedings of VLDB*, pages 101–110, Cairo, Egypt, Sept. 2000.