Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets

Gurmeet Singh Manku
IBM Almaden Research Center
manku@almaden.ibm.com

Sridhar Rajagopalan
IBM Almaden Research Center
sridhar@almaden.ibm.com

Bruce G. Lindsay
IBM Almaden Research Center
bruce@almaden.ibm.com

Abstract

In a recent paper [MRL98], we had described a general framework for single pass approximate quantile finding algorithms. This framework included several known algorithms as special cases. We had identified a new algorithm, within the framework, which had a significantly smaller requirement for main memory than other known algorithms. In this paper, we address two issues left open in our earlier paper.

First, all known and space efficient algorithms for approximate quantile finding require advance knowledge of the length of the input sequence. Many important database applications employing quantiles cannot provide this information. In this paper, we present a novel non-uniform random sampling scheme and an extension of our framework. Together, they form the basis of a new algorithm which computes approximate quantiles without knowing the input sequence length.

Second, if the desired quantile is an extreme value (e.g., within the top 1% of the elements), the space requirements of currently known algorithms are overly pessimistic. We provide a simple algorithm which estimates extreme values using less space than required by the earlier more general technique for computing all quantiles. Our principal observation here is that random sampling is quantifiably better when estimating extreme values than is the case with the median.

1 Introduction

This article continues our study [MRL98] of the problem of computing quantiles of large sequences of online or disk-resident datasets in a single pass while using as little main memory as possible. We will denote the length of the input sequence by N. For $\phi \in (0, 1]$, the element in position $\lceil \phi N \rceil$ in the sorted sequence of the input is said to be the ϕ-quantile. The quantile corresponding to $\phi = 0.5$ has a special name, the *median*. For $\epsilon \in [0, 1]$, an element belonging to the input sequence is said to be an ϵ-approximate ϕ-quantile if its rank is between $\lceil (\phi - \epsilon)N \rceil$ and $\lceil (\phi + \epsilon)N \rceil$. Obviously, we assume that $\epsilon \leq \phi$ and $\epsilon \leq (1 - \phi)$.

SIGMOD '99 Philadelphia PA

1.1 Database Applications

Quantiles characterize distributions of real world data sets and are less sensitive to outliers than the moments (mean and variance). They can be used by business intelligence applications to distill summary information from huge data sets.

Quantiles are used by query optimizers to provide selectivity estimates for simple predicates on table values [SALP79]. Equi-depth histograms [PIHS96] are simply $\frac{i}{p}$-quantiles, for $i \in \{1, 2, \ldots, p-1\}$, computed over column values of database tables for a suitable p.

Splitters are used in parallel database systems, such as DB2 and Informix [Inf, DB2] for value range data partitioning. They are also used in distributed sorting to assign data elements to processors [DNS91].

Approximate quantiles can be substituted for exact quantiles in all the applications just described. In practice, it is acceptable if the difference in rank between the true and the approximate quantile is guaranteed to be less than 1% of the total number of data elements.

Probabilistic guarantees on the correctness of the output are also acceptable in practice as long as such guarantees are very close to 100%. For example, a set of splitters dividing a very large data set of size N into 100 approximately equal parts is acceptable if, with probability at least 99.99%, the rank of each splitter is guaranteed to be no more than $0.001N$ elements away from that of the corresponding exact splitter.

Extreme quantiles are often of much interest in real world datasets. Extreme values characterize outliers and represent skew in the data. For instance, the 95^{th}quantile in a quarterly sales table for all franchises of a company is useful to compute.

1.2 Motivation for Unknown N

If one views quantile computation as an aggregation operator in relational databases, the input might be an intermediate table for whose size, at best, a crude estimate from the query optimizer is available. Approximate quantiles can also be used for maintaining equi-depth histograms of a dynamically growing table. Such a histogram should be accurate at all times irrespective of the current size of the table.

1.3 Challenges to Meet

The efficiency and the correctness of the algorithm should be *data independent*. It should not be influenced by the arrival distribution or the value distribution of the input. It should not require *a priori* knowledge of the size of the data set. The algorithm should provide explicit and tunable performance guarantees. Its performance should degrade gracefully as the approximation guarantee desired is made tighter.

The algorithm should require only a *single pass* over the data. Multiple passes over large data sets not only degrade performance but also are incompatible with most DBMS GROUP BY implementations. The main memory requirements of the algorithm should be as small as possible. Main memory is of concern when histograms over multiple columns of a table are to be computed simultaneously. GROUP BY algorithms also compute multiple aggregation results concurrently, further increasing the desirability of a small and predictable memory footprint. The algorithm should be simple to understand, parallelizable, and should scale well on SMP and MPP configurations.

1.4 Our Contributions

In our previous article [MRL98], we developed a general framework for identifying approximate quantiles of large data sets in a single pass using little main memory. All algorithms which fit in this framework, including the one proposed in [MRL98] and other previously known ones [ARS97, MP80, AS95], required that N, the size of the input sequence, be known in advance. In this article, we build upon our previous work by augmenting our framework and devising a novel non-uniform sampling technique. The resulting algorithm solves the approximate quantile finding problem without requiring advance knowledge of N. The new algorithm does not entail a significant main memory overhead when compared with algorithms that know N.

We also propose a simple strategy which requires significantly less memory when ϕ is tiny (or large) and close to 0 (or 1). The algorithm makes use of a simple computational fact: the extreme values of random samples can be computed using less space than medians, and an interesting statistical fact: the rank distribution of an extreme value of a random sample is more tightly clustered around its expected rank than is the case with quantiles close to the median. This allows us to improve upon space requirements for extreme value computation significantly.

1.5 Related Work and Connections

Absence of *a priori* knowledge of N enforces an online view of the problem. Essentially, the algorithm is required to have available an estimate of the quantiles for any prefix of the input sequence. Clearly, it could be employed as an online aggregation operator [Hel97], thereby providing more controllable and user friendly user interfaces.

Synopsis data structures, a term coined by Gibbons and Matias [GM99, GM98], summarize the information content of massive data sets. The synopsis has a memory footprint substantially smaller than the base data set. It is designed to support fast computation of approximate answers to a limited set of queries. One example of a synopsis data structure is a set of approximate histograms over several combinations of column values of a table.

Gibbons, Matias and Poosala [GMP97] propose an algorithm for computing approximate quantiles that satisfy a different error metric. The algorithm dynamically adjusts a set of bucket boundaries on the fly, possibly requiring more than one pass over the data set. Chaudhuri, Motwani and Narsayya [CMN98] also present an approximate quantile finding algorithm that employs block sampling. Their error metrics differ from ours and the algorithm can possibly require multiple passes.

2 Antecedents

The theory literature has focused on discovering bounds on the number of *comparisons* needed to find exact quantiles of datasets. The celebrated paper by Blum, Floyd, Pratt, Rivest and Tarjan [BFP+73] shows that any quantile of a data set of size N can be computed with at most $5.43N$ comparisons. The paper also establishes a lower bound of $1.5N$ comparisons for the problem. For an account of progress since then, see the survey by Mike Paterson [Pat97].

Frances Yao [Yao74] showed that any deterministic algorithm that computes an *approximate* quantile requires $\Omega(N)$ comparisons. Curiously, this lower bound is easily beaten by employing randomization. The folklore algorithm that outputs the median of a random sample of size $O(\epsilon^{-2} \log \delta^{-1})$ requires a number of comparisons that is independent of N. For a comprehensive survey of this aspect of the literature, see the survey by Paterson [Pat97].

2.1 Single Pass Algorithms

Quantile finding algorithms that require only a single pass over the data were first studied by Ira Pohl [Poh69] who established that any deterministic algorithm that computes the exact median of N elements in one pass needs to store at least $N/2$ data elements. Munro and Paterson [MP80] proved a general result that memory to store $\Theta(N^{\frac{1}{p}})$ elements is necessary and sufficient to compute the exact median of N elements in p passes.

When N is large, computation of *exact* quantiles in a single pass is impractical due to the incredibly large main memory requirement. This motivates a search for algorithms that compute *approximate* quantiles. The ideas in the paper by Munro and Paterson [MP80] can be used to construct a single pass algorithm that computes ϵ-approximate quantiles of N elements in $O(\epsilon^{-1}\log^2(\epsilon N))$ space.

In our previous paper [MRL98], we presented a general framework for computing approximate quantiles in a single pass, that includes two previously known algorithms, one by Munro and Paterson [MP80] and another by Alsabti, Ranka and Singh [ARS97], as special cases. We also described a third algorithm, also within the framework, which significantly improved upon the main memory requirements with respect to the earlier algorithms. Theoretically though, the space complexity of that algorithm is $O(\epsilon^{-1}\log^2(\epsilon N))$, the same as that of Munro and Paterson's algorithm. We also presented a very simple randomized algorithm that achieves further reduction in space at the cost of probabilistic guarantees on the correctness of the output. The randomized algorithm requires only $O(\epsilon^{-1}\log^2(\epsilon^{-1}\log^2\log\delta^{-1}))$ space, where δ is the probability that the algorithm fails to produce a quantile within the promised approximation guarantee. Note that the space complexity is independent of N.

The principal drawback is that *all the above algorithms require that N be known in advance.*

2.2 Unknown N Algorithms

A very simple sampling scheme called *reservoir sampling* [Vit85] generates a sample of size s without advance knowledge of N. Folklore analysis shows that if the sample has size $O(\epsilon^{-2}\log\delta^{-1})$, the ϕ-quantile of the sample is an ϵ-approximate quantile of the input dataset with probability at least $1-\delta$. The quadratic dependence of s on ϵ^{-1} makes the scheme impractical for small values of ϵ because the entire sample has to be stored in main memory. In comparison, if N is known, a random sampling algorithm we proposed in [MRL98] requires only $O(\epsilon^{-1}\log^2(\epsilon^{-1}\log^2\log\delta^{-1}))$ space. This motivates the search for a more sophisticated sampling technique which works without knowing N but requires little space for small values of ϵ. This paper presents one such scheme.

The random sampling scheme we present is non-uniform, i.e., the probability that an element of the input is included in the sample is not the same for all elements. Elements that are early in the sequence are included with larger probability than those that arrive later. This non-uniformity improves the space overhead required by the reservoir methods to levels comparable to the best known algorithms that know N. The principal challenges are the algorithmic and analysis issues associated with non-uniform sampling. We feel that the methods used in this paper might generate interest in employing more sophisticated sampling methods for solving other problems.

Problem 1 *Given ϕ, ϵ and δ, design a single pass algorithm that computes, with probability at least $1-\delta$, an ϵ-approximate ϕ-quantile of an input sequence using as little main memory as possible, without knowing the length of the sequence in advance.*

2.3 Extreme values

When $\phi \leq \epsilon$, the smallest element in the input sequence, denoted by MIN, is an ϵ-approximate ϕ quantile. Computing MIN requires only $O(1)$ space. This motivates the following questions: Is MIN a special case? Do we require significantly larger amounts of memory when ϕ is slightly larger than ϵ, a situation in which MIN does not qualify? The expected rank of the MIN has nothing to do with ϕ. Is there an estimator whose expected rank is ϕN and which can be computed using very little space and reliably?

We will answer both of these questions (see section 7) by providing a sampling method and an estimator which has an expected rank of ϕN and which works when as long as ϕ and ϵ are both small and not necessarily exactly the same.

Problem 2 *Given ϕ close to 0, ϵ and δ, design a single pass algorithm that computes, with probability at least $1-\delta$, an ϵ-approximate ϕ-quantile of a data set of arbitrary size using as little main memory as possible.*

3 The Unknown N Algorithm

The algorithm is parameterized by three integers b, k and h. It uses b buffers each of which can store k elements. We will later compute values of b, k and h as functions of ϵ and δ. Buffers are always labeled empty, partial or full. Initially, all b buffers are labeled empty. With each buffer X, we also associate a positive integer $w(X)$, which denotes its *weight*. Various algorithms can be composed from an interleaved sequence of three operations on buffers: NEW, COLLAPSE and OUTPUT.

3.1 NEW Operation

NEW takes as input an empty buffer and an integer r that represents the sampling rate. It is invoked only

if there is an empty buffer and at least one outstanding element in the input sequence. The operation simply populates the buffer by choosing a a single random element from a block of r input elements each. It repeats this operation over k successive blocks of r elements each. Thus it consumes exactly rk input elements, choosing k of them for populating the buffer. The NEW operation returns the buffer after assigning it weight r. Further, if the buffer was not completely filled because there were less than rk elements left in the input stream, the buffer is marked partial. In the normal course, the buffer is marked full.

Notice that choosing $r = 1$ amounts to no sampling. If r is larger then sampling is introduced. The larger r is the sparser the sample. The algorithm will dynamically change the value of r during execution leading to a variable rate of sampling.

3.2 COLLAPSE Operation

COLLAPSE takes $c \geq 2$ full input buffers, denoted by $X_1, X_2, \ldots, X_c$, and outputs a buffer, Y, each of size k. In the end, all but one input buffer is marked empty. The output Y is stored in the buffer that is marked full. Thus, Y is logically different from $X_1, X_2, \ldots, X_c$ but physically occupies space corresponding to one of them.

The weight of the output buffer $w(Y)$ is the sum of weights of input buffers, $\sum_{i=1}^{c} w(X_i)$. We now describe the elements stored in Y. Consider making $w(X_i)$ copies of each element in X_i and sorting all the input buffers together, taking into account the multiple copies. The elements in Y are simply k equally spaced elements in this (sorted) sequence. If $w(Y)$ is odd, these k elements are in positions $jw(Y) + \frac{w(Y)+1}{2}$, for $j = 0, 1, \ldots, k-1$. If $w(Y)$ is even, we have two choices: We could either choose elements in positions $jw(Y) + \frac{w(Y)}{2}$ or those in positions $jw(Y) + \frac{w(Y)+2}{2}$, for $j = 0, 1, \ldots, k-1$. The COLLAPSE operator alternates between these two choices on successive invocations with even $w(Y)$.

It is easy to see that multiple copies of elements need not actually be materialized. COLLAPSE can be performed *in situ*; no additional space is required. In the end, only the buffer that stores the output is labeled full, the rest are labeled empty.

3.3 OUTPUT Operation

OUTPUT is performed exactly once, just before termination. It takes $c \geq 2$ input buffers, $X_1, X_2, \ldots, X_c$. All buffers are full with the exception of the last one which might be partial. Similar to COLLAPSE, this operator makes $w(X_i)$ copies of each element in X_i and sorts all the input buffers together, taking the multiple copies into account. The element at position $\lceil \phi(kw(X_1) + kw(X_2) + \cdots + kw(X_{c-1}) + k_c w(X_c)) \rceil$ is output, where k_c denotes the size of the last buffer.

3.4 Definition of Weighted ϕ-quantile

As shown in Figure 1, the algorithm can be looked upon as being composed of two stages. The first stage accepts the input sequence and invokes successive NEW operations. The output of the first stage feeds the second stage which runs the deterministic algorithm. The assignment of weights to NEW buffers effectively feeds a *weighted sequence* of elements to the second stage where the weight of an element is the weight of the NEW buffer it lies in. The *weighted ϕ-quantile* of such a weighted sequence is defined in a natural way as follows. Imagine taking all NEW buffers together and making as many copies of each element as the weight of the NEW buffer it lies in. The ϕ-quantile of this set of copies is the weighted ϕ-quantile of the sequence.

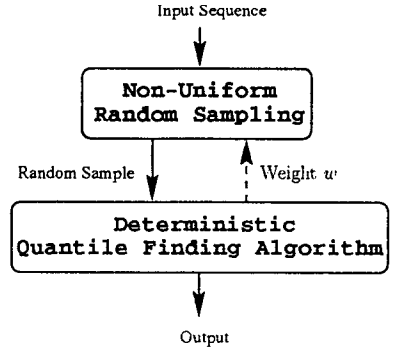

Figure 1: **The Big Picture.**

3.5 A Tree Representation

An algorithm for computing approximate quantiles consists of a series of invocations of NEW and COLLAPSE, terminating with OUTPUT. NEW populates empty buffers with input. COLLAPSE reclaims buffer space by collapsing a chosen subset of full buffers. OUTPUT is invoked on the final set of buffers. The sequence of operations carried out by such an algorithm can be represented by a tree. The vertex set of the tree (except the root) is the set of all the (logical) buffers (initial, intermediate or final) produced during the computation. Clearly, there could be many more of these than b, the number of physical buffers used by the algorithm. The leaves of the tree correspond to initial buffers that get populated from the incoming data stream. An edge is drawn from each input buffer to the output buffer of a COLLAPSE operation. The root corresponds to the final OUTPUT operation. The children of the root are the final buffers produced. We draw broken edges from the children to the root.

See Figure 2 depicting one such tree possible with $b = 5$ buffers, where each NEW has been invoked with

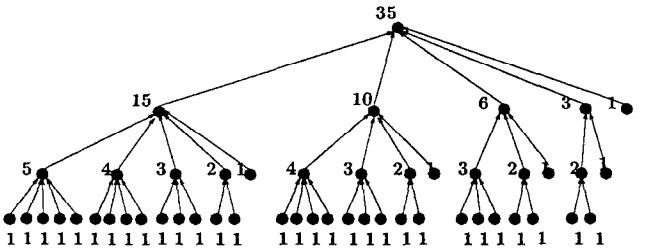

Figure 2: **A tree with $b = 5$ buffers when each NEW is invoked with sampling rate $r = 1$. Each node is labeled with its weight.**

sampling rate 1. The labels of nodes represent their weights. Leaves get populated left to right.

3.6 COLLAPSE Policy

The algorithm manages $b \geq 2$ buffers each of size k. With each full buffer, we also associate an integer value to denote its *level* in the tree. NEW buffers are assigned level zero when the sampling rate r is one. During the course of the algorithm, if there is an outstanding element in the input sequence, we check whether there are any empty buffers. If so, we invoke NEW (the determination of sampling rate r will be more fully explained in Section 3.7). If there are no empty buffers, we have no choice but to reclaim space by invoking COLLAPSE on some buffers. The question then is: Which subset of full buffers do we COLLAPSE? We now describe our choice.

NEW buffers are assigned level zero (until the onset of sampling). Let ℓ be the smallest level of any full buffer. If there is exactly one buffer at level ℓ, we increment its level until there are at least two at the lowest level. COLLAPSE is invoked on the set of buffers at level ℓ. The output of COLLAPSE is assigned level $\ell + 1$. See Figure 2 for the tree formed with $b = 5$ buffers using the buffer management scheme just described.

3.7 Non-uniform Sampling

The height of the tree increases by one whenever COLLAPSE is invoked on the entire set of b buffers. Creation of the first buffer at level h marks the onset of sampling. Thereafter, NEW is invoked with sampling rate $r = 2$ and each NEW buffer is assigned level 1. This continues until the height of the tree increases further to $h + 1$. At this point, the sampling rate is halved, i.e., NEW is invoked with $r = 4$ and NEW buffers are assigned level 2. In general, whenever the first buffer at height $h + i$ is produced for $i \geq 0$, the sampling rate is halved and subsequent NEW operations are invoked with rate $r = 2^{i+1}$, the NEW buffer being assigned level $i + 1$. See Figure 3 for the tree formed by the buffer management policy just described.

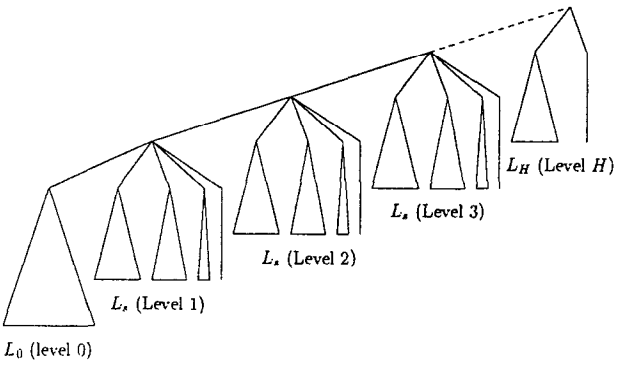

Figure 3: **The tree for computing a weighted ϕ-quantile of samples.**

OUTPUT is invoked whenever a current estimate of the quantiles is desired or if the input stream runs dry. OUTPUT does not destroy or modify the state. Therefore, it can be invoked as many times as required. It is this feature of our algorithm that makes it amenable for online aggregation environments [Hel97].

4 Analysis of the Algorithm

There are two possible sources of error which our algorithm is subject to. The first is the sampling error, and the second is the error introduced by the algorithm. Correspondingly, in our analysis, the permissible error ϵ is split into two parts: $(1 - \alpha)\epsilon$ and $\alpha\epsilon$, for some $\alpha \in (0, 1)$. The first part is used to account for the sampling error, and the second part is used to account for the error introduced by the deterministic tree that consumes the samples (See Figure 1). More precisely, we will establish two bounds. First, in Section 4.1, the sampling scheme will be shown to guarantee that with probability at least $1 - \delta$, both the weighted $(\phi - \alpha\epsilon)$-quantile and the weighted $(\phi + \alpha\epsilon)$-quantile of the sample set are ϵ-approximate ϕ-quantiles of the input sequence seen so far. Second, in Section 4.2, the deterministic tree will be shown to guarantee that with probability 1, it computes a weighted $\alpha\epsilon$-approximate ϕ-quantile of the weighted sequence of (sample) elements fed to it. The two guarantees taken together ensure that the output is an ϵ-approximate ϕ-quantile of the input sequence at all times but for an event of probability at most $1 - \delta$ which accounts for the failure of the sampling step.

Analysis of the sampling scheme and the deterministic tree will yield inequalities linking together $b, k, h, \epsilon, \delta$ and α. Computing b, k and h as functions of ϵ and δ will then be a matter of solving an optimization problem subject to a set of constraints we derive.

4.1 The Sampling Constraint

The non-uniformity of our sampling scheme is implicit in the definition of NEW. Our analysis employs an interesting variant of Hoeffding's inequality [Hoe63]. We state the lemma here without proof and refer the interested reader to Hoeffdings original article:

Lemma 1 *(Hoeffding's Inequality) Let $X_1, X_2, \ldots, X_t$ denote independent random variables satisfying $0 \leq X_i \leq n_i$ for $i = 1, 2, \ldots, t$. Let $X = \sum_{i=1}^{t} X_i$. Let $\mathbf{E}X$ denote the expected value of X. Then, for any $\lambda > 0$, $Pr[|X - \mathbf{E}X| \geq \lambda] \leq \exp \frac{-2\lambda^2}{\sum_{i=1}^{t} n_i^2}$.*

Consider partitioning N, the size of the input sequence, into t disjoint non-empty subsets of sizes $n_1, n_2, \ldots, n_t$ in an arbitrary way. Thus $N = \sum_{i=1}^{t} n_i$. From within each subset, we choose one element (the *representative* for this subset) uniformly randomly. These t representatives constitute our sample. Each element in the sample is assigned a weight that equals the size of the subset it was drawn from. Therefore, the cumulative weight of all elements in the sample is N. Consider the weighted $(\phi + \alpha\epsilon)$-quantile and the weighted $(\phi - \alpha\epsilon)$-quantile of the sample. Let δ denote the probability that the ranks of both of these elements in the sorted sequence of the input sequence (of size N) lies in the range $\lceil(\phi \pm \epsilon)N\rceil$.

Lemma 2 $\delta \leq 2\exp\left[-2(1-\alpha)^2\epsilon^2 \frac{(\sum_{i=1}^{t} n_i)^2}{\sum_{i=1}^{t} n_i^2}\right]$

Proof: Let SMALL denote the set of input elements whose rank is smaller than $\lceil(\phi - \epsilon)N\rceil$ in the sorted input. Let LARGE denote the set of elements whose rank is larger than $\lceil(\phi + \epsilon)N\rceil$. Our sample is bad iff either the weighted sum of sample elements drawn from SMALL is more than $\lceil(\phi - \alpha\epsilon)N\rceil$ or the weighted sum of sample elements drawn from LARGE is more than $N - \lceil(\phi + \alpha\epsilon)N\rceil$. We will bound the probability that the first event occurs.

Define t Bernoulli variables, $X_1, X_2, \ldots, X_t$. The random variable X_i takes the value n_i if the representative of the i^{th} subset lies in SMALL; otherwise it takes the value zero. Let $X = \sum_{i=1}^{t} X_i$. It follows that $\mathbf{E}X = \lceil(\phi - \epsilon)N\rceil - 1$. The probability that X assumes a value at least $\lceil(\phi - \alpha\epsilon)N\rceil$ can be bounded by invoking Hoeffding's inequality as follows: $Pr[X - \mathbf{E}X > (1-\alpha)\epsilon N] \leq \exp \frac{-2(1-\alpha)^2\epsilon^2(\sum_{i=1}^{t} n_i)^2}{\sum_{i=1}^{t} n_i^2}$.

By a symmetric argument, one can show that the same bound holds for the probability that weighted sum of elements drawn from LARGE is more than $N - \lceil(\phi + \alpha\epsilon)N\rceil$. Taken together, we get the desired result. □

Let H denote the highest level of the tree. Let L_d denote the number of leaves in the tree before sampling starts, i.e., the number of NEW buffers with weight 1. Let L_H denote the number of leaves at level H. Let L_s denote the number of leaves at any other level other than 0 and H. See Figure 3.

Application of Lemma 2 to our sampling scheme yields $\delta \leq 2\exp\left[-2(1-\alpha)^2\epsilon^2 X\right]$ where X equals $\frac{[L_d k + 2L_s k + 2^2 L_s k + \cdots + 2^{H-1} L_s k + 2^H L_H k]^2}{2^0 L_d k + 2^2 L_s k + 2^4 L_s k + \cdots + 2^{2H-2} L_s k + 2^{2H} L_H k}$ which simplifies to $\frac{k[L_d + (2^H - 2)L_s + 2^H L_H]^2}{L_d + \frac{4^H - 4}{3} L_s + 4^H L_H}$. It can be shown[1] that X takes its minimum value for some $L_H \geq 0$. The minimum value is $X \geq \frac{2^H - 1}{4^H}\left[4L_d + \frac{8}{3}(2^H - 2)L_s\right]$.

It follows that when $H = 1$, $X = L_d k$. As $H \to \infty$, $X \to \frac{8}{3} L_s k$. We can also show that $X \geq \min[L_d k, \frac{8}{3} L_s k]$. It follows that $\delta \leq 2\exp[-2(1-\alpha)^2\epsilon^2 \min[L_d k, \frac{8}{3} L_s k]]$ which is equivalent to

$$\boxed{\min\left[L_d k, \tfrac{8}{3} L_s k\right] \geq \frac{\log(2\delta^{-1})}{2(1-\alpha)^2\epsilon^2}} \qquad (1)$$

Thus, we have proved the following lemma:

Lemma 3 *For any dataset, ϵ, δ, ϕ, and any choice of $0 < \alpha < 1$: if equation 1 is satisfied, then both the weighted $\phi - \alpha\epsilon$-quantile and the $\phi + \alpha\epsilon$-quantile of the sample are ϵ approximate ϕ quantiles of the dataset with probability $1 - \delta$.*

Notice that equation 1, places a restriction on k, the size of each buffer, and L_d and L_s parameters which are determined by the shape of the collapse tree. We now derive two other such conditions, each of which comes from considering the errors that are introduced by the computational process. Unlike the first condition, these will depend on ϵ and α only and not on δ.

4.2 The Tree Constraints

Let C denote the total number of COLLAPSE operations in the tree, i.e., the number of non-leaf non-root nodes. Let W denote the sum of weights of all COLLAPSE operations. Let w_{max} denote the weight of the heaviest child of the root. The main lemma in our earlier paper, which applies *mutatis mutandis* to the new sampling based algorithm is:

Lemma 4 *The weighted difference in rank between the true weighted ϕ-quantile of the sequence fed to the algorithm and that of the output produced by the algorithm is at most $\frac{W - C - 1}{2} + w_{max}$.* □

For a proof of this lemma, we refer the interested reader to our earlier paper [MRL98]. The following lemma provides an upper bound for W, the sum of weights of all COLLAPSE operations.

[1] We minimize $y = (a+x)^2(b+x)^{-1}$, where $a = 2^{-H}[L_d + (2^H - 2)L_s]$ and $b = 4^{-H}[L_d + \frac{4^H - 4}{3} L_s]$. Setting $y' = (a+x)(2b - a + x)(b+x)^2$ to zero yields $x = a - 2b$. The second derivative $y'' = 2(b-a)^2(b+x)^3$ is positive at $x = a - 2b$. The minimum value for y is $4(a-b)$.

Lemma 5 *Let $\ell_1, \ell_2, \ldots, \ell_L$ denote the sequence of full leaves in the tree from left to right. Let h_i denote the distance of ℓ_i from the final root. Let w_i denote the weight that was assigned to ℓ_i when it was created with* NEW. *Then $W \leq \sum_{i=1}^{L} w_i(h_i - 1)$* □

Note that the partial buffer that might result when the input sequence terminates, does not participate in Lemma 5.

We first handle the case $H \geq 1$. In Figure 3, L_d and L_H denote the number of full leaves at level 0 and level H respectively. L_s denotes the number of full leaves at all other levels. The size of the partial buffer is k' where $0 \leq k' < k$.

Application of Lemma 5 yields $W \leq L_d(h+H-1) + 2L_s(h+H-2) + 2^2L_s(h+H-3) + \cdots + 2^{H-1}L_s h + 2^H(h-1)L_H$ which simplifies to

$$W \leq L_d(h+H-1) + L_s[(h+1)2^H - 2(h+H)] + L_H(h-1)2^H$$

The weighted sum of elements in leaf buffers is

$$\begin{aligned} S &= L_d k + 2L_s k + 2^2 L_s k + \cdots + 2^{H-1}L_s k + 2^H L_H k + 2^H k' \\ &= k[L_d + L_s(2^H - 2) + L_H 2^H] + 2^H k' \end{aligned}$$

The analysis of the tree shown in Figure 3 is simplified if we weaken the upper bound in Lemma 4 to $\frac{W}{2} + w_{max}$. Then the tree computes a weighted $\alpha\epsilon$-approximate quantile of the sample S if the constraint $\frac{W}{2} + w_{max} \leq \alpha\epsilon S$ is satisfied.

Setting w_{max} equal to $L_d + (2^H - 2)L_s$ pessimistically, we obtain $\frac{1}{2}[(L_d(h+H-1) + L_s[(h+1)2^H - 2(h+H)] + L_H(h-1)2^H] + L_d + (2^H - 2)L_s \leq \alpha\epsilon k[L_d + (2^H - 2)L_s + L_H 2^H] + \alpha\epsilon 2^H k'$ which can be tightened by dropping the trailing term containing k'. The resulting inequality is equivalent to $\frac{L_d(h+H-1)+L_s[(h+1)2^H-2(h+H)]+L_H(h-1)2^H}{L_d+L_s(2^H-2)+L_H 2^H} \leq 2\alpha\epsilon k - 2$. Note that addition of $L_H 2^H$ in the denominator on the left hand side of the inequality is accompanied by addition of $(h-1)L_H 2^H$ in the numerator. It is true that $\frac{a+(h-1)\Delta}{b+\Delta} \leq \frac{a}{b}$ iff $\frac{a}{b} \geq h-1$ for any positive integers a, b, Δ and h. It can be verified that $\frac{L_d(h+H-1)+L_s[(h+1)2^H-2(h+H)]}{L_d+L_s(2^H-2)} \geq (h-1)$ for any value of H. Thus, we can weaken the previous inequality to $\frac{L_d(h+H-1)+L_s[(h+1)2^H-2(h+H)]}{L_d+L_s(2^H-2)} \leq 2\alpha\epsilon k - 2$.

Let β denote the ratio $\frac{L_d}{L_s}$. It can be shown that the left hand side is less than $h + 1 + c$, where $c = \max\left[\frac{(\beta-2)(H-2)}{\beta+2^H-2}\right]_{H\geq 1}$. This yields

$$\boxed{h + 3 + c \leq 2\alpha\epsilon k} \qquad (2)$$

The analysis is much simpler for the Munro-Paterson COLLAPSE policy [MP80] as $\beta = 2$.

We now handle the case $H = 0$, i.e., sampling has not kicked in; all elements in the input sequence are fed to the tree. Let L be the current number of full leaves, where $0 < L \leq L_d$. So far, $Lk + k'$ input elements have been processed by the tree, where k' is the size of the partial buffer in the end. The height of the tree is at most h. From Lemma 4, the difference in rank between the output of the algorithm and the exact ϕ-quantile of the input sequence is at most $\frac{W}{2} + w_{max}$. Using Lemma 5, we obtain $\frac{L(h-1)}{2} + w_{max} \leq \epsilon(Lk + k')$. Using a pessimistic value of $w_{max} = L$, we get $\frac{L(h-1)}{2} + L \leq \epsilon(Lk + k')$ which can be made tighter by dropping the term containing k'. The resulting inequality simplifies to

$$\boxed{h + 1 \leq 2\epsilon k} \qquad (3)$$

4.3 Putting it All Together

Equations 2 and 3 ensure that the deterministic tree always computes a weighted $\alpha\epsilon$-approximate ϕ-quantile of the sample fed to it. Equation 1 ensures that with probability at least $1 - \delta$, the output of this tree is no more than another $(1 - \alpha)\epsilon N$ elements away from the exact ϕ-quantile of N elements for any value of N. Taken together, the two constraints ensure that the overall algorithm always computes an ϵ-approximate ϕ-quantile with probability at least $1 - \delta$ without knowing the size of the input sequence in advance. Thus, provided all the constraints specified in equations 1, 2, and 3 are satisfied, the output value will be an ϵ approximate ϕ quantile with probability at least $1 - \delta$.

4.4 Space Complexity

The space complexity for the algorithm can be computed by fixing $\alpha = 0.5$ and using Munro-Paterson COLLAPSE policy, which is explained in detail in [MRL98]. The number of leaves $L_d = 2^b$ and $L_s = 2^{b-1}$. Eqn 1 simplifies to $2^b k \geq 2\epsilon^{-2}\log(2\delta^{-1})$. The height $h = b - 1$, $\beta = 2$ and $c = 0$ in Eqn 2, which simplifies to $b + 2 \leq \epsilon k$, which is tighter than Eqn 3. Solving these two inequalities for b and k yields the space complexity.

Theorem 1 *It is possible to compute, with probability at least $1 - \delta$, an ϵ-approximate ϕ-quantile of an arbitrarily large dataset in a single pass without requiring a priori knowledge of its size, using $O(\epsilon^{-1}\log^2\epsilon^{-1} + \epsilon^{-1}\log^2\log\delta^{-1})$ space.* □

Our random sampling scheme is easy to implement in practice as it requires us to pick a single element from a block of r elements where r is a power of two. Our sampling is without replacement. Typically, algorithms that employ random sampling require sampling without replacement, which is not as easy to implement.

4.5 Computing b and k

Computation of b and k now reduces to an optimization problem where we minimize bk, the amount of main memory required, subject to three constraints, namely Equations 1, 2 and 3. We also have the constraints $0 < \alpha < 1$, $b > 0$, $k > 0$ and $h > 0$.

If the tree is allowed to grow to height $h \geq 3$ before sampling begins, the number of leaves $L_d = \binom{b+h-2}{h-1}$ and $L_s = \binom{b+h-3}{h-1}$. These can be plugged in Eq 1.

Optimal values for b and k for values of ϵ and δ of practical interest can be computed by searching for b and h in the interval $[2, 50]$. For fixed values of b and h, the three constraints imply a lower bound on k, which can be computed as follows. Substituting values of b, h, ϵ and δ in Eqn 1 yields an inequality of the form $k \geq c_1(1-\alpha)^{-2}$ where c_1 is some constant. Eqn 2 yields an inequality of the form $k \geq c_2\alpha^{-1}$ where c_2 is another constant. Solve the equation $c_1(1-\alpha)^{-2} = c_2\alpha^{-1}$ for α. Then $\min[\lceil c_2\alpha^{-1} \rceil, \frac{h+1}{2\epsilon}]$ is a lower bound for k. The latter term comes from Equation 3. Identify that combination of b and h that minimizes the product bk.

4.6 Performance Comparison

Table 1 lists b, k and total memory required by the new algorithm for practical values of ϵ and δ. The memory requirements for our old algorithm that knows N a priori [MRL98] are also listed along with. The new algorithm requires no more than twice the memory required by the old one. Figure 4 compares the memory requirements as N varies. The new algorithm requires a constant amount of space, no matter what the value of N is. The old algorithm can take advantage of the fact that sampling need not be carried out for small values of N and save on memory requirements.

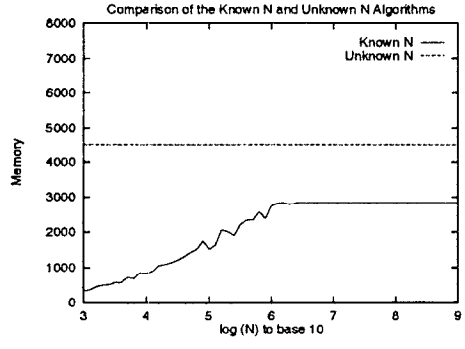

Figure 4: **Comparison of memory requirements for $\epsilon = 0.01$ and $\delta = 10^{-4}$.**

4.7 Multiple Quantiles

If a multitude of quantiles is desired simultaneously, the algorithm remains the same as before. Its analysis, however, requires a small change: δ in Eqn 1 gets replaced by $p\delta$ where p is the number of quantiles being computed simultaneously. The proof of correctness is simple: Let $\delta' = \delta/p$. The deterministic algorithm to which samples are fed computes any number of weighted quantiles of the samples simultaneously, each of which is $\alpha\epsilon$-approximate. Eqn 1 confirms that the probability that a particular quantile fails to be ϵ-approximate is at most δ'. It follows that the probability that *any* quantile fails to be ϵ-approximate is at most $p\delta'$ which is simply δ.

From Theorem 1, we deduce that the dependence of the total amount of memory required on the number of quantiles p, is $O(\log^2 \log p)$. Therefore, the cost of computing additional quantiles is small.

We can actually establish an upper bound on main memory requirements, independent of p. The trick lies in pre-computing a total of $\lceil \epsilon^{-1} \rceil$ quantiles for $\phi' = \frac{\epsilon}{2}, \frac{3\epsilon}{2}, \frac{5\epsilon}{2}$, and so on, each one of which is $\frac{\epsilon}{2}$-approximate. To output a ϕ-quantile, simply select that quantile from the pre-computed set that corresponds to a position closest to ϕ. It is easy to see that the output is ϵ-approximate. This pre-computation requires $O(\epsilon^{-1}\log^2(\epsilon^{-1}\log(\epsilon\delta)^{-1}))$ space. It is very useful when ϕ is not known in advance, as is the case when quantiles are used for constructing equi-depth histograms.

In Table 2, we plot the memory requirements as the number of quantiles increases, for different values of ϵ, keeping δ fixed at 10^{-4}. The last column lists the upper bound on memory requirements for an arbitrary number of quantiles.

ϵ, p	1	4	16	64	∞
0.100	0.28 K	0.29 K	0.30 K	0.31 K	0.71 K
0.050	0.68 K	0.70 K	0.71 K	0.73 K	1.64 K
0.010	4.68 K	4.78 K	4.87 K	4.97 K	11.23 K
0.005	10.51 K	10.76 K	10.97 K	11.17 K	25.16 K
0.001	67.61 K	69.01 K	70.18 K	71.24 K	97.66 K

Table 2: **Memory requirements for multiple quantiles.**

As expected, the amount of main memory required grows slowly as a function of p, the number of distinct quantiles requested. However, pre-computation of $\lceil \epsilon^{-1} \rceil$ equally spaced quantiles requires significantly more memory. This stems from the fact that memory requirements grow at least as fast as ϵ^{-1}, and pre-computation sets the approximation guarantee to $\frac{\epsilon}{2}$. Therefore we are better off using the pre-computation trick only if p is extremely large, or if p is not known at the outset.

5 Dynamic Buffer Allocation

One drawback of our algorithm is that we need to allocate all the memory required up front (Figure

	Unknown N Algorithm									Known N Algorithm		
ϵ, δ	*Number of Buffers b*			*Size of Buffer k*			*Total memory bk*			*Total memory*		
	10^{-3}	10^{-4}	10^{-5}	10^{-3}	10^{-4}	10^{-5}	10^{-3}	10^{-4}	10^{-5}	10^{-3}	10^{-4}	10^{-5}
0.1000	3	3	3	90	97	102	0.26 K	0.28 K	0.30 K	0.13 K	0.14 K	0.15 K
0.0500	3	3	4	216	231	181	0.63 K	0.68 K	0.71 K	0.32 K	0.35 K	0.38 K
0.0100	4	5	5	1140	958	991	4.45 K	4.68 K	4.84 K	2.45 K	2.68 K	2.83 K
0.0050	5	5	5	2060	2153	2232	10.06 K	10.51 K	10.90 K	5.77 K	6.25 K	6.56 K
0.0010	6	6	6	11109	11539	11914	65.09 K	67.61 K	69.81 K	39.71 K	42.61 K	44.49 K

Table 1: **Values for number of buffers b, size of each buffer k and total memory required by the new algorithm for different values of ϵ and δ. Also listed are memory requirements by our old algorithm that knows N a priori (N is assumed to be large enough to warrant sampling).**

4). If the input consists of a singleton element, our main memory usage is clearly outrageous. This can be ameliorated by allocating the set of b buffers one by one, as required. Still, for small values of $N \geq bk$, the algorithm uses significantly more memory than would have been required had N been known in advance. Is it possible to re-design the algorithm so that buffers are allocated even more slowly so that our main memory usage at all times is as close as possible to that required by an algorithm that knows N?

In practice, memory allocation would consist of a sequence of buffer allocation operations across time. For performance and simplicity, it is desirable that each buffer be contiguous and that its size remain unchanged.

We now design an algorithm whose memory requirements grow slowly with the size of the input. We start off by allocating one buffer initially. When it is full, we allocate another. When the second buffer is also full, we have a choice between invoking COLLAPSE and allocating a new buffer. In general, this choice has to be made when all buffers currently allocated are full. For $i \in \{1, b\}$, let L_i denote the number of leaves in the tree when the i^{th} buffer is allocated. For the first two buffers, $L_1 = 0$ and $L_2 = 1$. We call the sequence $\langle L_1, L_2, \ldots, L_b \rangle$ the *buffer allocation schedule* for the algorithm. When L_d NEW operations have been carried out, we start sampling and we follow the original algorithm. For simplicity, we assume that for all i, $L_i < L_d$, i.e., there is no buffer allocation once sampling kicks in.

If the input sequence has more than $L_d k$ elements, the constraints in Eq 1 and Eq 2 would ensure that the output is an ϵ-approximate ϕ-quantile. If the input sequence terminates at some point before L_d NEW operations have been invoked, we invoke OUTPUT operation as usual. However, we require a guarantee that the output is indeed an ϵ-approximate ϕ-quantile no matter what the current value of N is. Clearly, not all buffer allocation schedules can provide such a guarantee. We call a buffer allocation schedule *valid* if it provides such a guarantee.

It turns out that several buffer schedules are *valid*. To choose the *best* among the myriad of *valid* schedules, we need an objective function. If the objective is to minimize the maximum amount of memory ever required, our original algorithm which allocates all buffers at the outset is the best. If the objective is that the main memory requirements be as *close* as possible to that if N were known, we need to quantify the goodness of a valid buffer allocation schedule. Once such a quantitative measure is available, we can select the optimal schedule.

Another approach to select a reasonably good buffer allocation schedule is to let the user specify an upper limit on the main memory requirements for different values of N. There may or may not be a valid buffer schedule that meets these upper limits. By trial and error, the user can discover a schedule that is both valid and reasonably good.

We adopt the latter approach, letting the user specify upper limits on main memory usage for different values of N. How do we compute a valid schedule that meets these limits? We search for k and b by assigning increasingly large values to k. Fixing k fixes b and the buffer allocation schedule. We can then use Eq 3 to limit h, the height to which the tree is allowed to grow before we start sampling. This enables us to compute both L_d and L_s. From Eq 1, we obtain an upper bound for α. From Eq 2, we get a lower bound for α. If the range between these bounds does not intersect with the interval $(0, 1)$, the current schedule is rejected and we start all over again with a larger value of k. Otherwise, all constraints have been satisfied and the current schedule is accepted.

Figure 5 shows a valid schedule whose main memory requirements are always within the upper limits specified by the user.

6 Parallel Implementation

In a parallel setting, we assume P processing nodes. The input also consists of P separate input sequences,

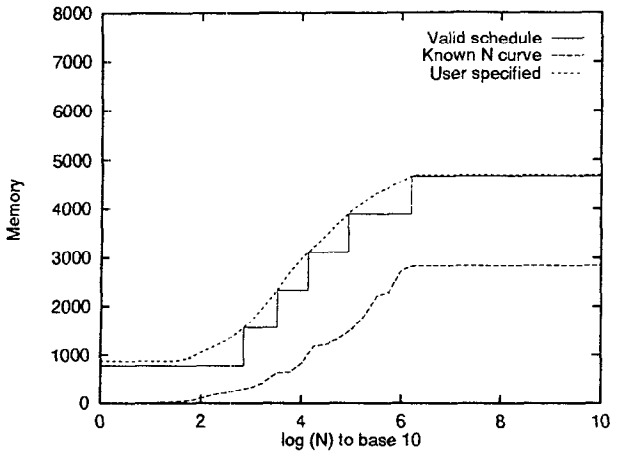

Figure 5: **A valid buffer allocation schedule within user specified memory constraints, for $\epsilon = 0.01$ and $\delta = 10^{-4}$.**

one per processor. Any input sequence may terminate at any time. We wish to compute approximate quantiles of the aggregate of all sequences taken together. Interprocessor communication should be minimal.

At each processor, we run the single processor algorithm outlined in the previous section. A processor, upon termination of its input sequence, is left with some full buffers and possibly a partial buffer. If there are at least two full buffers, a final COLLAPSE on the set of full buffers is invoked. Each processor finally has at most one full buffer and at most one partial buffer. Both buffers, tagged with their respective weights and sizes, are then shipped for further processing to a distinguished processor which we call Processor P_0.

Processor P_0 assigns level 0 to all incoming buffers. It retains the weights of incoming full buffers. To handle incoming partial buffers, it maintains an additional buffer B_0. The first partial buffer received is copied to B_0. When another partial buffer B_{in} arrives, the weights of B_{in} and B_0 are compared. We denote the weights by W_{in} and W_0 respectively. If they are equal, we copy as many elements from B_{in} as possible without overflowing B_0. If B_0 becomes full, it is added to the list of full buffers maintained by Processor 0. If all buffers are currently full and there still remain some uncopied elements in B_{in}, COLLAPSE is invoked. The remaining elements of B_{in} are then copied to B_0. If W_{in} and W_0 differ, then the buffer with smaller weight is shrunk in size by sampling at a rate equal to the ratio of the larger weight to the smaller. Moreover, the buffer just shrunk is assigned the larger weight. For example, if B_{in} has weight $W_{in} = 8$ and B_0 has weight $W_0 = 2$, then B_0 is shrunk in size by sampling at rate $W_{in}/W_0 = 4$, i.e., exactly one out of successive blocks of four elements in B_0 is selected. After shrinking, B_0 is assigned weight 8. At this point both B_0 and B_{in} have same weight and we process them as described before.

When all input buffers have been received by P_0, it invokes an OUTPUT operation on all its buffers taken together.

How much memory does P_0 require? P_0 is required to maintain at least two buffers. It can build any tree of buffers (See Figure 2). In the analysis that follows, we assume that the height of such a tree is h'.

When the degree or parallelism is very large, collecting output buffers at one node may deteriorate performance significantly. In such a case, we aggregate processors into multiple groups. One designated processor in each group collects the output buffers from all others in its group. In the end, the outputs from these processors can be collected at one processor. As far as theoretical analysis of such a scheme is concerned, luckily, all that matters is the increase in the height of the tree, which we denote by h'.

6.1 Parallel Sampling Constraint

Assuming that sampling is carried out in at least one processor, application of Lemma 1 to the set of samples from all processors yields

$$\delta \leq 2\exp\left[-2(1-\alpha)^2\epsilon^2 \mathrm{X}\right]$$

where $\mathbf{X} = \frac{(\sum A_i)^2}{\sum B_i}$ for $i = 1, 2, \ldots, P$. There is a small catch here[2]. Since all A_i and B_i values are non-negative, $\frac{(\sum A_i)^2}{\sum B_i} \geq \frac{\sum A_i^2}{\sum B_i} \geq \min\left[\frac{A_i^2}{B_i}\right]_{i=1,2,\ldots,P}$. We showed that $\frac{A_i^2}{B_i} \geq \min[L_d k, \frac{8}{3}L_s k]$, for $i = 1, 2, \ldots, P$. This leads to the following inequality:

$$\boxed{\min\left[L_d k, \tfrac{8}{3}L_s k\right] \geq \frac{\log 2\delta^{-1}}{2(1-\alpha)^2\epsilon^2}} \qquad (4)$$

Interestingly, the constraint is the same as before (Eq 1).

6.2 Parallel Tree Constraints

The analysis of the tree is slightly different because of the additional COLLAPSE in the end at a designated processor. Proceeding as before, we get an inequality of the form $\frac{\sum C_i}{\sum D_i} \leq 2\alpha\epsilon k - 2$. Since all C_i and D_i values are positive, from componendo-dividendo, $\frac{\sum C_i}{\sum D_i} \leq \max\left[\frac{C_i}{D_i}\right]_{i=1,2,\ldots,P}$. We already established that for any i, $\frac{C_i}{D_i} \leq h + h' + 3 + c$ where $c =$

[2]Processors where sampling has not started will have their A_i values less than L_d. We can club these values together with the L_d of that A_i which corresponds to the processor where sampling has begun. The rest of the analysis is then ok.

$max\left[\frac{(\beta-2)(H-2)}{\beta+2^H-2}\right]_{H\geq 1}$ and $\beta = \frac{L_d}{L_s}$. It follows that

$$\boxed{h + h' + 3 + c \leq 2\alpha\epsilon k} \quad (5)$$

Interestingly, the constraint is the same as before, only the height h has now been augmented with h', the additional height gained at the merging processor.

If sampling does not start at any processor, then following the same argument as in the non-parallel case, we obtain the constraint

$$\boxed{h + h' + 1 \leq 2\epsilon k} \quad (6)$$

6.3 Putting It All Together

Remarkably, the three constraints, namely Eqn 4, 5 and 6, are very similar to the tree constraints derived for the single processor case. Computing b and k amounts to optimizing bk subject to the three constraints.

7 Extreme Values

This section deals with an interesting special case of the order statistic problem. The case arises when the required quantile ϕ is small and close to ϵ, the required accuracy. For instance, if the required quantile ϕ is 1%, or equivalently, .01 and the desired accuracy is 1 in 1000 or .001.

We provide a simple algorithm which seems to outperform most other algorithms handily in the amount of memory required. As a motivating example, when ϕ and ϵ are set to exactly the same value, the minimum value in the dataset is an ϵ-approximate ϕ-quantile. Clearly, this value can be calculated using very little space. In this section we look at a simple generalization of this observation.

The method is to use a random sample and keep only the k largest elements of the sample in memory. The parameters k and the sample size s are chosen in conjunction so that the expected rank of the kth largest element of the sample is ϕN and so that the probability that it is an ϵ-approximate ϕ-quantile is at least δ. As k is increased, the sample size s has to be increased in correspondence. In particular, the relationship is $k = \phi s$. Thus, the sampling rate, which is $\frac{s}{N} = \frac{k}{N\phi}$ is dependent on N, the size of the data set.

The question is the following: Given ϵ, δ and ϕ, what is the smallest value of k (and consequently of s) such that the k^{th}largest value in s is an ϵ approximate ϕ quantile with probability $1 - \delta$? Computing a tight bound for k and s requires a tighter bound on the probability of tail events than that provided by Hoeffding's lemma. We elaborate on this next.

There are a number of bounds on the probability of tail events, alternately known as "large deviation theorems." We will state a form that is convenient in our context, usually known as Stein's lemma.

Let $\{X_i : 1 \leq i \leq N\}$ be i.i.d. $\{0,1\}$ Bernoulli with parameter q which is unknown. The simple likelihood test between two competing hypothesis p_1 and p_2 aims on figuring out which of p_1 and p_2 is closer to q. The test is simply to choose the p_j maximizing $P(\{X_i\}$ were generated by $p_j)$, which is, letting $\sum_i X_i = \ell$, $\binom{n}{\ell} p_j^\ell (1-p_j)^{N-\ell}$. Let p_1 be the truly better hypothesis[3] with respect to q. We say that the test fails if it chooses the wrong hypothesis, in this case, p_2. Stein's lemma places a bound on the probability that this test fails.

Lemma 6 *(Stein's Lemma) Let $\{X_i : 1 \leq i \leq s\}$ be i.i.d. Bernoulli with parameter q. Consider any p_1, p_2, $p_i \in [0,1]$. Then, the probability that the likelihood test fails is bounded by*

$$P(\textit{likelihood test fails}) \leq 2^{-sD(p_1;p_2)}$$

where $D(p_1;p_2)$ denotes the Kullback Leibler distance, $p_1 \log \frac{p_2}{p_1} + (1-p_1)\log\frac{1-p_2}{1-p_1}$. □

Let us assume that we choose a random sample with replacement, (this is not much different from a sample without replacement if the sample size is small with respect to N). Let X_i be the Bernoulli variable that is 1 iff the ith sample point has rank at most ϕN. Clearly, the probability that X_i is 1 is exactly ϕ.

By choosing $p_1 = q = \phi$, $p_2 = \phi - \epsilon$, applying Stein's lemma and summing the two probabilities, we get a bound on the probability that the kth largest element is either too small or too large in terms of the sample size s (since, clearly, if the computed approximate quantile is not an ϵ-approximate ϕ-quantile, the likelihood test fails and the sum is the union bound on the two bad events).

Thus, modulo our approximation, we obtain the condition:

$$\delta \leq 2^{-D(\phi;\phi-\epsilon)s} + 2^{-D(\phi;\phi+\epsilon)s}$$

Also, note that since the expected rank of the kth largest element of the sample is ϕN, we get

$$k/s = \phi$$

The two conditions above give us enough information to compute a bound for k in terms of ϕ, ϵ and δ. When subject to the conditions that ϕ is small, and ϵ is smaller, the above expression reduces to

$$\frac{\delta}{2} \leq 2^{-\epsilon s}$$

[3]We do not specify exactly what we mean by this, though for our purposes, we note that in the case that $q = p_1$, then p_1 is the truly better hypothesis. The principle notion here is that the better hypothesis is the closer one with respect to the Kullback Leibler distance (sometimes known as the relative entropy). For details check [CT91]

Consequently, choosing $s = \frac{1}{\epsilon}(\log\frac{1}{\delta} + 1)$ suffices. And thus, k, the amount of memory required is $k = \frac{\phi}{\epsilon}(1 + \log\frac{1}{\delta})$.

The advantage of this method is that the amount of space required is linear in $\frac{1}{\epsilon}$ and not quadratic, as is the case with the random sampling algorithm when ϕ is larger.

8 Conclusions

Important database applications that employ quantiles suffer from lack of reliable *a priori* knowledge of the length of input sequence whose quantiles need be computed. This motivates a search for quantile finding algorithms that do not require such knowledge.

We presented the first practical algorithm enjoying this property. The algorithm is part of the framework first proposed in [MRL98]. Moreover, it employs a novel non-uniform random sampling technique. Its performance in terms of main memory requirements is comparable to that of the best known algorithm that knows N. Furthermore, we improved upon the algorithm by delaying the allocation of buffers so that the main memory requirements during the course of execution are as close as possible to that of the best algorithm that knows N. We also proposed and analyzed a parallel version of the algorithm.

We also presented algorithms that require significantly less memory if the desired quantiles is an extreme value, i.e., close to the largest or smallest element in the sequence.

Acknowledgments

We thank Peter Haas for pointing out Lemma 1, a version of Hoeffding's inequality that uses the sum of squares in the denominator of the exponent.

References

[ARS97] K. Alsabti, S. Ranka, and V. Singh. A One-Pass Algorithm for Accurately Estimating Quantiles for Disk-Resident Data. In *Proc. 23rd VLDB Conference*, Athens, Greece, 1997.

[AS95] R. Agrawal and A. Swami. A One-Pass Space-Efficient Algorithm for Finding Quantiles. In *Proc. 7th Intl. Conf. Management of Data (COMAD-95)*, Pune, India, 1995.

[BFP+73] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time Bounds for Selection. In *J. Comput. Syst. Sci.*, volume 7, pages 448–461, 1973.

[CMN98] S Chaudhuri, R Motwani, and V Narasayya. Random Sampling for Histogram Construction: How much is enough? In *ACM SIGMOD 98*, volume 28, pages 436–447, Seattle, WA, USA, June 1998.

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, New York, 1991.

[DB2] DB2 MVS. .

[DNS91] D. DeWitt, J. Naughton, and D. Schneider. Parallel Sorting on a Shared-Nothing Architecture using Probabilistic Splitting. In *Proc. Intl. Conf. on Parallel and Distributed Inf. Sys.*, pages 280–291, Miami Beach, 1991.

[GM98] P G Gibbons and Y Matias. New Sampling-based Summary Statistics for Improving Approximate Query Answers. In *ACM SIGMOD 98*, volume 28, pages 331–342, Seattle, WA, USA, June 1998.

[GM99] P. B. Gibbons and Y. Matias. Synoptic Data Structures for Massive Data Sets. In *To appear in DIMACS Series in Discrete Math. and Thy. Comp. Sc.*, 1999.

[GMP97] P G Gibbons, Y Matias, and V Poosala. Fast Incremental Maintenance of Approximate Histograms. In *Proc. 23rd Intl. Conf. on Very Large Data Bases*, pages 466–475, August 1997.

[Hel97] J. M. Hellerstein. Online Processing Redux. *Bulletin of the IEEE Computer Society*, 1997.

[Hoe63] W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *American Statistical Association Jornal*, pages 13–30, March 1963.

[Inf] Informix. .

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

[MRL98] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate Medians and other Quantiles in One Pass and with Limited Memory. In *ACM SIGMOD 98*, volume 28, pages 426–435, Seattle, WA, USA, June 1998.

[Pat97] M. R. Paterson. Progress in Selection. Deptt. of Computer Science, University of Warwick, Coventry, UK, 1997.

[PIHS96] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. In *ACM SIGMOD 96*, pages 294–305, Montreal, June 1996.

[Poh69] I. Pohl. A Minimum Storage Algorithm for Computing the Median. Technical Report IBM Research Report RC 2701 (# 12713), IBM T J Watson Center, November 1969.

[SALP79] P. G. Selinger, M. M. Astrahan, R. A. Lories, and T. G. Price. Access Path Selection in a Relational Database Management System. In *ACM SIGMOD 79*, June 1979.

[Vit85] J S Vitter. Random Sampling with a Reservoir. *ACM Tran. Math. Software*, 11(1):37–57, 1985.

[Yao74] F. F. Yao. On Lower Bounds for Selection Problems. Technical Report MAC TR-121, Massachusetts Institute of Technology, 1974.