Approximate Medians and other Quantiles in One Pass and with Limited Memory

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

We present new algorithms for computing approximate quantiles of large datasets in a single pass. The approximation guarantees are explicit, and apply without regard to the value distribution or the arrival distributions of the dataset. The main memory requirements are smaller than those reported earlier by an order of magnitude.

We also discuss methods that couple the approximation algorithms with random sampling to further reduce memory requirements. With sampling, the approximation guarantees are explicit but probabilistic, i.e., they apply with respect to a (user controlled) confidence parameter.

We present the algorithms, their theoretical analysis and simulation results.

1 Introduction

This article studies the problem of computing order statistics of large sequences of online or disk-resident data using as little main memory as possible. We focus on computing *quantiles*, which are elements at specific positions in the sorted order of the input. The ϕ-quantile, for $\phi \in [0, 1]$, is defined to be the element in position $\lceil \phi N \rceil$ in the sorted sequence of the input. Here, and in the rest of this paper N denotes the number of elements in the input. For $\phi = 0.5$, the quantile is called the *median*. An element is said to be an ϵ-approximate ϕ-quantile if its rank is between $\lceil (\phi - \epsilon)N \rceil$ and $\lceil (\phi + \epsilon)N \rceil$. Clearly, there could be several elements in the dataset that qualify.

1.1 Database Applications

Quantiles are of interest to both database implementers and database users. They 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.

Obtaining an accurate estimate of predicate selectivity is valuable for query optimization [1]. To better estimate the cardinality of query result sets, quantiles can be, and are, used to characterize the distribution of stored data [2]. Equi-depth histograms [3], for instance, 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.

Parallel database systems [4, 5] employ value range data partitioning that requires generation of *splitters* to divide the data into approximately equal parts. Distributed parallel sorting can also use splitter values to assign data elements to the nodes where they will be sorted [6].

Approximate quantiles can be substituted for exact quantiles in all three applications, namely statistical data analysis, database query optimization and value range data partitioning. However, without any *guarantee* on the error, *i.e.*, number of elements between the true quantile and the approximate quantile, practitioners should be, and are, reluctant to use the algorithms that produce them. However, if an algorithm provides explicit and *a priori* guarantees on the accuracy of its output, quantiles can be understood and trusted for use.

1.2 Challenges to Meet

The efficiency and correctness of the algorithm should be *data independent*. Datasets originate from two sources, namely stored tables and intermediate query results. The sequence of values coming from a stored table is influenced by the insert order and data value clustering. The sequence of values coming from an intermediate result depends on the query plan: for example, a merge join will produce a table ordered on the join column. Even if the stored or intermediate table is clustered (sorted) on a column(s) different from the ones used for quantile computation, correlations between these columns are not unlikely. Thus, arrival orders and value distributions are hard to characterize. It is important that algorithms not depend on assumptions about either for efficiency or correctness.

We note that different applications demand different levels of accuracy. Indeed, the amount of error in the quantiles produced affects the performance of the application that uses them. This cost can sometimes be quantified. For example, equi-depth histogram errors lead to cardinality estimation errors in query optimization. The cost of partition imbalance for distributed sorting is proportional to the difference between completion times for the smallest and largest partitions. The quantile finding algorithm should be tunable to the level of accuracy required for the application, and its performance should degrade gracefully when the accuracy requirements on it are increased.

We require that only a *single pass* should be made over the data. Multiple passes over large data sets are unattractive for performance reasons and are incompatible with most DBMS GROUP BY implementations.

Finally, we must address the problem of main memory usage. Clearly, the entire input stream cannot be buffered in memory. For a given degree of accuracy, the algorithm should use as little main memory as possible. This is especially important for query optimization as it is desirable to compute histograms for multiple columns of a table in a single pass over a table. GROUP BY algorithms also compute multiple aggregation results concurrently.

We propose the following desiderata: An algorithm must 1) not require prior knowledge of the arrival or value distribution of its inputs, 2) provide explicit and tunable approximation guarantees, 3) compute results in a single pass, 4) be parallelizable and scale well on SMP and MPP configurations, 5) produce multiple quantiles at no extra cost, 6) use as little memory as possible, and 7) be simple to code and understand. As we shall see, the algorithms we present clearly meet the first five requirements. While they improve upon previously described algorithms in memory usage, they still impose non-trivial memory costs. The reader will judge the simplicity and understandability of our algorithms.

2 Antecedents

2.1 The Theory Literature

The theory literature has primarily focused on counting the number of *comparisons* needed to find the exact median (quantile). The celebrated paper of Blum, Floyd, Pratt, Rivest and Tarjan[7], shows that selection of the kth largest element out of N can be done using at most $5.43N$ comparisons. This paper also shows that at least $1.5N$ comparisons are required in the computation of the exact median. For an account of progress since then, see the survey by Mike Paterson [8]. The current best bounds are much tighter and are the product of sophisticated and deep analysis (see [8, 9, 10, 11, 12]). The upper bound is $2.9423N$ comparisons, and the lower bound $(2+\alpha)N$, where α is of the order of 2^{-40}, an extremely small number by most standards.

Frances Yao [13] showed that computing an *approximate* median requires $\Omega(N)$ comparisons for any deterministic algorithm. Curiously, this lower bound is easily beaten by resorting to randomization. The naive randomized algorithm, which outputs the median of a random sample of size $O(\frac{1}{\epsilon^2}\log\delta^{-1})$, uses a number of comparisons independent of N. For a comprehensive survey of this aspect of the literature, see the survey by Paterson [8].

Ira Pohl [14] established that any deterministic algorithm that computes the exact median in one pass needs to store at least $N/2$ data elements. Munro and Paterson [15] generalized this and showed that memory to store $O(N^{\frac{1}{p}})$ elements is necessary and sufficient for finding the exact median in p passes. The same bound holds for any ϕ-quantile for a constant ϕ. This result motivates a search for algorithms that produce *approximate* quantiles in a single pass with less memory.

2.2 The Database Literature

Jain and Chlamtac [16] proposed a simple algorithm for computing quantiles in a single pass using only a constant amount of memory. However, there are no *a-priori* guarantees on the error. Agrawal and Swami [17] proposed another one pass algorithm. The idea here is to adjust equi-depth histogram boundaries on the fly when they do not appear to be in balance. Again, there are no strong and *a-priori* guarantees on error.

Alsabti, Ranka and Singh [18] propose a data independent single pass algorithm with guaranteed error bounds. We describe this algorithm in more detail in Section 4.4.

Random sampling can be a very useful tool in this context. It has been used by DeWitt *et al* [6] for distributed sorting. We discuss this idea and various applications in detail in Section 5.

2.3 Bridging the gap

The ideas in the paper by Munro and Paterson can be applied to develop a one-pass algorithm for approximate quantiles. Indeed, though (or perhaps because) the original paper appeared as early as 1980, this fact appears to have been missed by the database community. In this paper, we will describe a general and uniform setting which includes the Munro and Paterson algorithm as a special case. In doing so, we will improve upon the Munro-Paterson algorithm in terms of space required. We show how these algorithms can be coupled with sampling to obtain further reduction in space.

Here is a statement of the problem we tackle in the next two sections: *Given N, ϕ and ϵ, design a single pass algorithm that computes an ϵ-approximate ϕ-quantile of a dataset of size N using as little main memory as possible.*

3 A Uniform Framework

Any algorithm in our framework is parameterized by two integers b and k. The algorithm will use b buffers each of which can store k elements. Thus, (but for a small amount of memory required for book-keeping purposes), the memory footprint will be bk elements. We also associate with each buffer X, a positive integer $w(X)$, which denotes its *weight*. Intuitively, the weight of a buffer is the number of input elements *represented* by each element in the buffer. Buffers are always labeled empty or full. Initially, all b buffers are labeled empty.

The values of b and k will be calculated to (1) enforce the approximation guarantee and (2) optimize bk under the first constraint.

Various algorithms can be composed from an interleaved sequence of three operations, namely NEW, COLLAPSE and OUTPUT, which we now describe.

3.1 NEW Operation

NEW takes as input an empty buffer. 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 input buffer with the next k elements from the input sequence, labels the buffer as full, and assigns it a weight of 1. If the buffer cannot be filled completely because there are less than k remaining elements in the input sequence, an equal number of $-\infty$ and $+\infty$ elements are added to make up for the deficit.

Let the size of the augmented dataset, consisting of the original dataset plus the $-\infty$ and $+\infty$ elements added to the last buffer, be βN for some $\beta \geq 1$. Let $\phi' = \frac{2\phi+\beta-1}{2\beta}$. It

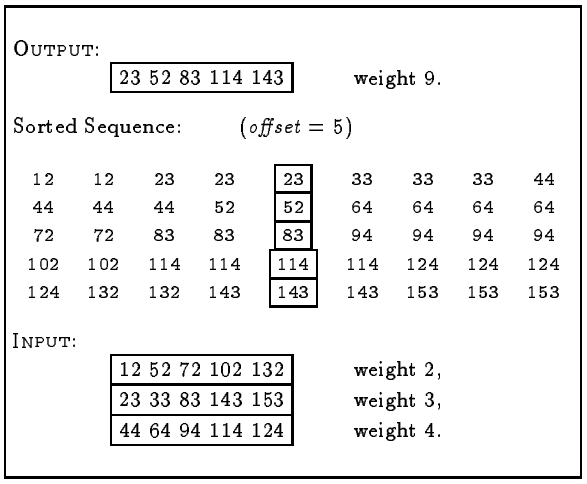

Figure 1: COLLAPSE illustrated.

is clear that the ϕ-quantile of the original dataset of size N corresponds to the ϕ'-quantile of the augmented dataset of size βN.

3.2 COLLAPSE Operation

COLLAPSE takes $c \geq 2$ full input buffers, $X_1, X_2, \ldots, X_c$, and outputs a buffer, Y, all 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. Figure 1 illustrates the COLLAPSE operation.

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 $j = 0, 1, \ldots, k-1$. We call the quantity $\frac{w(Y)+1}{2}$, the *offset* for this COLLAPSE. 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 $w(Y)$. In short, if we denote the offset for an output buffer Y by $offset(Y)$, the contents of Y can be described as consisting of elements in positions $jw(Y) + offset(Y)$, for $j = 0, 1, \ldots, k-1$.

It is easy to see that multiple copies of elements need not actually be materialized. The outputs to be stored in Y can be identified as follows: First sort the input buffers individually and then start merging them. While merging, if the element just selected originates from buffer X_i, a counter (initialized to zero) gets incremented $w(X_i)$ times. If the counter *hits* a value that corresponds to a position for which Y should be populated, the selected element is marked; otherwise it is left unmarked. In the end, all marked elements are collected together into one of the input buffers, which is labeled full; the remaining input buffers are labeled empty.

Let C denote the total number of COLLAPSE operations carried out during the course of the algorithm. Let W denote the sum of weights of the output buffers produced in all such operations.

Lemma 1 *The sum of offsets of all the* COLLAPSE *operations is at least* $\frac{W+C-1}{2}$.

Proof: Let $C = C_{odd} + C_{even}$, where C_{odd} and C_{even} are the number of COLLAPSE operations where the weight $w(Y)$ of the output buffer Y is odd and even respectively.

Further, let $C_{even} = C^1_{even} + C^2_{even}$, where C^1_{even} is the number of COLLAPSE operations where the offset for output buffer Y is $\frac{w(Y)}{2}$ and C^2_{even} is the number of COLLAPSE operations where the offset of output buffer Y is $\frac{w(Y)+2}{2}$.

Clearly, the sum of all offsets is $\frac{W+C_{odd}+2C^2_{even}}{2}$. Since COLLAPSE alternates between the two choices of offsets for even weights of the output buffer, $C^2_{even} \geq \frac{C_{even}-1}{2}$. Therefore, the sum of all offsets is at least $\frac{W+C-1}{2}$. □

3.3 OUTPUT Operation

OUTPUT is performed exactly once, just before termination. It takes $c \geq 2$ full input buffers, $X_1, X_2, \ldots, X_c$, of size k. It outputs a single element, corresponding to the approximate ϕ'-quantile of the augmented dataset. Recall that the ϕ-quantile of the original dataset corresponds to the ϕ' quantile of the augmented dataset, which consists of the original elements plus the $-\infty$ and $+\infty$ elements added to the last buffer.

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 of each element into account. The output is the element in position $\lceil \phi' kW \rceil$, where $W = w(X_1) + w(X_2) + \cdots + w(X_c)$.

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 and COLLAPSE reclaims some of them by collapsing a chosen subset of full buffers. OUTPUT is invoked on the final set of full buffers. Different buffer collapsing policies correspond to different algorithms. We now describe three interesting policies.

3.4 COLLAPSE Policies

Munro and Paterson [15]

If there is an empty buffer, invoke NEW; otherwise, invoke COLLAPSE on two buffers having the same weight.

Alsabti, Ranka and Singh [18]

Fill $b/2$ empty buffers by invoking NEW and then invoke COLLAPSE on them. Repeat this $b/2$ times and invoke OUTPUT on the resulting buffers.

New Algorithm

Associate with each buffer X an integer $\ell(X)$ that denotes its *level*. Let ℓ be the smallest among the levels of currently full buffers. If there is exactly one empty buffer, invoke NEW and assign it level ℓ. If there are at least two empty buffers, invoke NEW on each and assign level 0 to each one. If there are no empty buffers, invoke COLLAPSE on the set of buffers with level ℓ. Assign the output buffer, level $\ell + 1$.

4 Analysis

In this section, we will first see how the sequence of operations carried out by the algorithm can be looked upon as a tree of buffers. Next, we analyze the tree and show error bounds on the quality of the output. Finally, we show how we compute values of b and k to minimize memory for different collapsing policies.

4.1 A Tree Representation

The sequence of operations carried out by the 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 are 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 datastream. 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 Figures 2, 3 and 4 for the trees resulting from the Munro-Paterson algorithm, Alsabti-Ranka-Singh algorithm and the new collapsing policy proposed in this paper. The labels of nodes correspond to their weights. In all the figures, leaves get populated left to right. A COLLAPSE operation, corresponding to a non-leaf node, is invoked as soon as its operands, corresponding to the children of the node, are ready.

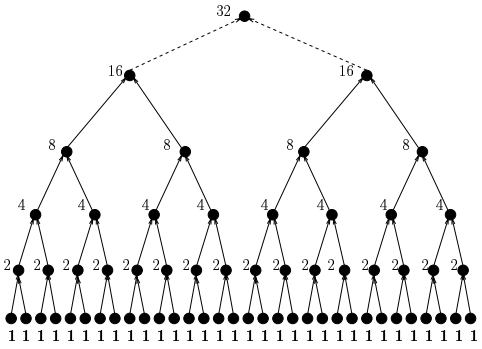

Figure 2: **The tree for Munro-Paterson algorithm for $b = 6$ buffers. Each node is labeled with its weight.**

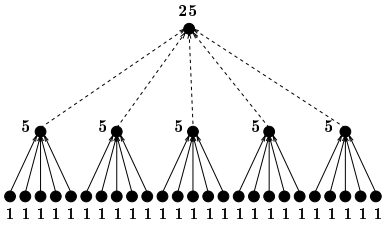

Figure 3: **The tree for Alsabti-Ranka-Singh algorithm for $b = 10$ buffers. Each node is labeled with its weight.**

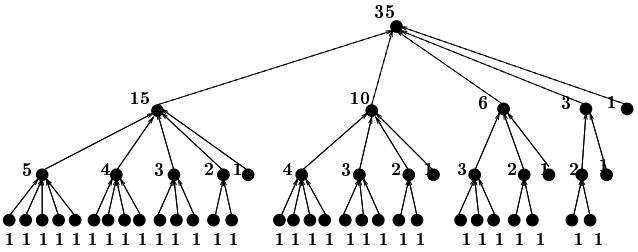

Figure 4: **The tree for the new collapsing scheme, with $b = 5$ buffers. Each node is labeled with its weight.**

It is possible to have collapsing policies that are not representable as trees. Also, it is possible to vary the buffer sizes. Such schemes are not in the scope of this paper.

4.2 Approximation Guarantees

Let L denote the number of leaves in the tree. Let C denote the number of COLLAPSE operations, 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. See Figure 5 for a quick reference to the list of symbols used in the analysis.

In this section, we will prove the following: *The difference in rank between the true ϕ-quantile of the original dataset and that of the output produced by the algorithm is at most* $\frac{W-C-1}{2} + w_{max}$.

We require a few simple results to ratify this claim, which we re-state as Lemma 5. The reader who wishes to skip the proof can jump to Section 4.3 without loss in continuity, noting that the claim holds for *any* tree whose leaves have weight 1 and whose non-leaves have at least two children. The leaves need not be at the same level.

Lemma 2 *The sum of weights of the top buffers, i.e., the children of the root, is L, the number of leaves.* □

Let Q be the output of the algorithm. We say that an element in the input sequence is *definitely-small* if we can assert that it is smaller than Q. Similarly, we say that an element is *definitely-large* if there is evidence that it is larger than Q.

Our analysis proceeds in two phases: First, we describe a procedure to identify the definitely-small and definitely-large elements and mark them. Next, we describe a counting technique to establish that there are a fairly large number of such elements.

The identification procedure starts with the top buffers, i.e., the children of the root. We mark elements in these buffers as definitely-small or definitely-large depending upon whether they are smaller or larger than Q. The element Q itself belongs to neither of the two categories. Some of the children of the root are leaves; we ignore them. Other children have at least two children and are the outputs of COLLAPSE operations. Consider one such output buffer and its children. All elements among the children that are smaller than a definitely-small element in the output can be marked definitely-small. Similarly, all elements among the children that are larger than a definitely-large element in the output can be marked definitely-large. See Figure 6 for an example. We continue in this fashion until all the leaves have been processed.

User Specified:

N	Size of dataset
ϕ	Quantile to be computed
ϵ	Approximation guarantee

Others:

b	Number of buffers
k	Size of each buffer
ϕ'	Quantile in the augmented dataset consisting of $-\infty$ and $+\infty$ elements
C	Number of COLLAPSEs
W	Sum of weights of all COLLAPSEs
w_{max}	Weight of heaviest COLLAPSE
L	Number of leaves in the tree
h	Height of tree

Figure 5: **List of important symbols used in the analysis.**

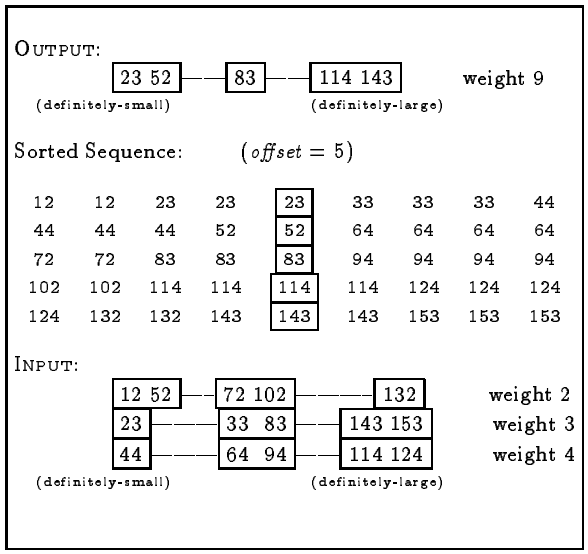

Figure 6: **Identification of definitely-small and definitely-large elements illustrated for an intermediate node in the tree, corresponding to the same input/output buffers as in Figure 1.**

Consider the sets of definitely-small and definitely-large elements among the children of the root. From Lemma 2, we infer that OUTPUT selects the element at position $\lceil \phi' kL \rceil$ in the sorted sequence of copies of elements. Though we cannot place an interesting bound on the number of definitely-small elements among these buffers, we can certainly place one on their *weighted sum*, where the weight of an element is the weight of the buffer it originates from. The same holds for definitely-large elements.

Let DS_{top} denote the weighted sum of definitely-small elements among the top buffers, i.e., children of the root. Let DL_{top} denote the weighted sum of definitely-large elements among these buffers.

Lemma 3

$$\lceil \phi' kL \rceil - w_{max} \leq DS_{top} \leq \lceil \phi' kL \rceil - 1$$
$$kL - \lceil \phi' kL \rceil - w_{max} + 1 \leq DL_{top} \leq kL - \lceil \phi' kL \rceil$$

□

Consider a node Y in the tree corresponding to a COLLAPSE operation. Let Y have $s \geq 0$ definitely-small elements. Then the weighted sum of these elements is $sw(Y)$. Consider the largest element in this set of definitely-small elements. This element is in position $(s-1)w(Y) + offset(Y)$ in the sorted sequence of elements of its children with each element having been duplicated as many times as the weight of the child it originates from. Therefore, the weighted sum of definitely-small elements among the children of Y (smaller than the largest definitely-small element in Y) is $(s-1)w(Y) + offset(Y)$ which can also be written as $sw(Y) + (w(Y) - offset(Y))$. See Figures 1 and 6 for an illustration of this argument. We can similarly argue that if there are ℓ definitely-large elements in Y, for a weighted sum of $\ell w(Y)$, then the weighted sum of definitely-large elements among its children is at least $\ell w(Y) - (w(Y) - offset(Y))$.

We have shown that *the weighted sum of both definitely-small and definitely-large elements among the children of each node Y is smaller by at most $w(Y) - offset(Y)$ than the weighted sum of definitely-small and definitely-large elements in Y itself.* This suggests a counting technique in which we maintain a set of buffers and the weighted sums of definitely-small and definitely-large elements in those buffers. The initial set is the set of top buffers, with the weighted sums being DS_{top} and DL_{top} respectively. We repeatedly replace a non-leaf in the set by its children and update the weighted sum of definitely-small and definitely-large elements. This process stops when we have traveled down towards all the leaves and no non-leaf remains in the set.

Let DS_{leaves} and DL_{leaves} denote the weighted sum of definitely-small and definitely-small elements in the set of leaf buffers. Since the weight of each leaf is one, DS_{leaves} and DL_{leaves} is, in fact, the number of definitely-small and definitely-large elements in the augmented dataset, exactly the values of interest.

Lemma 4

$$DS_{leaves} \geq DS_{top} - \frac{W - C + 1}{2}$$
$$DL_{leaves} \geq DL_{top} - \frac{W - C + 1}{2}$$

Proof: Starting at the top buffers, i.e., children of the root, the initial weighted sums of definitely-small and definitely-large elements are DS_{top} and DL_{top} respectively. We know that each COLLAPSE operation, corresponding to a node Y, diminishes the weighted sum of definitely-small and definitely-large elements by at most $w(Y) - offset(Y)$. Thus, as we travel down towards the leaves and hit a node Y, both DS_{top} and DL_{top} get diminished by $w(Y) - offset(Y)$. The total amount they are both diminished by, is the sum of weights of all COLLAPSE operations minus the sum of offsets of all COLLAPSE operations. The first quantity is exactly W. From Lemma 1, the second quantity is at least $\frac{W+C-1}{2}$. This gives us the desired bounds on DS_{leaves} and DL_{leaves}. □

Lemma 5 *The difference in rank between the true ϕ-quantile of the original dataset and that of the output produced by the algorithm is at most $\frac{W-C-1}{2} + w_{max}$.*

Munro-Paterson Algorithm

	Number of buffers b					*Size of buffer k*					*Total memory bk*				
ϵ, N	10^5	10^6	10^7	10^8	10^9	10^5	10^6	10^7	10^8	10^9	10^5	10^6	10^7	10^8	10^9
0.100	11	14	17	21	24	98	123	153	96	120	1.1 K	1.7 K	2.6 K	2.0 K	2.9 K
0.050	11	14	17	20	23	98	123	153	191	239	1.1 K	1.7 K	2.6 K	3.8 K	5.5 K
0.010	9	11	14	17	21	391	977	1221	1526	954	3.5 K	10.7 K	17.1 K	25.9 K	20.0 K
0.005	8	11	14	17	20	782	977	1221	1526	1908	6.3 K	10.7 K	17.1 K	25.9 K	38.2 K
0.001	6	9	11	14	17	3125	3907	9766	12208	15259	18.8 K	35.2 K	107.4 K	170.9 K	259.4 K

Alsabti-Ranka-Singh Algorithm

	Number of buffers b					*Size of buffer k*					*Total memory bk*				
ϵ, N	10^5	10^6	10^7	10^8	10^9	10^5	10^6	10^7	10^8	10^9	10^5	10^6	10^7	10^8	10^9
0.100	280	892	2826	8942	28282	6	6	6	6	6	1.7 K	5.4 K	17.0 K	53.7 K	169.7 K
0.050	198	630	1998	6322	19998	11	11	11	11	11	2.2 K	6.9 K	22.0 K	69.5 K	220.0 K
0.010	88	280	892	2826	8942	52	52	51	51	51	4.6 K	14.6 K	45.5 K	144.1 K	456.0 K
0.005	62	198	630	1998	6322	105	103	101	101	101	6.5 K	20.4 K	63.6 K	201.8 K	638.5 K
0.001	26	88	280	892	2826	592	517	511	503	501	15.4 K	45.5 K	143.1 K	448.7 K	1415.8 K

New Algorithm

	Number of buffers b					*Size of buffer k*					*Total memory bk*				
ϵ, N	10^5	10^6	10^7	10^8	10^9	10^5	10^6	10^7	10^8	10^9	10^5	10^6	10^7	10^8	10^9
0.100	5	7	10	15	12	55	54	60	51	77	0.3 K	0.4 K	0.6 K	0.8 K	0.9 K
0.050	6	6	8	7	8	78	117	129	211	235	0.5 K	0.7 K	1.0 K	1.5 K	1.9 K
0.010	7	12	9	10	10	217	229	412	596	765	1.5 K	2.7 K	3.7 K	6.0 K	7.7 K
0.005	3	8	8	8	7	953	583	875	1290	2106	2.9 K	4.7 K	7.0 K	10.3 K	14.7 K
0.001	3	5	5	9	10	2778	3031	5495	4114	5954	8.3 K	15.2 K	27.5 K	37.0 K	59.5 K

Sampling followed by New Algorithm for 99.99% confidence

	Number of buffers b					*Size of buffer k*					*Total memory bk*				
ϵ, N	10^5	10^6	10^7	10^8	10^9	10^5	10^6	10^7	10^8	10^9	10^5	10^6	10^7	10^8	10^9
0.100	5	5	5	5	5	31	31	31	31	31	0.2 K	0.2 K	0.2 K	0.2 K	0.2 K
0.050	5	5	5	5	5	76	76	76	76	76	0.4 K	0.4 K	0.4 K	0.4 K	0.4 K
0.010	7	12	6	6	6	217	229	472	472	472	1.5 K	2.7 K	2.8 K	2.8 K	2.8 K
0.005	3	8	7	7	7	953	583	937	937	937	2.9 K	4.7 K	6.6 K	6.6 K	6.6 K
0.001	3	5	5	9	9	2778	3031	5495	4114	4943	8.3 K	15.2 K	27.5 K	37.0 K	44.5 K

Table 1: **Number of buffers b, size of buffer k and total memory bk.**

Proof: Since there are L leaves and each leaf buffer is worth k elements, there are a total of kL elements in the augmented dataset. The true ϕ'-quantile of the augmented dataset lies at position $\lceil \phi' kL \rceil$. However, the output could be the element at a position as small as $DS_{leaves} + 1$ or as large as $kL - DL_{leaves}$. Thus the difference between the true ϕ'-quantile and that of the output of the algorithm could be as large as $\lceil \phi' kL \rceil - DS_{leaves} - 1$ or $kL - DL_{leaves} - \lceil \phi' kL \rceil$. From Lemma 4, we deduce that

$$\lceil \phi' kL \rceil - DS_{leaves} - 1 \leq \lceil \phi' kL \rceil - DS_{top} + \frac{W - C + 1}{2} - 1$$

Substituting $\lceil \phi' kL \rceil - DS_{top} \leq w_{max}$ from Lemma 3, we obtain

$$\lceil \phi' kL \rceil - DS_{leaves} - 1 \leq \frac{W - C - 1}{2} + w_{max}$$

Exactly the same bound can be established for the quantity $kL - DL_{leaves} - \lceil \phi' kL \rceil$ using a similar argument. Therefore, the difference in ranks of the element output at the root and that of the true ϕ'-quantile of the augmented dataset is no more than $\frac{W-C-1}{2} + w_{max}$. Since the true ϕ'-quantile of the augmented dataset is the same as the true ϕ-quantile of the original dataset, we get the desired result. □

4.3 Munro-Paterson Algorithm

The Munro-Paterson algorithm requires two buffers at the leaf level and one buffer at each other level of the tree, except the root. Therefore, the height of the tree, including the root, can be at most b. The two children of the root are the inputs to the final OUTPUT operation. The collapsing scheme described by Munro and Paterson in the original paper stipulates that there be exactly 2^{b-1} leaves and that the final OUTPUT operation be carried out on two buffers with weight 2^{b-2}. We will assume the same restrictions while analyzing their algorithm.

For the Munro-Paterson tree, the height is b. The sum of weights of all COLLAPSE operations is $W = (b-2)2^{b-1}$. The total number of COLLAPSE operations is $C = 2^{b-1} - 2$. The heaviest COLLAPSE operation is $w_{max} = 2^{b-2}$. Plugging in these values into Lemma 5, we obtain that the difference between the rank of the output of the algorithm and that of the true ϕ-quantile is at most $\frac{W-C-1}{2} + w_{max} = (b-2)2^{b-2} + \frac{1}{2}$. This number should be less than ϵN, for the output to be an ϵ-approximate quantile.

What are the optimal values of b and k? Our objective is to minimize bk, the amount of memory needed, subject to the constraints $(b-2)2^{b-2} + \frac{1}{2} \leq \epsilon N$ and $k2^{b-1} \geq N$. The first constraint ensures that the approximation error is at most ϵ. The second constraint ensures that the number of

elements in all the leaf buffers combined is at least N (there are $L = 2^{b-1}$ leaves, each worth k elements).

In practice, the optimal values can be calculated by first computing the maximum integral b that satisfies $(b-2)2^{b-2} \leq \epsilon N$. Since b must lie between 1 and $\log \epsilon N$, binary search can be used. Then one can compute the smallest integral k that satisfies $k2^{b-1} \geq N$.

The values of b, k and the total memory required are listed for practical values of N and ϵ in Table 1.

4.4 Alsabti-Ranka-Singh Algorithm

For the Alsabti-Ranka-Singh algorithm, we assume that b is even. The weighted sum of all COLLAPSE operations is $W = \frac{b^2}{4}$. The total number of COLLAPSE operations is $C = \frac{b}{2}$. The weight of the heaviest COLLAPSE operation is $w_{max} = \frac{b}{2}$. The number of leaves is $L = \frac{b^2}{4}$. Plugging in values into Lemma 5, we obtain that the difference between the ranks of the output of the algorithm and that of the true ϕ-quantile of the original dataset is at most $\frac{W-C-1}{2} + w_{max}$, which simplifies to $\frac{b^2}{8} + \frac{b}{4} - \frac{1}{2}$. This number should be less than ϵN.

The optimal values for b and k can be obtained by minimizing bk, subject to the constraints $\frac{b^2}{8} + \frac{b}{4} - \frac{1}{2} \leq \epsilon N$ and $k\frac{b^2}{4} \geq N$. In practice, the optimal value can be obtained by computing the largest integral b that satisfies the first constraint and then computing the smallest integral k that satisfies the second one. The values of b, k and the total memory required for practical values of N and ϵ are listed in Table 1.

4.5 The New Algorithm

For the new collapsing policy, the values of W, C and w_{max} are functions of the height of the tree, which we denote by h. The Munro-Paterson tree has a height of at most b. The Alsabti-Ranka-Singh tree has a height of 2. However, the height of the new tree is not restricted by b.

For height $h \geq 3$, the number of leaves in the tree is $L = \binom{b+h-2}{h-1}$. The number of COLLAPSE operations is $C = \binom{b+h-3}{h-2} - 1$. The weighted sum of all COLLAPSE operations is $W = (h-2)\binom{b+h-2}{h-1} - \binom{b+h-3}{h-3}$. The weight of the heaviest COLLAPSE is $w_{max} = \binom{b+h-3}{h-2}$.

Our objective is to minimize bk subject to two constraints. The first constraint is $\frac{W-C-1}{2} + w_{max} \leq \epsilon N$, which is equivalent to $(h-2)\binom{b+h-2}{h-1} - \binom{b+h-3}{h-3} + \binom{b+h-3}{h-2} \leq 2\epsilon N$. The second constraint is $kL \geq N$, which is equivalent to $k\binom{b+h-2}{h-1} \geq N$. In practice, optimal values for b and k can be computed by trying out different values of b in the range 1 and 30. For each b, compute the largest integral h that satisfies the first constraint. Then, compute the smallest integral k that satisfies the second constraint. Identify that value of b that minimizes bk.

The values of b, k and memory requirements for the new algorithm are listed in Table 1 for practical values of N and ϵ.

4.6 Performance Comparison

From Table 1, it can be deduced that the new algorithm is always better in terms of space required. Figure 7 shows how the amount of memory varies for increasing N, as ϵ is held constant at 0.01. The new algorithm is a clear winner.

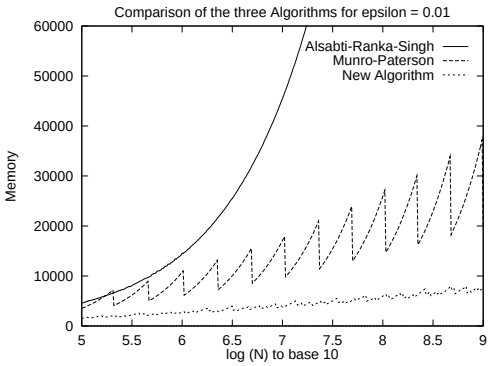

Figure 7: **Memory requirements for $\epsilon = 0.01$.**

The kinks in the curve for the Munro-Paterson algorithm merit an explanation. That algorithm stipulates that there be exactly 2^{b-1} leaves in the tree. Recall from Section 4.3 that we minimize bk, the total memory requirement, subject to two constraints, $(b-2)2^{b-2} + \frac{1}{2} \leq \epsilon N$ and $k2^{b-1} \geq N$. As N grows, the largest b that satisfies the first inequality increases. As b increases by 1 for two successive values of N, the smallest value of k that satisfies the second constraint diminishes by roughly half, resulting in a similar reduction in the product bk, the total memory required.

4.7 Multiple Quantiles

The analysis culminating in Lemma 5 holds for *any* number of quantiles simultaneously. Therefore, any algorithm in our framework computes multiple quantiles with the same approximation guarantees at no extra cost.

4.8 Space Complexity

The space complexity for the new algorithm can be determined by restricting $b = h$. The first constraint, i.e., $\frac{W-C-1}{2} + w_{max} \leq \epsilon N$ can be weakened to $2b\binom{2b}{b} \leq \epsilon N$, yielding $b = O(\log \epsilon N)$. Together with the second constraint, we obtain $k = O(\frac{1}{\epsilon} \log \epsilon N)$, for an overall space complexity of $O(\frac{1}{\epsilon} \log^2 \epsilon N)$. Asymptotically, the Munro-Paterson algorithm also has the same space complexity.

Theorem 1 *It is possible to compute an ϵ-approximate ϕ-quantile of a dataset of size N in a single pass using only $O(\frac{1}{\epsilon} \log^2 \epsilon N)$ memory, for any ϕ.* □

In Figure 7, the curves for the new algorithm and the Munro-Paterson algorithm are actually parabolic, though they appear to be almost straight lines for a limited range of $\log N$. Figure 7 shows that the constant for the new algorithm is clearly better. For the Alsabti-Ranka-Singh algorithm, $b = O(\sqrt{\epsilon N})$ and $k = O(\frac{1}{\epsilon})$, for a total memory requirement of $O(\sqrt{\frac{N}{\epsilon}})$. This explains the exponential curve for their algorithm in Figure 7.

4.9 Parallel Version

The new algorithm is easily parallelized by partitioning the input stream (either statically or dynamically) among the processors. The root nodes of each partition are concatenated to form an input stream for the final OUTPUT phase that outputs the different quantiles. This approach scales linearly with the degree of parallelism except for the final

Sampling followed by New Algorithm

	$\alpha\epsilon$			*Sample size S*			*Number of buffers b*			*Size of buffer k*			*Total memory bk*		
ϵ, δ	10^{-2}	10^{-3}	10^{-4}	10^{-2}	10^{-3}	10^{-4}	10^{-2}	10^{-3}	10^{-4}	10^{-2}	10^{-3}	10^{-4}	10^{-2}	10^{-3}	10^{-4}
0.100	0.0451	0.0446	0.0521	0.3 K	0.4 K	0.5 K	3	4	5	42	36	31	0.13 K	0.14 K	0.15 K
0.050	0.0255	0.0293	0.0272	1.1 K	1.5 K	2.0 K	4	5	5	79	71	76	0.32 K	0.35 K	0.38 K
0.010	0.0063	0.0057	0.0064	26.5 K	38.0 K	49.5 K	6	6	6	408	447	472	2.45 K	2.68 K	2.83 K
0.005	0.0031	0.0034	0.0032	106.0 K	152.0 K	198.1 K	6	7	7	962	893	937	5.77 K	6.25 K	6.56 K
0.001	0.0007	0.0007	0.0007	2.6 M	3.8 M	5.0 M	8	8	9	4964	5326	4943	39.71 K	42.61 K	44.49 K

Table 2: *Memory required by first sampling and then running the new approximate quantile finding algorithm for different ϵ and δ.*

phase. For moderate degrees of parallelism ($\approx$ 24 nodes), the final phase is insignificant for large data sets. For higher degrees of parallelism ($>$ 100), the outputs of the root gates can be partitioned arbitrarily to a smaller number of processors. The new roots can now be combined in a final step at a single processor.

5 A Sampling based Algorithm

In this section, we show how the deterministic algorithm presented in Section 3 can be coupled with sampling to obtain remarkable reductions in space for large N, the size of the dataset. Interestingly, the space required becomes independent of N.

We tackle the following problem: *Given N, ϕ, ϵ and δ, design a single pass algorithm to compute an ϵ-approximate ϕ-quantile of a dataset of size N using as little main memory as possible such that the output is guaranteed to be correct with probability at least $1-\delta$.*

We require the following inequality due to Hoeffding [19]:

Lemma 6 (Hoeffding's Inequality) *Let $X_1, X_2, \ldots, X_n$ be independent random variables with $0 \leq X_i \leq 1$ for $i = 1, 2, \ldots, n$. Let $X = \sum_{i=1}^{n} X_i$. Let $\mathbf{E}X$ denote the expectation of X. Then, for any $\lambda > 0$, $Pr[X - \mathbf{E}X \geq \lambda] \leq \exp \frac{-2\lambda^2}{n}$.* □

The big picture is the following. Let $\epsilon = \epsilon_1 + \epsilon_2$. We will later describe how ϵ_1 and ϵ_2 are to be fixed. Assume that it is possible to choose a sample, say, of size S, from a total of N elements, such that it is guaranteed, with probability at least $1-\delta$, that the set of elements between the pair of positions $\lceil(\phi \pm \epsilon_1)S\rceil$ in the sorted sequence of the sample are a subset of the set of elements between the pair of positions $\lceil(\phi \pm \epsilon)N\rceil$ in the sorted sequence of the original dataset. Then, we can run the new deterministic algorithm in Section 3 on the samples, stipulating an accuracy of ϵ_1. The algorithm guarantees that it outputs a quantile that is at most $\epsilon_1 S$ elements away from the true quantile of the sample. Coupled with the guarantee that such an element is no further than another ϵ_2 elements away in the original dataset, with probability at least $1-\delta$, the whole scheme works.

The crucial question is: Given N, ϵ_1, ϵ_2 and δ, how big a sample do we require?

Lemma 7 *Let $\epsilon = \epsilon_1 + \epsilon_2$. A total of $S \geq \frac{1}{2\epsilon_2^2}\log(2\delta^{-1})$ samples drawn from a population of N elements are enough to guarantee that the set of elements between the pair of positions $\lceil(\phi \pm \epsilon_1)S\rceil$ in the sorted sequence of the samples is a subset of the set of elements between the pair of positions $\lceil(\phi \pm \epsilon)N\rceil$ in the sorted sequence of the N elements.*

Proof: We say that a sample is *bad* if it does not satisfy the property mentioned in the lemma. Otherwise it is *good*.

Let sets $N_{\phi-\epsilon}$ and $N_{\phi+\epsilon}$ denote the set of elements preceding the $(\phi-\epsilon)$-quantile and succeeding the $(\phi+\epsilon)$-quantile among N elements respectively.

A sample of size S is bad if and only if more than $\lceil(\phi - \epsilon_1)S\rceil$ elements are drawn from $N_{\phi-\epsilon}$ or more than $S - \lceil(\phi + \epsilon_1)S\rceil$ elements are drawn from $N_{\phi+\epsilon}$.

The probability that more than $\lceil(\phi-\epsilon_1)S\rceil$ elements are drawn from $N_{\phi-\epsilon}$ can be bounded as follows. The process of drawing a sample of size S from a population of N elements corresponds to S independent Bernoulli trials (coin tosses) with probability $\phi - \epsilon$. The expected number of successful trials is $(\phi-\epsilon)S$. If the number of successful trials is more than $\lceil(\phi-\epsilon_2)S\rceil$, the sample is bad. The probability that it occurs is less than $e^{-2\epsilon_2^2 S^2}$, from Hoeffding's inequality.

A symmetric argument shows that the probability that more than $S - \lceil(\phi+\epsilon_1)S\rceil$ elements are drawn from $N_{\phi+\epsilon}$ is also less than $e^{-2\epsilon_2^2 S^2}$.

Thus the probability δ that the sample is bad is at most $2e^{-2\epsilon_2^2 S^2}$. Solving for S, we obtain $S \geq \frac{1}{2\epsilon_2^2}\log(2\delta^{-1})$. □

Interestingly, the number of samples S is independent of N.

5.1 Fixing ϵ_1 and ϵ_2

Let $\epsilon_1 = \alpha\epsilon$, for some $\alpha \in (0,1)$. Then, $\epsilon_2 = (1-\alpha)\epsilon$. As α approaches one, the number of samples increases. As α approaches zero, the approximation guarantee required of the deterministic algorithm increases. In either case, the memory requirements blow up. Clearly, there is an optimal value of α that minimizes memory.

How do we compute the optimal value of α? For practical values of ϵ and δ, one can compute the overall memory requirements for different values of $\alpha \in [0.2, 0.8]$ in increments of 0.001 and figure out where the minimum value lies. See Table 2 for memory requirements for different ϵ and δ.

Theoretically, the complexity can be determined by fixing α to be a constant, say 0.5. Then, the number of samples is $O(\epsilon^{-2}\log\delta^{-1})$. From Section 4.8, the new deterministic algorithm requires $O(\epsilon^{-1}\log^2\epsilon N)$ space for N elements, the space requirements for running it on the samples is $O(\epsilon^{-1}\log^2[\epsilon^{-1}\log\delta^{-1}])$.

Theorem 2 *It is possible to compute an ϵ-approximate ϕ-quantile with probability of success at least δ for an arbitrary sized dataset in a single pass using $O(\epsilon^{-1}\log^2\epsilon^{-1} + \epsilon^{-1}\log^2\log\delta^{-1})$ space.* □

5.2 To Sample or Not to Sample

When is it a good idea to sample and then use the new quantile finding algorithm? For a fixed value of ϵ and δ, there is a threshold value for N above which we are better off sampling. The threshold value can be obtained by first computing the minimum memory required for a fixed value of ϵ and δ, using the technique outlined earlier in this section, and then computing the maximum N whose ϵ-approximate quantile can be computed in the same amount of memory. See Figure 8 for the threshold values for ϵ between 0.1 and 0.0001.

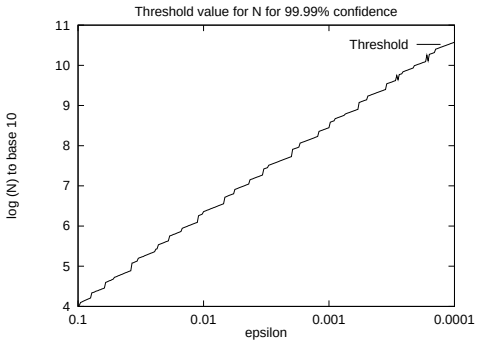

Figure 8: **Threshold value for N for confidence 99.99%.**

5.3 Multiple Quantiles

If we want to compute p different quantiles, each with an error bound of ϵ with confidence at least $1-\delta$ that all quantiles are ϵ-approximate, then the following approach works: Let $\epsilon = \epsilon_1 + \epsilon_2$. Let $S = \frac{1}{2}\epsilon_2^{-2}\log(2p\delta^{-1})$. We choose S samples and feed them to the deterministic algorithm, stipulating an accuracy of ϵ_1. We read off p quantiles instead of a single quantile in the final COLLAPSE operation. *All* quantiles are guaranteed, with probability at least $1-\delta$, to be ϵ-approximate. Optimal values for ϵ_1 and ϵ_2 can be calculated using the same technique outlined earlier in this section.

The proof for the correctness of the approach is simple: Let $\delta' = \delta/p$. Using Lemma 7, we can argue 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 2, 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 many quantiles at once is not too great. Note that multiple quantiles do not require any extra memory if we run a deterministic algorithm from Section . However, with random sampling, probabilistic guarantees on error bounds on multiple quantiles necessitates a slightly larger sized sample, which in turn increases memory requirements by a small $O(\log^2 \log p)$ factor.

6 Simulation Results

Our analysis of the new deterministic algorithm is for the worst case scenario. In practice, it is unlikely that any approximate quantile output at the root gate really deviates from its exact value by as much as ϵ. To ratify our claim, we present experimental results on datasets with different sequences of ranks of elements. Note that the exact values of data elements are of no consequence. It is the permutation of their ranks in sorted order that matters.

We study two kinds of permutations: sorted and random. We fix $\epsilon = 10^{-3}$ and compute 15 quantiles at positions $\frac{q}{16}$ for $q \in \{1, 2, \ldots, 15\}$. The results are tabulated in Table 3.

	Observed ϵ					
	Sorted			*Random*		
q, N	10^5	10^6	10^7	10^5	10^6	10^7
1	0.00008	0.00011	0.00013	0.00022	0.00010	0.00006
2	0.00007	0.00004	0.00005	0.00031	0.00010	0.00006
3	0.00005	0.00009	0.00007	0.00035	0.00015	0.00008
4	0.00014	0.00004	0.00001	0.00027	0.00012	0.00007
5	0.00003	0.00001	0.00006	0.00024	0.00011	0.00007
6	0.00006	0.00011	0.00004	0.00036	0.00012	0.00006
7	0.00001	0.00001	0.00007	0.00034	0.00012	0.00006
8	0.00009	0.00005	0.00002	0.00028	0.00014	0.00006
9	0.00022	0.00004	0.00007	0.00034	0.00013	0.00011
10	0.00013	0.00003	0.00006	0.00037	0.00013	0.00007
11	0.00002	0.00002	0.00004	0.00029	0.00017	0.00006
12	0.00003	0.00001	0.00003	0.00023	0.00021	0.00011
13	0.00009	0.00001	0.00000	0.00025	0.00021	0.00007
14	0.00002	0.00002	0.00000	0.00021	0.00019	0.00008
15	0.00003	0.00000	0.00001	0.00020	0.00021	0.00008

Table 3: **Final error obtained by running the new algorithm on different sized datasets with $\epsilon = 0.001$.**

It is clear that the actual error obtained is much better than the ϵ we started out with.

7 Conclusions and Future Work

We have described algorithms for computing approximate quantiles with guaranteed error bounds in a single pass over large online or disk-resident datasets. The algorithms require substantially less memory than previously published results. We have shown how further reduction in memory can be achieved by coupling sampling with an approximate quantile computing algorithm. More interestingly, above certain dataset sizes, the memory requirement becomes independent of the size of the dataset at the cost of probabilistic confidence in the approximation guarantee. We have derived important parameters of the algorithms over varying input sizes, error bounds and quantiles needed. The memory requirements, accuracy guarantees and experimental results for the new algorithms show that it is now practical and safe to both deliver order statistics aggregation to database users, and also to exploit them for query optimization and data partitioning.

The clearly delineated tradeoff between the accuracy of any quantile and the memory requirements poses a challenging choice to users of the algorithm. In some cases, further analysis may be able to quantify the cost benefit of the accuracy for particular applications. In other cases, the user's judgment may be required to choose an appropriate error bound.

Practical implementations in "real" Relational Database Management Systems will be challenged by the need to support additional parameters (ϕ, ϵ and δ) for SQL column functions which have only a single parameter up to this point. It will also require some ingenuity to handle multiple quantiles efficiently on the same column (e.g., SELECT QUANTILE (0.35, col1), QUANTILE (0.50, col1), ...). In addition, the non-trivial memory requirements will probably require some tricky extensions to the GROUP BY execution environment of the system.

References

[1] 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.

[2] G. Piatetsky-Shapiro, "Accurate Estimation of the Number of Tuples Satisfying a Condition", *in ACM SIGMOD 84*, Boston, June 1984.

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

[4] "DB2 MVS", .

[5] "Informix", .

[6] 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.*, pp. 280–291, Miami Beach, 1991.

[7] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan, "Time Bounds for Selection", *in J. Comput. Syst. Sci.*, vol. 7, pp. 448–461, 1973.

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

[9] D. Dor, *Selection Algorithms*, PhD thesis, Tel-Aviv University, 1995.

[10] D. Dor and U. Zwick, "Selecting the Median", *in Proc. 6th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 28–37, 1995.

[11] D. Dor and U. Zwick, "Finding the αn^{th} Largest Element", *Combinatorica*, vol. 16, pp. 41–58, 1996.

[12] D. Dor and U. Zwick, "Median Selection Requires $(2+\epsilon)n$ Comparisons", Technical Report 312/96, Department of Computer Science, Tel-Aviv University, Apr. 1996.

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

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

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

[16] R. Jain and I. Chlamtac, "The P^2 Algorithm for Dynamic Calculation for Quantiles and Histograms without Storing Observations", *CACM*, vol. 28, pp. 1076–1085, 1985.

[17] 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.

[18] 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.

[19] W. Hoeffding, "Probability Inequalities for Sums of Bounded Random Variables", *American Statistical Association Jornal*, pp. 13–30, Mar. 1963.