Continuously Maintaining Quantile Summaries of the Most Recent N Elements over a Data Stream

Xuemin Lin
University of New South Wales
Sydney, Australia

Hongjun Lu
Hong Kong University of Sci. & Tech.
Hong Kong

Jian Xu
University of New South Wales
Sydney, Australia

Jeffrey Xu Yu
Chinese University of Hong Kong
Hong Kong

Abstract

Statistics over the most recently observed data elements are often required in applications involving data streams, such as intrusion detection in network monitoring, stock price prediction in financial markets, web log mining for access prediction, and user click stream mining for personalization. Among various statistics, computing quantile summary is probably most challenging because of its complexity. In this paper, we study the problem of continuously maintaining quantile summary of the most recently observed N elements over a stream so that quantile queries can be answered with a guaranteed precision of ϵN. We developed a space efficient algorithm for pre-defined N that requires only one scan of the input data stream and $O(\frac{\log(\epsilon^2 N)}{\epsilon} + \frac{1}{\epsilon^2})$ space in the worst cases. We also developed an algorithm that maintains quantile summaries for most recent N elements so that quantile queries on any most recent n elements ($n \leq N$) can be answered with a guaranteed precision of ϵn. The worst case space requirement for this algorithm is only $O(\frac{\log^2(\epsilon N)}{\epsilon^2})$. Our performance study indicated that not only the actual quantile estimation error is far below the guaranteed precision but the space requirement is also much less than the given theoretical bound.

1 Introduction

Query processing against data streams has recently received considerable attention and many research breakthroughs have been made, including processing of relational type queries [4, 6, 8, 15, 20], XML documents [7, 14], data mining queries [3, 16, 22], v-optimal histogram maintenance [12], data clustering [13], etc. In the context of data streams, a query processing algorithm is considered efficient if it uses very little space, reads each data element just once, and takes little processing time per data element.

Recently, Greenwald and Khanna reported an interesting work on efficient quantile computation [10]. A *ϕ-quantile* ($\phi \in (0, 1]$) of an ordered sequence of N data elements is the element with rank $\lceil \phi N \rceil$. It has been shown that in order to compute exactly the ϕ-quantiles of a sequence of N data elements with only p scans of the data sequence, any algorithm requires a space of $\Omega(N^{1/p})$[19]. While quite a lot of work has been reported on providing approximate quantiles with reducing space requirements and one scan of data [1, 2, 17, 18, 9], techniques reported in [10] (Referred as GK-algorithm hereafter) are able to maintain an *ϵ-approximate* quantile summary for a data sequence of N elements requiring only $O(\frac{1}{\epsilon}\log(\epsilon N))$ space in the worst case and one scan of the data. A quantile summary is ϵ-approximate if it can be used to answer any quantile query within a precision of ϵN. That is, for any given rank r, an ϵ-approximate summary returns a value whose rank $r\prime$ is guaranteed to be within the interval $[r - \epsilon N, r + \epsilon N]$.

While quantile summaries maintained by GK-algorithm have their applications, such summaries do not have the concept of $aging$, that is, quantiles are computed for all N data elements seen so far, including those seen long time ago. There are a wide range of applications where data elements seen early could be outdated and quantile summaries for the most recently seen data elements are more important. For example, the top ranked Web pages among most recently accessed N pages should produce more accurate web page access prediction than the top ranked pages among all pages accessed so far as users' interests are changing. In financial market, investors are often interested in the price quantile of the most recent N bids. Detar *et. al.* considered such a problem of maintaining statistics over data streams with regard to the last N data elements seen so far and referred to it as the *sliding window* model [5]. However, they only provided algorithms for maintaining aggregation statistics, such as computing the number of 1's and the sum of the last N positive integers. Apparently, maintaining order statistics (e.g. quantile summary) is more complex than those simple aggregates. Several approximate join processing [4] and histogram techniques [11] based on the sliding window model have also been recently reported but they are not relevant to maintaining order statistics.

Motivated by the above, we studied the problem of space-efficient one-pass quantile summaries over the most recent N tuples seen so far in data streams. Different from the GK-algorithm where tuples in a quantile summary are merged based on capacities of tuples when space is needed for newly arrived data elements, we maintain quantile summary in partitions based on time stamps so that outdated data elements can be deleted from the summary without affecting the precision. Since quantile information is local to each partition, a novel merge technique was developed to produce an ϵ-approximate summary from partitions for all N data elements. Moreover, we further extended the technique to maintain a quantile summary for the most recent N data elements in such a way that quantile estimates can be obtained for the n most recent elements for any $n \leq N$.

To the best of our knowledge, no similar work has been reported in the literature. The contribution of our work can be summarized as follows.

- We extended the sliding window model to an n-of-N model. Under the sliding window model, quantile summaries are maintained for the N most recently seen elements in a data stream. Under the n-of-N model, quantile summary with a sliding window N can produce quantile estimates for any n ($n \leq N$) most recent elements. In other words, the sliding window model can be viewed as a special case of the n-of-N model.
- For the sliding window model, we developed a one-pass deterministic ϵ-approximate algorithm to maintain quantiles summaries. The algorithm requires a space of $O(\frac{\log(\epsilon^2 N)}{\epsilon} + \frac{1}{\epsilon^2})$.
- For the general n-of-N model, we developed another one-pass deterministic approximate algorithm that requires a space of $O(\frac{1}{\epsilon^2}\log^2(\epsilon N))$. The algorithm is ϵ-approximate for every $n \leq N$.
- The technique developed to merge multiple ϵ-approximate quantile summaries into a single ϵ-approximate quantile summary can be directly used for distributed and parallel quantile computation.

The rest of this paper is organized as follows. In section 2, we present some background information in quantile computation. Section 3 and 4 provide our algorithms for the sliding window model and the n-of-N model, respectively. Discussions of related work and applications are briefly presented in section 5. Results of a comprehensive performance study are discussed in section 6. Section 7 concludes the paper.

2 Preliminaries

In this section, we first introduce the problem of quantile computation over a sequence of data, followed by a number of computation models. Finally we review some most closely related work.

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
12	10	11	10	1	10	11	9	6	7	8	11	4	5	2	3

Figure 1. a data stream with arrival time-stamps

2.1 Quantile and Quantile Sketch

In this paper, for notation simplification we will always assume that a data element is a value and an ordered sequence of data elements in quantile computation always means an increasing ordering of the data values. Furthermore, we always use N to denote the number of data elements.

Definition 1 (Quantile). *A* ϕ-quantile *(*$\phi \in (0, 1]$*) of an ordered sequence of* N *data elements is the element with rank* $\lceil \phi N \rceil$*. The result of a quantile query is the data element for a given rank.*

Example 1. *Figure 1 shows a sample sequence of data generated from a data stream where each data element is represented by a value and the arrival order of data elements is from left to right. The total number of data elements in the sequence is 16. The sorted order of the sequence is 1, 2,3,4,5,6,7,8,9,10,10,10,11,11,11,12. So, 0.5-quantile returns the element ranked 8 (=0.5*16), which is 8; and 0.75-quantile returns an element 10 which ranks 12 in the sequence.*

Munro and Paterson showed that any algorithm that computes the exact ϕ-quantiles of a sequence of N data elements with only p scans of the data sequence requires a space of $\Omega(N^{1/p})$[19]. In the context of data streams, we are only able to see the data once. On the other hand, the size of a data steam is theoretically infinite. It becomes impractical to compute exact quantile for data streams. For most applications, keeping an quantile summary so that quantile queries can be answered with bounded errors is indeed sufficient.

Definition 2 (ϵ**-approximate).** *A quantile summary for a data sequence of* N *elements is* ϵ*-approximate if, for any given rank* r*, it returns a value whose rank* r' *is guaranteed to be within the interval* $[r - \epsilon N, r + \epsilon N]$*.*

Generally, a quantile summary may be in any form. Definition 2 does not lead to a specific query algorithm to find a data element from an ϵ-approximate summary within the precision of ϵN. To resolve this, a well-structured ϵ-approximate summary is needed to support approximate quantile queries. In our study, we use *quantile sketch* or *sketch* for brevity, a data structure proposed in [10] as the quantile summaries of data sequences.

Definition 3 (Quantile Sketch). *A* quantile sketch S *of an ordered data sequence* D *is defined as an ordered sequence of tuples* $\{(v_i, r_i^-, r_i^+) : 1 \leq i \leq m\}$ *with the following properties.*

1. *Each $v_i \in D$.*
2. *$v_i \leq v_{i+1}$ for $1 \leq i \leq m-1$.*
3. *$r_i^- < r_{i+1}^-$ for $1 \leq i \leq m-1$.*
4. *For $1 \leq i \leq m$, $r_i^- \leq r_i \leq r_i^+$ where r_i is the rank of v_i in D.*

Example 2. *For the sequence of data shown in Figure 1, {(1,1,1), (2,2,9), (3,3,10),(5,4,10),(10,10,10),(12,16,16)} is an example quantile sketch consisting of 6 tuples.*

Greenwald and Khanna proved the following theorem:

Theorem 1. *For a sketch S defined in Definition 3, if:*

1. $r_1^+ \leq \epsilon N + 1$,
2. $r_m^- \geq (1-\epsilon)N$,
3. *for* $2 \leq i \leq m$, $r_i^+ \leq r_{i-1}^- + 2\epsilon N$.

*then S is ϵ-*approximate. *That is, for each $\phi \in (0,1]$, there is a (v_i, r_i^-, r_i^+) in S such that $\lceil \phi N \rceil - \epsilon N \leq r_i^- \leq r_i^+ \leq \lceil \phi N \rceil + \epsilon N$.*

Theorem 1 states that for each rank r, we are able to find a tuple (v_i, r_i^-, r_i^+) from such an ϵ-approximate sketch by a linear scan, such that the rank r_i of v_i is within the precision of ϵN, that is, $r - \epsilon N \leq r_i \leq r + \epsilon N$. In the rest of paper, we will use the three conditions in Theorem 1 to define an ϵ-approximate sketch.

Example 3. *It can be verified that the sketch provided in Example 2 is 0.25-approximate with respect to the data stream in Figure 1. On the other hand, the sketch {(1,1,1), (3,2,10), (10,10,10), (12, 16, 16)} is 0.2813-approximate.*

Note that the three conditions presented in Theorem 1 are more general than those originally in [10] due to our applications to the most recent N elements. However, the proof techniques in Proposition 1 and Corollary 1 [10] may lead to a proof of Theorem 1; we omit the details from this paper.

2.2 Quantile Sketches for Data Streams

Quantile summaries, or in our case, quantile sketch can be maintained for data streams under different computation models.

Data stream model. Most previous work are working with the data stream model. That is, a sketch is maintained for all N data items seen so far.

Sliding window model. Under sliding window model, a sketch is maintained for the most recently seen N data elements. That is, for any $\phi \in (0,1]$, we compute ϕ-quantiles against the N most recent elements in a data stream seen so far, where N is a pre-fixed number.

n-of-N model. Under this model, a sketch is maintained for N most recently seen data elements. However, quantile queries can be issued against any $n \leq N$. That is, for any $\phi \in (0,1]$ and any $n \leq N$, we can return ϕ-quantiles among the n most recent elements in a data stream seen so far.

Consider the data stream in Figure 1.

- Under the data stream model, since a sketch is maintained for all N data elements seen so far, a 0.5-quantile returns 10 at time t_{11}, and 8 at time t_{15}.
- With the width N of the sliding window being 12, a 0.5-quantile returns 10 at time t_{11}, and 6 at time t_{15}, which is ranked sixth in the most recent $N = 12$ elements.
- Assume that the sketch is maintained since t_0 with the n-of-N model, at time t_{15}, quantile queries can be answered for any $1 \leq n \leq 16$. A 0.5-quantile returns 6 for $n = 12$ and 3 for $n = 4$.

It can be seen that different sketches have different applications. For example, in databases, a sketch maintained under the data stream model is useful for estimating the sizes of relational operations which is required for query optimization. For a Web server, it is more appropriate to maintain a sketch on accessed Web pages under the sliding window model so that a page access prediction for cache management can be based on most recent user access patterns. In a security house, a sketch of bid/ask prices maintained under the n-of-N model is more appropriate so that it can answer quantile queries from clients with different investment strategies.

2.3 Maintaining ϵ-Approximate Sketches

In our study, we are interested in deterministic techniques with performance guarantees. Such algorithms are only available under the data stream model or the model of disk residence data.

Manku, Rajagoipalan and Lindsay [17] developed the first deterministic one-scan algorithm, with polylogarithmic space requirement, for approximately computing ϕ-quantiles with the precision guarantee of ϵN. Greenwald-Khanna algorithm [10] reduced the space complexity to $O(\frac{1}{\epsilon}\log(\epsilon N))$ for a data stream with N elements seen so far. The GK-algorithm [10] maintains a sketch by one-pass scan of a data stream to approximately answer quantile queries. A generated sketch is guaranteed ϵ-approximate.

For presentation simplification, the GK-algorithm uses two parameters g_i and Δ_i to control r_i^- and r_i^+ for each tuple (v_i, r_i^-, r_i^+) in a generated sketch such that for each i,

- $\sum_{j \leq i} g_j \leq r_i \leq \sum_{j \leq i} g_j + \Delta_i$,
- $r_i^- = \sum_{j \leq i} g_j$, (a)
- $r_i^+ = \sum_{j \leq i} g_j + \Delta_i$. (b)

The GK-algorithm maintains the following invariants to ensure that a generated sketch $\{(v_i, r_i^-, r_i^+) : 1 \leq i \leq m\}$ is ϵ-approximate.

Invariant 1: For $2 \leq i \leq m$, $g_i + \Delta_i < 2\epsilon N$.

Invariant 2: v_1 is the first element in the ordered data stream.

Invariant 3: v_n is the last element in the ordered data stream.

2.4 Challenges

Note that in the sliding window model, the actual contents of the most recent N tuples change upon new elements arrive, though N is fixed. This makes the existing summary techniques based on a whole dataset not trivially applicable.

Example 4. *GK-algorithm generates the following sketch regarding the data stream as depicted in Figure 1 if* $\epsilon = 0.5$.

$$\{(1, 1, 1), (10, 9, 9), (12, 16, 16)\}$$

The data elements involved in the sketch are 1, 10, *and* 12. *If we want to compute the quantiles against the most recent* 4 *elements (i.e.,* 2, 3, 4, 5*), this sketch is useless.*

Clearly, it is desirable that the data elements outside a sliding window should be removed from our considerations. As the contents in a sliding window continuously change upon new elements arrive, it seems infeasible to remove the exact outdated elements without using a space of $O(N)$. Therefore, the challenges are to develop a space-efficient technique to continuously partition a data stream and then summarize partitions to achieve high approximate accuracy. The quantile summary problem for n-of-N model seems even harder.

3 One-Pass Summary for Sliding Windows

In this section, we present a space-efficient summary algorithm for continuously maintaining an ϵ-approximate sketch under a sliding window. The basic idea of the algorithm is to continuously divide a stream into the buckets based on the arrival ordering of data elements such that:

- Data elements in preceding buckets are generated earlier than those in later buckets.
- The capacity of each bucket is $\lfloor \frac{\epsilon N}{2} \rfloor$ to ensure ϵ-approximation.
- The algorithm issues a new bucket only after the preceding blocks are full.
- For each bucket, we maintain an $\frac{\epsilon}{4}$-approximate sketch continuously by GK-algorithm instead of keeping all data elements.
- Once a bucket is full, its $\frac{\epsilon}{4}$-approximate sketch is compressed into an $\frac{\epsilon}{2}$-approximate sketch with a space of $O(\frac{1}{\epsilon})$.
- The oldest bucket is *expired* if currently the total number of elements is $N + 1$; consequently the sketch of the oldest bucket is removed.
- Merge local sketches to approximately answer a quantile query.

Figure 2 illustrates the algorithm. Note that GK-algorithm has been applied to our algorithm only because it has the smallest space guarantee among the existing techniques. Our algorithm will be able to accommodate any new algorithms for maintaining an ϵ-approximate sketch.

The rest of the section is organized as follows. We successively present our novel merge technique, compress technique, sketch maintenance algorithm, and query algorithm.

3.1 Merge Local Sketches

Suppose that there are l data streams D_i for $1 \leq i \leq l$, and each D_i has N_i data elements. Further suppose that each S_i ($1 \leq i \leq l$) is an η-approximate sketch of D_i. In this subsection, we will present a novel technique to merge these l sketches such that the merged sketch is η-approximate with respect to $\cup_{i=1}^{l} D_i$. This technique is the key to ensure ϵ-approximation of our summary technique, although used only in the query part.

Algorithm 1 depicts the merge process. Each S_i (for $1 \leq i \leq l$) is represented as $\{(v_{i,j}, r^-_{i,j}, r^+_{i,j}) : 1 \leq j \leq |S_i|\}$.

Algorithm 1 Merge

Input:

$\{(S_i, N_i) : 1 \leq i \leq l\}$; each S_i is η-approximate.

Output:

S_{merge}.

Description:

```
S_merge := ∅; r_0^- := 0; k := 0;
for 1 ≤ i ≤ l do
    r_{i,0}^- := 0;
end for
while ∪_{i=1}^l S_i ≠ ∅ do
    choose the tuple (v_{i,j}, r_{i,j}^-, r_{i,j}^+) with the smallest v_{i,j};
    S_i := S_i − {(v_{i,j}, r_{i,j}^-, r_{i,j}^+)};
    k := k + 1; v_k := v_{i,j};
    r_k^- := r_{k−1}^- + r_{i,j}^- − r_{i,j−1}^-;
    if k = 1 then
        r_k^+ := η Σ_{i=1}^k N_i + 1
    else
        r_k^+ := r_{k−1}^- + 2η Σ_{i=1}^k N_i;
        S_merge := S_merge ∪ {(v_k, r_k^-, r_k^+)}.
    end if
end while
```

We first prove that S_{merge} is a sketch of $\cup_{i=1}^{l} D_i$. This is an important issue. Suppose that for a given rank r, we can find a tuple (v_k, r^-_k, r^+_k) in S_{merge} such that $r - \epsilon N \leq r^-_k \leq r^+_k \leq r + \epsilon N$. There is no guarantee that r_k (the rank of v_k) is within $[r - \epsilon N, r + \epsilon N]$ unless r_k is between r^-_k and r^+_k.

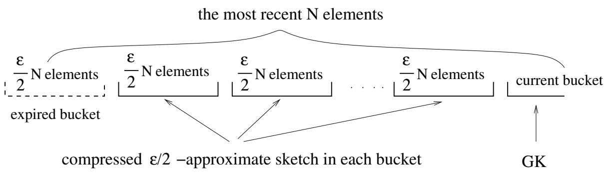


Figure 2. our summary technique

Lemma 1. *Suppose that there are l data streams D_i for $1 \le i \le l$ and each D_i has N_i data elements. Suppose that each S_i ($1 \le i \le l$) is an η-approximate sketch of D_i. Then, S_{merge} generated by the algorithm Merge on $\{S_i : 1 \le i \le l\}$ is a sketch of $\cup_{i=1}^{l} D_i$ which has $\sum_{i=1}^{l} |S_i|$ tuples.*

Proof. It is immediate that S_{merge} has $\sum_{i=1}^{l} |S_i|$ tuples.

Based on the algorithm Merge, the first 3 properties in the sketch definition can be immediate verified. We need only to prove that for each $(v_k, r_k^-, r_k^+) \in S_{merge}$, $r_k^- \le r_k \le r_k^+$ where r_k is the rank of v_k in the ordered $\cup_{i=1}^{k} D_i$.

For each $(v_k, r_k^-, r_k^+) \in S_{merge}$, $(v_{i,j_{k,i}}, r_{i,j_{k,i}}^-, r_{i,j_{k,i}}^+)$ denotes the last tuple of S_i ($1 \le i \le l$) merged into S_{merge} no later than obtaining (v_k, r_k^-, r_k^+). Note that if $j_{k,i} = 0$ (i.e. no tuple in S_i has been merged yet), then we make $v_{i,0} = -\infty$ and $r_{i,0}^- = r_{i,0} = r_{i,0}^+ = 0$.

It should be clear that $v_k \ge v_{i,j_{k,i}}$ for $1 \le i \le l$. Consequently, $r_k \ge \sum_{i=1}^{l} r_{i,j_{k,i}} \ge \sum_{i=1}^{l} r_{i,j_{k,i}}^-$. According to the algorithm Merge (line 9), it can be immediately verified that for each k,

$$r_k^- = \sum_{i=1}^{l} r_{i,j_{k,i}}^-. \tag{1}$$

Thus, $r_k \ge r_k^-$.

Now we prove that $r_k \le r_k^+$ for $k \ge 2$, as it is immediate that $r_1 \le r_1^+$. Suppose that $r_k = \sum_{i=1}^{l} p_i$ where each p_i denotes the number of elements from S_i not after v_k in the merged stream. Assume v_k is from stream D_α. Clearly, $p_\alpha = r_{\alpha,j_{k,\alpha}} \le r_{\alpha,j_{k,\alpha}}^+ \le r_{\alpha,j_{k,\alpha}-1}^- + 2\eta N_\alpha$.

For each $p_i \ne 0$ ($i \ne \alpha$), note that $r_{i,j_{k,i}}^- + 2\eta N_i \ge r_{i,j_{k,i}+1}^+$. If $p_i > r_{i,j_{k,i}}^- + 2\eta N_i$, then $r_k > r_{i,k_i}^- + 2\eta N_i \ge r_{i,k_i+1}$; consequently, $(v_{i,j_{k,i}}, r_{i,j_{k,i}}^-, r_{i,j_{k,i}}^+)$ is not the last tuple from S_i merged into S_{merge} before v_k. Contradict! Therefore, $p_i \le r_{i,j_{k,i}}^- + 2\eta N_i$ for each $p_i \ne 0$ ($i \ne \alpha$).

By the algorithm Merge (line 9), for $k \ge 2$, $r_{k-1}^- = \sum_{i=1, i\ne\alpha}^{l} r_{i,j_{k,i}}^- + r_{\alpha,j_{k,\alpha}-1}^-$. Thus, $r_k \le r_k^+$. □

Theorem 2. *Suppose that there are l data streams D_i for $1 \le i \le l$ and each D_i has N_i data elements. Suppose that each S_i ($1 \le i \le l$) is an η-approximate sketch of D_i. Then, S_{merge} generated by the algorithm Merge on $\{S_i : 1 \le i \le l\}$ is an η-approximate sketch of $\cup_{i=1}^{l} D_i$.*

Proof. The property 1 and property 3 in the definition of "η-approximate" (in Theorem 1) are immediate. From the equation (1), the property 2 immediately follows. □

3.2 Sketch Compress

In this subsection, we will present a sketch compress algorithm to be used in our sketch construction algorithm. The algorithm is outlined in Algorithm 2. It takes an $\frac{\xi}{2}$-approximate sketch of a data set with N elements as an input, and produces an ξ-approximate sketch with at most $\lfloor \frac{1}{\xi} \rfloor + 2$ tuples.

Algorithm 2 Compress

Input:
an $\frac{\xi}{2}$-approximate sketch S;

Output:
an ξ-approximate sketch S_{comd} with at most $\lfloor \frac{1}{\xi} \rfloor + 2$ tuples

Description:

1: $S_{comd} := \emptyset$;
2: Add to S_{comd} the first tuple of S;
3: **for** $1 \le j \le \lfloor \frac{1}{\xi} \rfloor$ **do**
4: let s be the first tuple $(v_{k_j}, r_{k_j}^-, r_{k_j}^+)$ in S such that $j\lfloor \xi N \rfloor - \frac{\xi N}{2} \le r_{k_j}^- \le r_{k_j}^+ \le j\lfloor \xi N \rfloor + \frac{\xi N}{2}$;
5: $S_{comd} := S_{comd} \cup \{s\}$.
6: **end for**
7: Add the last tuple of S to S_{comd}.

Note that S is an ordered sequence of tuples. In Algorithm 2, s is the first tuple satisfying the condition from the beginning of S. Clearly, the algorithm can be implemented in a linear time with respect to the size of S.

Suppose that S_{comd} has the same ordering as S. Then it can be immediately verified that S_{comd} is a sketch. According to the condition given in line 4 in Algorithm 2, the property 3 in the definition (in Theorem 1) of ξ-approximate is immediate. Therefore, S_{comd} is an ξ-approximate sketch of the data stream.

Theorem 3. *Suppose that S is an $\frac{\xi}{2}$-approximate sketch. Then, S_{comd} generated by the algorithm Compress on S is ξ-approximate, which has at most $\lfloor \frac{1}{\xi} \rfloor + 2$ tuples.*

Note that regarding the same precision guarantee, our new compress technique takes about half of the number of tuples given by the compress technique in [10].

3.3 Sketch Construction and Maintenance

We are now ready to present our algorithm that continuously maintains a sketch to answer approximately a quantile query under the sliding window model. According to Theorem 2, in our sketch maintenance algorithm we need only to maintain a good approximate sketch for each bucket. The algorithm, outlined in Algorithm 3, follows the scheme as depicted in Figure 2.

Note that in each bucket, the algorithm keeps 1) its sketch, 2) its time-stamp, and 3) the number of elements contained in the bucket. The algorithm is quite straightforward. For a new data item, if the total number k of elements remained in our consideration is over the size of the sliding window N, the sketch for the oldest bucket is dropped (line 4-6). If the current bucket size exceeds $\lfloor \frac{\epsilon}{2} N \rfloor$, its corresponding sketch will be compressed to a sketch with a constant number $O(\frac{1}{\epsilon})$ of tuples (line 9) and a new bucket is created as the current bucket (line 10). The new data element is always inserted into a sketch of the current bucket by GK-algorithm for maintaining $\frac{\epsilon}{4}$-approximation (line 13). Note

Algorithm 3 SW

Description:

```
1:  k := 0;
2:  for all new generated data element d do
3:      k := k + 1;
4:      if k = N + 1 then
5:          Drop the sketch of the oldest bucket;
6:          k := k − ⌊εN/2⌋;
7:      end if
8:      if BucketSize(current) ≥ ⌊(ε/2)N⌋ then
9:          Compress(current, ξ = ε/2);
10:         current := NewBucket();
11:     end if
12:     increase BucketSize(current) by 1;
13:     GK(current ∪ {d}, ε/4);
14: end for
```

that in Algorithm 3, we use *current* to represent a sketch for the current bucket, and we also record the number of elements in the current bucket as $BucketSize(current)$. Note that when a new bucket is issued (line 10), we initialize only its sketch ($current$) and its bucket size ($BucketSize$), as well as assign the the time-stamp of the new bucket as the arrival time-stamp of the current new data element.

From the algorithm Compress, it is immediate that the local sketch for each bucket (except the current one) is $\frac{\epsilon}{2}$-approximate restricted to their local data elements, while the current (last) one is $\frac{\epsilon}{4}$-approximate. Further, the following space is required by the algorithm SW.

Theorem 4. *The algorithm SW requires $O(\frac{\log(\epsilon^2 N)}{\epsilon} + \frac{1}{\epsilon^2})$ space.*

Proof. The sketch in each bucket produced by the algorithm GK takes $O(\frac{\log(\epsilon^2 N)}{\epsilon})$ space which will be compressed to a space of $O(\frac{1}{\epsilon})$ once the bucket is full. Clearly, there are $O(\frac{1}{\epsilon})$ buckets. The theorem immediately follows. □

3.4 Querying Quantiles

In this subsection, we present a new query algorithm which can always answer a quantile query within the precision of ϵN, in light of Theorem 2. Note that in the algorithm SW, the number N' of data elements in the remaining buckets may be less than N - the maximum difference $(N - N')$ is $\lfloor \frac{\epsilon N}{2} \rfloor - 1$. Consequently, after applying the algorithm Merge on the remaining local sketches, we can guarantee only that S_{merge} is an $\frac{\epsilon}{2}$-sketch of the N' element. There is even no guarantee that S_{merge} is a sketch of the N elements.

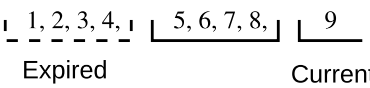


Figure 3. $\epsilon = 1$ and $N = 8$

For example, suppose that a stream arrives in the order 1, 2, 3, 4, ..., 9 as depicted in Figure 3. Upon the element 9 arrives, the first bucket (with 4 elements) is expired and its local sketch is dropped. The local $\frac{\epsilon}{2}$-approximate sketch could be $(5, 1, 1)$ and $(7, 3, 3)$ for the second bucket and $(9, 1, 1)$ for the current bucket. After applying the algorithm Merge on them, the first tuple in S_{merge} is $(5, 1, 3.5)$. Note that in this example, the most recent 8 elements should be 2, 3, 4, 5, ... 9, and the rank of 5 is 4. As the rank 4 is not between 1 and 3.5, S_{merge} is not a sketch for these 8 elements.

To solve this problem, we use a "lift" operation, outlined in Algorithm 4, to lift the value of r_i^+ by $\lfloor \frac{\epsilon N}{2} \rfloor$ for each tuple i.

Algorithm 4 Lift

Input:

S, ζ;

Output:

S_{lift};

Description:

```
S_lift := ∅;
for all tuple a_i = (v_i, r_i^-, r_i^+) in S do
    update a_i by r_i^+ := r_i^+ + ⌊(ζ/2)N⌋)
    insert a_i into S_lift
end for
```

Theorem 5. *Suppose that there is a data steam D with N elements and there is an $\frac{\zeta}{2}$-approximate sketch S of the most recent N' elements in D where $0 \leq N - N' \leq \lfloor \frac{\zeta}{2} N \rfloor$. Then, S_{lift} generated by the algorithm Lift on S is an ζ-approximate sketch of D.*

Proof. We first prove that S_{lift} is a sketch of D. Suppose that:

- $(v_i, r_i^-, r_i^+ + \lfloor \frac{\zeta}{2} N \rfloor)$ is a tuple in S_{lift} and (v_i, r_i^-, r_i^+) is a tuple in S;
- r_i is the rank of v_i in the ordered N' data items;
- r_i' is the rank of v_i in the ordered D.

Clearly, $r_i' \geq r_i \geq r_i^-$, and $r_i' \leq r_i + \lfloor \frac{\zeta}{2} N \rfloor$. As S is a sketch of the N' data items, $r_i \leq r_i^+$. Consequently, $r_i' \leq r_i^+ + \lfloor \frac{\zeta}{2} N \rfloor$. The other sketch properties of S_{lift} can be immediately verified.

The ζ-approximation of S_{lift} can be immediately verified from the definition (in Theorem 1). □

Our query algorithm is presented in Algorithm 5.

According to the theorems 1, 2, and 5, the algorithm SW_Query is correct; that is, for each rank $r = \lceil \phi N \rceil$ ($\phi \in (0, 1]$) we are always able to return an element v_i such that $|r - r_i| \leq \epsilon N$.

3.5 Query Costs

Note that in our implementation of the algorithm SW_Query, we do not have to completely implement the

Algorithm 5 SW_Query

Step 1: Apply the algorithm Merge ($\eta = \epsilon/2$) to the local sketches to produce a merged $\frac{\epsilon}{2}$-approximate scheme S_{merge} on the non-dropped local sketches.

Step 2: Generate a sketch S_{lift} from S_{merge} by the algorithm Lift by asking $\zeta = \epsilon$.

Step 3: For a given rank $r = \lceil \phi N \rceil$, find the first tuple (v_i, r_i^-, r_i^+) in S_{lift} such that $r - \epsilon N \leq r_i^- \leq r_i^+ \leq r + \epsilon N$. Then return the element v_i.

---

algorithm Merge and the algorithm Lift, nor materialize the S_{merge} and S_{lift}. The three steps in the algorithm SW_Query can be implemented in a pipeline fashion:

> Once a tuple is generated in the algorithm Merge, it passes into Step 2 for the algorithm Lift. Once the tuple is lifted, it passes to the Step 3. Once the tuple is qualified for the query condition in Step 3, the algorithm terminates and returns the data element in the tuple.

Further, the algorithm Merge can be run in a way similar to the l-way merge-sort fashion [21]. Consequently, the algorithm Merge runs in a $O(m \log l)$ where m is the total number of tuples in the l sketches. Since the algorithm Merge takes the dominant costs, the algorithm SW_Query also runs in time $O(m \log l)$.

Note that in our algorithms, we apply GK-algorithm, which records (v_i, g_i, Δ_i) instead of (v_i, r_i^-, r_i^+) for each i. In fact, the calculations of r_i^- and r_i^+ according to the equations (a) and (b) can be also pipelined with the steps 1-3 of the algorithm SW_Query. Therefore, this does not affect the time complexity of the query algorithm.

4 One-Pass Summary under n-of-N

In this section, we will present a space-efficient algorithm for summarising the most recent N elements to answer quantile queries for the most recent n ($\forall n \leq N$) elements.

As with the sliding window model, it is desirable that a good sketch to support the n-of-N model should have the properties:

- every data element involved in the sketch should be included in the most recent n elements;
- to ensure ϵ-approximation, a sketch to answer a quantile query against the most recent n elements should be built on the most recent $n - O(\epsilon n)$ elements.

Different from the sliding window model, in the n-of-N model n is an arbitrary integer between 1 and N. Consequently, the data stream partitioning technique developed in the last section cannot always accommodate such requests; for instance when $n < \lfloor \frac{\epsilon N}{2} \rfloor$, Example 4 is also applicable.

In our algorithm, we use the EH technique [5] to partition a data stream. Below we first introduce the EH partitioning technique.

4.1 EH Partitioning Technique

In EH, each bucket carries a time stamp which is the time stamp of the earliest data element in the bucket. For a stream with N data elements, elements in the stream are processed one by one according to their arrival ordering such that EH maintains at most $\lceil \frac{1}{\lambda} \rceil + 1$ "i-buckets" for each i. Here, an i-bucket consists of i data elements consecutively generated. EH proceeds as follows.

Step 1: When a new element e arrives, it creates a 1-bucket for e and the time-stamp of the bucket is the time-stamp of e.

Step 2: If the number of 1-buckets is full (i.e., $\lceil \frac{1}{\lambda} \rceil + 2$), then merge the two oldest i-buckets into an $i{+}1$-bucket (carry the oldest time-stamp) iteratively from $i = 1$. At every level i, such merge is done only if the number of buckets becomes $\lceil \frac{1}{\lambda} \rceil + 2$; otherwise such a merge iteration terminates at level i.

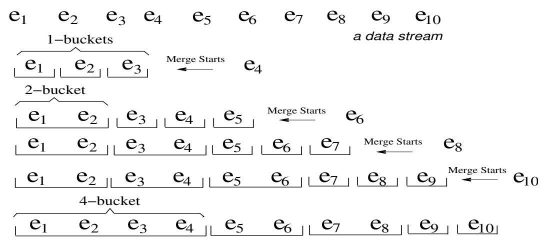


Figure 4. EH Partition: $\lambda = 0.5$

Figure 4 illustrates the EH algorithm for 10 elements where $\lambda = \frac{1}{2}$ and each e_i is assumed to have the time stamp i. The following theorems are immediate [5]:

Theorem 6. *For N elements, the number of buckets in EH is always $O(\frac{\log(\lambda N)}{\lambda})$.*

Theorem 7. *The number N' of elements in a bucket b has the property that $N' - 1 \leq \lambda N$ where N is the number of elements generated after all elements in b.*

4.2 Sketch Construction

In our algorithm, for each bucket b in EH, we record 1) a sketch S_b to summarize the data elements from the earliest element in b up to now, 2) the number N_b of elements from the earliest element in b up to now, and 3) the time-stamp t_b.

Once two buckets b and b' are merged, we keep the information of b if b is earlier than b'. Figure 5 illustrate the algorithm.

We present our algorithm below in Algorithm 6. Note that to ensure ϵ-approximation, we choose $\lambda = \frac{\epsilon}{\epsilon+2}$ in our algorithm to run the EH partitioning algorithm, and GK-algorithm is also applied to maintaining an $\frac{\epsilon}{2}$-approximate sketch.

Algorithm 6 nN

Upon a new data element e arrives at time t,

Step 1 - create a new sketch: Record a new 1-bucket e, its time stamp t_e as t, and $N_e = 0$. Initialize a sketch S_e.

Step 2 - drop sketches: If the number of 1-buckets is full (i.e., $\lceil \frac{1}{\lambda} \rceil + 2$), then do the following iteratively from $i = 1$ till j where the current number (before this new element arrives) of j-buckets is not greater than $\lceil \frac{1}{\lambda} \rceil$.

- get the two oldest buckets b_1 and b_2 among the i-buckets; and then
- drop b_1 and b_2 from the i-th bucket list; and then
- drop the sketch S_{b_2} (assume the b_1 is older than b_2); and then
- add b_1 together with its time stamp into $i + 1$-buckets list.

Scan the sketch list from oldest to delete the *expired* buckets b - (S_b, N_b, t_b); that is $N_b \geq N$.

Step 3 - maintain sketches: for each remaining sketch S_b, add e into S_b by GK-algorithm for $\frac{\epsilon}{2}$-approximation and $N_b := N_b + 1$.

By Theorem 6, the algorithm nN maintains $O(\frac{\log(\epsilon N)}{\epsilon})$ sketches, and each sketch requires a space of $O(\frac{\log(\epsilon N)}{\epsilon})$ according to GK-algorithm. Consequently, the algorithm nN requires a space of $O(\frac{\log^2(\epsilon N)}{\epsilon^2})$.

4.3 Querying Quantiles

In this subsection, we show that we can always get an ϵ-approximate sketch among the sketches maintained Algorithm 6 to answer a quantile query for the most recent n ($\forall n \leq N$) elements. Our query algorithm, described in Algorithm 7 consists of 3 steps. First, we choose an appropriate sketch among the maintained sketch. Then, we use the algorithm Lift, and finally check the query condition.

Theorem 8. *Algorithm 7 is correct; that is,*

- *the algorithm is always able to return a data element;*
- *the element returned by the algorithm meets the required precision.*

Proof. To prove the theorem, we need only to prove that

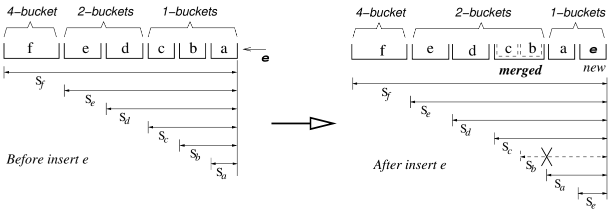


Figure 5. Algorithm nN: $\eta = 0.5$

Algorithm 7 nN_Query

Input:

Sketches maintained in the algorithm nN where $\lambda = \frac{\epsilon}{\epsilon+2}$, n ($n \leq N$), and $r = \lceil \phi n \rceil$ ($\phi \in (0, 1]$).

Output:

an element v' in the most recent n elements such that $r - \epsilon n \leq r' \leq r + \epsilon n$ (r' is the rank of v' in the most recent n elements).

Step 1: For a given n ($n \leq N$), scan the sketch list from oldest and find the first sketch S_{b_n} such that $N_{b_n} \leq n$.

Step 2: Apply the algorithm Lift to S_{b_n} to generate S_{lift} where $\zeta = \epsilon$.

Step 3: For a given rank r, find the first tuple (v_i^+, r_i^+, r_i^-) in S_{lift} such that that $r - \epsilon n \leq r_i^- \leq r_i^+ \leq r + \epsilon n$. Return v_i.

S_{lift} is an ϵ-approximate sketch of the most recent n elements.

Note that in Algorithm 6, S_{b_n} is maintained, by GK-algorithm for $\frac{\epsilon}{2}$-approximation, over the most recent N_{b_n} elements.

Case 1: $n < N_{b_n}$. Suppose that b'_n is the bucket in EH, which is just before b_n. Then, $N_{b'_n} > n$ since N_{b_n} is the largest number which is not larger than n. Consequently, $n - N_{b_n} \leq N_{b'_n} - N_{b_n} - 1$. According to Theorem 7, $n - N_{b_n} \leq \frac{\epsilon}{2+\epsilon} N_{b_n}$. This implies that $n - N_{b_n} < \frac{\epsilon}{2} n$; thus

$$n - N_{b_n} \leq \lfloor \frac{\epsilon}{2} n \rfloor \qquad (2)$$

Note that since Theorem 7 also covers an expired bucket, the inequality (2) holds if S_{b_n} is the oldest. Based on the inequality (2) and Theorem 5, S_{lift} is an ϵ-approximate sketch of the most recent n elements.

Case 2: $n = N_{b_n}$. It is immediate that S_{lift} is an ϵ-approximate sketch of the most recent n elements according to Theorem 5. In fact, in this case we do not have to use the algorithm Lift; however for the algorithm presentation simplification, we still include the operation for this case. □

Note that we can also apply a pipeline paradigm to the steps 2 and 3, in a way similar to that in section 3.5, to speed-up the execution of Algorithm 7. Consequently, Algorithm 7 runs in $O(\frac{\log(\epsilon N)}{\epsilon})$ time.

5 Discussions and Applications

The algorithm developed by Alsabti-Ranka-Singn (ARS) [2] is to compute quantiles approximately by one scan of datasets. Although this partition based algorithm was originally designed for disk-resident data, it may be modified to support the sliding window. However, this algorithm requires a space of $\Omega(\sqrt{\frac{N}{\epsilon}})$ [17]. Further, there is also a merge technique in the algorithm ARS but it was specifically designed to support the algorithm ARS; for instance, it does not support the sketches generated by GK-algorithm to retain ϵ-approximation. Therefore, the merge technique in the algorithm ARS is not applicable to our algorithm SW. In the next section, we will also compare the modified ARS with the algorithm SW by experiments.

Note that in applying the EH partitioning technique [5], we may have another option - maintaining only local sketches for each bucket and then merge two local sketches when their corresponding buckets are merged. Though this can retain ϵ-approximation, we found that in some cases we have to keep all the tuples from two buckets; this prevent us to work out a good space guarantee. In the next section, we will report the space requirements of a heuristic based on this option.

Now we show that the techniques developed in this paper can be immediately applied to other important quantile computation problems. Below we list three of such applications.

Distributed and Parallel Quantile Computation: In many applications, data streams are distributed. In light of our merge technique and Theorem 2, we need only to maintain a local ϵ-approximate sketch for each local data stream to ensure the global ϵ-approximation.

Most Recent T Period: In some applications, users may be interested in computing quantiles against the data elements in the most recent T period. The algorithm nN may be immediately modified to serve for this purpose. In the modification, to ensure ϵ-approximation we need only to change the expiration condition of a sketch: a sketch is expired if its time stamp is expired. Then we use the oldest remaining sketch to answer a quantile query. The space requirement is $O(\frac{\log^2(\epsilon N)}{\epsilon^2})$ where N is the maximum number of elements in a T period. Note that in this application, N is also unknown in each T; however, it can be ϵ-approximated by EH. We omit the details from the paper due to the space limitation.

Constrain based Sliding Window: In many applications, the users may be interested in only the elements meeting some constraints in the most recent N elements. In this application, we can also modify the algorithm nN to support an ϵ-approximate quantile computation for the elements satisfying the constraints in the most recent N elements. In the modification of the algorithm nN, we only allow the elements meeting the constraints to be added while each sketch still count the number of elements (even not meeting the constraints) "seen" so far and the number of elements seen so far meeting the constraints. Then, we use the oldest sketch to approximately answer a quantile query. The space requirement is $O(\frac{\log^2(\epsilon N)}{\epsilon^2})$. Similar to the most recent T period model, we need to ϵ-approximate the number of qualified elements in the most recent N elements.

Notation	Definition (Default Values)
N	The size of sliding window (800K)
ϵ	The guaranteed precision (0.05)
ϕ	$(1/q, ..., (q-1)/q)$ for a given q
D_d	Data distribution (Uniform)
Q_p	The query patterns (Random)
I	The length of a rank interval (N)
$\#Q$	The number of queries (100,000)

Table 1. System parameters.

6 Performance Studies

In our experiments, we modified ARS-algorithm [2] to support the sliding window model. In our modification, we partition a data stream in the way as shown in [17], which leads to the minimum space requirement. We implemented our algorithms SW and nN, as well as a heuristic - algorithm nN'. The algorithm nN' also adopts the EH partitioning technique; however, in nN' we maintain local sketches for each bucket in EH, and then use GK-algorithm [10] to merge two local sketches once the corresponding buckets have to be merged. As discussed in the last section, we did not obtain a good bound on the space requirements of nN'.

All the algorithms are implemented using C++. We conducted experiments on a PC with an Intel P4 1.8GHz CPU and 512MB memory using synthetic and real datasets.

The possible factors that will affect ϕ-quantile queries are shown in Table 1 with default values. The parameters are grouped in three categories: i) N (window size), ϵ (guaranteed precision), and ϕ (quantiles), ii) data distributions (D_d) to specify the "sortedness" of a data stream, and iii) query patterns (Q_p), lengths of rank intervals (I) to be queried, and the number of queries ($\#Q$).

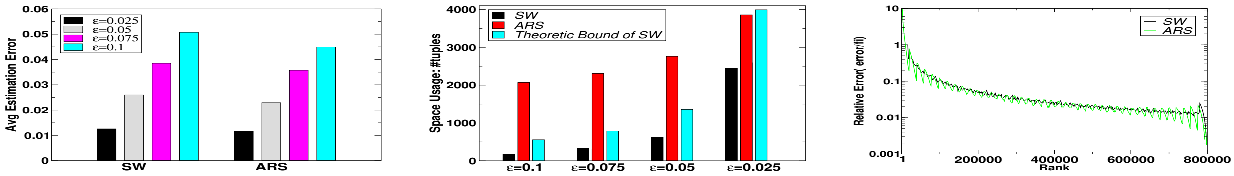
In our experiments, the data distributions (D_d) tested take the following random models: *uniform*, *normal*, and *exponential*. We also examined sorted or partially sorted data streams, such as global ascending, global descending, and partially sorted.

We use the *estimation error* as the error metrics to evaluate the tightness of ϵ as an upper-bound. The estimation error is represented as $\frac{|r'-r|}{M}$ where r is an exact rank to be queried, r' is the rank of a returned element, and M is the number of most recent elements to be queried. Here, M is N in a sliding window and is n in the n-of-N model.

As for query patterns (Q_p), in addition to the random ϕ-quantile queries, we also consider the 80-20 rules such as 80% of queries will focus on a small rank interval (with length I) and 20% of queries will access arbitrary ranks.

6.1 Sliding Window Techniques

In this subsection, we evaluate the performance of our sliding window techniques. In our experiments, we examined the average and maximum errors of ϕ-quantile queries based on a parameter q, and make ϕ-quantiles be in the form of $1/q, 2/q, ..., (q-1)/q$.



(a) Avg Errors ($q = 32$)

(b) Space Consumption

(c) Rank vs Errors($\epsilon = 0.025$)

Figure 6. Avg/Max Errors, Space Consumptions, and Rank (sliding window)

Overall Performance. An evaluation of overall performance of SW and ARS is shown in Figure 6 for the sliding window model N. In this set of experiments, $N = 800K$ and a data stream is generated using a uniform distribution with $1000K$ elements. We tested four different ϵs: 0.1, 0.075, 0.05 and 0.025.

Figure 6 (a) illustrates the average errors when $q = 32$. It shows that ARS performs similarly to nN regarding accuracy. This has been confirmed by a much larger set of queries using another error metrics: *relative* error $(\frac{|r'-r|}{r})$. Figure 6 (c) shows the relative errors of all ϕ-quantile queries in the form of $1/q, 2/q, ..., (q-1)/q$ ($\forall q \in [1, 800K]$ when $\epsilon = 0.025$. Overall, the relative error decreases while q increases but is not necessarily monotonic. The fluctuations of SW is much smaller than ARS, though their average relative errors are very close.

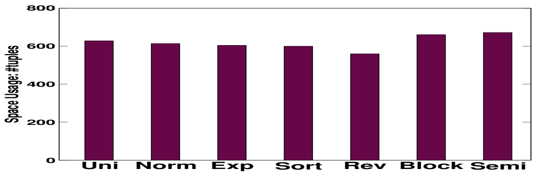


Figure 7. Space Consumptions vs Distributions

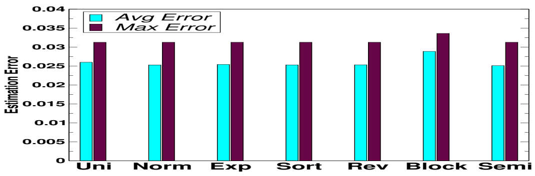
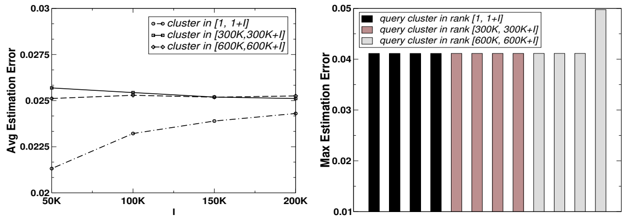


Figure 8. Errors vs Distributions ($q = 32$)

Figure 6 (b) gives the space consumptions of SW and ARS, respectively, together with the theoretical upper-bound of SW. Note that we measure the the space consumption by the maximum number of total sketch-tuples happened in temporary storage during the computation. In the algorithm ARS, each sketch-tuple needs only to keep its data element; thus, its size is about $1/3$ of those in SW. Therefore, the number of tuples we reported for the algorithm ARS is $1/3$ of the actual number of tuples for a fair comparison. We also showed the theoretical upper-bound of SW derived from that of GK-algorithm and the algorithm SW. Although a space of $\Omega(\sqrt{\frac{N}{\epsilon}})$ required by ARS-algorithm is asymptotically much larger than the theoretical space upper-bound of SW, there is still a chance to be smaller than that of SW when N is fixed and ϵ is small. This has been caught by the experiment when $\epsilon = 0.025$; it



(a) Avg Errors

(b) Max Errors

Figure 9. Query Patterns

is good to see that even in this situation, the actual space of SW is still much smaller than that of ARS.

Our experiment clearly demonstrated that the modified ARS is not as competitive as the algorithm SW. They have similar accuracy; however the algorithm SW requires only a much smaller space than that in the algorithm ARS. Further, our experiments also demonstrated that the the actual performance of SW is much better than the worst-case based theoretical bounds regarding both accuracy and space.

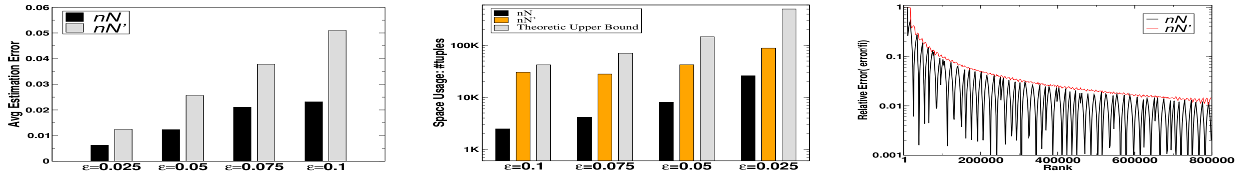
Next we evaluate our SW techniques against the possible impact factors, such as data distributions and query patterns.

Data Distributions. We conducted another set of experiments to evaluate the impacts of 7 different distributions on the algorithm SW. These include three random models: uniform, normal, and exponential, as well as four sorted models: sorted, reverse sorted, block sorted, and semi-block sorted. Block sorted is to divide data into a sequence of blocks, B_1, B_2, ... B_n such that elements in each block are sorted. Semi-block sorted means that elements in B_i are smaller than those in B_j if $i < j$; however, each block is not necessarily sorted.

Figures 7 and 8 report the experiment results. Note that in the experiments, $N = 800K$, the total number of elements is $1000K$, and $\epsilon = 0.05$. The experiment results demonstrated that the effectiveness of algorithm SW is not sensitive to data distributions.

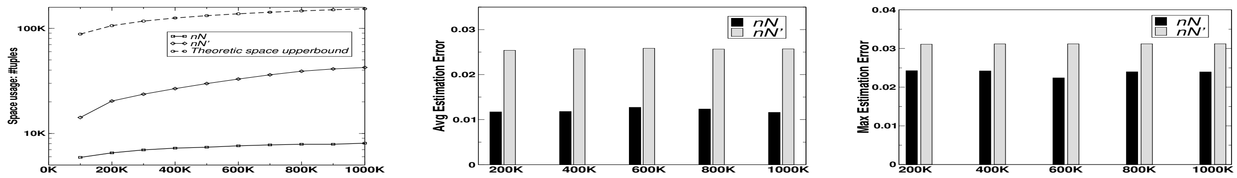
Query Patterns. Finally, we run $10K$ random ϕ-quantile queries, using our SW techniques, on a sliding window of $N = 800K$ in the data stream used in the experiments in Figure 6.

In light of 80-20 rules, we allocate 80% of ϕ-quantile queries in a small interval I. In this set of experiments, we tested four values of I: $S = 50K, 100K, 150K$ and $200K$. We also tested an impact of the positions of these intervals with lengths I; specifically we allocate the intervals at three



(a) Avg Errors ($q = 32$) (b) Space Consumption (c) Rank ($\epsilon = 0.025$)

Figure 10. Avg/Max Errors, Space Consumptions, and Rank (n-of-N)



(a) Space (b) Avg Errors (c) Max Errors

Figure 11. Space and Errors (n-of-N)

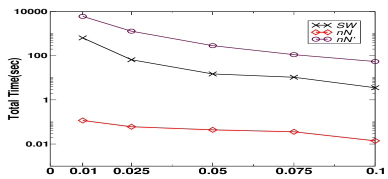
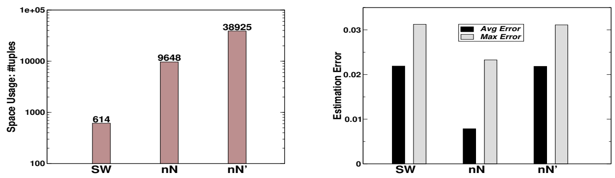


Figure 12. Query Performance (CPU)



(a) Space (b) Errors (Avg,Max)

Figure 13. Real Dataset

positions $[1, 1+I]$, $[300K, 300K+I]$ and $[600K, 600K+I]$.

The experiment results have been shown in Figure 9. Figure 9 (a) shows the average errors, and Figure 9 (b) shows the maximum errors. Note that in Figure 9 (b), the four lines in each query cluster correspond, respectively, to the four different I values: $50K$, $100K$, $150K$, and $200K$.

The maximum errors approach the theoretical upper-bound simply because the number of queries is large and therefore, the probability of having a maximum error is high.

6.2 n-of-N Techniques

We repeated a similar set of tests, to evaluate nN and nN', to those in Figure 6. The results are shown in Figure 10 for the n-of-N model, where we make $n = 400K$ and $N = 800K$. We use the same data stream as in the experiments in Figure 6. The algorithm nN clearly outperforms nN', as well as those worst-case based theoretical bounds.

We conducted another set of experiments to examine the impacts of n and N on the algorithm nN and nN'. The data stream used is the same as above with $1000K$ elements. The experiment results are reported in Figure 11.

Figure 11 (a) shows the space consumptions for the algorithm nN where N changes from $N = 100K$ to $N = 1000K$. Note that the theoretic space upper-bound means the one for the algorithm nN.

Figure 11 (b) and (c) show the average and maximum errors, respectively, when $N = 1000K$, and n changes from $n = 200K$ to $n = 1000K$. For each n, we run randomly 320 queries.

6.3 Query Costs

In Figure 12, we show the total query processing costs (CPU) for running $20K$ queries, where all the parameters take the default values and the number of the data stream elements is $1000K$.

Note that the algorithm nN_Query is the most efficient one as there is no merge operation.

6.4 Real Dataset Testing

The topic detection and tracking (TDT) is an important issue in information retrieval and text mining (`http://www.ldc.upenn.edu/Projects/TDT3/`). We have archived the news stories received through Reuters real-time datafeed, which contains 365,288 news stories and 100,672,866 (duplicate) words. All articles such as "the" and "a" are removed before term stemming.

We use $N = 800K$ and $\epsilon = 0.05$. Figure 13 shows space consumptions and average/max errors using $q = 32$, for SW, nN and nN'. They follow similar trends to those for synthetic data.

In summary, our experiment results demonstrated that the algorithm SW and the algorithm nN perform much better than their worst case based theoretical bounds, respectively. In addition to good theoretical bounds, the algorithm SW and the algorithm nN also outperform the other techniques.

Table 2. Comparing Our Results with Recent Results in Quantile Computation

Authors	Year	Computation Model	Precision	Space Requirement
Alsabti, *et. al.* [2]	1997	data sequence	ϵN, deterministic	$\Omega(\sqrt{\frac{N}{\epsilon}})$
Manku, *et. al.* [17]	1998	data sequence	ϵN, deterministic	$O(\frac{\log^2(\epsilon N)}{\epsilon})$
Manku, *et. al.* [18]	1999	data stream/appending only	ϵN, conf = $1-\delta$	$O(\epsilon^{-1}\log^2\epsilon^{-1}+\epsilon^{-1}\log^2\log\delta^{-1})$
Greenwald and Khanna [10]	2001	data stream/appending only	ϵN, deterministic	$O(\frac{1}{\epsilon}\log(\epsilon N))$
Gilbert *et. al.* [9]	2002	data stream/with deletion	ϵN, conf = $1-\delta$	$O((\log^2\lvert U\rvert\log\log(\lvert U\rvert/\delta))/\epsilon^2)$
Lin, *et. al.* [this paper]	2003	data stream/sliding window	ϵN, deterministic	$O(\frac{\log(\epsilon^2 N)}{\epsilon}+\frac{1}{\epsilon^2})$
Lin, *et. al.* [this paper]	2003	data stream/n-of-N	ϵN, deterministic	$O(\frac{\log^2(\epsilon N)}{\epsilon^2})$

7 Conclusions

In this paper, we presented our results on maintaining quantile summaries for data streams. While there is quite a lot of related work reported in the literature, the work reported here is among the first attempts to develop space efficient, one pass, deterministic quantile summary algorithms with performance guarantees under the sliding window model of data streams. Furthermore, we extended the sliding window model to propose a new n-of-N model which we believe has wide applications. As our performance study indicated, the algorithms proposed for both models provide much more accurate quantile estimates than the guaranteed precision while requiring much smaller space than the worst case bounds.

In Table 2, we compare our results with the recent results in quantile computation under various models.

An immediate future work is to investigate the problem of maintaining other statistics under our new n-of-N model. Furthermore, the technique developed in this work that merges multiple ϵ-approximate quantile sketches into a single ϵ-approximate quantile sketches is expected to have applications in distributed and parallel systems. One possible work is to investigate the issues related to maintain distributed quantile summaries for large systems for a sliding window.

References

[1] R. Agrawal and A. Swami. A one-pass space-efficient algorithm for finding quantiles. In S. Chaudhuri, A. Deshpande, and R. Krishnamurthy, editors, *COMAD*, 1995.

[2] K. Alsabti, S. Ranka, and V. Singh. A one-pass algorithm for accurately estimating quantiles for disk-resident data. In *The VLDB Journal*, pages 346–355, 1997.

[3] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *VLDB*, 2002.

[4] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD*, 2003.

[5] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows: (extended abstract). In *ACM-SIAM*, pages 635–644, 2002.

[6] A. Dobra, M. N. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD*, 2002.

[7] M. Garofalakis and A. Kumar. Correlating xml data streams using tree-edit distance embeddings. In *SIGMOD-SIGACT-SIGART*, 2003.

[8] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *SIGMOD*, pages 13–24, 2001.

[9] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *VLDB*, 2002.

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

[11] S. Guha and N. Koudas. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In *ICDE*, 2002.

[12] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *STOC*, pages 471–475, 2001.

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS*, pages 359–366, 2000.

[14] A. Gupta and D. Suciu. Stream processing of xpath queries with predicates. In *SIGMOD*, 2003.

[15] J. Kang, J. Naughton, and S. Viglas. Evaluation window joins over undounded streams. In *ICDE*, 2003.

[16] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.

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

[18] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *SIGMOD*, pages 251–262, 1999.

[19] J. Munro and M. Paterson. Selection and sorting wiith limited storage. In *TCS12*, 1980.

[20] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD*, 2003.

[21] R. Ramakrishan. *Database Management Systems*. McGraw-Hill, 2002.

[22] Y. Zhu and D. Shasha. Statstream:statistical monitoring of thousands of data streams in real time. In *VLDB*, 2002.