IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 14, 2020, accepted February 13, 2020, date of publication February 18, 2020, date of current version February 27, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2974919

A Survey of Approximate Quantile Computation on Large-Scale Data

ZHIWEI CHEN[1] **AND AOQIAN ZHANG**[2]
[1]School of Software, Tsinghua University, Beijing 100084, China
[2]Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada

Corresponding author: Aoqian. Zhang (aoqian.zhang@uwaterloo.ca)

This work was supported in part by the National Key Research and Development Plan under Grant 2019YFB1705301, and in part by the National Natural Science Foundation of China under Grant 61572272 and Grant 71690231.

ABSTRACT As data volume grows extensively, data profiling helps to extract metadata of large-scale data. However, one kind of metadata, order statistics, is difficult to be computed because they are not mergeable or incremental. Thus, the limitation of time and memory space does not support their computation on large-scale data. In this paper, we focus on an order statistic, quantiles, and present a comprehensive analysis of studies on approximate quantile computation. Both deterministic algorithms and randomized algorithms that compute approximate quantiles over streaming models or distributed models are covered. Then, multiple techniques for improving the efficiency and performance of approximate quantile algorithms in various scenarios, such as skewed data and high-speed data streams, are presented. Finally, we conclude with coverage of existing packages in different languages and with a brief discussion of the future direction in this area.

INDEX TERMS Data profiling, order statistics, approximate quantile, streaming model, distributed model.

I. INTRODUCTION

Data profiling is a set of activities to describe the metadata about given data [1]. It is crucial for data analysis, especially for large-scale data. It helps researchers to understand data distribution [2], discover duplicates [3], detect anomalies [4], determine thresholds [5], etc. Such information provides guidance for other data preprocessing work such as data cleaning [6], which can subsequently improve the performance of data mining dramatically [7]. When preprocessing large-scale data, data profiling is attached great importance to and faces its own challenges. Because of large data size, classic brutal methods are not applicable any more for their intolerable complexity of both time and space. Researchers have spent decades on figuring out new ways to compute the metadata which can be calculated easily on small data. The metadata can be divided into two categories based on scalability: aggregation statistics and order statistics [8].

Aggregation statistics are named for their property that they are mergeable and incremental, which makes them relatively easy to be computed no matter how large the data is. For examples, sum, mean values, standard deviations, min or max values are all aggregation statistics. For streaming models [9], [10], where data elements come one by one with time, we can trace and update aggregated results covering all arrived data by incrementing new results continuously. Time complexity and space complexity are both $O(1)$. As for distributed models [11], where data are stored in nodes of a distributed network, the overall aggregation statistics can be obtained by merging results from each node. The total communication cost of this computation is $O(|v|)$, where $|v|$ is the number of network nodes. However, order statistics, such as quantiles, heavy hitters, etc., do not preserve such property. So, we cannot compute them by merging existing results with newly produced results in a straight way. In order to compute them, many customized data structures or storage structures are proposed for these order statistics, trying to turn them into a mergeable or incremental form in some way.

In this summary, we focus on one order statistic, quantiles. They help to generate the description of the data distributions without parameters. In other words, they are able to reflect the cumulative distribution function (cdf), thus the probability distribution function (pdf), of data at low computational cost. Pdf is widely used in data cleaning and data querying. For example, in data cleaning, it is applied to demonstrate the distance distribution among values of the same attribute so as to identify misplaced attribute values [12]. And in data querying, it helps to set an appropriate correlation filter, improving efficiency for set correlation query over set records in databases [13]. Therefore, quantiles are regarded as one of

The associate editor coordinating the review of this manuscript and approving it for publication was Shiqiang Wang.

the most fundamental and most important statistics in data quality analysis in both theory and practice. For instance, many data analysis tools, including Excel, MATLAB, Python, etc., have quantile-computing functions as built-in components or libraries. In the Sawzall language, which is the basic for all Google's log data analysis, quantile is one of the seven basic statistic operators defined, along with sum, max, top-k, etc. [14]. Besides, quantiles are widely used in data collection and running-state monitoring in sensor networks [15], [16]. When a dataset contains dirty values, compared with mean values and standard deviations, quantiles and median absolute deviations are more objective and more accurate to reflect data center and data deviation [17]. They are less sensitive to outliers. In temporal data, where imprecise timestamps are prevalent, even if some timestamps are delayed very long or have inconsistent granularity, quantiles are still able to specify appropriate temporal constraints on time interval, helping to clean the data [18]. In addition, quantile algorithms have been widely used as subroutines to resolve more complicated problems, such as histograms and dynamic geometric computations [19].

A quantile is the element at a certain rank in the dataset after sort. Algorithmic studies can be traced back to 1973 at least when linear-time selection was invented [20]. In classic methods of computing ϕ-quantile over a dataset of size N, where $\phi \in (0, 1)$, first we sort all elements and then return the one ranking $\lfloor \phi N \rfloor$. Its time complexity is $O(N \log N)$ and space complexity is $O(N)$ obviously. However, in large-scale data, the method is infeasible under restrictions of memory size. Munro *et al.* has proved that any exact quantile algorithm with p-pass scan over data requires at least $\Omega(N^{1/p})$ space [21]. Besides, in streaming models, quantile algorithms should also be streaming, which means they are permitted to scan each element only once and need to update quantile answers instantaneously when receiving new elements. There is no way to compute quantiles exactly under such condition. Thus, approximation is introduced in quantile computation. Approximate computation is an efficient way to analyze large-scale data under restricted resources [22]. On one hand, it raises computational efficiency and lower computational space. On the other hand, large scale of the dataset can dilute approximation effects. Large-scale data is usually dirty, which also makes approximate quantile endurable and applicable in industry. Significantly, the scale of data is relative, based on the availability of time and space. So, the rule about how to choose between exact quantiles and approximate quantiles differs in heterogeneous scenarios, depending on the requirement for accuracy and the contradiction between the scale of data and that of resources. When the cost of computing exact quantiles is intolerable and the results are not required to be totally precise, approximate quantiles are a promising alternative.

We denote approximation error by ϵ. A ϵ-approximate ϕ-quantile is any element whose rank is between $r - \epsilon N$ and $r + \epsilon N$ after sort, where $r = \lfloor \phi N \rfloor$. For example, we want to calculate 0.1-approximate 0.3-quantile of the dataset

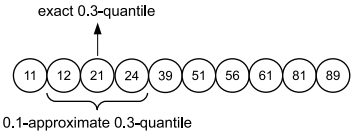


FIGURE 1. An example of a ϵ-approximate ϕ-quantile, where $\epsilon = 0.1$ and $\phi = 0.3$.

11, 21, 24, 61, 81, 39, 89, 56, 12, 51. As shown in Figure 1, we sort the elements as 11, 12, 21, 24, 39, 51, 56, 61, 81, 89 and compute the range of the quantile's rank, which is $[(0.3 - 0.1) \times 10, (0.3 + 0.1) \times 10] = [2, 4]$. Thus the answer can be one of 12, 21, 24. In order to further reduce computation space, approximate quantile computation is often combined with randomized sampling, making the deterministic computation becomes randomized. In such case, another parameter δ, or randomization degree, is introduced, meaning the algorithm answers a correct quantile with a probability of at least $1 - \delta$.

There are 3 basic metrics to assess an approximate quantile algorithm [23]:

- **Space complexity** It is necessary for streaming algorithms. Due to the limitation of memory, only algorithms using sublinear space are applicable [24]. It corresponds to communication cost in distributed models. In a distributed sensor network, communication overhead consumes more power and limits the battery life of power-constrained devices, such as wireless sensor nodes [25]. So, quantile algorithms are aimed at low space complexity or low communication cost.
- **Update time** It is the time spent on updating quantile answers when new element arrives. Fast updates can improve the user experience, so many streaming algorithms take update time as a main consideration.
- **Accuracy** It measures the distance between approximate quantiles and ground truth. Intuitively, the more accurate an algorithm is, the more space and the longer time it will consume. They are on the trade-off relationship. We use approximation error, maximum actual approximation error and average actual approximation error as quantitative indicators to measure accuracy.

We collected and studied researches about approximate quantile computation, then completed this survey. The survey includes 72 papers, which propose algorithms, varying from data sampling to data structure transformation, and techniques for optimization. Important algorithms are listed in Table 1. The remaining parts of this survey are organized as follows: In Section II, we introduce deterministic quantile algorithms over both streaming models and distributed models. Section III discusses randomized algorithms [26]. Section IV introduces some techniques and algorithms for improving the performance and efficiency of quantile algorithms in various scenarios. Section V presents a few off-the-shelf tools in industry for quantile computation. Finally, Section VI makes the conclusion and proposes interesting directions for future research.

TABLE 1. Approximate quantile algorithms.

Algorithm	Randomization	Model	Space complexity / Communication cost
MRL98 [27]	Deterministic	Streaming	$O(\frac{1}{\epsilon}\log^2(\epsilon N))$
GK01 [28]	Deterministic	Streaming	$O(\frac{1}{\epsilon}\log(\epsilon N))$
Lin SW [29]	Deterministic	Streaming	$O(\frac{1}{\epsilon}\log(\epsilon^2 N)+\frac{1}{\epsilon^2})$
Lin n-of-N [29]	Deterministic	Streaming	$O(\frac{1}{\epsilon^2}\log^2(\epsilon N))$
Arasu FSSW [30]	Deterministic	Streaming	$O(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\log N)$
Arasu VSSW [30]	Deterministic	Streaming	$O(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\log(\epsilon N)\log N)$
GK-UN [31]	Deterministic	Streaming	$O(\frac{1}{\epsilon}\log(\epsilon PC(S^u)))$
Q-digest [16]	Deterministic	Distributed	$O(\frac{1}{\epsilon}\lvert v\rvert\log\sigma)$
GK04 [32]	Deterministic	Distributed	$O(\frac{\lvert v\rvert}{\epsilon}\log^2 N)$
Cormode05 [33]	Deterministic	Distributed	$O(\frac{\lvert v\rvert}{\epsilon^2}\log N)$
Yi13 [34]	Deterministic	Distributed	$O(\frac{\lvert v\rvert}{\epsilon}\log N)$
MRL99 [35]	Randomized	Streaming	$O(\frac{1}{\epsilon}\log^2\frac{1}{\epsilon})$
Agarwal13 [36]	Randomized	Streaming	$O(\frac{1}{\epsilon}\log^{1.5}\frac{1}{\epsilon})$
Felber15 [37]	Randomized	Streaming	$O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$
Huang11 [38]	Randomized	Distributed	$O(\frac{1}{\epsilon}\sqrt{\lvert v\rvert h})$
Haeupler18 [39]	Randomized	Distributed	\$O($\log\log N+\log\frac{1}{\epsilon}$)

II. DETERMINISTIC ALGORITHMS

An algorithm is deterministic while it returns a fixed answer given the same dataset and query condition. Furthermore, quantile algorithms are classified based on their application scenarios. In streaming models, where data elements arrive one by one in a streaming way, algorithms are required to answer quantile queries with only one-pass scan, given the data size N [27] or not [28], [30], [31], [40], [41]. Except to answering quantile queries for all arrived data, Lin *et al.* [29] concentrates on tracing quantiles for the most recent N elements over a data stream. In distributed models, where data or statistics are stored in distributed architectures such as sensor networks, algorithms are proposed to merge quantile results from child nodes using as low communication cost as possible to reduce energy consumption and prolong equipment life [16], [32], [34], [36], [38].

A. STREAMING MODEL

The most prominent feature of streaming algorithms is that all data are required to be scanned only once. Besides, the length of the data stream may be uncertain and can even grow arbitrarily large. Thus, classic quantile algorithms are infeasible for streaming models because of the limitation of memory. Therefore, the priority of approximate quantile algorithms for streaming models is to minimize space complexities. In 2010, Hung and Ting [42] have proved that any comparison-based ϵ-approximate quantile algorithm over streaming models needs space complexity of at least $\Omega(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$, which sets a lower bound for these algorithms.

Both Jain and Chlamtac [43] and Agrawal and Swami [44] proposed algorithms to compute quantiles with one-pass scan. However, neither of them clarified the upper or lower bound of approximation error. In 1997, Alsabti *et al.* [40] improved the algorithm and came up with a version with guaranteed error bound, referred to as ARS97. Its basic idea is sampling and it includes the following steps:

1) Divide the dataset into r partitions.
2) For each partition, sample s elements and store them in a sorted way.
3) Combine r partitions of data, generating one sequence for querying quantiles.

ARS97 is targeted at disk-resident data, rather than streaming models. Nevertheless, its idea to partition the entire dataset and maintain a sampled sorted sequence for quantile querying inspires quantile algorithms over streaming models afterwards.

The inspired algorithm is referred to as MRL98 proposed by Manku *et al.* [27]. It requires the prior knowledge of the length N of data stream. Similar with ARS97, MRL98 divides the data stream into b blocks, samples k elements from each block and puts them into b buffers. Each buffer X is given a weight $w(X)$, representing the number of elements covered by this buffer. The algorithm consists of 3 operations:

- **NEW** Put the first bk elements into buffers successively and set their weights to 1.
- **COLLAPSE** Compress elements from multiple buffers into one buffer. Specifically, each element from an input buffer X_i would be duplicated $w(X_i)$ times. Then these duplicated elements are sorted and merged into a sequence, where k elements are selected at regular intervals and stored in the output buffer Y, whose weight $w(Y)=\sum_i w(X_i)$
- **OUTPUT** Select an element as the quantile answer from b buffers.

NEW and OUTPUT are straightforward, so the algorithm's space complexity depends mainly on how to trigger COLLAPSE. MRL98 proposed a tree-structure trigger strategy. Each buffer X is assigned a height $l(X)$ and l is set to $min_i l(X_i)$. $l(X)$ is set as the following standard:

- If only one buffer is empty, its height is set to l.
- If there are two or more empty buffers, their heights are set to 0.

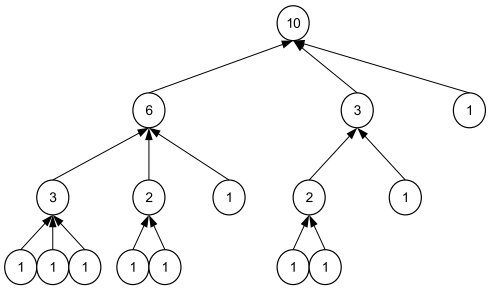


FIGURE 2. The collapsing strategy with 3 buffers. A node corresponds to a buffer and its number denotes the weight.

- Otherwise, buffers of height l are collapsed, generating a buffer of height $l+1$.

By tuning b and k, MLR98 can narrow the approximation error within ϵ. Figure 2 demonstrates the trigger strategy when $b=3$. The height of the strategy tree is logarithmic, thus the space complexity is $O(\frac{1}{\epsilon}\log^2(\epsilon N))$.

The limitation of MRL98 is that it needs to know the length of the data stream at first. However, in more cases, the length is uncertain and may even grow arbitrarily large. In such case, Greenwald and Khanna [28] proposes celebrated GK01 algorithm, distinguished by its innovative data structure, *Summaries*, or S for short. The basic idea is that when N increases, the set of ϵ-approximate answers for querying ϕ-quantile expands as well, so correctness can be retained even if removing some elements. S is a collection of tuples in the form of (v_i, g_i, Δ_i), where v_i is the element with the property $v_i \leq v_{i+1}$, g_i and Δ_i are 2 integers satisfying following conditions:

$$\sum_{j\leq i} g_j \leq r(v_i)+1 \leq \sum_{j\leq i} g_j + \Delta_i \tag{1}$$

$$g_i + \Delta_i \leq \lfloor 2\epsilon N \rfloor \tag{2}$$

$r(v_i)$, whose bounds are guaranteed by (1), is the ground truth of v_i's rank. Obviously, there are at most $g_i+\Delta_i-1$ elements between v_{i-1} and v_i. (2) makes sure that the range is within $\lfloor 2\epsilon N \rfloor - 1$. Therefore, for any $\phi \in (0,1)$, there always exists a tuple (v_i, g_i, Δ_i), where $r(v_i) \in [\lfloor(\phi-\epsilon)N\rfloor, \lfloor(\phi+\epsilon)N\rfloor]$, and thus v_i is a ϵ-approximation ϕ-quantile. To find the approximate quantile, we can find the least i satisfying:

$$\sum_{j\leq i} g_j + \Delta_i > 1 + \lfloor \epsilon N \rfloor + max_i(g_i+\Delta_i)/2 \tag{3}$$

and return v_{i-1} as the answer. S also supports several operations:

- **INSERT** Search S to find the least i so that $v_i > v$ and insert $(v, 1, \lfloor 2\epsilon N \rfloor)$ right before the tuple $t_i = (v_i, g_i, \Delta_i)$.
- **DELETE** Update g_{i+1} as $g_{i+1} = g_{i+1} + g_i$ and delete t_i. To maintain (2), t_i is removable only when it satisfies

$$g_i + g_{i+1} + \Delta_{i+1} \leq \lfloor 2\epsilon N \rfloor \tag{4}$$

- **COMPRESS** It can be found that DELETE is adding g of the deleting tuple to that of its predecessor. So we can delete multiple successive tuples, $t_{i+1}, t_{i+2}, \ldots, t_{i+k}$ at the same time by updating g_i as $g_i = g_i + g_{i+1} + g_{i+2} + \cdots + g_{i+k}$ and removing them. GK01 proposed a complicated COMPRESS strategy to reduce the size of S as small as possible: it executes when $t_i, t_{i+1}, \ldots, t_{i+k}$ are removable on the arrival of every $\frac{1}{2\epsilon}$ elements.

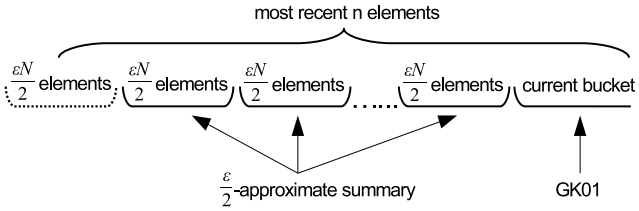


FIGURE 3. Structure of SW model.

It proves that the maximum size of S is $\frac{11}{2\epsilon}\log(2\epsilon N)$. So, its space complexity is $O(\frac{1}{\epsilon}\log(\epsilon N))$.

GK01 is for computing quantiles over all arrived data. Sometimes quantiles of the most recent N elements in a stream are required. Lin *et al.* [29] expanded GK01 and proposed two algorithms for such case: SW model and n-of-N model.

SW model is for answering quantiles over the most recent N elements instantaneously, where N is predefined. The model puts the most recent N elements into several buckets in their arriving order. Rather than original elements, each buckets stores a *Summary* [28] covering $\frac{\epsilon N}{2}$ successive elements. The buckets have 3 states as illustrated in Figure 3:

- A bucket is active when its coverage is less than $\frac{\epsilon N}{2}$. At this time, it maintains a $\frac{\epsilon}{4}$-approximate S computed by GK01.
- A bucket is compressed when its coverage reaches $\frac{\epsilon N}{2}$. The $\frac{\epsilon}{4}$-approximate S would be compressed to $\frac{\epsilon}{2}$-approximate S by an algorithm COMPRESS.
- A bucket is expired if it is the oldest bucket when the coverage of all buckets exceeds N. Once expired, the bucket is removed from the bucket list.

By maintaining and merging buckets, SW model answers quantile queries of elements covered by all unexpired buckets. However, when an old bucket is just expired and the active bucket has not been full, the coverage of all unexpired buckets N' is less than N. The difference between N and N' is at most $\lfloor\frac{\epsilon N}{2}\rfloor - 1$. An algorithm LIFT was proposed to resolve the problem: if $0 \leq N - N' \leq \lfloor\frac{\epsilon N}{2}\rfloor$, it can convert a $\frac{\epsilon}{2}$-approximate S' covering N' elements to a ϵ-approximate S covering N elements. Its worst case space complexity is $O(\frac{1}{\epsilon}\log(\epsilon^2 N) + \frac{1}{\epsilon^2})$.

The distinction of n-of-N model is that it answers quantile queries instantaneously over the most recent n elements where n is any integer not larger than a predefined N. It takes advantage of the EH-partition technique [45] as shown in Figure 4. The technique classifies buckets, marking buckets at level i as $i-bucket$ whose S covers all elements arriving since the bucket's timestamp. There are at most $\lceil\frac{1}{\lambda}\rceil + 1$ buckets at each level, where $\lambda \in (0,1)$. When a new element arrives, the

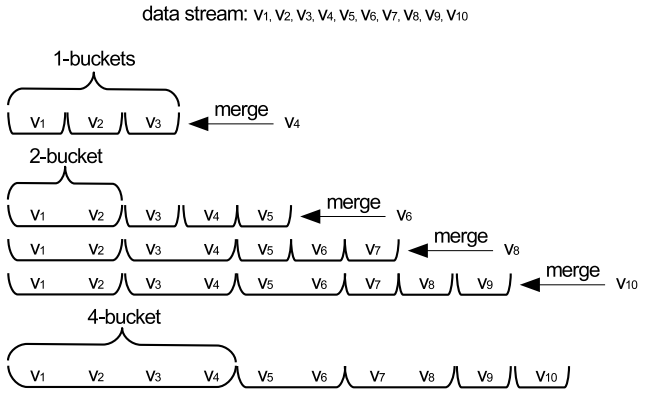


FIGURE 4. Structure of the EH-partition technique in n-of-N model, where $\lambda = 0.5$.

model creates a $1-bucket$ and sets its timestamp to the current timestamp. When the number of $i-bucket$s reaches $\lceil \frac{1}{\lambda} \rceil + 2$, the two oldest buckets at level i are merged into a $2i-bucket$ carrying the oldest timestamp iteratively until buckets at all levels are less than $\lceil \frac{1}{\lambda} \rceil + 2$. Because each bucket covers elements arriving since its timestamp, merging two buckets is equal to removing the later one. Lin *et al.* also proved that for any bucket b, its coverage N_b satisfies

$$N_b - 1 \leq \lambda N \tag{5}$$

In order to guarantee that quantiles are ϵ-approximate, λ is set to $\frac{\epsilon}{\epsilon+2}$. Each bucket preserves a $\frac{\epsilon}{2}$-approximate S. Quantiles are queried as follows:

1) Scan the bucket list until finding the first bucket b making $N_b \leq n$.
2) Use LIFT to convert S_b to a ϵ-approximate S covering n elements.
3) Search S to find the quantile answer.

According to (5), $n - N_b \leq \frac{\epsilon N_b}{\epsilon+2}$. Mark its predecessor bucket as b' and we have $N_{b'} > n$, thus $n - N_b \leq N_{b'} - N_b - 1$, and furthermore $n - N_b \leq \lfloor \frac{\epsilon n}{2} \rfloor$. So, LIFT can be applied to S_b. The worst case space complexity is $O(\frac{1}{\epsilon^2} \log^2(\epsilon N))$.

Arasu and Manku [30] generalized SW model and came up with fixed- and variable- size sliding window approximate quantile algorithms. A window is fixed-size when insertion and deletion of elements must appear in pairs after initialization. The fixed-size sliding window model defines blocks and levels as Figure 5. Each level preserves a partitioning of the data stream into non-overlapping blocks of equal size. Blocks and levels are both numbered sequentially. Assuming the window size is N, the block b in level l contains a *Summary* [28], denoted as $F(N, \epsilon)$ in this model, which covers elements with arriving positions in the range $[b2^l \frac{\epsilon N}{4}, (b+1)2^l \frac{\epsilon N}{4} - 1]$. Similar with SW model, a bucket is assigned one of 3 states at any point of time:

- A block is active if all elements covered by it belong to the current window.
- A bucket is expired if it covers at least one element which is older than the other N elements.

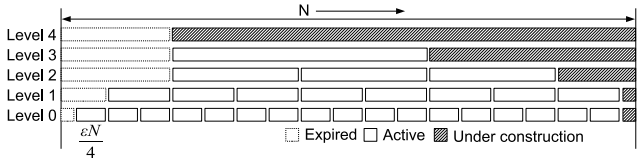


FIGURE 5. Levels and blocks in the fixed-size sliding window model.

- A bucket is under construction if some of its elements belong to the current window while others are yet to arrive.

The highest level with active blocks or blocks under construction L is $\log_2(\frac{4}{\epsilon})$. Blocks in levels above are marked expired. For each active block in level l, a $F(N, \epsilon_l)$ is retained, where $\epsilon_l = \frac{\epsilon}{2(2L+2)} 2^{(L-l)}$. When a block is under construction, GK01 computes its $F(N, \frac{\epsilon_l}{2})$, and it is converted to a $F(N, \epsilon_l)$ in the same way as COMPRESS in SW model. The $F(N, \epsilon_l)$ occupies $O(\frac{1}{\epsilon_l})$ space. Arasu *et al.* proved that using a set of *Summaries* covering $N_1, N_2, \ldots, N_s$ elements each with approximate error $\epsilon_1, \epsilon_2, \ldots, \epsilon_s$, a ϵ-approximate quantile can be computed, where $\epsilon = \frac{\epsilon_1 N_1 + \epsilon_2 N_2 + \cdots + \epsilon_s N_s}{N_1 + N_2 + \cdots + N_s}$. Thus, the fixed-size sliding window model can computes ϵ-approximate quantiles over the last N elements using $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log N)$ space. Contrary to a fixed-size window, a variable-size window bears no limitation on insertion and deletion, so the window size keeps changing. Arasu *et al.* used $V(n, \epsilon)$ to denote a *epsilon*-approximate *Summary* covering n elements, where n is the current size of the window. When a new element arrive, it becomes $V(n+1, \epsilon)$, and when the oldest element leaves, it gets $V(n-1, \epsilon)$. Besides, $F_n(N, \epsilon)$ is defined as a restriction of $F(N, \epsilon)$ to the last n elements as Figure 6. $F_n(N, \epsilon)$ is the same as $F(N, \epsilon)$ except that only blocks whose elements all belong to the most n recent elements, instead of N elements, are assumed active. $V(n, \epsilon)$ can be constructed by a set of $F_n(N, \epsilon)$ in the form of $\{F_n(2^k, \frac{\epsilon}{2}), F_n(2^{k-1}, \frac{\epsilon}{2}), \ldots, F_n(\frac{2}{\epsilon}, \frac{\epsilon}{2})\}$, where k is an integer satisfying $2^{k-1} < n \leq 2^k$. $V(n, \epsilon)$ is maintained under various operations as follows:

- **INSERT** Update all $F_n(N, \epsilon)$ in $V(n, \epsilon)$ by incrementing n. If $n + 1 = 2^k + 1$, create $F_{2^k}(2^{k+1}, \frac{\epsilon}{2})$ from $F_{2^k}(2^k, \frac{\epsilon}{2})$ and insert the new element into it, thus getting $F_{2^k+1}(2^{k+1}, \frac{\epsilon}{2})$.
- **DELETE** Compute $F_{n-1}(2^k, \frac{\epsilon}{2})$ from $F_n(2^k, \frac{\epsilon}{2})$. If $n - 1 = 2^{k-1}$, remove $F_{2^k-1}(2^k, \frac{\epsilon}{2})$.
- **QUERY** In order to query ϵ-approximate quantiles over the most $n' \leq n$ recent elements, find the integer l satisfying $2^{l-1} < n \leq 2^l$. Then query $F_n(2^l, \frac{\epsilon}{2})$ and return the answer.

The space complexity of the variable-size sliding window model is $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log(\epsilon N) \log N)$.

In 2019, Liang *et al.* [31] extended the problem and resolved approximate quantile queries over uncertain data streams. More specifically, it focuses on maintaining quantile *Summaries* [28] over data streams whose elements are drawn from individually domain space, represented by continuous

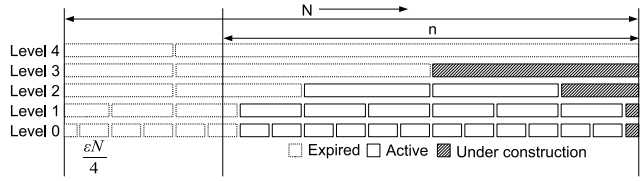


FIGURE 6. $F_n(N, \epsilon)$, a restriction of $F(N, \epsilon)$ to the last n elements.

or discrete pdf. An uncertain data stream S^u is a sequence of elements as $\{e_1^u, e_2^u, \ldots\}$, each of which is drawn from a domain D_i with pdf $f_i : D_i \rightarrow (0, 1]$ such that $\sum_{p \in D_i} f_i(p) = 1$. So, S^u is a concise representation of exponential or infinite number of possible worlds $\mathbb{W}$. Each world $W = \{p_i | pi \in D_i, i = 1, 2, \ldots\}$ is a deterministic stream with probability $Pr(W) = \prod_{p_i \in W} f_i(p_i)$. Therefore, the definition of approximate quantiles is generalized as the element $p \in D_i$ of $e_i^u \in S^u$ such that $\sum_{W \in \mathbb{W}} Pr(W) q(r, r_W^p) - \sum_{W \in \mathbb{W}} Pr(W) q(r, r_W^{p_{min}}) \leq \epsilon N$, where r_W^p is the rank of p in W and p_{min} is the element minimizing $\sum_{W \in \mathbb{W}} Pr(W) q(r, r_W^p)$. Liang *et al.* proposed 2 error metric functions as $q(r, r_W^p)$, the squared error function $q(r, r_W^p) = (r - r_W^p)^2$ and the related error function $q(r, r_W^p) = r - r_W^p$. Following GK01, an online algorithm, namely UN-GK, is introduced. UN-GK adjusts tuples in *Summaries* as $v_i = p_i \in D_j$ of $e_j^u \in S^u$, $g_i = PC_{min}(p_i) - PC_{min}(p_{i-1})$, and $\Delta_i = PC_{max}(p_i) - PC_{min}(p_i)$, where $PC_{min}(p)$ and $PC_{max}(p)$ is the lower bound and the upper bound of probabilistic cardinality [46] of elements in S^u no larger than p. And (2) is modified as $g_i + \Delta_i \leq 2\epsilon PC(S^u)$, where $PC(S^u)$ is the probabilistic cardinality of all elements in S^u as $PC(S^u) = |S^u|$. In this way, a ϵ-approximate quantile can be queried anytime over an uncertain data stream, and its space complexity is $O(\frac{1}{\epsilon} \log(\epsilon PC(S^u)))$.

B. DISTRIBUTED MODEL

In distributed models such as sensor networks, communication between nodes consumes much energy and cuts down the battery life of power-constrained devices. In addition, data transmission takes most of the running time of algorithms. Therefore, the priority of approximate quantile algorithms over distributed models is to reduce the communication cost by decreasing the size of transmitted data.

In 2004, Shrivastava *et al.* [16] designed an approximate quantile algorithm on distributed sensor networks with fixed-universe data, named as q-digest. Fixed-universe data refers to elements from a definite collection. Q-digest uses a unique binary tree structure to compress and store elements so that the storage space is cut down. The size of the fixed-universe collection is denoted as σ and the compress coefficient is denoted as k. Figure 7 is the structure of a q-digest tree, which exists in each network node. Each node of the tree covers elements ranging from $v.min$ to $v.max$, recording the sum of their frequencies as $count(v)$. Each leaf represents one element in the universe, in other words, $v.min = v.max$. Take node d as an example, its coverage is [7, 8] and there

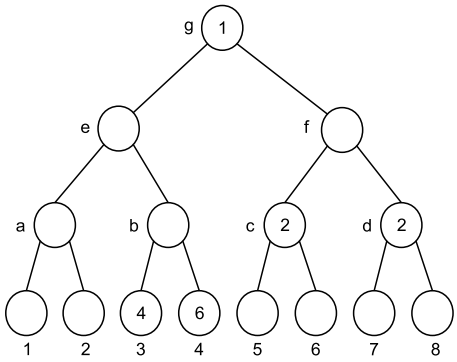


FIGURE 7. Structure of a q-digest tree, where $n = 15, k = 5, \sigma = 8$.

are 2 elements in this range. Besides, each node v of the tree must satisfy:

$$count(v) \leq \lfloor \frac{n}{k} \rfloor \tag{6}$$

$$count(v) + count(v_p) + count(v_s) > \lfloor \frac{n}{k} \rfloor \tag{7}$$

where v_p is v's parent node and v_s is its sibling node. (6) guarantees the upper bound of v' coverage to narrow approximation error while (7) demonstrates when to merge two small nodes so that the storage cost can be reduced.

Q-digest proposed an algorithm COMPRESS to merge nodes of a tree:

1) Scan nodes from bottom to top, from left to right, and sum up $count(v)$ and $count(v_s)$ when (7) is violated.
2) Store the sum in v_p.
3) Remove the count in v and v_s.

In order to reduce data communication, we number nodes in the tree from top to bottom, from left to right and only transmits nonempty nodes in the form of $(id(v), count(v))$, in which way every transmission contains only $O(\log \sigma + \log N)$ data. For example, node c in Figure 7 is transformed to (6, 2). When a network node receives q-digest trees from other nodes, it merges them with its own tree by adding up *count* of nodes representing the same elements and applying COMPRESS. After merging all q-digest trees in the network, we can query quantiles by traversing the ultimate tree in postorder, summing up *count* of passed nodes until the sum exceeds $\lfloor \phi N \rfloor$. The queried quantile is $v.max$ of the current node. The total communication cost of q-digest is $O(\frac{1}{\epsilon} |v| \log \sigma)$, where $|v|$ denotes the number of network nodes.

In the same year, Greenwald and Khanna [32] proposed an algorithm on distributed models, referred to as GK04. Unlike q-digest, GK04 does not require that all data be fixed-universe. It maintains a collection of $\frac{\epsilon}{2}$-approximate *Summaries* [28], denoted as S_v, in each node v in the sensor network. More specifically, $S_v = \{S_v^1, S_v^2, \ldots, S_v^k\}$, where S_v^i covers n_v^i elements, and S_v^i is classified as $class(S_v^i) = \lfloor \log n_v^i \rfloor$. In the network, data are transmitted in the form of S_v. When data arrives at a node, it combines its own S_v with the coming S_v by iteratively merging S_v of the same

class from bottom to top. Once the iteration ends, a pruning algorithm is applied to S_v^i to reduce the number of its tuples to $\log \frac{n_v}{\epsilon} + 1$ at most. The pruning would bring up the approximation error of S_v^i from ϵ' to $\epsilon' + \frac{\epsilon}{2 \log n_v}$ at most, so a ϵ-approximate S is computed after merging S_v from all nodes. GK04 only transmits $O(\frac{1}{\epsilon} \log^2 N)$ data, where N is the number of all data in the network. Moreover, if the height of the network topology is far smaller than N, Greenwald *et al.* improved GK04 to reduce the total communication cost to $O(\frac{|v|}{\epsilon} \log N \log(\frac{h}{\epsilon}))$. Its basic idea is to apply a new operation, REDUCE, which makes sure that all nodes transmits less than $O(\log \frac{h}{\epsilon})$ *Summaries*, after merging and pruning.

Q-digest and GK04 are both offline algorithms that compute quantiles over stationary data in distributed models. Besides, there are also algorithms focusing on distributed models whose nodes receiving continuous data streams. In 2005, Cormode *et al.* [33] proposed a quantile algorithm for the scenario that multiple mutually isolated sensors are connected with one coordinator, which traces updating quantiles in real time. Its goal is to ensure ϵ-approximate quantiles at the coordinator while minimizing communication cost between nodes and the coordinator. In general, each remote node maintains a local approximate *Summary* [28] and informs the coordinator after certain number of updates. In the coordinator, the approximation error ϵ is divided into 2 parts as $\epsilon = \alpha + \beta$:

- α is the approximation error of local *Summaries* sent to the coordinator.
- β is the upper bound on the deviation of local *Summaries* since the last communication.

Intuitively, larger β allows for large deviation, thus less communication between nodes and the coordinator. But because ϵ is fixed, α is smaller, increasing the size of *Summaries* sent to the coordinator each time. In other words, α and β are on the trade-off relationship. To resolve the trade-off, Cormode *et al.* introduces a prediction model in each remote node that captures the anticipated behavior of its local data stream. With the model, the coordinator is able to predict the current state of a local data stream while computing the global *Summary* and the remote node can check for the deviation between its *Summary* and the coordinator's prediction. The algorithm proposes 3 concise prediction models:

- **Zero-Information Model** assumes that there is no local update at any remote node since the last communication.
- **Synchronous-Updates Model** assumes that at each time step, each local node receives one update to its distribution.
- **Update-Rates Model** assumes that updates are observed at each local node at a uniform rate with a notion of global time.

Using prediction models to reduce communication times, the total communication cost of this algorithm is $O(\frac{|v|}{\epsilon^2} \log N)$.

In 2013, Yi and Zhang [34] proposed an algorithm in the same scenario and optimized the cost to $O(\frac{|v|}{\epsilon} \log N)$. The algorithm divides the whole tracking period into $O(\log N)$ rounds. A new round begins whenever N doubles. It first resolves the median-tracking problem, which can be easily generalized to the quantile-tracking problem. Assume that M is the cardinality of data, fixed at the beginning of a round, and m is the tracking median at the coordinator. The coordinator maintains 3 data structures. The first one is a dynamic set of disjoint intervals, each of which contains between $\frac{\epsilon M}{8}$ and $\frac{\epsilon M}{2}$ elements. The others are 2 counters $C.\Delta(L)$ and $C.\Delta(R)$, recording the number of elements received by all remote nodes to the left and the right of m since last update respectively. They are guaranteed with an absolute error at most $\frac{\epsilon M}{8}$ by asking each remote node to send an update whenever it receives $\frac{\epsilon M}{8|v|}$ elements to the left or the right of m. When $|C.\Delta(L) - C.\Delta(R)| \geq \frac{\epsilon M}{2}$, m is updated as follows:

1) Compute the total number of elements to the left and the right of m, $C.L$ and $C.R$ and let $d = \frac{1}{2}|C.L - C.R|$.
2) Compute the new median m' satisfying that $|r(m) - r(m') - d| \leq \frac{\epsilon M}{4}$, where r is the rank of element in all data. Replace m with m'. m' can be found quickly with the set of intervals. First find the first separating element e_1 of the intervals to the left of M. Then compute n_1, the number of elements at all remote sites that are in the interval $[e_1, m]$. If $|n_1 - d| \leq \frac{\epsilon M}{2}$, e_1 is m'. Otherwise find the next separating element e_i and count the number n_i until $|n_i - d| \leq \frac{\epsilon M}{2}$, then set m' to e_i.
3) Set $C.\Delta(L)$ and $C.\Delta(R)$ to 0.

Yi *et al.* has proved that m is at most ϵM elements away from the ground truth. Step 1 needs to exchange $O(|v|)$ messages. As for Step 2, m' can be found after at most $O(1)$ searches and the cost of each search is $O(|v|)$, making the total cost $O(|v|)$. Because each update increases N by a factor of $1 + \frac{\epsilon}{2}$, m is updated at most $O(\frac{1}{\epsilon})$ times. So, the total communication cost is $O(\frac{|v|}{\epsilon})$ each round and $O(\frac{|v|}{\epsilon} \log N)$ for the whole algorithm.

III. RANDOMIZED ALGORITHMS

Generally, randomized approximate quantile algorithms are combined with random sampling. They first sample a part of data and compute overall approximate quantiles with this portion. With less data taken into computation, the computation cost is cut down. In fact, many deterministic quantile algorithms propose randomized versions in this way [30], [34], [35].

A. STREAMING MODEL

Back to 1971, Vapnik and Chervonenkis [47] proposed a randomized quantile algorithm with space complexity of $O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon})$. This benchmark were raised to $O(\frac{1}{\epsilon} \log^2 \frac{1}{\epsilon})$ by Manku *et al.* [35] in 1999, referred to as MRL99. MRL99 does not require prior knowledge of the data size N and occupies less space in experiments when ϕ is an extreme value. The algorithm are improved based on MRL98 [27] so they have the identical frame with only minor differences in the operation NEW:

1) For each buffer, randomly select an element among r consecutive elements.
2) Repeats this operation k times to get k initial elements.
3) Set the buffer's weight to r.

Notice that MRL99 equals with MRL98 if $r = 1$. Because of the collapsing strategy as Figure 2, the larger weight a buffer has, the greater chance there its data must be retained while collapsing. If r is kept consistent, the probability of newly arrived data being selected will go down continuously. So r should keep changed dynamically, meaning the sampling is nonuniform. MRL99 initially sets r to 2 and traces a parameter h, representing the maximum height of all buffers. When a buffer's height reaches $h+i$ for the first time, where $i \geq 0$, r is doubled. By tuning h, b and k, MRL99 manages to compute approximate quantiles with a probability of at least $1 - \delta$.

Recalling two sliding window models in Section II-A, Arasu *et al.* proposed the fact that the quantile of a random sample of size $O(\frac{1}{\epsilon^2}\log\delta^{-1})$ is an ϵ-approximate quantile of N elements with the probability at least $1 - \delta$. To sample elements of specific size, a fast alternative is to randomly select one out of 2^k successive elements, where $k = \lfloor \log_2 N/(\frac{1}{\epsilon^2}\log\delta^{-1})\rfloor$. In this way, k grows logarithmically along with the data stream as required, so the approximation error and the randomization degree are guaranteed.

Another algorithm was proposed by Agarwal *et al.* [36], which is based on *Summaries* [28]. Its basic idea is to sample tuples from multiple *Summaries* and merge them to compute approximate quantiles with low space complexity. There are two situations while merging: same-weight merges and uneven-weight merges. Same-weight merges are for merging two Ss covering the same number of elements. It contains following steps:

1) Combine the two Ss in a sorted way.
2) Label the tuples in order and classify them by label parity.
3) Equiprobably select one class of tuples as the merged result S_{merged}.

Assuming each S covers k elements, if we have $k = O(\frac{1}{\epsilon}\sqrt{\log(\frac{1}{\epsilon\delta})})$, the algorithm will answer quantile queries with a probability of at least $1-\delta$. Uneven-weight merges are for merging two Ss of different sizes, which can be reduced to same-weight merges by a so-called logarithmic technique [32]. The space complexity of the algorithm is $O(\frac{1}{\epsilon}\log^{1.5}\frac{1}{\epsilon})$.

However, Agarwal *et al.* just proposed and analyzed the algorithm in theory without implementation. Afterwards, Felber and Ostrovsky [37] came up with a randomized algorithm whose space complexity is $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ but also did not realize it. Besides, this algorithm is not actually useful but only suitable for theoretical study because its hidden coefficient of O is too large.

B. DISTRIBUTED MODEL

As for distributed models, Huang *et al.* [38] proposed a randomized quantile algorithm which brings down total communication cost from $O(|v|\log^2\frac{N}{\epsilon})$ in GK04 [32] to $O(\frac{1}{\epsilon}\sqrt{|v|h})$, where h denotes the height of the network topology. It contains two version: the flat model and the tree model. For the flat model, all other nodes are assumed to be directly connected to the root node. The algorithm is designed as following steps:

1) Sample elements in node v with a probability of p and compute their ranks in v, denoted as $r(a, v)$ where a is a sampled element.
2) Transmit sampled elements, as well as ones from its child nodes, to its parent node v_p.
3) Find predecessors of a, denoted as $pred(a, v_s)$, in v's sibling nodes v_s.
4) Estimate the rank of a in v_s, denoted as $\hat{r}(a, v_s)$, according to $r(pred(a, v_s), v_s)$.
5) Compute the approximate rank of a in v_p as $r(a, v_p) = \sum \hat{r}(a, v_s) + r(a, v)$.

If elements are not uniformly distributed in the network and some nodes contain the majority, the total communication cost will increase dramatically as the effect of load imbalance. The algorithm resolves the problem by tuning p based on data amount in each node:

- If the amount is greater than $N/\sqrt{|v|}$, p is set to $1/(\epsilon N_v)$.
- Otherwise, p is set to $\Theta(\sqrt{|v|}/(\epsilon N))$.

The inconsistent probability of sampling makes sure that $O(\frac{1}{\epsilon})$ elements are sampled at most in each node no matter how many elements there exist at first. After transmitting all sampled elements to the root node, the element a whose rank $r(a, v_{root})$ is closest to $\lfloor \phi N \rfloor$ is returned as the queried quantile. For the tree model, things become more complicated for two reasons. First, an intermediate node may suffer from heavy traffic going through if it has too many descendants without any data reduction. Second, each message needs $O(h)$ hops to reach the root node, leading to the total communication of $O(h\sqrt{|v|}/\epsilon)$. To resolve the first problem, Huang *et al.* proposed a algorithm MERGE in a systematic way to reduce data size. As for the second problem, the basic idea is to partition the routing tree into t connected components, each of which has $O(|v|/t)$ nodes. Then each component is shrunk into a "super node". Now the height of the tree reduces to t. By setting $t = |v|/h$, the desired space bound becomes $O(\frac{1}{\epsilon}\sqrt{|v|h})$.

In 2018, Haeupler *et al.* [39], gave a drastically faster gossip algorithm, referred as Haeupler18, to compute approximate quantiles. Gossip algorithms [48] are algorithms that allow nodes in a distributed network to contact with each other randomly in each round and gradually converge to get final results. The algorithm contains two phases. In the first phase, each node adjusts its value so that the quantiles around ϕ-quantile become the median quantiles approximately. And in the second phase, nodes compute their approximate median quantiles to get the global result. Haeupler *et al.* proved that the algorithm requires $O(\log\log N + \log\frac{1}{\epsilon})$ rounds to solve the ϵ-approximate ϕ-quantile problem with high probability.

IV. IMPROVEMENT

So far, in the discussion about approximate quantile algorithms, they are generally used with constant approximation error and indiscriminate performance on data regardless of data distribution. However, in some cases, we may have known that the data is skewed [49]–[53]. In other cases, quantile queries over high-speed data streams need to be updated and answered highly efficiently [54], [55]. In addition, there are also techniques for optimizing quantile computation with the help of GPUs [56], [57]. This section presents several techniques for improving the performance and efficiency of approximate quantile algorithms in various scenarios.

A. SKEWNESS

The first algorithm is known as t-digest, proposed by Dunning and Ertl [53]. T-digest is for computing extreme quantiles such as the 99th, 99.9th and 99.99th percentiles. Totally different from q-digest, its basic idea is to cluster real-valued samples like histograms. But they differ in three aspects. First, the range covered by clusters may overlap. Second, instead of lower and upper bounds, a cluster is represented by a centroid value and an accumulated size on behalf of the number of elements. Third, clusters whose range is close to extreme values contain only a few elements so that the approximation error is not absolutely bounded, but relatively bounded, which is $\phi(1-\phi)$. T-digest can be applied to both streaming models and distributed models because the proposed cluster is a mergeable structure. The merge is restricted by the size bound of clusters. Dunning *et al.* proposed several scale functions to define the bound. The standard is that the size of each cluster should be small enough to get accurate quantiles, but large enough to avoid winding up too many clusters. A scale function is

$$f(\phi) = \frac{\delta}{2\pi} \sin^{-1}(2\phi - 1) \tag{8}$$

where δ is the compression parameters and the size bound is defined as

$$W_{bound} = f(\frac{W_{left} + W}{N}) - f(\frac{W_{left}}{N}) \leq 1 \tag{9}$$

where W_{left} and W are respectively the weight of clusters whose centroid values are smaller than that of the current cluster and of current cluster. As Figure 8 shows, (8) is non-decreasing and is steeper when ϕ is closer to 0 or 1, which means clusters covering extreme values have smaller size, making the algorithm more accurate for computing extreme quantiles and more robust for skewed data.

The second algorithm, proposed by Lin *et al.* [51] and elaborated by Liu *et al.* [52], is aimed at streaming models, using nonlinear interpolation. The algorithm maintains two buffers, the quantile buffer $Q = \{q_1, q_2, \ldots, q_m\}$, where q_i is the approximate ϕ_i-quantile, and the data buffer B of size n, holding the most recent n elements. Q is estimated from observed data and incrementally updated when B is full-filled. In order to estimate the extreme quantiles accurately, the nonlinear interpolation $F(x)$, which is an approximate distribution function estimated from a training set stream, is leveraged together with B and Q to update Q.

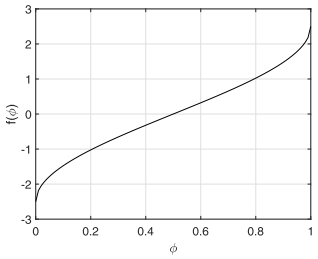


FIGURE 8. Function of (8) with $\delta = 10$.

The third algorithm was proposed by Cormode *et al.* [49] for skewed data. Again, the algorithm is an improved version of GK01 for two problems. The first problem is the biased quantiles problem. Low-biased quantiles are the set of elements whose rank $\lfloor \phi^i N \rfloor$ for $i = 1, 2, \ldots, \log_{1/\phi} N$, and high-biased quantiles are symmetry by reversing the ordering relation. The definition is easy to be generalized to approximate quantiles. The problem is computing the first k elements in high-biased approximate quantiles. Similar with GK01 that g_i and Δ_i are restricted by $g_i + \Delta_i \leq \lfloor 2\epsilon N \rfloor$, the algorithm generalizes the restriction as $g_i + \Delta_i \leq f(r_i, N)$, where $f(r_i, N)$ is an appropriate function. r_i is the rank of v_i equaling $\sum_{j=1}^{i-1} g_j$. For the biased quantiles problem, $f(r_i, N)$ is set to $2\epsilon r_i$. The restriction is tighter than that in GK01 so the correctness is guaranteed. Cormode *et al.* proved that the space lower bound is $\Omega(\frac{1}{\epsilon} min(k \log \frac{1}{\phi}, \log(\epsilon N)))$. The other problem is targeted quantiles problem that quantiles meeting a set of pairs $T = \{(\phi_i, \epsilon_i)\}$ are required to be maintained. In such case, $f(r_i, N)$ is set to $\frac{2\epsilon_i r_i}{\phi_i}$ if $\phi_i N \leq r_i \leq N$ and $2\epsilon_i(N - r_i)/(1 - \phi_i)$ if $0 \leq r_i \leq \phi_i N$.

For problems resolved by q-digest [16] that all elements are selected from a fixed-universe collection, Cormode *et al.* [50] combined the binary tree structure in q-digest and standard dictionary data structures [58], proposing a new deterministic algorithm to compute biased quantiles with space complexity of $O(\frac{1}{\epsilon} \log \sigma \log(\epsilon N))$, where σ denotes the size of the fixed-universe collection.

B. HIGH-SPEED DATA STREAMS

In order to compute quantiles over high-speed data streams, both computational cost and per-element update cost need to be low. Zhang and Wang [55] proposed an algorithm for both fixed- and arbitrary- size high-speed data streams. Let N and n denote the number of elements in the entire data stream and elements seen so far. For the fixed-size data streams, where N is given, a multi-level summary structure $S = \{s_0, s_1, \ldots, s_L\}$ is maintained, where s_i is the summary at level i, as shown in Figure 9. Each element in s is stored with its upper and lower bound of rank, $r_{min}(e)$ and $r_{max}(e)$. The data stream is divided into blocks of size $b = \lfloor \frac{1}{\epsilon} \log \epsilon N \rfloor$ and s_i covers a disjoint bag B_i. Among bags, B_0 contains the most recent blocks even though it may be incomplete. The structure is maintained as follows:

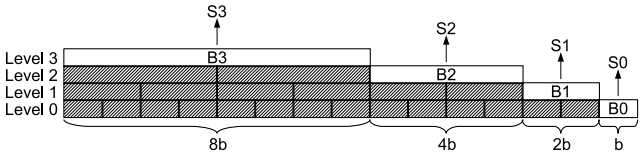


FIGURE 9. A multi-level summary structure with $L = 3$.

1) Insert the new element to s_0.
2) If $|s_0| < b$, the procedure is done. Otherwise, compress s_0 to generate a sketch s_c of size $\lfloor \frac{b}{2} \rfloor$ and send it to level 1. COMPRESS would raise the approximation error from ϵ_0 to $\epsilon_0 + \frac{1}{b}$.
3) If s_1 is empty, set s_1 to s_c and the procedure is done. Otherwise, merge s_1 with s_c and empty s_0. Finally, compress the merged s and send it to level 2.
4) Repeat the steps above until an empty s_i is found.

To answer quantiles, the algorithm first sorts s_0 and merges s at all levels. Then it searches the merged s and return the element satisfying that $r_{min}(e) \geq r - \lfloor \epsilon N \rfloor$ and $r_{max}(e) \leq r + \lfloor \epsilon N \rfloor$. As for the arbitrary-size data streams, the basic idea is to partition the data stream into disjoint sub-streams d_i with the size $\frac{2^i}{\epsilon}$ in their arrival order, and use the algorithm for the fixed-size data streams on each sub-stream because their length is known now. The computational cost and the per-element update cost of both algorithms are $O(N \log(\frac{1}{\epsilon} \log \epsilon N))$ and $O(\log \log N)$. Compared to GK01, the experimental results over high-speed data streams are reported to achieve about $200 \sim 300x$ speedup.

Besides, in order to lighten the burden of massive continuous quantile queries with different ϕ and ϵ, Lin *et al.* [54] proposed 2 techniques for processing queries. The first technique is to cluster multiple queries as a single query virtually while guaranteeing accuracy. Its basic idea is to cluster the queries that share some common results. The second technique is to minimize both the total number of times for reprocessing and the number of clusters. It adopts a trigger-based lazy update paradigm.

C. GPU

Govindaraju *et al.* [56] studied optimizing quantile computation using graphics processors, or GPU for short. GPUs are well designed for rendering and allow many rendering applications to raise memory performance [59]. In order to utilize the high computational power of GPUs, Govindaraju *et al.* proposed an algorithm based on sorting networks. Sorting networks are a set of sorting algorithms mapped well to mesh-based architectures [60]. Operations in the algorithm, including comparisons and comparator mapping, are realized by color blending and texture mapping in GPUs. The theoretical algorithm they used is GK04 [32] and by taking advantage of high computational power and memory bandwidth of GPUs, the algorithm offers great performance for quantile computation over streaming models.

V. APPROXIMATE QUANTILE COMPUTATION TOOLS

Over the past decades, many off-the-shelf, open-source packages or tools for quantile computation have been developed and available to users. Some of them are based on exact quantile computation while others implement approximate quantile computation. In this section, we review such tools, focusing on their algorithmic theories and application scenarios.

MATLAB is a famous numerical computing environment and programming language. It provides comprehensive packages for computing various statistics or functions. Quantiles are included as its in-tool function.[1] The function *quantile* receives a few parameters, including a numerical array, the cumulative probability and the computation kind. It computes quantiles in either exact or approximate way. It computes exact quantiles by the classic algorithm that uses sort. And it implements t-digest [53] for approximate quantiles. So, this function is suitable to compute quantiles over distributed models, which benefit from parallel computation.

The distributed cluster-computing framework, Spark, provides approximate quantile computation since version 2.0.[2] It implements GK01 and can be called in many programming languages such as Python, Scala, R and Java. Computing on a Spark dataframe, the function contains 3 parameters as a dataframe, the name of a numerical column, a list of quantile probabilities and the approximation error. Since version 2.2, the function has been upgraded to support computation over multiple dataframe columns.

In Java, Google publishes an open-source set of core libraries, named as Guava.[3] It includes new collection types, graph libraries, support for concurrency, etc. Also, quantile computation is included in its functions as *median*() and *percentiles*(). The quantiles are exact results so the average time complexity of its implementation is $O(n)$ while the worst case time complexity is $O(N^2)$. It optimizes multiple quantile computation on the same dataset with indexes, improving the performance in some degree. Another package that supports quantile computation is Apache Beam.[4] It is a unified programming model for batch and streaming data processing on execution engines. Unlike Guava, it implements approximate quantile computation.

The programming language, Rust,[5] implements approximate quantile computation over data streams with a moderate amount of memory. It implements the algorithms, GK01 [28] and CKMS [49], so that no boatload of information is stored in memory. Besides, it implements a variant of quantile computation, an ϵ-approximate frequency count for a stream, outputting the k most frequent elements, with the algorithm Misra-Gries [61], which helps with an estimation

[1] https://www.mathworks.com/help/stats/quantile.html
[2] https://spark.apache.org/docs/2.0.0
[3] https://github.com/google/guava
[4] https://beam.apache.org/
[5] https://www.rust-lang.org/

of quantiles as well. As for C++, a peer-reviewed set of core libraries, Boost, provides exact quantile computation as its in-tool function.[6]

VI. FUTURE DIRECTIONS AND CONCLUSIONS

In this survey, we have presented a comprehensive survey of approximate quantile computation. Readers who want to know more about the performance of basic quantile algorithms can read a paper by Luo *et al.* [23], which implements a part of the quantile algorithms above and compares them by experiments.

Even though approximate quantiles have been studied for more than three decades, there is still plenty of room for improvement. The explosion of data also brings new challenges to this field. There are many studies to resolve quantile computation in large-scale data, but not enough. Besides, with the development of industries, new scenarios keep appearing so that quantile computation should be optimized specifically as well. And the evolvement of techniques brings new direction and new potential to quantile computation. Here we present a few future directions for quantile computation.

First, almost all quantile algorithms require at least one-pass scan over the entire dataset, no matter it is applied on streaming models or distributed models. However, at some time, the cost of scanning the entire data is intolerable if quantile queries are demanded to be answered in time. In such case, only a portion of the entire dataset is permitted into the process of the whole algorithm. This condition is more restricted than that in existing randomized quantile algorithms. Even if randomized algorithms sample a part of data to save the space of computation, the process of sampling requires the participation of the entire dataset, which means that they need to scan all data once. To resolve the problem, we need to determine the sampling methods first. Unlike fine-grained sampling in existing randomized algorithms, we may use coarse-grained sampling, such as sampling in the unit of block. The method should also take the way of storage into account. For example, if the dataset is stored distributedly in HDFS [62], we may sample part of blocks, avoiding scanning the entire data. The sampling methods may be exclusive, varying based on applications. After determining sampling methods, we need to consider how much data is sampled to balance the accuracy and the occupation of time and space. One direction is to simulate the idea in machine learning [63] that data are trained (or sampled in our situation) continuously until the quantile result converges.

Second, many new computation engines are being developed. They can be classified into two categories in general. One is streaming computation engines and the other is distributed computation engines. Streaming computation engines, represented by Spark Streaming, Storm and Flink [64], are aimed at real-time streaming needs with minor difference in some ways. For example, Storm and Flink behave like true streaming processing systems with lower latencies while Spark Streaming can handle higher throughput at the cost of higher latencies. Distributed computation engines, represented by Spark and GraphLab [65], are designed to analyze distributed data such as datasets with graph properties.They have their pros and cons in heterogeneous scenarios as well. The characteristics of these engines may be utilized in the implementation of algorithms. As reviewed in Section V, only a small fraction of approximate quantile algorithms is implemented in them such as GK01 in Spark. But many other algorithms are still needed to be implemented and optimized purposefully. For example, Spark [66] is a framework for computation in distributed clusters, supporting parallel computation naturally and having potential of benefitting many quantile algorithms such as MRL98. Its streaming version, Spark Structured Streaming [67], supports stream processing, as well as window operations. It may help quantile algorithms over streaming models, such as Lin SW and Lin n-of-N [29], to be implemented in a more efficient way, even though the theoretical complexity remains the same. Many quantile algorithms are implemented with basic languages while others are even without implementation. Transplanting them to new computation engines is not an easy work and there is great room for optimization.

Third, with the appearance of new specific application scenarios, the requirements of approximate quantile algorithms evolve as well. For example, nowadays, more and more attention is being paid to the correlation of data, and data graph is one way to present the correlation. In a data graph, each edge is usually associated with a weight, representing frequencies of the appearance of the correlation. One may need to determine a threshold for pruning edges by their weights for better performance in analysis [68], [69]. And we can use quantiles to determine the threshold. However, unlike random sampling in a dataset, the edges in a graph are correlated (the frequencies of two edges connecting to the same vertex are correlated). Maybe it is a challenge, as well as an opportunity, to efficiently compute approximate quantiles in a correlated data graph. Furthermore, if the graph is constrained by some patterns [70], [71], the algorithms may be improved and optimized correspondingly. Other cases include computing quantiles in the Blockchain network [72]. Unlike traditional distributed models, which have a master node and multiple slave nodes, the Blockchain network is decentralized, making merging quantile algorithms infeasible. Further research is needed for computing approximate quantiles in such network. Haeupler *et al.* [39], which uses gossip algorithms, provides a good direction, but there is much more to be done.

[6] https://www.boost.org/

REFERENCES

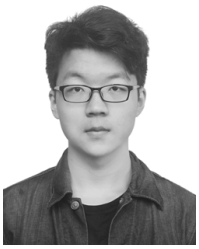
[1] Z. Abedjan, L. Golab, and F. Naumann, "Profiling relational data: A survey," *VLDB J.*, vol. 24, no. 4, pp. 557–581, Jun. 2015.

[2] Z. He, Z. Cai, S. Cheng, and X. Wang, "Approximate aggregation for tracking quantiles and range countings in wireless sensor networks," *Theor. Comput. Sci.*, vol. 607, pp. 381–390, Nov. 2015.

[3] Y. Wang, S. Song, L. Chen, J. X. Yu, and H. Cheng, "Discovering conditional matching rules," *ACM Trans. Knowl. Discovery Data*, vol. 11, no. 4, pp. 1–38, Jun. 2017.

[4] A. Zhang, S. Song, J. Wang, and P. S. Yu, "Time series data cleaning: From anomaly detection to anomaly repairing," *Proc. VLDB Endowment*, vol. 10, no. 10, pp. 1046–1057, Jun. 2017.
[5] S. Song, L. Chen, and H. Cheng, "Efficient determination of distance thresholds for differential dependencies," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 9, pp. 2179–2192, Sep. 2014.
[6] S. Song, A. Zhang, J. Wang, and P. S. Yu, "SCREEN: Stream data cleaning under speed constraints," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Victoria, Australia, 2015, pp. 827–841.
[7] S. Song, C. Li, and X. Zhang, "Turn waste into wealth: On simultaneous clustering and cleaning over dirty data," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, Sydney, NSW, Australia, Aug. 2015, pp. 1115–1124.
[8] S. Guha and A. McGregor, "Stream order and order statistics: Quantile estimation in random-order streams," *SIAM J. Comput.*, vol. 38, no. 5, pp. 2044–2059, Jan. 2009.
[9] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proc. 25st ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst. (PODS)*, Madison, WI, USA, Jun. 2002, pp. 1–16.
[10] S. Muthukrishnan, "Data streams: Algorithms and applications," *Found. Trends Theor. Comput. Sci.*, vol. 1, no. 2, pp. 117–236, 2005.
[11] A. Segall, "Distributed network protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 1, pp. 23–34, Jan. 1983.
[12] Y. Sun, S. Song, C. Wang, and J. Wang, "Swapping repair for misplaced attribute values," in *Proc. 36st Int. Conf. Data Eng. (ICDE)*, Dallas, TX, USA, Apr. 2020.
[13] F. Gao, S.-X. Song, L. Chen, and J.-M. Wang, "Efficient set-correlation operator inside databases," *J. Comput. Sci. Technol.*, vol. 31, no. 4, pp. 683–701, Jul. 2016.
[14] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the data: Parallel analysis with sawzall," *Sci. Program.*, vol. 13, no. 4, pp. 277–298, 2005.
[15] G. Cormode, T. Johnson, F. Korn, S. Muthukrishnan, O. Spatscheck, and D. Srivastava, "Holistic UDAFs at streaming speeds," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Paris, France, Jun. 2004, pp. 35–46.
[16] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: New aggregation techniques for sensor networks," in *Proc. 2nd Int. Conf. Embedded Netw. Sensor Syst.*, Baltimore, MD, USA, Nov. 2004, pp. 239–249.
[17] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, "Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median," *J. Experim. Social Psychol.*, vol. 49, no. 4, pp. 764–766, Jul. 2013.
[18] S. Song, Y. Cao, and J. Wang, "Cleaning timestamps with temporal constraints," *Proc. VLDB Endowment*, vol. 9, no. 10, pp. 708–719, Jun. 2016.
[19] P. Indyk, "Algorithms for dynamic geometric problems over data streams," in *Proc. 36th Annu. ACM Symp. Theory Comput. (STOC)*, Chicago, IL, USA, Jun. 2004, pp. 373–380.
[20] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan, "Time bounds for selection," *J. Comput. Syst. Sci.*, vol. 7, no. 4, pp. 448–461, Aug. 1973.
[21] J. I. Munro and M. S. Paterson, "Selection and sorting with limited storage," *Theor. Comput. Sci.*, vol. 12, no. 3, pp. 315–323, Nov. 1980.
[22] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–33, Mar. 2016.
[23] G. Luo, L. Wang, K. Yi, and G. Cormode, "Quantiles over data streams: Experimental comparisons, new analyses, and further improvements," *The VLDB J.*, vol. 25, no. 4, pp. 449–472, Feb. 2016.
[24] S. Ganguly and A. Majumder, "Cr-precis: A deterministic summary structure for update data streams," in *Proc. Combinatorics, Algorithms, Probabilistic Experim. Methodologies, (ESCAPE)*, Hangzhou, China, Apr. 2007, pp. 48–59.
[25] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *Proc. 13th Int. Conf. Very Large Data Bases, (VLDB)*, Toronto, ON, Canada, Aug./Sep. 2004, pp. 588–599.
[26] R. Motwani and P. Raghavan, "Randomized algorithms," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 33–37, 1996.
[27] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate medians and other quantiles in one pass and with limited memory," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Seattle, WA, USA, Jun. 1998, pp. 426–435.
[28] M. Greenwald and S. Khanna, "Space-efficient online computation of Quantile summaries," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Santa Barbara, CA, USA, May 2001, pp. 58–66.
[29] X. Lin, H. Lu, J. Xu, and J. X. Yu, "Continuously maintaining quantile summaries of the most recent N elements over a data stream," in *Proc. 20th Int. Conf. Data Eng. (ICDE)*, Boston, MA, USA: Mar./Apr. 2004, pp. 362–373.
[30] A. Arasu and G. S. Manku, "Approximate counts and quantiles over sliding windows," in *Proc. 23rd ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst. (PODS)*, Paris, France, 2004, pp. 286–296.
[31] C. Liang, M. Li, and B. Liu, "Online computing quantile summaries over uncertain data streams," *IEEE Access*, vol. 7, pp. 10916–10926, 2019.
[32] M. B. Greenwald and S. Khanna, "Power-conserving computation of order-statistics over sensor networks," in *Proc. 23rd ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst. (PODS)*, Paris, France, Jun. 2004, pp. 275–285.
[33] G. Cormode, M. N. Garofalakis, S. Muthukrishnan, and R. Rastogi, "Holistic aggregates in a networked world: Distributed tracking of approximate Quantiles," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Baltimore, MD, USA, Jun. 2005, pp. 25–36, 2005.
[34] K. Yi and Q. Zhang, "Optimal tracking of distributed heavy hitters and Quantiles," *Algorithmica*, vol. 65, no. 1, pp. 206–223, Oct. 2011.
[35] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Random sampling techniques for space efficient online computation of order statistics of large datasets," in *Proc. SIGMOD ACM SIGMOD Int. Conf. Manage. Data*, Philadelphia, PA, USA, Jun. 1999, pp. 251–262.
[36] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi, "Mergeable summaries," *ACM Trans. Database Syst.*, vol. 38, no. 4, pp. 1–28, Nov. 2013.
[37] D. Felber and R. Ostrovsky, "A randomized online quantile summary in o(1/epsilon * log(1/epsilon)) words," in *Proc. Approximation, Randomization, Combinat. Optim. Algorithms Techn. (APPROX/RANDOM)*, Princeton, NJ, USA, Aug. 2015, pp. 775–785.
[38] Z. Huang, L. Wang, K. Yi, and Y. Liu, "Sampling based algorithms for quantile computation in sensor networks," in *Proc. Int. Conf. Manage. Data SIGMOD*, Athens, Greece, Jun. 2011, pp. 745–756.
[39] B. Haeupler, J. Mohapatra, and H.-H. Su, "Optimal gossip algorithms for exact and approximate quantile computations," in *Proc. ACM Symp. Princ. Distrib. Comput. (PODC)*, Egham, U.K., Jul. 2018, pp. 179–188.
[40] K. Alsabti, S. Ranka, and V. Singh, "A one-pass algorithm for accurately estimating quantiles for disk-resident data," in *Proc. 23rd Int. Conf. Very Large Data Bases*, Athens, Greece, Aug. 1997, pp. 346–355.
[41] F. Chen, D. Lambert, and J. C. Pinheiro, "Incremental quantile estimation for massive tracking," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Boston, MA, USA, Aug. 2000, pp. 516–522.
[42] R. Y. S. Hung and H. Ting, "An $\Omega(\frac{1}{\varepsilon} log \frac{1}{\varepsilon})$ Space Lower Bound for Finding ε-Approximate Quantiles in a Data Stream," in*Proc. 4th Int. Workshop Frontiers in Algorithmics (FAW)*, Wuhan, China, Aug. 2010, pp. 89–100.
[43] R. Jain and I. Chlamtac, "The p^2 algorithm for dynamic calculation of quantiles and histograms without storing observations," *Commun. ACM*, vol. 28, no. 10, pp. 1076–1085, Oct. 1985.
[44] R. Agrawal and A. N. Swami, "A one-pass space-efficient algorithm for finding quantiles," in *Proc. 7th Int. Conf. Manage. Data (COMAD)*, Pune, India, Dec. 1995, pp. 28–43.
[45] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM J. Comput.*, vol. 31, no. 6, pp. 1794–1813, Jan. 2002.
[46] C. Liang, Y. Zhang, P. Shi, and Z. Hu, "Learning accurate very fast decision trees from uncertain data streams," *Int. J. Syst. Sci.*, vol. 46, no. 16, pp. 3032–3050, Mar. 2014.
[47] V. N. Vapnik and A. Y. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," in *Measures of Complexity*. Cham, Switzerland: Springer, 2015, pp. 11–30.
[48] S. P. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: Design, analysis and applications," in *Proc. 24th Annu. Joint Conf. IEEE Comput. Commun. Societies*, Miami, FL, USA, Mar. 2005, pp. 1653–1664.
[49] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Effective computation of biased quantiles over data streams," in *Proc. 21st Int. Conf. Data Eng. (ICDE)*, Tokyo, Japan, Apr. 2005, pp. 20–31.
[50] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Space- and time-efficient deterministic algorithms for biased quantiles over data streams," in *Proc. the 25th ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst. (PODS)*, Chicago, IL, USA, 2006, pp. 263–272.

[51] Z. Lin, J. Liu, and N. Lin, "Accurate quantile estimation for skewed data streams," in *Proc. IEEE 28th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, Montreal, QC, Canada, Oct. 2017, pp. 1–7.
[52] J. Liu, W. Zheng, Z. Lin, and N. Lin, "Accurate quantile estimation for skewed data streams using nonlinear interpolation," *IEEE Access*, vol. 6, pp. 28438–28446, 2018.
[53] T. Dunning and O. Ertl, "Computing extremely accurate quantiles using t-digests," 2019, *arXiv:1902.04023*. [Online]. Available: https://arxiv.org/abs/1902.04023
[54] X. Lin, J. Xu, Q. Zhang, H. Lu, J. Xu Yu, X. Zhou, and Y. Yuan, "Approximate processing of massive continuous quantile queries over high-speed data streams," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 5, pp. 683–698, May 2006.
[55] Q. Zhang and W. Wang, "A fast algorithm for approximate quantiles in high speed data streams," in *Proc. 19th Int. Conf. Sci. Stat. Database Manage. (SSDBM)*, Banff, Canada, Jul. 2007, p. 29.
[56] N. K. Govindaraju, N. Raghuvanshi, and D. Manocha, "Fast and approximate stream mining of quantiles and frequencies using graphics processors," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Baltimore, MD, USA, Jun. 2005, pp. 611–622.
[57] Y. Zhou, H. Wang, and C.-T. Cheng, "Parallel computing method of data stream quantiles with GPU," *J. Comput. Appl.*, vol. 30, no. 2, pp. 543–546, Mar. 2010.
[58] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 1989.
[59] O. Rubinstein, D. Reed, and J. Alben, "Gpu rendering to system memory," U.S. Patent Appl. 10 833 694, Oct. 27, 2005.
[60] K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIPS*, Atlantic City, NJ, USA, Apr./May 1968, pp. 307–314, 1968.
[61] J. Misra and D. Gries, "Finding repeated elements," *Sci. Comput. Program.*, vol. 2, no. 2, pp. 143–152, 1982.
[62] D. Borthakur, "HDFS architecture guide," *Hadoop Apache Project*, vol. 53, nos. 1–13, p. 2, 2008.
[63] C. M. Bishop and N. M. Nasrabadi, "Pattern recognition and machine learning," *J. Electron. Imag.*, vol. 16, no. 4, 2007, Art. no. 049901.
[64] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Chicago, IL, USA, May 2016, pp. 1789–1792.
[65] J. Wei, K. Chen, Y. Zhou, Q. Zhou, and J. He, "Benchmarking of distributed computing engines spark and GraphLab for big data analytics," in *Proc. IEEE Second Int. Conf. Big Data Comput. Service Appl. (BigDataService)*, Oxford, U.K., Mar. 2016, pp. 10–13.
[66] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. 2nd USENIX Workshop Hot Topics Cloud Comput.*, Boston, MA, USA, Jun. 2010, p. 95.
[67] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia, "Structured streaming: A declarative API for real-time applications in apache spark," in *Proc. Int. Conf. Manage. Data SIGMOD*, Houston, TX, USA, Jun. 2018, pp. 601–613.
[68] X. Zhu, S. Song, X. Lian, J. Wang, and L. Zou, "Matching heterogeneous event data," in *Proc. Int. Conf. Manage. Data IGMOD*, Snowbird, UT, USA, Jun. 2014, pp. 1211–1222.
[69] Y. Gao, S. Song, X. Zhu, J. Wang, X. Lian, and L. Zou, "Matching heterogeneous event data," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 11, pp. 2157–2170, 2018.
[70] X. Zhu, S. Song, J. Wang, P. S. Yu, and J. Sun, "Matching heterogeneous events with patterns," in *Proc. IEEE 30th Int. Conf. Data Eng. (ICDE)*, Chicago, IL, USA, Mar./Apr. 2014, pp. 376–387.
[71] S. Song, Y. Gao, C. Wang, X. Zhu, J. Wang, and P. S. Yu, "Matching heterogeneous events with patterns," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 8, pp. 1695–1708, Aug. 2017.
[72] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. IEEE Secur. Privacy Workshops*, San Jose, CA, USA, May 2015, pp. 180–184.



ZHIWEI CHEN received the B.E. degree from the School of Software, Nanjing University, China, in 2018. He is currently pursuing the M.E. degree with the School of Software, Tsinghua University, China. His research interest includes data quality analytics.

AOQIAN ZHANG received the B.E. degree in computer software and the Ph.D. degree in software engineering from the School of Software, Tsinghua University, in 2013 and 2018, respectively. He is currently a Postdoctoral Fellow with the Data System Group, Cheriton School of Computer Science, University of Waterloo.