# Quantiles on Streams

Chiranjeeb Buragohain
Amazon.com
Seattle, WA 98104, USA
chiran@amazon.com

Subhash Suri
Dept. of Computer Science,
University of California
Santa Barbara, CA 93106, USA
suri@cs.ucsb.edu

## SYNONYMS

Median; histogram; selection; order statistics

## DEFINITION

Quantiles are order statistics of data: the $\phi$-quantile ($0 \leq \phi \leq 1$) of a set $S$ is an element $x$ such that $\phi|S|$ elements of $S$ are less than or equal to $x$ and the remaining $(1-\phi)|S|$ are greater than $x$. This article describes data stream (single-pass) algorithms for computing an approximation of such quantiles.

## HISTORICAL BACKGROUND

The need to summarize data has been around since the earliest days of data processing. Large volumes of raw, unstructured data easily overwhelm human ability to comprehend or digest, and tools that help identify the major underlying trends or patterns in data have enormous value. Quantiles characterize distributions of real world data sets in ways that are less sensitive to outliers than simpler alternatives such as the mean and the variance. Consequently, quantiles are of interest to both database implementers and users: for instance, they are a fundamental tool for query optimization, splitting of data in parallel database systems, and statistical data analysis.

Quantiles are closely related to the familiar concepts of frequency distributions and histograms. The cumulative frequency distribution $F()$ is commonly used to summarize the distribution of a (totally ordered) set $S$. Specifically, for any value $x$,

$$F(x) = \text{Number of values less than } x. \tag{1}$$

The quantile $Q(\phi)$, or the $\phi$-th quantile is simply the inverse of $F(x)$. Specifically, if the set $S$ has $n$ elements, then the element $x$ has the property that

$$Q(F(x)/n) = x. \tag{2}$$

Thus, the 1/2-quantile is just the familiar median of a set, while 0- and 1-quantiles are the minimum and the maximum elements of the set. Histograms are another popular method for summarizing data into a smaller number of "buckets": the buckets only retain the information how many elements fall between two consecutive bucket boundaries, but not their precise values. It is easy to see that a sequence of quantiles resembles a *histogram* of the underlying set, and provides a natural and complete summary of the entire *distribution* of the data values.

In computer science, sorting and selection (another name for quantile computation) are two of the most basic problems, with long and intellectually rich history of research. Indeed, the computational complexity of selection, namely, determining the element of a given rank, is one of the earliest celebrated problems, and the elegant, linear-time algorithm of Blum et al. [2] is a classical result, taught regularly in the undergraduate algorithms and data structures course. For more recent theoretical results on the complexity of selection in the classical comparison-complexity model, please refer to the survey by Paterson [16].

## SCIENTIFIC FUNDAMENTALS

The problem of quantile computation, while well-solved in the classical model of computation, assumes a new and challenging character within the constraints of single-pass computation (the streaming model). Indeed, when the algorithm's memory is limited and significantly smaller than the size of the data set $S$, it is not possible to compute the quantile precisely, and the best possible solution is an approximation. This was formalized in a 1980 paper by Munro and Paterson [15] who proved that any algorithm that determines the median of a set by making at most $p$ sequential scans of the input requires at least $\Omega(n^{1/p})$ working memory. Thus, computing the true median will require memory linear in the size of the set.

Against this backdrop, therefore, the main focus of recent research has been on achieving provable-quality approximation of the quantiles. In particular, an $\epsilon$-approximate quantile summary of a sequence of $n$ elements is a data structure that can answer quantile queries about the sequence to within a precision of $\epsilon n$. In other words, when queried for a $\phi$-quantile, for $0 \leq \phi \leq 1$, the structure returns an element $x$ that is guaranteed to be in the $[\phi - \epsilon,\ \phi + \epsilon]$ quantile range. The key evaluation metric for the performance of these approximation schemes is the size (memory footprint) of their summary data structures, although other factors such as simplicity of implementation are also desirable.

The discussion in this article will focus on algorithms that operate in the *data stream* model: the algorithm is endowed with a finite memory, which is significantly smaller in size than the size of the input; the input is presented to the algorithm in an arbitrary (perhaps adversarial) order; and any data not explicitly stored by the algorithm is irretrievably lost. Thus, the algorithm is restricted to a single scan of the data in the input order, and after this scan it must output an approximation of the quantiles of the input values.

Before discussing the state of the art for this problem, it may help to consider a real-world scenario for the use of quantiles in data streams, both to illustrate a motivating application and to appreciate the scale of the problem. A web site, such as a search engine, consists of several web server hosts; the users' queries (requests) are collectively handled by these servers (using some scheduling protocol); and the overall performance of the web site is characterized by the latency (delay) encountered by the users. The distribution of the latency values is typically very skewed, and a common practice is to track some particular quantiles, for instance, the 95th percentile latency. In this context, one can ask several questions.

- What is the 95th percentile latency of a single web server?
- What is the 95th percentile latency of the entire web site (over all the servers)?
- What is the 95th percentile latency of the website during the last one hour?

The Yahoo website, for instance, handles more than 3.4 billion hits per day, which translates to 40000 requests per second. The Wikipedia website handles 30000 requests per second at peak, with 350 web servers.

While all three questions relate to computing of quantiles, they have different technical nuances, and often require different algorithmic approaches. In particular, the first question is the most basic version, asking for a determination of quantiles for a stream of data; the second extends the setting to *distributed input*, and thus demands an algorithm in the distributed computing model. The third problem is an instance of the *sliding window* model, where the computation must occur over a subset (time window) of the stream, and this subset is continuously changing. This article is primarily focused on the stream setting (problem 1), also known as the cash register model, but will also discuss, when appropriate, extensions to these other models.

## Randomized Algorithms

One can estimate the quantiles of a stream by the quantiles of a random sample of the input. The key idea is to maintain a random sample of appropriate size and when asked for a quantile of the input set, simply report the corresponding quantile of the random sample. If the size of the input stream, $N$ is known, then the following simple algorithm can compute a random sample of size $k$ in one-pass: choose each element independently with probability $k/N$ to include in the sample. If the size of the full data stream is not known in advance, or if the ability to answer queries during reading the stream is required as well, then the reservoir sampling algorithm of Vitter [19] can be used instead. To maintain a sample of size $k$, the reservoir sampling algorithms begins by including the first $k$ stream elements in the sample; from then on, the $i$th element from the stream is chosen with

probability $i/n$. If the $i$th element is chosen, one of the elements from the current sample is evicted uniformly at random to keep the size of the sample constant at $k$. While straightforward to implement, random sampling has the disadvantage of needing a rather large sample to achieve expected approximation accuracy. Specifically, in order to estimate the quantiles with precision $\epsilon n$, with probability at least $1-\delta$, a sample of size $\Theta(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ is required, where $0 < \delta < 1$.

In [5], Cormode and Muthukrishnan proposed a more space-efficient data structure, called Count-Min sketch, which is inspired by Bloom filters. Count-Min sketch allows $\epsilon$-approximation of quantiles using $O(\frac{1}{\epsilon}\log^2 n \log(\frac{\log n}{\phi\delta}))$ memory. Although the space needed by Count-Min is worse than the two deterministic schemes discussed below, it has the advantage of allowing general updates to the streams: past elements can be deleted as well as their values updated.

## Deterministic Algorithms

The first deterministic streaming algorithm for quantiles was proposed by Manku, Rajagopalan and Lindsay [13, 14], building on the prior work by Munro and Paterson's [15]. This algorithm has space complexity $O(\frac{1}{\epsilon}\log^2 \epsilon n)$, meaning that using memory that grows poly-logarithmically in the stream size and inversely with the accuracy parameter $\epsilon$, the quantiles can be estimated with precision $\epsilon n$. This result has since been improved by two groups: in [9], Greenwald and Khanna propose a $O(\frac{1}{\epsilon}\log \epsilon n)$ memory scheme, and in [18], Shrivastava, Buragohain, Agrawal and Suri propose a $O(\frac{1}{\epsilon}\log U)$ memory scheme, where $U$ is the size of domain from which the input is drawn.

The Greenwald-Khanna (GK) algorithm is based on the idea that if a sorted subset $\{v_1, v_2, \cdots\}$ of the input stream $S$ (of current size $n$) can be maintained such that the ranks of $v_i$ and $v_{i+1}$ are within $2\epsilon n$ of each other, then an arbitrary quantile query can be answered with precision $\epsilon n$. Their main contribution is to show how to maintain such a subset of values using a data structure of size $O(\frac{1}{\epsilon}\log \epsilon n)$. The Q-Digest scheme of Shrivastava et al. [18] approaches the quantile problem as a histogram problem over a universe of size $U$; thus $\log U$ is the number of bits needed to represent each element. Q-Digest maintains a set of buckets dynamically, merging those that are light (containing few items of the stream) and splitting those that are heavy, with an aim to keep the relative sizes of all the buckets nearly equal. Specifically, using a $O(\frac{1}{\epsilon}\log U)$ size data structure, Q-Digest ensures that the input stream is divided into $O(1/\epsilon)$ buckets, with each bucket containing $O(\epsilon n)$ items. Thus, the rank of any item can be determined with precision $\epsilon n$ by locating its bucket.

In theoretical terms, the GK scheme has better performance when the input is drawn from a large universe, but the stream itself has only modest size. The Q-Digest, on the other hand, is superior when the stream size is huge but elements are drawn from a smaller universe. The GK algorithm is very clever, but requires a sophisticated analysis. The Q-Digest is simpler to understand, analyze, and implement, and it lends itself to easy extensions to distributed settings.

## Practical Considerations

A detailed study of the empirical performance of quantile algorithms was carried out by Cormode et al. [3] on IP stream datasets. They concluded that with careful implementation, a commodity hardware machine (dual Pentium 2.8GHz CPU and 4GB RAM) can keep up with a 2Gbit/second stream (310,000 packets/second). Performance numbers can depend also on the input distribution. For example, the deterministic algorithms presented above can have different memory usage and accuracy depending on the order in which the input values are presented, but sketching techniques such as the Count-Min sketch are not affected by the order of the input. The input value distribution can also impact perceived accuracy of the approximate quantiles. For example, for skewed distributions, the *numeric value* of the exact $\phi$-th quantile can be arbitrarily far from the numeric values of the $(\phi \pm \epsilon)$-th quantile.

## Extensions

Given the fundamental nature of quantiles and their widespread applications in data processing, it is no surprise that there are multiple extensions of the basic setting that have been considered so far. There are many interesting and practically-motivated applications, such as the latency of the web site mentioned earlier, where quantiles must be computed over distributed data, or over a sliding window portion of the stream etc. In the following, the current state of algorithms for these variants are briefly discussed.

**Quantiles in distributed streams.** In many settings, data of interest are naturally distributed across multiple sources, such as servers in a web application and measurement devices in a sensor network. In these applications, it is necessary to compute the quantile summary of the entire data, but *without* creating a centralized repository of data, which could be undesirable because of the additional latency, communication overhead, or energy constraints of untethered sensors. The efficiency of an algorithm in this distributed setting is measured by the amount of information each node in the system must transmit during the computation.
One natural approach for distributed approximation of quantiles is for each node (server, sensor, etc.) to compute a local summary of its data, and arrange the nodes in a virtual hierarchy that guides them to merge these summaries into a final structure computed at the root of the hierarchy. The tree-based Q-digest [18] algorithm extends rather easily to the distributed setting, as the the histogram boundaries of the Q-Digest are aligned to binary partition of the original value space $U$. The space complexity of the distributed version remains the same as the stream version, namely, $O(\frac{1}{\epsilon}\log U)$. The GK algorithm is little more complicated to extend to distributed streams, but Greenwald and Khanna themselves developed such an extension in [10] However, the space complexity of their distributed data structure grows to $O(\frac{1}{\epsilon}\log^3 n)$ [10]. The Bloom filter based Count-Min sketch [5] also extends easily to the distributed setting also without any increase in the space complexity.

**Quantiles in sliding windows.** In many applications, the user is primarily interested in the most recent portion of the data stream. This poses the *sliding window* extension of the quantiles problem, in which the desired quantile summary for the most recent $N$ data elements—the window slides with the arrival of each new element, as the oldest element of the window is discarded and the new arrival added. In [12], Lin et al. presented such a sliding window scheme for quantile summaries, however, the space requirement for their algorithm is $O(\frac{1}{\epsilon^2} + \frac{1}{\epsilon}\log \epsilon^2 N)$. This was was soon improved by Arasu and Manku [1] to $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\log N)$.

**Biased estimate of quantiles.** The absolute measure of approximation precision is quite reasonable so long as the error $\epsilon n$ is quite small compared to the rank of the quantile sought, namely, $\phi n$. This holds as long as the quantiles of interest are in the middle of the distribution. But if $\phi$ is either close to 0 or 1, one may prefer a *relative* error, so that the estimated quantile is in the range $[(1-\epsilon),\ (1+\epsilon)]\phi n$. This variant was solved by Gupta and Zane [11] using random sampling techniques with a $O(\frac{1}{\epsilon^3}\log n \log\frac{1}{\delta})$ size data structure. The space bound has since been improved by Cormode et al. [4] to $O(\frac{1}{\epsilon}\log U \log(\epsilon n))$ using a deterministic algorithm.

**Duplicate insensitive quantiles.** In some distributed settings, a single event can be observed multiple times. For example in the Internet, a single packet is observed at multiple routers; in wireless sensor networks, due to the broadcast nature of the medium, and to add fault-tolerance, data can be routed along multiple paths. Summaries such as quantiles or the number of distinct items are clearly not robust against duplication of data; on the other hand, simpler statistics such as minimum and maximum are not affected by duplication. Flajolet and Martin's distinct counting algorithm [8] is the seminal work in this direction. Cormode et al. have introduced algorithms based on sampling to compute various duplicate insensitive aggregates [6]. Their Count-Min sketch can be also easily adapted to compute duplicate insensitive quantiles.

## KEY APPLICATIONS

Internet-scale network monitoring and database query optimization are two important applications that originally motivated the need for quantiles summaries over data streams. Gigascope [7] is a streaming database system that employs statistical summaries such as quantiles for monitoring network applications and systems. Quantile estimates are also widely used in query optimizers to estimate the size of intermediate results, and use those estimates to choose the best execution plan [13]. Distributed quantiles have been used to succinctly summarize the distribution of values occurring over a sensor network [18]. In a similar context, distributed quantiles are also used to summarize performance of websites and distributed applications [17].

## FUTURE DIRECTIONS

The field of computing approximate quantiles over streams have led to a fertile research program and is expected to bring up new challenges in both theory and implementation. Although there is an obvious lower bound of $\Omega(\frac{1}{\epsilon})$ memory required to compute $\epsilon$-approximate quantiles, no non-trivial lower bound on memory is known. Since the current best algorithms requires $O(\frac{1}{\epsilon}\log(\epsilon n))$ or $O(\frac{1}{\epsilon}\log(U))$ memory, it will be useful to either lower the memory usage or prove a better lower bound.

Another direction in which improvements are highly desirable is running time. The current deterministic algorithms require amortized running time of $O(\log\frac{1}{\epsilon} + \log\log(\epsilon n))$ or $O(\log\frac{1}{\epsilon} + \log\log(U))$ per item. In high data-rate streams, even such low processing times are not fast enough: what is desired is a $O(1)$ insert time, or even a sublinear time quantile algorithm. As of now, there is no memory efficient sub-linear time quantile algorithm known except for random sampling.

## RECOMMENDED READING


[1] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *Proc. of the PODS '04*, 2004.

[2] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[3] G. Cormode, F. Korn, S. Muthukrishnan, T. Johnson, O. Spatscheck, and D. Srivastava. Holistic udafs at streaming speeds. In *Proc. of ACM SIGMOD*, 2004.

[4] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Space and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proc. of PODS '06*, 2006.

[5] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[6] G. Cormode, S. Muthukrishnan, and W. Zhuang. What's different: Distributed, continuous monitoring of duplicate-resilient aggregates on data streams. In *Proc. of the 22nd International Conference on Data Engineering*, 2006.

[7] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *Proc. ofthe ACM SIGMOD*, 2003.

[8] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[9] J. M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proc. the 20th ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD)*, 2001.

[10] J. M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proc. of 23rd ACM Symposium on Principles of Database Systems (PODS)*, 2004.

[11] A. Gupta and F. Zane. Counting inversions in streams. In *Proc. of ACM-SIAM SODA*, 2003.

[12] 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. of ICDE*, pages 362–374, 2004.

[13] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proc. of ACM SIGMOD '98*, pages 426–435, New York, NY, USA, 1998. ACM Press.

[14] 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. of ACM SIGMOD '99*, pages 251–262, New York, NY, USA, 1999. ACM Press.

[15] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, pages 315–323, 1980.

[16] M. S. Paterson. Progress in selection. In *Scandinavian Workshop on Algorithm Theory*, pages 368–379, 1996.

[17] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming Journal*, 13(4):227–298, 2005.

[18] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *Proc. of SenSys'04*, 2004.

[19] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 1985.