

Space Efficient Quantile Summary for Constrained Sliding Windows on a Data Stream

Jian Xu Xuemin Lin Xiaofang Zhou

Computer Science & Engineering
The University of New South Wales
Sydney 2052, Australia
{xujian, lxue}@cse.unsw.edu.au

School of ITEE
The University of Queensland
Brisbane QLD 4072, Australia
zxf@itee.uq.edu.au

Abstract. In many online applications, we need to maintain quantile statistics for a sliding window on a data stream. The sliding windows in natural form are defined as the most recent N data items. In this paper, we study the problem of estimating quantiles over other types of sliding windows. We present a uniform framework to process quantile queries for time constrained and filter based sliding windows. Our algorithm makes one pass on the data stream and maintains an ϵ -approximate summary. It uses $O(\frac{1}{\epsilon^2} \log^2 \epsilon N)$ space where N is the number of data items in the window. We extend this framework to further process generalized constrained sliding window queries and proved that our technique is applicable for flexible window settings. Our performance study indicates that the space required in practice is much less than the given theoretical bound and the algorithm supports high speed data streams.

1 Introduction

In many applications, massive volume of data is generated and collected from distributed sources and needs to be processed in real time. The data can be modelled as an unbounded data sequence arriving at a port of the system thus is given the name “**data stream**”. The uprising requests of processing data streams call for a revolutionary view and new techniques. In stream processing, a data item can be accessed at its arriving time only once. As the size of a data stream is often unbounded, processing needs to make as small size memory footprints as possible. On one hand, one pass of strict ordered scan of data makes save both on access time and storage. On the other hand, many operations such as join and aggregations become non-trivial under data stream model. Elegant algorithms are needed to meet the space and time restrictions.

New semantic for processing data streams keeps being discovered. Among those, processing data streams over sliding windows receives much research attention[1, 2]. As data arrives in order, it is natural to define “old” data which is observed earlier and “recent” ones which has just arrived. In many applications, only recent data items are interested to the user. Therefore, computations need to focus on a sliding window which contains only a portion of most recent data in the stream. In this paper, we study quantile problems on sliding windows.

Quantile information unveils important information on the data distribution of a data set. The task is to compute ϕ quantile, which is the value of the data at rank $\lceil \phi N \rceil$ (after sorting) for a set of N data elements. Early in 1980, Munro

and Paterson proved in [5] an $\Omega(N^{\frac{1}{p}})$ lower bound on space requirement for computing exact quantiles over N data elements in p passes of scanning the data. In data stream, as it's impossible to get accurate quantiles, it is desirable to have estimations with accuracy guarantees. The work [7, 8] studied the problem of estimating approximate quantiles for whole data stream. The space requirement is $O(\frac{1}{\epsilon} \log^2 \epsilon N)$ for ϵ -approximate quantiles. In a recent paper [4], this problem was re-studied for append only data stream. A deterministic algorithm using a space of $O(\frac{1}{\epsilon} \log \epsilon N)$ was presented. In [3], the authors studied the problem on a data stream that has both append and deletion. The algorithm uses $O(\log^2 |U| \log(\log(|U|)/\delta)/\epsilon^2)$ space, given the value domain U of data and the algorithm estimates ϵ -approximate quantiles with confidence $1 - \delta$. The only work we know for estimating quantiles on sliding windows is [6]. In this paper, the authors gave a solution for estimating quantiles on a sliding window containing a fixed number of data items. The algorithm uses $O(\frac{1}{\epsilon^2} + \frac{1}{\epsilon} \log \epsilon^2 N)$ space for sliding window containing N data items. In addition to the natural form of sliding window, a window may be constrained to other parameters such as “*data arrived in most recent 5 minutes*” or “*phone calls originated from CSE in the most recent 10000 calls inside campus*”. Challenges on such sliding windows lie on two aspects. Firstly, the number of data items in the window is unknown and changes over time. The technique in [6] does not work without an apriori to the window size. Secondly, constraints may have various of forms and a practical query processing should be able to handle different window queries in a uniform way rather than hacking from one specific solution to another. Our contribution can be summarized as following:

1. We give a sub-linear deterministic algorithm for estimating ϵ -approximate quantiles over time constrained and filter based sliding windows.
2. We give a uniform framework for estimating ϵ -approximate quantiles under an extended sliding window definition, using sub-linear space.

2 Preliminary

In this section, we introduce some background of quantile estimation over data streams and define terms to use in the later sections.

2.1 Quantile computations on data stream

Definition 1 (Quantile). A ϕ -quantile ($\phi \in (0, 1]$) of an ordered sequence of N data items is the data item at rank $\lceil \phi N \rceil$.

Figure 1 is an example for quantiles on a data stream of 8 items. As the

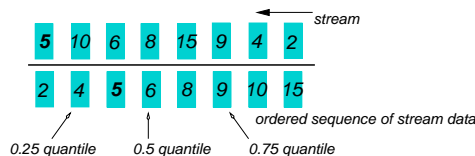


Figure 1. quantiles of a sequence

difference between ϕN and $\lceil \phi N \rceil$ is smaller than 1, without loss of generality, we

use ϕN as the requested quantile point instead of $\lceil \phi N \rceil$ in the paper. Definition 2 gives the quantitative approximation measurement.

Definition 2 (ϵ -approximate quantile). An ϵ -approximate answer for a quantile query $Q(\phi)$ is a data item of which the rank r satisfies $|r - \phi N| \leq \epsilon N$. Where N is the number of data elements in the sequence.

Generally, a summary which estimates ϵ -approximate quantiles may be in any form. In our study, we use *quantile sketch*, or *sketch* for brevity, to denote a summary structure proposed in [4], as the quantile summary of a data stream.

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

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

An ϵ -approximate sketch is also written as ϵ -sketch for short. And Greenwald and Khanna proved the following lemma in [4].

Lemma 1 (GK ϵ -sketch). An ϵ -sketch satisfies the following conditions.

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

To query the sketch for quantile ϕ , the algorithm scans the sketch and returns a tuple (v, r^-, r^+) which $r^- > \phi N - \epsilon N$ and $r^+ < \phi N + \epsilon N$. It was shown in [4] that maintaining such a sketch requires $O(\frac{1}{\epsilon} \log \epsilon N)$ space. We use the term **GK-sketch** to refer to an instance of a sketch maintained by algorithm in [4].

2.2 Constrained sliding windows

The sliding window studied in [6] can be viewed as of a natural-form. It is defined on the number of data items in the window. For the windows defined on other constraints, we call them “*constrained sliding windows*”. Fig. 2 shows the difference between the two types of windows. A strict definition on accepted

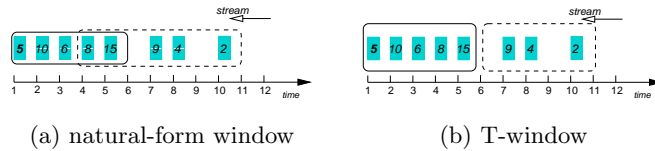


Figure 2. two types of sliding windows

constraints to a sliding window will be derived in section 4, after we show the framework for estimating quantiles for two typical constrained windows.

3 The sliding window techniques

In this section, we present a framework for estimating approximate quantiles over constrained sliding windows and show how quantiles can be estimated for *time constrained* and *filter based* sliding windows.

3.1 The framework

The basic idea of our algorithm is to partition the stream along its incoming order and build sketches on each partition so that at any time, a query can be answered by extracting an appropriate sketch from the summary.

The validity of partitioning a data stream while still keeping the accuracy of quantile estimations is based on the following theorem.

Theorem 1. *Given an $\frac{\epsilon}{2}$ -sketch S_n on the most recent n data items, then a ϕ -quantile query estimation from S_n is an ϵ -approximate ϕ -quantile for the most recent N data items, if $n \leq N \leq (1 + \frac{\epsilon}{2})n$.*

To build a summary, we partition the stream into k buckets b_1, b_2, \dots, b_k . For each bucket, we maintain an $\frac{\epsilon}{2}$ -sketch which covers all data items that arrived after the bucket was constructed. Let the number of data covered by the sketch in b_i be n_i . The buckets in the summary satisfy that $n_i - n_{i+1} \leq \frac{\epsilon}{2}n_{i+1} + 1$ and $n_i - n_{i+2} > \frac{\epsilon}{2}n_{i+2} + 1$, for $1 \leq i < k - 1$, namely $\frac{\epsilon}{2}$ -partition.

We build in each bucket an $\frac{\epsilon}{2}$ -approximate GK-sketch and according to theorem 1, a sketch can be extracted from the summary for any length of window. We use (a, b) -summary to denote a summary built on a -partition and b -sketch. e.g. The summary we just built is an $(\frac{\epsilon}{2}, \frac{\epsilon}{2})$ -summary.

Lemma 2. *The number of buckets maintained in an ϵ -partitioned summary, if $n_1 = N$, is $\Theta(\frac{1}{\epsilon} \log \epsilon N)$.*

As $O(\frac{1}{\epsilon} \log \epsilon N)$ space is needed for a GK-sketch. We have,

Theorem 2. *The total space needed by an (ϵ, ϵ) -summary is $O(\frac{1}{\epsilon^2} \log^2 \epsilon N)$.*

Let $d_i = n_i - n_{i+1}$ and we group buckets with same d_i to form “ d -groups”. (see Fig. 3) Observe that dropping of buckets will always occur on the second bucket in an d -group, promoting the first one into the $2d$ -group. The d_i thus takes a form of $2^j, j = 0, 1, 2, \dots$. Use the space bound we derived, there will be $\log \epsilon N$ such d -groups and each contains no more than $\lfloor \frac{1}{\epsilon} \rfloor + 1$ buckets. We give algorithm 1 to efficiently maintain the summary structure.

Algorithm 1 Summary Maintenance

Upon a new data item v arrives:

step 1: Allocate a new bucket for this data item and append it to 1-group.

step 2: Iteratively checking the d -groups, from 1-group upwards. If the i -group is full (i.e. contains $\lfloor \frac{1}{\epsilon} \rfloor + 2$ buckets), drop the second bucket in this group and move the first one to $2i$ -group. Stop iteration when at i_0 -group, no bucket is dropped.

step 3: Add v to sketches maintained in all the buckets in the summary.

It is important to note here that although in the analysis of the algorithm, we use N, n to refer to the number of data items in the window, it does not

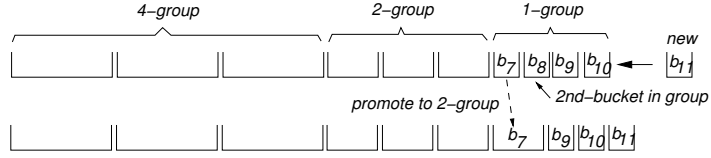


Figure 3. drop redundant buckets in summary

have to be unveiled to the algorithm. Therefore, the sliding window does not have to define on “item counting”. Additionally, for any window length, we can find a corresponding sketch in the summary. This enables answering queries for different window lengths using one single summary.

3.2 Quantile summary for two constrained sliding windows.

T-window quantile query: A T-window query with parameters ϕ , T and ϵ can be written in SQL-like syntax as:

```
SELECT  $\phi$ -quantile FROM stream
RANGE= data arrived in last  $T$  time
PRECISION =  $\epsilon_0$ 
```

Here, the parameter ϕ and T can vary from query to query while ϵ_0 is pre-fixed when the summary is built.

We build an $(\frac{\epsilon}{2}, \frac{\epsilon}{2})$ -summary and record the time when each bucket is built. To query a quantile, we choose the earliest constructed bucket with time-stamp inside the query window and estimate quantile using the sketch in this bucket.

N,f-window quantile query: An N,f-window quantile query with parameters ϕ , N , ϵ and filter function f can be written in SQL-like syntax as:

```
SELECT  $\phi$ -quantile FROM stream WHERE  $f(v) = true$ 
RANGE= most recent  $N$  data items.
PRECISION =  $\epsilon_0$ 
```

The underlined parameters vary from query to query while f and ϵ_0 are fixed when building the summary structure.

A single GK-sketch can handle filter based quantile query on whole stream by adding only the data items that pass the filter into the sketch. While according to [6], an approximate quantile summary for an N-item window can be maintained. However, neither can handle this N, f -window query.

To process the N, f -window quantile query, we build an $(\frac{\epsilon}{2}, \frac{\epsilon}{2})$ -summary. A data item v is added into the summary if $f(v) = true$. On each bucket b_i , we record the number n_i of data items observed (whatever added or not to the summary) after this bucket is built. To query a quantile, we scan the summary and choose the first bucket b_k that $n_k \leq N$, and return the quantile estimated by the sketch in it.

4 Processing general queries

In this section, we propose a uniform summarization technique for processing quantile queries on sliding windows with general constraints.

4.1 Sliding window and regression function

A sliding window has the following properties.

1. **Monotony.** An expired data item won't re-enter a window.
2. **FIFO.** Early arrived data items expires from a window before late items.

We first define *monotonic boolean function*, as following.

Definition 4 (Monotonic Boolean Function). A monotonic boolean function $F : V \mapsto \{true, false\}$ satisfies,

$$\exists t_0, F(v)|_{t_0} = false \longrightarrow \forall t > t_0, F(v)|_t = false.$$

Here, $t > t_0$ means a time t later than t_0 and V is the domain of the attribute(s) tested by F .

We define *regression function*(RF) to fully specify a sliding window.

Definition 5 (Regression Function (RF)). A function R is a regression function if it satisfies the following two conditions:

1. R is a monotonic boolean function.
2. For two data items v_1 and v_2 which v_1 arrives earlier than v_2 , if $R(v_1)$ and $R(v_2)$ are both true at time t_0 , then there is no time $t > t_0$ that $R(v_1) = true$ while $R(v_2) = false$.

We use the term *R-window*, or simply R to refer to a sliding window constrained by R . And the sliding windows studied in the last section is described using regression function like this:

1. $R_N(v) = true$ iff v is among the most recent N items.
2. $R_T(v) = true$ iff v is observed in recent T time.
3. $R_f(v) = f(v)$, $f(v) : V \mapsto \{false, true\}$

We specially address on R_f that, for a given function f , a data item is determined to be inside($f(v) = true$) or outside($f(v) = false$) of the window at the time it is observed and the status never changes afterwards. We define *filter* RF specially for this kind of regression function.

Definition 6 (Filter RF). Function R is a filter RF if

1. R is a regression function,
2. If at time t_0 , $\exists v, R(v) = true$, then $R(v) = true$ for all $t > t_0$.

It is easy to see that composition of two RFs under **AND** or **OR** is still a RF. However, only filter-RFs keep this property under **NEG** operation.

We are now ready to show how a query on a generally constrained window, possibly a boolean expression formed by regression functions, be processed in a uniform manner in our framework.

4.2 Processing quantile queries for a general R -window

A constraint R can be written in DNF.

$$\begin{aligned} R = & (a_{11} \wedge a_{12} \dots \wedge a_{1m_1} \wedge b_{11} \wedge b_{12} \dots \wedge b_{1n_1}) \\ & \vee (a_{21} \wedge a_{22} \dots \wedge a_{2m_2} \wedge b_{21} \wedge b_{22} \dots \wedge b_{2n_2}) \\ & \dots \\ & \vee (a_{j1} \wedge a_{j2} \dots \wedge a_{jm_j} \wedge b_{j1} \wedge b_{j2} \dots \wedge b_{jn_j}) \end{aligned}$$

Here, regression functions are divided into classes a and b , where class b is the class of filter-RFs and a for non-filter RFs. We use t_i , $1 \leq i \leq j$ to denote the j conjunction clauses. We build the summary in the following steps.

step 1: scan for pure class a clause If there exists one or more $n_i = 0$ ($1 \leq i \leq j$). We form a “pure class a sub-constraint” $A = \bigvee_{\{i:n_i=0\}} t_i$.

step 2: scan for pure class b clause If there exists one or more $m_i = 0$ ($1 \leq i \leq j$). We form a “pure class b sub-constraint” $B = \bigvee_{\{i:m_i=0\}} t_i$.

The remaining clauses form constraint $C = \bigvee_{\{i:m_i * n_i \neq 0\}} t_i$ so $R = A \vee B \vee C$.

step 3: build quantile summary Algorithm 2 demonstrates this procedure.

Algorithm 2 Build Summary

Input:

Constraints $R = A \vee B \vee C$.

Output:

Quantile Summary for R -window.

Description:

- 1: **if** B is not an empty constraint **then**
 - 2: Build a GK-sketch $S2$ on error parameter ϵ ;
 - 3: **end if**
 - 4: **if** C is not an empty constraint **then**
 - 5: Form constraint $B_s = (\bigvee_{\{i:t_i \text{ in } C\}} t_i^b) \wedge \neg B$; ($t_i^b = \bigvee_{k=1}^{n_i} b_{ik}$)
 - 6: **end if**
 - 7: Build an $(\frac{\epsilon}{2}, \frac{\epsilon}{2})$ -summary $S1$ with constrains B_s and $A \vee C$;
 - 8: **return** $S = (S1, S2)$;
-

When a new data item arrives in the data stream, the summary is updated using algorithm 3. It is easy to verify that the space usage of the whole summary is bounded by $O(\frac{1}{\epsilon^2} \log^2 \epsilon n_1 + \frac{1}{\epsilon} \log \epsilon n_2)$, where n_1, n_2 is the number of items inserted into $S1, S2$ respectively. To query the summary for ϕ -quantile on an

Algorithm 3 Summary Maintenance

Input:

Summary $S(S1, S2)$ built in algorithm 2 and a data item v .

Output:

Updated summary.

Description:

- if** $B(v) == true$ **then**
 - Add v into sketch $S2$.
 - else if** $B_s(v) == true$ **then**
 - Add v into summary $S1$. (use algorithm 1)
 - end if**
-

R -window, first extract sketch sk_R from the $(\frac{\epsilon}{2}, \frac{\epsilon}{2})$ -summary $S1$ for the query window. Then *merge* sk_R with $S2$. The merged sketch is an ϵ -sketch and we query it for ϕ -quantile.

We omit the detail of operation *merge* here due to space limitation. A general version (of merging multiple sketches) can be found in [6] in which it is proved that the merged sketch is ϵ -approximate. The whole diagram of maintenance and query of the summary is shown in figure 4.

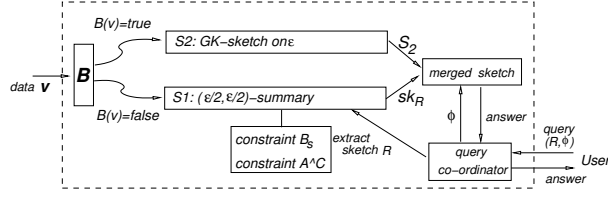


Figure 4. Working flow of the framework

5 Performance Study

We present performance evaluations as well as analysis on practical issues of our algorithm in this section. The algorithms are implemented in C++ and compiled using GNU gcc(v2.95). We run experiments on a P4 2.0GHz , 512MB memory PC with Debian Linux as its operating system.

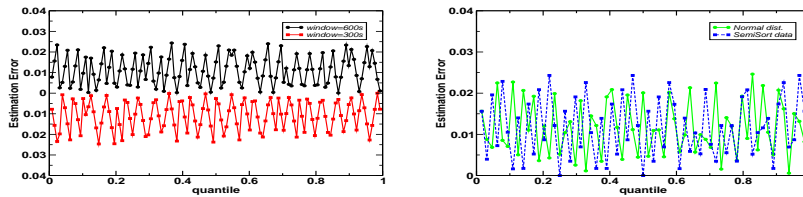
5.1 Performance evaluation on synthetic data sets

We test the algorithm using a set of T-window queries. The parameters that affect experiments are listed in table 1 with their default values used in this subsection.

Table 1. T-window query: system parameters.

Notation	Definition (Default Values)
ϵ	The guaranteed precision (0.05)
ϕ	$(1/q, \dots, (q-1)/q)$ for a given q ($q=64$)
D	Data distribution of the stream (uniform)
Sr	Stream Rate (approx. 400 tuples/s)
Mw	Max window length(on time) (600s)
Ss	Stream Size ($1.5 * Mw * Sr = 360K$)
Qw	Query window length(on time) (600s)

Figure 5(a) shows the estimation errors for quantiles from $1/128..127/128$ on $Qw = 600s$ and $Qw = 300s$. Figure.5(b) shows the experiment results using normal distributed data and semisort data. From the result we can see that under a guaranteed error setting of 0.05, the estimation errors are much lower than the guaranteed value. We also have similar observations on summaries with other ϵ , omitted here due to space limitation.

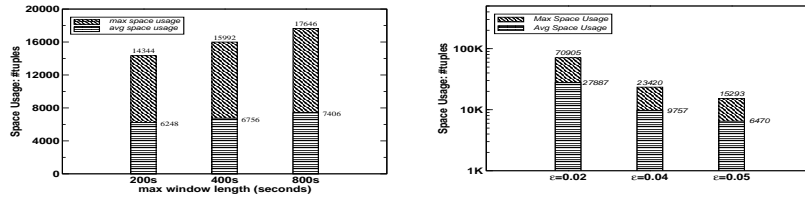


(a) on diff. windows ($q = 128$)

(b) on diff. data sets ($q = 64$)

Figure 5. estimating accuracy evaluation

We also measure the memory space used by the summary. Figure 6 shows the space consumption under different settings. We present two statistics in the result graphs. The "max space usage" is the peak value of memory consumption and "avg space usage" represents the average space consumption. Figure 6(a) shows the space consumption when Mw varies from 200s, 400s, to 800s. We can see that the space consumption increases very slowly on window enlargement, both for max and avg space usage. In Fig. 6(b), we compare the space usage when precision requirement changes. The experiment results show that the space usage increases apparently when ϵ gets smaller and the trend follows our analysis of the space usage.

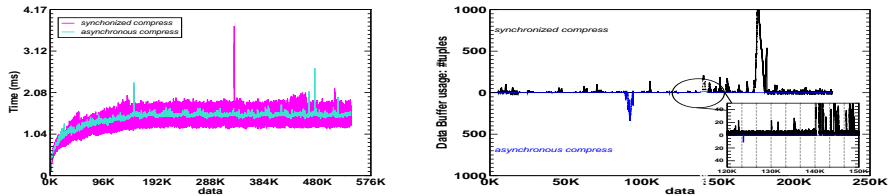


(a) on diff. max window size

(b) on diff. error guarantees

Figure 6. performance of space consumption: T-window

We conducted a set of experiments to measure the timing characteristics of the algorithm. Compressing of the GK-sketch dominates the running time of the algorithm. As we maintain multiple sketches in the summary, avoiding a synchronized compression of all sketches can have benefits that both the overall runtime performance is less fluctuating and the use of data buffer for incoming stream items can be reduced.



(a) timing characteristics

(b) buffer usage

Figure 7. runtime performance: synchronized v.s. asynchronous

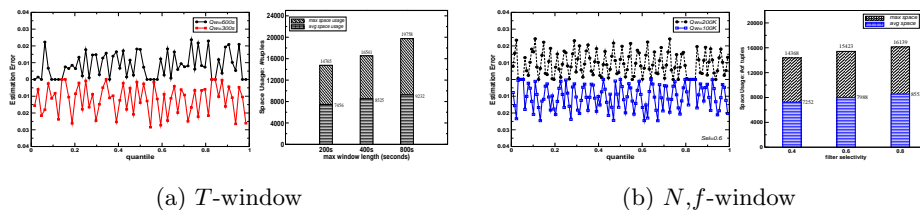
We tune our algorithm to compress sketches asynchronously. The result is shown in Fig. 7 where in sub-figure (a), the fluctuation of asynchronous compress (pale green) curve inside) is much less than that of synchronized compression. Time needed for an individual insertion is calculated by an average of a batch of 16 consecutive insertions to reduce measuring error. Figure 7(a) also suggests that the algorithm supports high speed data stream (> 500 tuples/s). Figure 7(b) shows the buffer usage of the algorithm, the stream rate is 500 tuples per second. The difference on buffer usage between synchronized and

asynchronous compressions is apparent from the experiment that asynchronous compression needs generally less buffer in most of the time.

5.2 Real data set evaluations

We also test the algorithm using a real data set. We use data from TDT(Topic Detection and Tracking) data repository. The data collected from news articles and stories is hashed to numerical values for the experiments.

Figure8 shows the accuracy and space consumption tests on default settings using the real data set for T-window queries(Fig.8(a)). We also test a set of N,f-window queries, in which the selectivity of filter function is set to 0.6 and window lengths are 100K and 200K respectively. The results are shown in Fig.8(b).



(a) T-window

(b) N,f-window

Figure 8. real data experiments

6 Conclusion

In this paper, we study the problem of processing approximate quantile queries for constrained sliding windows on a data stream. We present a framework that estimates ϵ -approximate quantiles using $O(\frac{1}{\epsilon^2} \log^2 \epsilon N)$ space. We extend the framework to process generally constrained sliding window queries, using $O(\frac{1}{\epsilon^2} \log^2 \epsilon n_1 + \frac{1}{\epsilon} \log \epsilon n_2)$ space ($n_1 + n_2 = N$).

References

1. B. Babcock, M. Datar, R. Motwani, and L. O’Callaghan. Sliding window computations over data streams. In *ACM PODS*, 2003.
2. M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *13th ACM-SIAM symposium on Discrete algorithms*, 2002.
3. A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *VLDB*, 2002.
4. M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *ACM SIGMOD*, 2001.
5. J.I.Munro and M.S.Paterson. Selection and sorting wiith limited storage. In *TCS12*, 1980.
6. 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 *ICDE*, 2004.
7. G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *ACM SIGMOD*, 1998.
8. G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *ACM SIGMOD*, 1999.