

Variance Estimation over Sliding Windows

Linfeng Zhang
Department of Electrical and Computer
Engineering
Iowa State University
Ames, Iowa 50011
zhanglf@iastate.edu

Yong Guan
Department of Electrical and Computer
Engineering
Iowa State University
Ames, Iowa 50011
guan@iastate.edu

ABSTRACT

Capturing characteristics of large data streams has received considerable attention. The constraints in space and time restrict the data stream processing to only one pass (or a small number of passes). Processing data streams over sliding windows make the problem more difficult and challenging. In this paper, we address the problem of maintaining ϵ -approximate variance of data streams over sliding windows. To our knowledge, the best existing algorithm requires $O(\frac{1}{\epsilon^2} \log N)$ space, though the lower bound for this problem is $\Omega(\frac{1}{\epsilon} \log N)$. We propose the first ϵ -approximation algorithm to this problem that is optimal in both space and worst case time. Our algorithm requires $O(\frac{1}{\epsilon} \log N)$ space. Furthermore, its running time is $O(1)$ in worst case.

Categories and Subject Descriptors

F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity; H.2.8 [Database management]: Database Applications—*Data mining*

General Terms

Algorithms

Keywords

Data Mining, Data Streams, Sliding Windows, Variance Estimation

1. INTRODUCTION

The problem of capturing characteristics of large data streams has received considerable attention [2, 9]. Many characteristics of large data streams, such as sum, mean, variance, diameter, frequency, quantile, top- k list (hot list), distribution, etc., have been widely studied. If we have sufficient large space and do not have time constraints, we can obtain any characteristics that we want precisely. However, the issue in processing large data streams is that in many

cases we probably do not have enough space to keep each element in the data streams. That is, we have to gather the interested characteristics with one pass and we have no chance to go through the data streams again.

An even greater challenge is to process data streams over sliding windows. An algorithm can work over sliding windows if it can not only gather the data streams characteristics, but also update the characteristics when new data is inserted and old data is expired. Unfortunately, many previous data stream algorithms cannot work over sliding windows. In this paper, we are interested in estimating element variance in data streams over sliding windows, while using as little space and operation as possible and making only one pass over the data streams.

1.1 Motivation

Variance estimation can be useful in many different research fields, e.g., network monitoring, intrusion detection, financial monitoring, weather forecast, disaster forecast, etc. Variance is an important statistical measure to evaluate the variability of random variables. Also, variance is often related to anomaly and status change, therefore variance is a powerful measure for anomaly detection and forecast. For instance, in weather forecast, variance is an important variable to accurately forecast precipitation, temperature, wind direction and speed, etc. In disaster forecast and reduction field, before disasters, such as earthquake, tornado and flood, happen, there usually exists some anomaly that can be observed and evaluated using variance.

To calculate the variance of a fixed series of elements is trivial. However, in many cases, different data have different weights (importance) according to the time. That is, there is time decay in the data streams. For instance, for a network intrusion detection system, the current network status is more important than that of one day ago, because we can use the current network status to detect ongoing anomaly and intrusions and accordingly take some measures to reduce the potential damage. Sliding window model is a general time decay model that only considers the most recent N elements evenly, and discards the influence of all older elements, where N is the window size. Sliding window model can provide recent data information by removing the stale data. Unfortunately, it is nontrivial to calculate the variance of data streams in sliding window model. For example, when a computer network is under attacks, it may show large variance in terms of connection numbers, traffic volume and delays, etc. However, it is difficult to maintain accurate variance of these variables over sliding windows when the network has heavy traffic.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODS'07, June 11–13, 2007, Beijing, China.

Copyright 2007 ACM 978-1-59593-685-1/07/0006 ...\$5.00.

Recently several algorithms are proposed for capturing different kinds of characteristics of large data streams over sliding windows [1, 3, 4, 5, 6, 7, 8, 10, 11]. For the problem of variance estimation over sliding windows, the best algorithm was proposed by Babcock et al. [3]. However, as mentioned by the authors of [3], there is still a gap between the optimal space bound and their upper bound. Also, the running time of their algorithm is not constant in worst case. Consequently, it is desired to find an efficient variance estimation algorithm over sliding windows which needs less space and runs in constant time.

1.2 Problem Definition

DEFINITION 1. Variance: Let $\{x_1, x_2, \dots, x_N\}$ be a series of N integers (which may be negative). The variance of these N numbers is defined by

$$V = \sum_{i=1}^N (x_i - \mu)^2,$$

where $\mu = \frac{1}{N} \sum_{i=1}^N x_i$ denotes the **mean** of these N integers.

To calculate the exact variance of fixed N numbers is trivial. We only need to keep two variables: one is the sum of all numbers, and the other is the sum of all numbers' squares. However, calculating exact variance over sliding windows is more difficult and challenging. For a stream of numbers, if we want to calculate the exact variance of the most recent N numbers, the memory requirement is $\Omega(N)$. In some cases, it is impossible to keep so much memory space for variance estimation purpose. Consequently, we have to find efficient and accurate approximation algorithms to estimate the variance over sliding windows.

DEFINITION 2. ϵ -Approximation: Let $\hat{V}$ denote the estimation of the real value V . $\hat{V}$ is an ϵ -approximation of V if

$$|V - \hat{V}| \leq \epsilon V,$$

where $0 < \epsilon < 1$ denotes the relative error.

Problem Statement

In this paper, we study the problem stated as follows: *Given an arbitrary window size N and an error bound ϵ , how to maintain ϵ -approximate variance of a stream of integers over sliding windows with size N in one pass?*

1.3 Our Contributions

In this paper, we address the problem of maintaining ϵ -approximate variance of data streams over sliding windows. To our knowledge, the best existing algorithm for this problem by Babcock et al. [3] requires $O(\frac{1}{\epsilon} \log N)$ space, and its running time is $O(1)$ amortized and $O(\frac{1}{\epsilon} \log N)$ in worst case. Datar et al. [4] presented a space lower bound of $\Omega(\frac{1}{\epsilon} \log N)$ for maintaining the sum and hence the variance of the last N elements when assuming that the maximum absolute value of the integer elements is at most polynomial in N . Therefore, there is a gap of $\frac{1}{\epsilon}$ between the optimal bound and the best existing upper bound.

In this study, we propose the first ϵ -approximation algorithm to maintain variance of data streams over sliding windows that is optimal in both space and worst case time.

Our algorithm requires $O(\frac{1}{\epsilon} \log N)$ space. Furthermore, its running time is $O(1)$ in worst case.

The rest of this paper is structured as follows: Section 2 discusses the related work. We introduce our algorithm and give the analysis about error bound, space requirement and running time in Section 3. We finally conclude in Section 4.

2. RELATED WORK

Datar et al. [4] proposed an algorithm to solve the *Basic-Counting* problem over sliding windows using *Exponential Histograms* (EH). Given the sliding window size N , and the relative error ϵ , the space taken by their algorithm is $O(\frac{1}{\epsilon} \log N)$, and the time taken to process each element is $O(1)$ amortized and $O(\log N)$ in worst case. Gibbons and Tirthapura [6] presented an algorithm based on a data structure called the *wave* that uses the same space as in [4]. However, their algorithm only takes $O(1)$ time to process each element in worst case. They also presented three extensions of the sliding window model for the distributed streams.

The EH algorithm [4] can estimate a class of aggregate functions over sliding windows. It applies to any function f satisfying the following properties for all multisets X, Y :

1. $f(X) \geq 0$.
2. $f(X) \leq \text{poly}(|X|)$.
3. $f(X \cup Y) \geq f(X) + f(Y)$.
4. $f(X \cup Y) \leq C_f(f(X) + f(Y))$, where constant $C_f \geq 1$.

However, variance does not satisfy the fourth property. For instance, suppose both X and Y have zero (or very small) variance, but different means. Then the variance of $X \cup Y$ may not be bounded by any constant C_f . Therefore, the EH algorithm cannot be utilized in variance estimation directly. Consequently, Babcock et al. [3] proposed a new algorithm for maintaining variance over sliding windows of a data stream. In their algorithm, elements in the data stream are partitioned into buckets, and the synopses of these buckets are maintained, which can be used to merge adjacent buckets to save space according to particular combination constraints. This algorithm requires $O(\frac{1}{\epsilon} \log N)$ space, and its running time is $O(1)$ amortized and $O(\frac{1}{\epsilon} \log N)$ in worst case. However, as mentioned by the authors of [3], there is a gap of $\frac{1}{\epsilon}$ between the optimal bound $\Omega(\frac{1}{\epsilon} \log N)$ and their upper bound.

Besides the bit counting and variance problems, several other problems of capturing characteristics of data streams over sliding windows are studied. Zhu and Shasha [11] introduced *basic windows* to compute the statistics over a sliding window. Golab et al. [7] utilized the basic windows mechanism in their deterministic algorithm to identify frequent items over sliding windows. Babcock et al. [3] proposed an algorithm for maintaining a k -median clustering over sliding windows of a data stream. Feigenbaum et al. [5] investigated the diameter problem in the streaming and sliding-window models, and the diameter is the maximum distance between a pair of elements within a data stream. Lin et al. [8] considered the problem of continuously maintaining ϵ -approximate quantiles over sliding windows in a data stream. Arasu and Manku [1] studied the problem of maintaining ϵ -approximate counts and quantiles over sliding windows in a data stream. Tirthapura et al. [10] considered the problem of maintaining sketches of recent elements of a asynchronous data stream in which the observed order of data is not the same as the time order in which the data was generated.

3. ALGORITHM

We first describe our algorithm, and then analyze its error bound, space requirement and running time.

3.1 Algorithm Description

Let N denote the window size and ϵ be the relative error parameter. An existing element is called *active* if it is one of the most recent N elements within the current window, or *expired* if it leaves the current window. For each element x_i , an index pos_i is used to record its position in the data stream, which is an indicator of “active” or “expired” by comparing with pos – the position index of the most recent element.

Similar to the algorithm in [3], elements in the data stream are partitioned into buckets. For each bucket B_i , we keep a timestamp t_i , which equals the position index of the oldest element in this bucket. Besides the timestamp, the following statistics information (n_i, μ_i, V_i) is also maintained:

- n_i : number of elements in the bucket;
- μ_i : mean of elements in the bucket;
- V_i : variance of elements in the bucket.

Figure 1 shows the description of our algorithm. To estimate the variance of the data stream in current sliding window, we maintain a triplet $(n_{all}, \mu_{all}, V_{all})$ which objects are the count, mean and variance of the combination of all active buckets in current window respectively. Note that n_{all} , μ_{all} and V_{all} are all initialized to zero at the beginning of the data stream.

Our algorithm has three steps when a new element arrives. When a new element x_t comes, the current index pos is updated. The new element constitutes a new bucket B_1 with $t_1 = pos$, $n_1 = 1$, $\mu_1 = x_t$, $V_1 = 0$, and an old bucket B_i becomes B_{i+1} . We update $(n_{all}, \mu_{all}, V_{all})$ by including the new bucket B_1 using Lemma 1.

Also, we check the oldest bucket B_m which has timestamp t_m . If it is expired, we first update $(n_{all}, \mu_{all}, V_{all})$ by excluding the expired bucket B_m using Lemma 1. Then we delete bucket B_m , which means our algorithm only maintains the buckets which elements are all active in the current window. We can use a wraparound counter with maximum $N - 1$ to represent the timestamps (i.e., position indices), since the oldest bucket will be deleted as soon as it expires. Therefore, the timing information of each bucket can be represented by $\lceil \log N \rceil$ bits.

We then check whether there are qualified pairs of adjacent buckets that can be merged to save space. Let $B_A = B_{i+1} \cup B_{i+2}$ and $B_B = \bigcup_{j=1}^i B_j$. We merge two adjacent buckets B_{i+1} and B_{i+2} if and only if they satisfy the following merging rules:

$$\text{RULE 1. } V_{A,B} - V_B \leq \frac{\epsilon}{5} V_B.$$

$$\text{RULE 2. } n_A \leq \frac{\epsilon}{10} n_B.$$

$$\text{RULE 3. } n_A + n_B \leq \frac{N}{2}.$$

We will explain why we set such merging rules when analyzing the error bound of our algorithm. When two adjacent buckets merge into a merged bucket, the merged bucket's timestamp is set to be the timestamp of the older one, and the merged bucket's statistics information can be calculated using Lemma 1. The merging step checks newer buckets first before checking older buckets.

Step 1: Insert New Element x_t .

Let $pos = pos + 1 \bmod N$.

Create a new bucket for x_t . The new bucket becomes B_1 with $t_1 = pos$, $n_1 = 1$, $\mu_1 = x_t$, $V_1 = 0$.

(Note that an old bucket B_i becomes B_{i+1} .)

Update $(n_{all}, \mu_{all}, V_{all})$ by including new bucket B_1 .

Step 2: Delete Expired Bucket.

Let B_m be the oldest bucket.

if there are at least 2 buckets and $t_m = pos$

Update $(n_{all}, \mu_{all}, V_{all})$ by excluding bucket B_m .

Delete bucket B_m .

endif

Step 3: Merge Buckets.

Let $i = 1$, and $B_B = B_1$.

while B_{i+2} exists

Let $B_A = B_{i+1} \cup B_{i+2}$.

if $n_A + n_B > \frac{N}{2}$

return

endif

if $n_A \leq \frac{\epsilon}{10} n_B$ and $V_{A,B} - V_B \leq \frac{\epsilon}{5} V_B$

Delete B_{i+2} , and let $B_{i+1} = B_A$.

(Note that bucket B_{j+1} becomes B_j for $j \geq i+2$.)

else

Let $i = i + 1$, and $B_B = B_B \cup B_i$.

endif

endwhile

Figure 1: Algorithm Description

3.2 Error Bound

Before determining the space requirement and running time of our algorithm, we first prove that it can maintain ϵ -approximate variance of data streams over sliding windows. Lemma 1 gives the equations to calculate the statistics information of the combination of two buckets.

LEMMA 1. When two buckets B_i and B_j are merged into a single bucket $B_{i,j}$, the count, mean and variance of $B_{i,j}$ can be computed from those of B_i and B_j as follows:

$$\begin{aligned} n_{i,j} &= n_i + n_j, \\ \mu_{i,j} &= \frac{n_i \mu_i + n_j \mu_j}{n_i + n_j}, \\ V_{i,j} &= V_i + V_j + \frac{n_i n_j}{n_i + n_j} (\mu_i - \mu_j)^2. \end{aligned} \quad (1)$$

PROOF. This lemma and its proof can be found in [3]. $\square$

Using Lemma 1, we can immediately get the following lemma which calculates the statistics information of the combination of three buckets.

LEMMA 2. When three buckets B_i , B_j and B_k are merged into a single bucket $B_{i,j,k}$, the count, mean and variance of $B_{i,j,k}$ can be computed as follows:

$$\begin{aligned} n_{i,j,k} &= n_i + n_j + n_k, \\ \mu_{i,j,k} &= \frac{n_i \mu_i + n_j \mu_j + n_k \mu_k}{n_i + n_j + n_k}, \\ V_{i,j,k} &= V_i + V_j + V_k + \frac{n_i(n_j + n_k)}{n_i + n_j + n_k} (\mu_i - \mu_{j,k})^2. \end{aligned}$$

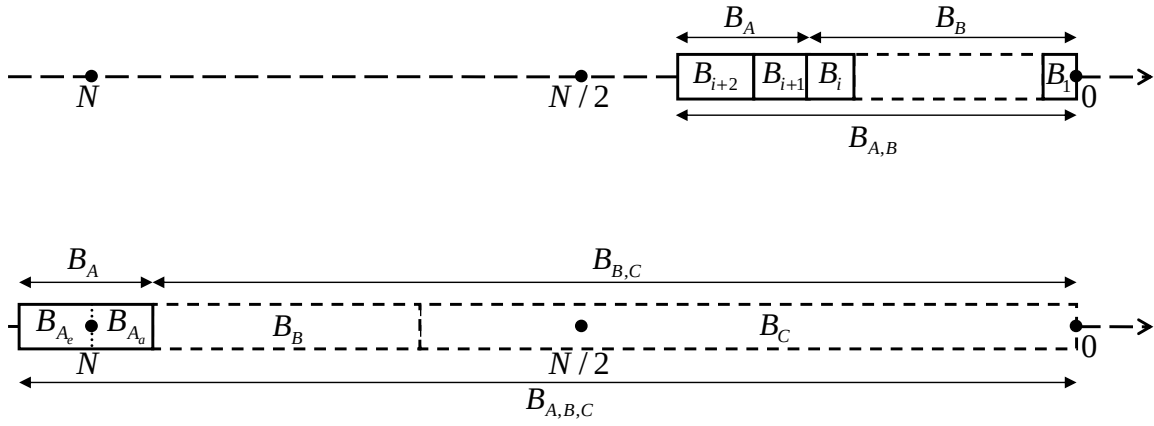


Figure 2: An Illustration of the Buckets

Now let us analyze how estimation error is generated. For a bucket B_i which has $n_i = 1$, there is no estimation error when it expires, because either the single element or none element of B_i is active when the current window slides over B_i . Consequently, only the buckets generated by merging which have more than one element can introduce estimation error when expiring.

For instance, as shown in Figure 2, consider any merged bucket B_A which has more than one element, when it expires, it is split by the sliding window into two virtual buckets: the expired bucket B_{A_e} which contains all expired elements of B_A , and the active bucket B_{A_a} which contains all active elements of B_A . In Figure 2, we use dotted boxes to denote the virtual buckets B_B and B_C . The virtual bucket B_B contains all newer elements when bucket B_A generates by merging, and the virtual bucket B_C contains all newer elements after B_B when bucket B_A expires. Note that B_C may become larger when more elements in B_A expires with the slide of the current window. Because we do not have the exact variance of bucket B_{A_a} , we have to return the estimated variance $\hat{V} = V_{B,C}$, which is the variance of all active buckets kept in memory space (i.e., V_{all}), to estimate the real variance.¹

Let δ denote the relative error of the variance estimation of our algorithm. If the real variance $V = V_{A_a,B,C} = 0$, then according to Lemma 1 or 2, $\hat{V} = V_{B,C} \leq V_{A_a,B,C} = 0$, and $\delta = 0 < \epsilon$. When $V = V_{A_a,B,C} \neq 0$, δ is defined by

$$\delta = \frac{V - \hat{V}}{V} = \frac{V_{A_a,B,C} - V_{B,C}}{V_{A_a,B,C}}.$$

We will prove that $0 \leq \delta < \epsilon$ when $V = V_{A_a,B,C} \neq 0$ by utilizing the two virtual buckets B_B and B_C . We first provide the relationship between n_B and n_C .

LEMMA 3. *For any merged bucket B_A , let virtual bucket B_B contains all newer elements when bucket B_A generates by merging, and let virtual bucket B_C contains all newer elements after B_B when bucket B_A expires. We have*

$$n_B < n_C.$$

¹Although we may not know exact V_B and V_C since one merged bucket may expand the boundary between virtual buckets B_B and B_C , we can still calculate $V_{B,C}$ which is the variance of the combination of all active buckets.

PROOF. By construction, $n_{A_a} + n_B + n_C = N$. Thus, $n_C = N - (n_{A_a} + n_B)$. On the other hand, $n_{A_a} < n_A$, since n_{A_a} represents the number of active elements of the bucket A, and the expired portion of A is assumed to be non-empty. Thus, $n_C > N - (n_A + n_B)$. Now, by applying Rule 3 of merging rules (i.e., $n_A + n_B \leq \frac{N}{2}$), it follows that $N - (n_A + n_B) \geq \frac{N}{2}$. This implies that $n_C > \frac{N}{2}$. Since $n_{A_a} + n_B + n_C = N$, it results that $n_B < \frac{N}{2}$ since $n_{A_a} > 0$. Thus, $n_B < n_C$. $\square$

To prove $0 \leq \delta < \epsilon$, we first relax the relative error δ to an auxiliary parameter θ , and then prove that $\theta < \epsilon$, where

$$\theta = \frac{V_A + \frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2}.$$

LEMMA 4. $0 \leq \delta \leq \theta$.

PROOF. Equation (1) in Lemma 1 shows that the variance of a bucket is larger than or equal to that of its sub-bucket. We know that $B_{A,B,C} = B_{A_e} \cup B_{A_a,B,C} = B_A \cup B_{B,C}$, and thus $B_{B,C} \subset B_{A_a,B,C} \subset B_{A,B,C}$. Consequently, we get

$$V_{A,B,C} \geq V_{A_a,B,C} \geq V_{B,C}.$$

Therefore,

$$\begin{aligned} 0 \leq \delta &= \frac{V_{A_a,B,C} - V_{B,C}}{V_{A_a,B,C}} \leq \frac{V_{A,B,C} - V_{B,C}}{V_{B,C}} \\ &= \frac{V_A + \frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B + V_C + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2} \\ &\leq \frac{V_A + \frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2} = \theta. \end{aligned}$$

$\square$

Consequently, if we can prove that $\theta \leq \epsilon$, then our algorithm is an ϵ -approximation algorithm. We consider all possibilities of μ_C , which is the mean of virtual bucket B_C , in the following three cases separately. We will show that in each of these three cases, θ is less than ϵ . Without loss of generality, we assume that $\mu_A \leq \mu_B$. When $\mu_A > \mu_B$, we can make similar analysis and get the same conclusion, since actually we only utilize the order information between μ_A and μ_B .

CASE 1. $\mu_B \leq \mu_C \leq 2\mu_B - \mu_A$

LEMMA 5. In Case 1, $\theta < \epsilon$.

PROOF. According to Rule 2, $n_A \leq \frac{\epsilon}{10}n_B < \frac{n_B}{10}$. Furthermore, Rule 1 shows that $V_{A,B} - V_B \leq \frac{\epsilon}{5}V_B$. We only consider that $V_{A,B} \neq 0$, thereby $V_B \neq 0$. Thus, we get

$$\begin{aligned} \epsilon &\geq 5 \frac{V_{A,B} - V_B}{V_B} = 5 \frac{V_A + \frac{n_A n_B}{n_A + n_B}(\mu_A - \mu_B)^2}{V_B} \\ &\geq 5 \frac{V_A + \frac{n_A n_B}{0.1 n_B + n_B}(\mu_A - \mu_B)^2}{V_B} \\ &= 5 \frac{V_A + \frac{n_A}{1.1}(\mu_A - \mu_B)^2}{V_B}. \end{aligned}$$

Therefore,

$$\epsilon > 4 \frac{V_A + n_A(\mu_A - \mu_B)^2}{V_B}. \quad (2)$$

Because $\mu_B \leq \mu_C$ in Case 1, and the mean of a merged bucket should be bounded between the means of its two sub-buckets, thus $\mu_B \leq \mu_{B,C} \leq \mu_C$. Therefore, in Case 1 that $\mu_B \leq \mu_C \leq 2\mu_B - \mu_A$, we get

$$\mu_B \leq \mu_{B,C} \leq 2\mu_B - \mu_A.$$

We have assumed that $\mu_A \leq \mu_B$, therefore

$$0 \leq \mu_B - \mu_A \leq \mu_{B,C} - \mu_A \leq 2(\mu_B - \mu_A).$$

We get $|\mu_A - \mu_{B,C}| \leq 2|\mu_A - \mu_B|$. Consequently,

$$\begin{aligned} \theta &= \frac{V_A + \frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2} \\ &\leq \frac{V_A + \frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B} \\ &\leq \frac{V_A + n_A \frac{n_B+n_C}{n_A+n_B+n_C} 4(\mu_A - \mu_B)^2}{V_B} \\ &\leq \frac{V_A + 4n_A(\mu_A - \mu_B)^2}{V_B} \\ &\leq 4 \frac{V_A + n_A(\mu_A - \mu_B)^2}{V_B}. \end{aligned}$$

Compared with inequality (2), we get $\theta < \epsilon$. $\square$

CASE 2. $\mu_C > 2\mu_B - \mu_A$

LEMMA 6. In Case 2, $\theta < \epsilon$.

PROOF. In this case that $\mu_C > 2\mu_B - \mu_A$, we get

$$\mu_B - \mu_A < \mu_C - \mu_B.$$

We have assumed that $\mu_A \leq \mu_B$, therefore $\mu_B < \mu_C$. Also, the mean of a merged bucket should be bounded between the means of its two sub-buckets, therefore $\mu_B < \mu_{B,C} < \mu_C$. Consequently,

$$\begin{aligned} |\mu_A - \mu_{B,C}| &< \mu_C - \mu_A = \mu_C - \mu_B + \mu_B - \mu_A \\ &< 2|\mu_B - \mu_C|. \end{aligned} \quad (3)$$

Therefore,

$$\begin{aligned} \theta &= \frac{V_A + \frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2} \\ &= \frac{V_A}{V_B + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2} + \frac{\frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2} \\ &< \frac{V_A}{V_B} + \frac{\frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{\frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2} \\ &= \frac{V_A}{V_B} + \frac{n_A}{n_B} \cdot \frac{(n_B + n_C)^2}{(n_A + n_B + n_C)n_C} \cdot \left(\frac{|\mu_A - \mu_{B,C}|}{|\mu_B - \mu_C|}\right)^2. \end{aligned}$$

Equation (1) in Lemma 1 shows that the variance of a bucket is larger than or equal to the sum of variance of both sub-buckets. Therefore $V_A \leq V_{A,B} - V_B$. According to Rule 1, we get

$$\frac{V_A}{V_B} \leq \frac{V_{A,B} - V_B}{V_B} \leq \frac{\epsilon}{5}.$$

According to Rule 2, we get

$$\frac{n_A}{n_B} \leq \frac{\epsilon}{10}.$$

From inequality (3), we get

$$\left(\frac{|\mu_A - \mu_{B,C}|}{|\mu_B - \mu_C|}\right)^2 < 4.$$

Let $g(n_C) = \frac{(n_B+n_C)^2}{(n_A+n_B+n_C)n_C}$. Consequently,

$$\theta < \frac{\epsilon}{5} + \frac{\epsilon}{10} \cdot g(n_C) \cdot 4 = \frac{\epsilon}{5}(1 + 2g(n_C)).$$

The derivative of $g(n_C)$ is

$$\begin{aligned} g'(n_C) &= \frac{(n_B + n_C)[(n_A - n_B)n_C - (n_A + n_B)n_B]}{(n_A + n_B + n_C)^2 n_C^2} \\ &< -\frac{(n_A + n_B)(n_B + n_C)n_B}{(n_A + n_B + n_C)^2 n_C^2} < 0. \end{aligned}$$

Therefore, $g(n_C)$ decreases with the increase of n_C . According to Lemma 3 that $n_B < n_C$,

$$g(n_C) < g(n_B) = \frac{4n_B}{n_A + 2n_B} < \frac{4n_B}{2n_B} = 2.$$

Therefore,

$$\theta < \frac{\epsilon}{5}(1 + 2g(n_C)) < \epsilon.$$

$\square$

CASE 3. $\mu_C < \mu_B$

LEMMA 7. In Case 3, $\theta < \epsilon$.

PROOF. We claim that for any instance B_C in Case 3, we can find an equal or even worse instance $B_{C'}$ in terms of θ which mean belongs to either Case 1 or Case 2, such that $\theta < \theta'$ where

$$\theta' = \frac{V_A + \frac{n_A(n_B+n_{C'})}{n_A+n_B+n_{C'}}(\mu_A - \mu_{B,C'})^2}{V_B + \frac{n_B n_{C'}}{n_B+n_{C'}}(\mu_B - \mu_{C'})^2}.$$

If $\mu_C < \mu_B$, we construct $B_{C'}$ from B_C by adding $2(\mu_B - \mu_C)$ to each element in B_C . For bucket $B_{C'}$,

$$\begin{aligned} n_{C'} &= n_C, \\ \mu_{C'} &= 2\mu_B - \mu_C > \mu_B. \end{aligned}$$

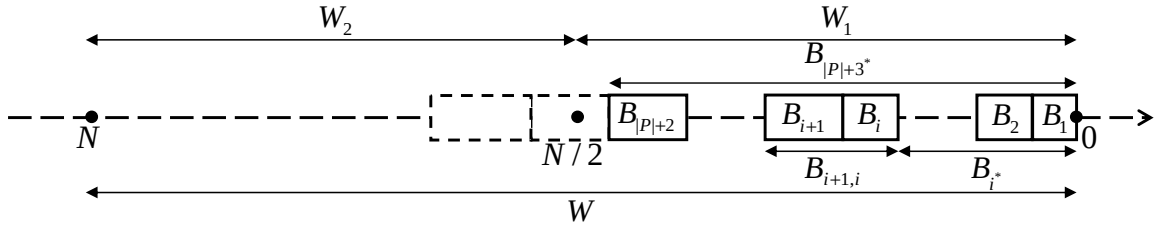


Figure 3: An Illustration of the Buckets and Windows

Consequently, the mean $\mu_{C'}$ of bucket $B_{C'}$ belongs to either Case 1 or Case 2, and thus $\theta' < \epsilon$ according to Lemma 5 and 6. Since μ_C and $\mu_{C'}$ are symmetric about μ_B , $\mu_{B,C}$ and $\mu_{B,C'}$ should also be symmetric about μ_B . Therefore $\mu_{B,C} + \mu_{B,C'} = 2\mu_B$, and

$$\mu_{B,C'} - \mu_B = \mu_B - \mu_{B,C} > 0.$$

We have assumed that $\mu_A \leq \mu_B$, therefore

$$\begin{aligned} |\mu_A - \mu_{B,C}| &= |(\mu_B - \mu_{B,C}) - (\mu_B - \mu_A)| \\ &= |(\mu_{B,C'} - \mu_B) - (\mu_B - \mu_A)| \\ &\leq |(\mu_{B,C'} - \mu_B) + (\mu_B - \mu_A)| \\ &= |\mu_A - \mu_{B,C'}|, \end{aligned}$$

and

$$\begin{aligned} \theta &= \frac{V_A + \frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2} \\ &= \frac{V_A + \frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_{C'})^2} \\ &\leq \frac{V_A + \frac{n_A(n_B+n_{C'})}{n_A+n_B+n_{C'}}(\mu_A - \mu_{B,C'})^2}{V_B + \frac{n_B n_{C'}}{n_B+n_{C'}}(\mu_B - \mu_{C'})^2} \\ &= \theta'. \end{aligned}$$

Consequently, we get $\theta \leq \theta' < \epsilon$ in Case 3. $\square$

We have shown that, when the real variance $V = 0$, our algorithm returns the estimation $\hat{V} = 0$. When the real variance $V \neq 0$, according to Lemma 4, 5, 6 and 7, we get $0 \leq \delta \leq \theta < \epsilon$ where δ denotes the relative error of the variance estimation of our algorithm. Consequently, we get the following theorem about the error bound of our algorithm.

THEOREM 1. *Our algorithm can maintain ϵ -approximate variance of data streams over sliding windows.*

Furthermore, we make the observation that our algorithm only makes one-sided error. That is,

$$(1 - \epsilon)V \leq \hat{V} \leq V.$$

To help understand our algorithm, we shortly explain why we set such three merging rules and why these three merging rules can bound the estimation error to $O(\epsilon)$. As shown in Lemma 4,

$$0 \leq \delta \leq \theta = \frac{V_A + \frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2}.$$

First of all, Rule 1 implies that $O(\frac{V_A}{V_B}) = O(\epsilon)$. Furthermore, when μ_C is close to μ_B (i.e., Case 1), Rule 1 and 2 can

guarantee that

$$O\left(\frac{\frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B}\right) \approx O(\epsilon).$$

However, when μ_C is far way from μ_A and μ_B (i.e., Case 2), in worst case

$$\begin{aligned} O\left(\frac{\frac{n_A(n_B+n_C)}{n_A+n_B+n_C}(\mu_A - \mu_{B,C})^2}{V_B + \frac{n_B n_C}{n_B+n_C}(\mu_B - \mu_C)^2}\right) &\approx O\left(\frac{n_A}{n_B} \cdot \frac{(n_B + n_C)^2}{(n_A + n_B + n_C)n_C}\right) \\ &\approx O\left(\epsilon \cdot \frac{N}{n_C}\right). \end{aligned}$$

Therefore we have to add a constraint on n_C (i.e., Rule 3), such that $O(n_C) = O(N)$ to bound relative error to $O(\epsilon)$.

3.3 Space Requirement

Now consider any two adjacent buckets B_i and B_{i+1} when $i \geq 2$ after the merging procedure completes. Let bucket $B_{i+1,i} = B_{i+1} \cup B_i$ and bucket $B_i^* = \bigcup_{j=1}^{i-1} B_j$, and Figure 3 shows an illustration. When $n_{i+1,i} + n_{i^*} = n_{i+2^*} \leq \frac{N}{2}$, according to the merging rules, our algorithm has the following invariant:

INVARIANT 1. *For $i \geq 2$, if B_{i+1} exists, $B_{i+1,i}$ has either*

Property 1: $n_{i+1,i} = n_{i+2^*} - n_{i^*} > \frac{\epsilon}{10} n_{i^*}$, or

Property 2: $V_{i+1,i,i^*} - V_{i^*} = V_{i+2^*} - V_{i^*} > \frac{\epsilon}{5} V_{i^*}$.

Let W denote the current N -sized window, W_1 denote the current $\frac{N}{2}$ -sized window, and W_2 denote the past $\frac{N}{2}$ -sized window. We define three sets, P , P_1 and P_2 . P is all adjacent bucket pairs in W_1 except the first pair (i.e. $B_1 \cup B_2$). P_1 is a subset of P which contains all adjacent buckets with Property 1, and P_2 is a subset of P which contains all adjacent bucket pairs with Property 2.

$$P = \{B_{i+1,i} : n_{i+2^*} \leq \frac{N}{2}, i \geq 2\},$$

$$P_1 = \{B_{i+1,i} : n_{i+2^*} > (1 + \frac{\epsilon}{10})n_{i^*}, B_{i+1,i} \in P\},$$

$$P_2 = \{B_{i+1,i} : V_{i+2^*} > (1 + \frac{\epsilon}{5})V_{i^*}, B_{i+1,i} \in P\}.$$

Then from Invariant 1, we get

$$\begin{aligned} P &= P_1 \cup P_2, \\ |P| &\leq |P_1| + |P_2|. \end{aligned}$$

Let m denote the number of buckets within the current N -sized window W . Let m_1 denote the number of buckets within the current $\frac{N}{2}$ -sized window W_1 . Let m_2 denote

the number of buckets within the past $\frac{N}{2}$ -sized window W_2 . Then

$$m_1 = |P| + 2.$$

Consider any window W_2 , all buckets in it are from a particular old W_1 after potential merging. According to Rule 3, any bucket which passes the boundary between W_1 and W_2 will not change any more in W_2 before it expires. Consequently, at any time,

$$m_2 \leq \max(m_1).$$

In addition, there is at most one bucket which expands the boundary between W_1 and W_2 . Consequently,

$$m \leq m_1 + m_2 + 1 \leq 2\max(m_1) + 1 = 2\max(|P|) + 5. \quad (4)$$

Therefore, if we can find the upper bound of $|P|$, we get the bound of m (i.e., the number of buckets in our algorithm) accordingly.

LEMMA 8. *In the case that every $B_{i+1,i}$ in P has Property 1 (i.e., $P = P_1$), we have*

$$|P_1| < 2\lceil \frac{10}{\epsilon} \rceil \log N - 1.$$

PROOF. In this case, since any adjacent bucket pair $B_{i+1,i}$ has Property 1 when $i \geq 2$, then for any index $i + 2j \leq |P_1| + 3$ with $i \geq 2$,

$$\begin{aligned} n_{i+2^*} &> (1 + \frac{1}{10}\epsilon)n_{i^*}, \\ n_{i+4^*} &> (1 + \frac{1}{10}\epsilon)n_{i+2^*} > (1 + \frac{2}{10}\epsilon)n_{i^*}, \\ n_{i+6^*} &> (1 + \frac{1}{10}\epsilon)n_{i+4^*} > (1 + \frac{3}{10}\epsilon)n_{i^*}, \\ &\dots \\ n_{i+2j^*} &> (1 + \frac{j}{10}\epsilon)n_{i^*}. \end{aligned}$$

Thus when $j = \lceil \frac{10}{\epsilon} \rceil$, $n_{i+2j^*} > 2n_{i^*}$. Consequently, n_{i^*} will be doubled when i increases every $2\lceil \frac{10}{\epsilon} \rceil$. We know that $n_{2^*} = n_1 = 1$, therefore

$$\begin{aligned} 2^{\lfloor \frac{|P_1|+1}{2\lceil \frac{10}{\epsilon} \rceil} \rfloor} \cdot n_{2^*} &\leq n_{|P_1|+3^*} \leq \frac{N}{2}, \\ \frac{|P_1|+1}{2\lceil \frac{10}{\epsilon} \rceil} &< N, \\ |P_1| &< 2\lceil \frac{10}{\epsilon} \rceil \log N - 1. \end{aligned}$$

□

Let R be the upper bound on the absolute value of the data elements.

LEMMA 9. *In the case that every $B_{i+1,i}$ in P has Property 2 (i.e., $P = P_2$), we have*

$$|P_2| < 2\lceil \frac{5}{\epsilon} \rceil \log(\frac{3NR^2}{2}) + 1.$$

PROOF. The proof is similar to that of Lemma 8. In this case, the maximum element number bound $\frac{N}{2}$ in window W_1 becomes the maximum variance bound $\frac{NR^2}{2}$. V_{i^*} will be doubled when i increases every $2\lceil \frac{5}{\epsilon} \rceil$.

We claim that $V_{4^*} \geq \frac{2}{3}$. First, $V_{4^*} > 0$, otherwise $V_{4^*} = V_{2^*} = 0$ which violates Property 2. Because $V_{4^*} > 0$, and all elements in the data stream are integers, then V_{4^*} achieves

its minimum value $\frac{2}{3}$ if and only if the three elements in B_{4^*} are $\{c_1, c_1, c_1 \pm 1\}$, where c_1 is an integer. Consequently,

$$\begin{aligned} 2^{\lfloor \frac{|P_2|+1}{2\lceil \frac{5}{\epsilon} \rceil} \rfloor} \cdot V_{4^*} &\leq V_{|P_2|+3^*} \leq \frac{NR^2}{2}, \\ \frac{|P_2|+1}{2\lceil \frac{5}{\epsilon} \rceil} &< \frac{3NR^2}{2}, \\ |P_2| &< 2\lceil \frac{5}{\epsilon} \rceil \log(\frac{3NR^2}{2}) + 1. \end{aligned}$$

□

LEMMA 10. *In any case, we have*

$$|P| < 2\lceil \frac{10}{\epsilon} \rceil \log N + 2\lceil \frac{5}{\epsilon} \rceil \log(\frac{3NR^2}{2}).$$

PROOF. We check the size of subsets P_1 and P_2 separately. First, consider how many pairs of adjacent buckets that have Property 1 (belong to P_1) can exist to contain at most $\frac{N}{2}$ elements. Any pair of adjacent buckets which does not belong to P_1 also contributes to contain some number of elements. Further, n_{i^*} is still doubled after every $2\lceil \frac{10}{\epsilon} \rceil$ pairs of adjacent buckets in P_1 . Consequently, Lemma 8 is still true for $|P_1|$. Similarly, any pair of adjacent buckets which does not belong to P_2 also contributes to the increase of variance, and Lemma 9 is still true for $|P_2|$. Therefore,

$$|P| \leq |P_1| + |P_2| < 2\lceil \frac{10}{\epsilon} \rceil \log N + 2\lceil \frac{5}{\epsilon} \rceil \log(\frac{3NR^2}{2}).$$

□

Consequently, we get the space bound of our algorithm as shown in the following theorem.

THEOREM 2. *Our algorithm can maintain ϵ -approximate variance of data streams over sliding windows using $O(\frac{1}{\epsilon} \log N)$ space.*

PROOF. According to inequality (4) and Lemma 10, we get

$$\begin{aligned} m &\leq 2\max(|P|) + 5 \\ &< 4\lceil \frac{10}{\epsilon} \rceil \lceil \log N \rceil + 4\lceil \frac{5}{\epsilon} \rceil \lceil \log(\frac{3NR^2}{2}) \rceil + 5. \end{aligned}$$

Therefore, our algorithm is $O(\frac{1}{\epsilon} \log(NR^2))$ in space. Assume that R is at most polynomial in N , the space requirement is in fact $O(\frac{1}{\epsilon} \log N)$. □

3.4 Running Time

Obviously, the running time of our algorithm is $O(\frac{1}{\epsilon} \log N)$ in worst case. Although Step 1 and 2 are both $O(1)$, Step 3 needs $O(\frac{1}{\epsilon} \log N)$ in worst case. Now we propose a mechanism which makes our algorithm an $O(1)$ running time algorithm in worst case. The baseline idea is that we only check constant number of pairs of adjacent buckets each time in the third (merging) step.

Let c be an integer which is large than 1. When each element arrives, we still insert it as a new bucket (Step 1) and delete expired bucket if any (Step 2). However, instead of checking all pairs of adjacent buckets within W_1 in Step 3, we only run the kernel **while** loop c times. After that, the merging procedure is hanged up and will be resumed with the same status when a new element arrives. Note that before the merging procedure completes, each new element

is stored in an individual bucket, and window W_1 does not slide (although window W slides and the expired buckets are still deleted). Since many new buckets may be queued before the merging procedure completes, we need extra space to keep them. However, such extra space can still be efficiently bounded.

LEMMA 11. *Let Y denote the maximum number of buckets that window W_1 can have after merging procedure completes which is in $O(\frac{1}{\epsilon} \log N)$ as shown in Theorem 2. Let Z denote the maximum number of new arriving elements (buckets) during the merging procedure. Then*

$$Z \leq \lceil \frac{Y}{c-1} \rceil.$$

PROOF. Let y_i denote the number of buckets that window W_1 can have after the i th merging procedure completes. Let z_i denote the number of new arriving elements during the i th merging procedure. Then for the $i+1$ th merging procedure, in the beginning, window W_1 can contain at most $y_i + z_i$ buckets and at most $y_i + z_i - 2$ pairs of adjacent buckets which need to check after the first bucket B_1 . Because we check c pairs of adjacent buckets when receiving a new element, we have

$$z_{i+1} \leq \lceil \frac{y_i + z_i - 2}{c} \rceil.$$

We claim that for any index $i \geq 1$, $z_i \leq \lceil \frac{Y}{c-1} \rceil$. We use induction to prove this claim. First, $z_1 = 1$. Suppose for an index $i > 1$, the claim is true. Then for index $i+1$,

$$\begin{aligned} z_{i+1} &\leq \lceil \frac{y_i + z_i - 2}{c} \rceil \leq \lceil \frac{Y + \lceil \frac{Y}{c-1} \rceil - 2}{c} \rceil \\ &\leq \lceil \frac{Y + \frac{Y}{c-1}}{c} \rceil \leq \lceil \frac{Y}{c-1} \rceil. \end{aligned}$$

Therefore, we get $Z \leq \lceil \frac{Y}{c-1} \rceil$. $\square$

Lemma 11 shows that we can process a new element using constant time while the extra space and thus the total space are still in $O(\frac{1}{\epsilon} \log N)$. Further, it is clear that our algorithm can update $(n_{all}, \mu_{all}, V_{all})$ in $O(1)$, therefore our algorithm can answer a query of the variance in the current sliding window with constant time. Consequently, we get the final theorem which concludes the optimal properties of our algorithm:

THEOREM 3. *Our algorithm can maintain ϵ -approximate variance of data streams over sliding windows using $O(\frac{1}{\epsilon} \log N)$ space. Furthermore, the running time of processing a new element and answering a query is $O(1)$ in worst case. Our algorithm is optimal in both space and worst case time.*

4. CONCLUSIONS

In this paper, we address the problem of maintaining ϵ -approximate variance in data streams over sliding windows. Our proposed algorithm only needs $O(\frac{1}{\epsilon} \log N)$ space and $O(1)$ running time in worst case. Therefore, it is optimal in both space and worst case time. In the future, we will continue to extend our algorithm to other data stream statistics problems over sliding windows.

5. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their invaluable feedback. This work was partially sponsored by NSF under grants No. CNS-0644238, CNS-0626822, and DUE-0313837, ARDA under contract number NBCHC030107, and Carver Trust Foundation.

6. REFERENCES

- [1] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2004)*, Paris, France, June 2004.
- [2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2002)*, pages 1–16, Madison, USA, June 2002.
- [3] B. Babcock, M. Datar, R. Motwani, and L. O’Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2003)*, pages 234–243, San Diego, USA, June 2003.
- [4] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pages 635–644, San Francisco, USA, Jan. 2002.
- [5] J. Feigenbaum, S. Kannan, and J. Zhang. Computing diameter in the streaming and sliding-window models. *Algorithmica*, 41(1):25–41, 2004.
- [6] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA 2002)*, Winnipeg, Manitoba, Canada, Aug. 2002.
- [7] L. Golab, D. DeHaan, E. D. Demaine, A. López-Ortiz, and J. I. Munro. Identifying frequent items in sliding windows over on-line packet streams. In *Proceedings of the Internet Measurement Conference (IMC 2003)*, Miami, USA, Oct. 2003.
- [8] 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 *Proceedings of the 20th International Conference on Data Engineering (ICDE 2004)*, Boston, USA, Mar. 2004.
- [9] S. Muthukrishnan. Data streams: Algorithms and applications. Technical report, Rutgers University, Piscataway, USA, 2003.
- [10] S. Tirthapura, B. Xu, and C. Busch. Sketching asynchronous streams over sliding windows. In *Proceedings of the 25th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2006)*, Denver, USA, July 2006.
- [11] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB 2002)*, Hong Kong, China, Aug. 2002.