Approximately Processing Multi-granularity Aggregate Queries over Data Streams

Shouke Qin Weining Qian Aoying Zhou*
Department of Computer Science and Engineering, Fudan University, Shanghai 200433, China
{skqin,wnqian,ayzhou}@fudan.edu.cn

Abstract

Aggregate monitoring over data streams is attracting more and more attention in research community due to its broad potential applications. Existing methods suffer two problems, 1) The aggregate functions which could be monitored are restricted to be first-order statistic or monotonic with respect to the window size. 2) Only a limited number of granularity and time scales could be monitored over a stream, thus some interesting patterns might be neglected, and users might be misled by the incomplete changing profile about current data streams. These two impede the development of online mining techniques over data streams, and some kind of breakthrough is urged. In this paper, we employed the powerful tool of fractal analysis to enable the monitoring of both monotonic and non-monotonic aggregates on time-changing data streams. The monotony property of aggregate monitoring is revealed and monotonic search space is built to decrease the time overhead for accessing the synopsis from $O(m)$ to $O(\log m)$, where m is the number of windows to be monitored. With the help of a novel inverted histogram, the statistical summary is compressed to be fit in limited main memory, so that high aggregates on windows of any length can be detected accurately and efficiently on-line. Theoretical analysis show the space and time complexity bound of this method are relatively low, while experimental results prove the applicability and efficiency of the proposed algorithm in different application settings.

1. Introduction

Query and mining over data streams have been attracting much attention in database community recently because of the motivation from real-life applications. As well-known, the conventional database technologies were invented for the management of collections of static data, but querying and mining dynamic data streams are still challenges. The constraints imposed on data stream processing include limited memory consumption, sequentially linear scanning, and on-line accurate result reporting over evolving data streams. One of the most interesting topics is querying and monitoring aggregates with a user specified threshold, including frequent items [12] or itemsets [19, 16] mining, burst detection [21, 20], change detection [5], and aggregate monitoring over time windows [7].

In this paper, we focus on a specific form of query requests, say, given a set of given windows $w_i, i = 1, 2, ..., m$, report current timestamp and the all windows w_p if an aggregate over w_p, denoted as $F(w_p)$, is larger than the predefined threshold $T(w_p)$. It has wide applications in monitoring network traffic data [8], Gamma ray data [21], and Web log mining [7].

However, existing techniques suffer from two major problems when being applied to real-life applications. First, they are designed for processing aggregations with distributive property (e.g., count) but not for algebraic aggregates such as average or variance. The significant difference between algebraic and distributive aggregations lies in that the later is monotonic with respect to the time dimension, while the former is not. However, algebraic aggregate functions are used widely in analysis of streaming data. For example, the following four models are frequently used to remove the effect of noisy data: Moving Average, S-shaped Moving Average, Exponentially Weighted Moving Average, Non-Seasonal Holt-Winters [14]. Though some methods are proposed to maintain variance and K-medians over a stream [3], the cost for finding bursts using these synopsis is still $O(m)$. Thus, the method cannot support multi-granularity aggregate queries efficiently, especially when the number of windows is large.

Second, most existing techniques are designed for querying over a single window or a small number of windows. Since users often do not know the most appropriate window size they need to monitor, they tend to monitor a large number of windows simultaneously. Although processing

*This work is partially finished when the author is visiting Computer Science Division of University of California at Berkeley.

multiple queries separately is a trivial solution, and aggregate window of continuous queries can be shared [2], it inevitably results in redundant storage and computation, and finally bad performance.

To address these two problems, and considering the basic requirements of data stream processing, in this paper, we propose a novel approximate method for continuously monitoring a set of aggregate queries with different window size parameters over a data stream.

1.1 Related Work

Monitoring and mining data stream has received considerable attention recently [13, 21, 8, 14, 5, 7, 20]. Kleinberg focuses on modelling and extracting structure from text stream in [13]. Our work is different in that we focus on distributive and algebraic aggregates on different sliding windows. There are also some works about finding changes in data stream [8, 14, 5]. Although their objectives and application backgrounds are different, one common feature they all bear is that they can monitor the bursty behavior on only one granularity. The method in [21] uses a Shifted Wavelet Tree (SWT) to detect bursts on multiple windows. In 2005, a more efficient aggregate monitoring method is put forward in [7], which uses a novel index schema to detect bursty behaviors. However, these two methods can only detect monotonic aggregates, and the maximum window size and the number of monitored windows are both limited. Since they are both based on the assumption that for two windows with different size, if w_j is contained in w_j, there must be $F(w_i) > F(w_j)$. Actually, it holds only for F which is monotonic with respect to the window size, such as sum or $count$. For non-monotonic F such as $average$, the assumption does not hold. An adaptive burst detecting method based on Inverted Histogram is proposed in [20]; however, no any overall bound for the error of burst detection is given there.

Fractal geometry was known to be suitable to describe the essence in the complex shapes and forms of nature [15]. As a tool, fractal techniques have been successfully applied to the modelling and compression of data sets [18, 11, 17, 4]. However, existing fractal techniques are difficult to be employed directly in the data stream scenario, for the building of a fractal model is of polynomial time complexity, and multiple scans of the whole data set is necessary.

Our contributions can be summarized as follows:

- We incorporate the tool of fractal analysis into aggregate query processing. Then our method provides the ability for processing not only the distributive aggregates but also the algebraic ones. We also propose a method for processing data which does not obey the scaling relationship of exact fractal models.
- The monotonic property of the synopsis search space is described. To construct such a monotonic search space for multi-granularity aggregate query processing, a novel approach is presented, which could decreases the time overhead of query processing from $O(m)$ to $O(\log m)$, where m is the number of windows being monitored.
- An efficient synopsis, called Inverted Histogram (IH), is employed, and the algorithm for query processing is given. We analyze the storage and computation cost, and prove the error bound of our algorithm. We also complement our analytical results with an extensive experimental study. The results demonstrate that the proposed method can monitor multi-granularity aggregate queries accurately with high efficiency.

The remainder of this paper is organized as follows. Section 2 is for the preliminary knowledge of fractal and scaling relationship. Then the problem statement is presented in this section. Section 3 introduces the concept of monotonic search space and presents the method for building such a search space. The details of aggregates monitoring algorithm, including the maintenance of IH and the extension of basic algorithm, is introduced in Section 4. Section 5 shows the results and then analyzes the experiments. Finally, Section 6 is for concluding remarks.

2 Preliminaries and Problem Statement

Fractal analysis is a powerful tool for analyzing time series data and data streams. Power-law scaling relationship, a basic characteristic of fractals, could be adopted to handle data streams.

Pieces of a fractal object when enlarged are similar to larger pieces or to that of the whole [15]. If these pieces are identical re-scaled replicas of the others, the fractal is exact. When the similarity is presented only in a statistical sense, and it is statistically observed on the given feature with different granularity, the fractal is statistical. Determined fractals such as the von Koch curve and Cantor set are both exact; however, most natural objects are statistical fractals.

When a quantitative property, q, is measured on a time scale s, its value depends on s according to the following scaling relationship [15]:

$$q = ps^d \tag{1}$$

This is called power law scaling relationship of q with s. p is a factor of proportionality and d is the scaling exponent. The value of d can be easily determined as the slope of the linear least squares fitting to the data pairs on the plot of $\log q$ versus $\log s$:

$$\log q = \log p + d \log s \tag{2}$$

Data points for exact fractals are lined up along the regression slope, whereas those of statistical fractals scatter around it since the two sides of equation (2) are equal only in distribution.

For discrete time series data, equation (1) can be transformed to $q(st) = s^d p(t)$ where t is the basic time unit and $s(s \in \mathbb{R})$ is scaling measurement [6]. Simply, d can be determined by $d = \frac{\log(q(st)/p(t))}{\log s}$. The power-law scaling relationship is also obeyed when q is an aggregate function or some second order statistic values. The details will not be discussed here due to the space limitation.

A data stream X can be considered as a sequence of points $x_1, ..., x_n$ with subscript in ascending order. Each element in the stream can be a value in the range $[0..R]$. F can be not only monotonic aggregates with respect to the window size (e.g., *sum* and *count*) but also non-monotonic ones such as *average*. Thereafter, it can be seen that F can be extended to some even more complicated statistic values, such as *variance*.

To monitor aggregations on evolving data streams is in fact to answer aggregate queries continuously on subsequences (windows) with different lengthes. The formal definition of a monitoring task over a data stream can be described as follows.

Definition 1 *Assuming that F is an aggregate function, w_{l_i} is the latest sliding window of a stream with the length of l_i, $1 \leq l_i \leq n$, and $T(w_{l_i})$ is the threshold on $F(w_{l_i})$ given by users beforehand. The result is reported on window w_{l_i} when x_n comes and if $F(w_{l_i}) \geq T(w_{l_i})$.*

Here, the definition is similar to the burst defined in [21, 7]. Compared with previous works, the major differences of ours lie in: 1) F can be algebraic aggregates, 2) a large number of windows can be monitored simultaneously.

To reach such a goal, a brute-force approach is to check the synopsis for each window whenever a new point arrives. However, with this approach the cost of answering a query is $O(m)$, where m is the number of windows. Though some previous effort tried to reduce the cost for maintaining the synopsis, it is argued that reporting query result from synopsis also need substantial cost. The cost is extremely expensive, especially when monitoring large number of windows.

We propose an alternative method which organize all the windows into a monotonic search space. Here, a *search space* is a permutation of all the windows. Our basic idea is to sort the windows, so that when a result is reported on a window, all its predecessors should be reported as results. We call such search space as a *monotonic search space*. Thereby, a binary search can be applied on the monotonic search space, and the time complexity for accessing synopsis is $O(\log m)$.

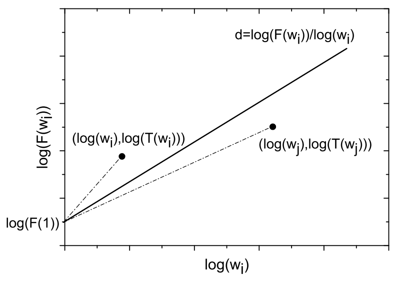


Figure 1. Sketch map for power law scaling of aggregate function F.

3 Construction of a Monotonic Search Space

3.1 Monotonic Search Spaces over Exact Fractals

Here, it is assumed that the monitored data stream X is exact fractal. The extensions of the basic method for handling statistical fractals and non-fractal data will be given next.

The aggregate result on unit time scale is denoted by $F(1)$ $(F(1) > 0)$. A line with slope $d = \log(F(w_i)/F(1))/\log(w_i)$ can be obtained on stream X. The line can be drawn on a sketch map as Fig.1 shows. Here, d is the scaling exponent of F. Different F has different d.

Assuming that d is known in advance. $T(w_i)$ and $T(w_j)$ are two user-given thresholds on window w_i and w_j respectively. We plot the two points $(\log(w_i), \log(T(w_i)))$ and $(\log(w_j), \log(T(w_j)))$ on the sketch map. Provided that the point $(\log(w_i), \log(T(w_i)))$ is over the line d and the point $(\log(w_j), \log(T(w_j)))$ is below the line d, that is to say, the following slope inequalities hold:

$$\frac{\log(T(w_i)/F(1))}{\log(w_i)} > d = \frac{\log(F(w_i)/F(1))}{\log(w_i)}$$

$$\frac{\log(T(w_j)/F(1))}{\log(w_j)} < d = \frac{\log(F(w_j)/F(1))}{\log(w_j)}$$

then, when we have a detection for changes on stream X, there must be a result reported on window w_j and no result on window w_i. The reason behind is that $\log(T(w_i)) > \log(F(w_i))$ is always holds since $\log(T(w_j)/F(1)) < \log(F(w_j)/F(1))$ is guaranteed on the overall data stream.

For extending to a more complex condition, assuming that d is not known *a priori*, and m thresholds $T(w_1), T(w_2), ..., T(w_m)$ on m different windows are given by users beforehand. To monitor the aggregates on those windows, we first sort these levels with their ratios $log(T(w_i)/F(1))/log(w_i)$ in ascending order. This implies that $F(w_i)/T(w_i)$ $i = 1, 2, ..., m$ are ordered in ascending order. Hence, the property of the search space follows:

Property 1 *$T(w_1), T(w_2), ..., T(w_m)$ are m thresholds given for m different windows $w_1, w_2, ..., w_m$, and $F(1)$ is the aggregate result of the unit time scale. Provided that $\log(T(w_i)/F(1))/\log(w_i)$ is ordered in ascending order, a window w_i is a result to be reported implies that all its predecessors are results to be reported.*

With this property, we can sort the search space and decrease the time overhead in great deal when answering aggregate monitoring queries. It endows the stream monitor with high scalability for the tough monitoring tasks on fluctuating data streams. However, this property is strictly abided by exact fractals. For statistical fractals or non-fractal data, the scaling exponent d in equation (1) is not a constant on overall data stream. As a result, the order of thresholds is variable from point to point in a data stream. This brings great challenges for using this monotony property in real life applications.

3.2 Approximating Real-life Data with Fractals

Real-life data may not be exact fractals. For statistic fractals in real world applications, this property is abided in distribution. That is to say, the rule is only obeyed by the statistic variables on the data stream, such as expectation and variance.

In this case, we sort the thresholds with Property 1; it means that we use exact fractals to approximate statistic fractals. This approximation will induce two kinds of errors. One is the error of fractal approximation, denoted by err_{FA}; the other is the error of missorting the thresholds, denoted by err_{MS}.

The fractal approximation error err_{FA} is induced by transforming real-life data stream to exact fractal which follows the power law scaling relationship. Let the aggregate result on ith-level window w_i be $F(w_i)$ and its expectation and standard deviation be $E(F(w_i))$ and $D(F(w_i))$. We assume that $\frac{F(w_i)-E(F(w_i))}{D(F(w_i))} \sim Norm(0,1)$. The threshold given by users on $F(w_i)$ is denoted by $T(w_i)$. Because the fractal transformation uses $E(F(w_i))$ to approximate $F(w_i)$ for scaling in power law relationship, this will not induce any error when $F(w_i) < T(w_i)$. However, when $F(w_i) \geq T(w_i)$, the change of aggregate on window w_i may be neglected. err_{FA} is the probability of such kind of error is occurring on w_i.

$$err_{FA} = P(F(w_i) \geq T(w_i))$$

$$= P(\frac{F(w_i) - E(F(w_i))}{D(F(w_i))} \geq \frac{T(w_i) - E(F(w_i))}{D(F(w_i))})$$

Therefore, the err_{FA} is limited when $T(w_i)$ is given.

Having different purpose in comparison with other approximation techniques, such as histogram, wavelet and FFT(Fast Fourier Transformation), the fractal approximation is not used to compress data and reconstruct them later. So, the err_{FA} will not be affected by the distance between fractal data and original data but the deviation of a given threshold to the expected value of the aggregate result on corresponding window. The distance error to original data will be considered in our synopsis data structure—IH, which will be discussed later.

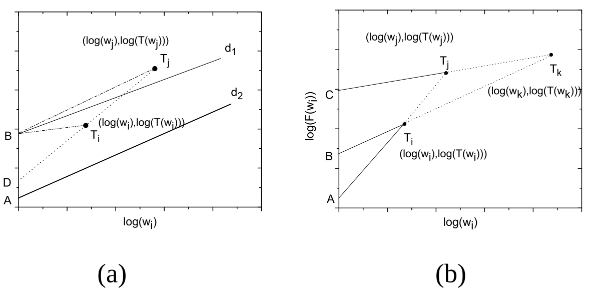


Figure 2. (a) Power law scaling relationship on real life data stream. (b) Piecewise monotonic search space on real life data stream.

The missorting error err_{MS} originates from using invariant $F(1)$ to sort the thresholds. For exact fractals, the value of $F(1)$ is invariant. However, for statistic fractals the $F(1)$ is time-changing. For example, if the monitored aggregate function F is sum, the value of $F(1)$ is equal to $\frac{1}{n}\sum_{i=1}^{n} x_i$ which is kept varying, induced by the newly coming x_n of stream X. Thus, constructing a monotonic search space with an invariant $F(1)$ as in exact fractals does induce errors consequently.

Let us consider a simplified case shown in Fig.2.(a), where two thresholds T_i and T_j are given by users for two different levels of windows. It can be seen that the monotonic search space achieved on d_1 will not hold on d_2. The reason is that the two scaling relationships have different $\log(F(1))$. So, the order of monitored windows based on $log(T(w_i)/F(1))/log(w_i)$ is variable on different scaling relationships. It will be explained in details as follows.

The value of point B on the vertical axis is $\log(F(1))$ of d_1. Because the slope of the line BT_i is smaller than that of BT_j, the monotonic search space is $\langle w_i, w_j \rangle$. It means that when the aggregate monitoring query is submitted, we first detect the time scale w_i. If the result of aggregate monitoring query is found on w_i, the time scale w_j will be detected in the second step. Otherwise, the query processing will halt and no result reported. In this way, $\langle w_i, w_j \rangle$ can support the queries on d_1. However, when a new point x_{n+1} comes, the current power-law scaling relationship d_2 is obtained. The value of point A on the vertical axis is $\log(F(1))$ of d_2. Since the slope of line AT_i is larger than that of AT_j, the monotonic search space of d_2 is $\langle w_j, w_i \rangle$. Apparently, it conflicts with the monotonic search space of d_1.

From the analysis above, a rule ensuring the invariant monotonic search space could be observed and summarized as follows. Assuming that the line T_jT_i intersects the ver-

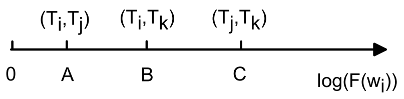


Figure 3. Building monotonic search space.

tical axis at point D. If the point $\log(F(1))$ of any d_i is above point D on the vertical axis, the d_i has a same monotonic search space $\langle w_i, w_j \rangle$ as that of d_1. Similarly, if the point $\log(F(1))$ of any d_i is below point D on the vertical axis, the d_i has an inverted monotonic search space $\langle w_j, w_i \rangle$, compared with that of d_1.

Property 2 *Given two thresholds $T(w_i)$ and $T(w_j)$. Suppose the level of w_i is smaller than that of w_j, and the line through points $(\log(w_i), \log(T(w_i)))$ and $(\log(w_j), \log(T(w_j)))$ intersects the vertical axis at D. Then, if $\log(F(1))$ is above D, the monotonic search space is $\langle w_i, w_j \rangle$; otherwise, if $\log(F(1))$ is below D, $\langle w_j, w_i \rangle$ is a monotonic search space.*

The simple rule observed hereinabove is only adaptable to double thresholds. Here, it is extended to a more complex case with three thresholds. For the cases with more than three thresholds, the extension is a easy and trivial job. As shown in Fig.2.(b), the three thresholds are plotted as T_i, T_j and T_k. Pairwise connecting the three points, lines T_iT_j, T_iT_k and T_jT_k can be obtained. They intersect the vertical axis at points A, B and C, respectively. The vertical axis can be upwards divided into four disjointed intervals $(-\infty, A]$, $(A, B]$, $(B, C]$ and $(C, +\infty)$. For the $\log(F(1))$ falling in $(A, B]$, $\langle w_i, w_j \rangle$, $\langle w_k, w_i \rangle$ and $\langle w_k, w_j \rangle$ can be obtained based on Property 2. These binary monotonic relationships can be merged into $\langle w_k, w_i, w_j \rangle$ which is the monotonic search space of the interval $(A, B]$.

Proposition 1 *Given m thresholds $\{T(w_i)\}_{i=1}^{m}$ and corresponding points $\{(\log(w_i), \log(T(w_i)))\}_{i=1}^{m}$. Draw the lines connecting any pair of points, then the vertical axis is divided into several intervals. Each interval has a unique monotonic search space.*

No matter what the value of $\log(F(1))$ on scaling relationship d_i is, it can be located at one of the intervals. Then, the monotonic search space of the corresponding interval can be used to support aggregate monitoring queries. A novel method to build the intervals and their monotonic search spaces will be presented in next subsection.

3.3 Building the Search Space

Considering the case of three thresholds in Fig.2.(b) as an example. The vertical axis is plotted as Fig.3. Suppose the line T_iT_j connecting the pair (T_i, T_j) intersects with the axis $\log(F(w_i))$ at point A, where $w_i < w_j$. The pair (T_i, T_j) is called the divide condition of divide point A. For the other two points B and C, it could be deduced similarly.

Algorithm 1 buildMonotonicSearchSpace($T(w_i), w_i$)

```
 1: create a new point $T_i$;
 2: $T_i \leftarrow (\log(w_i), \log(T(w_i)))$;
 3: if $m \geq 1$ then
 4:     for $j$ = 1 to $m$ do
 5:         line $T_i$ and $T_j$ intersects vertical axis at $D_i$;
 6:         if $w_i < w_j$ then
 7:             add $\langle w_j, w_i \rangle$ to the leftwards intervals of $D_i$;
 8:             add $\langle w_i, w_j \rangle$ to the rightwards intervals of $D_i$;
 9:         else
10:             add $\langle w_i, w_j \rangle$ to the leftwards intervals of $D_i$;
11:             add $\langle w_j, w_i \rangle$ to the rightwards intervals of $D_i$;
12:         end if
13:     end for
14: end if
15: increase $m$ by 1;
```

Binary monotony relationships is generated first and then merged finally. If a $\log(F(1))$ falls in the interval $(A, B]$, and the $<$ is given to the divide conditions of all the leftward divide points of interval $(A, B]$, the binary monotony relationship $\langle w_i, w_j \rangle$ can be obtained. When given the $>$ to the divide conditions of all the rightward divide points of interval $(A, B]$, the binary monotony relationship $\langle w_k, w_i \rangle$ and $\langle w_k, w_j \rangle$ can be obtained. Then, all the three binary monotony relationships could be merged, and $\langle w_k, w_i, w_j \rangle$ is formed, which is the monotonic search space for the interval $(A, B]$.

The procedure described here can be performed off-line, for the order of thresholds are always determined before online monitoring aggregates. It can also be executed online for inserting some new thresholds instantly given by users. Therefore, Algorithm 1 is in fact an incremental algorithm.

Algorithm 1 has two input parameters $T(w_i)$, w_i and no output. Given a new threshold $T(w_i)$, it obtains a new point T_i in Line 2. Then, at Line 5, it draws a line through points T_i and T_j which intersects the vertical axis at point D_i. In the process of updating the monotonic search space of each interval, from Line 4 to 13, the algorithm adds the newly generated binary monotonic relationship into each entry.

The time cost of Algorithm 1 for adding a new window into existing search spaces is $O(m)$, where m is the number of monitored windows at present. The space cost of Algorithm 1 is very small. This is because the entries are stored in hard disk and need not to be maintained in memory. Furthermore, the distribution of $\log(F(1))$ can be obtained, we need only to store the entries covering the possible value of $\log(F(1))$.

4 Aggregate Monitoring

In this section, an efficient monitoring algorithm is proposed to monitor both monotonic and non-monotonic aggregates over time-changing data streams. This algorithm

Algorithm 2 monitorAggregatesFalsePositively

```
1: locate $x_n$ at interval $I_k$;
2: $A \leftarrow$ retrieve monotonic search space of $I_k$;
3: $low \leftarrow 0, high \leftarrow A.size() - 1$;
4: while $low \leq high$ do
5:    $mid \leftarrow (low + high)/2$;
6:    if $T(A[mid]) \leq$ComputeAGG($A[mid]$) then
7:       $low \leftarrow mid + 1$;
8:    else
9:       if $T(A[mid]) >$ComputeAGG($A[mid]$) then
10:          $high \leftarrow mid - 1$;
11:      end if
12:   end if
13: end while
14: return $high$;
```

can monitor the changes of aggregate results on multiple granularity with sub-linear search space. To our knowledge, the existing research works have to monitor those windows with linear search space, resulting in the poor scalability for the stream monitoring. Furthermore, the algorithm could be extended to monitor algebraic aggregates such as variance. The proposed algorithm can be used as a framework to offer efficient solutions for other kinds of monitoring tasks as well.

When a new point x_n is arriving and being inserted into the histogram, the Algorithm 2 is invoked for answering aggregate monitoring query over current stream. The algorithm has no input parameter. The output is the alarm information if any. Here, we turn to the windows which is experiencing changes. The algorithm perform binary search on the monotonic search space from Line 4 to Line 13 for answering aggregate monitoring queries. ComputeAGG($A[mid]$) is the process for retrieving $F(w_i)$ ($w_i = A[mid]$) from IH false positively or false negatively.

Algorithm 2 can explore the search space of the overall stream in $O(\log m)$ time, where m is the number of monitored windows. If the concern is only about whether or not some querying result occurs when x_n comes, rather than on which window the change occurs or how many changes are induced on the overall stream, the time cost could be $O(1)$. We can claim the following theorem when assuming the space cost of a B-bucket histogram is B without loss any generality.

Theorem 1 *Algorithm 2 can monitor aggregates in real time on m granularity over a data stream in $O(n \log m)$ time and $O(B)$ space.*

This algorithm can be implemented upon any synopsis structure provided that it is for computing aggregate results. IH is one of these synopses. It is actually a novel histogram we proposed for detecting aggregation bursts. Some details will be presented in next subsection. The aggregate result $F(w_i)$ on the latest i-length window $(1 \leq i \leq n)$ can be obtained from IH. In practice, $F(w_i) = f_{ss}(i)$, and $f_{ss}(i) = \Sigma_{j=n-i+1}^{n} x_j$, that is to say, it is the sum of the last i values of stream X, called suffix sum. Due to the limited space available for computing aggregates over a data stream, two alternative approaches, namely, false positive oriented and false negative oriented, could be employed to approximate aggregate computing. The former answers aggregate query on w_i with a value not less than the real $F(w_i)$, whereas the latter answers aggregate query on w_i with a value not exceeding the real $F(w_i)$. In real world, it depends on application which one of the false positive oriented and false negative oriented algorithm is preferred.

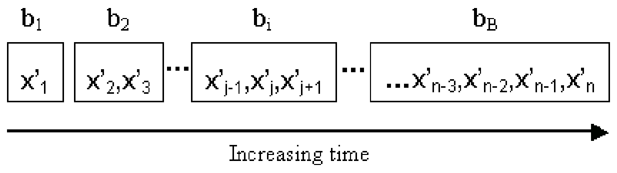
It can be seen that Algorithm 2 is a framework for monitoring both distributive and algebraic aggregates. The kind of aggregate which can be monitored by Algorithm 2 is determined by the kind of aggregate which can be computed by IH. Thus, in the following, IH is introduced first, and then the aggregate monitoring tasks are analyzed.

4.1 Aggregate Estimation Using IH

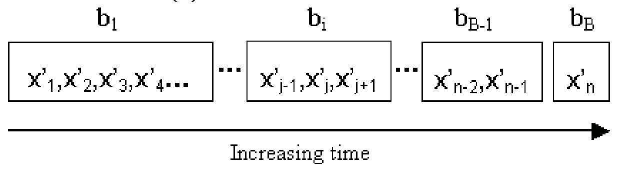
Now, we are at the point to introduce a novel synopsis, called Inverted Histogram (abbrev. IH). It is used to approximately calculate the aggregation value when given a window. Though existing approximate V-optimal histogram [10] can also do this job, it suffers a lot of workload to monitor aggregate. The reason is that the absolute error and size of the bucket constructed afterwards are getting larger and larger as the stream proceeds. Thus, the errors induced by using the newly created buckets are also increasing. However, for our purpose, what we expect is that the recent bucket is of high precision, in other words, the width of the recent buckets should be smaller than that of old ones.

Our basic idea is to invert the order of buckets, and then the smaller bucket is used to store the newly coming points. Fig.4.(b) illustrates the idea [20]. The oldest point x'_1 and the latest point x'_n of stream x' are in b_1 and b_B, respectively. In Fig.4.(a) which is also duplicated from [20], $x'_i = f_{ps}(i)$, $f_{ps}(i)$ is the prefix sum, i.e., $f_{ps}(i) = \Sigma_{j=1}^{i} x_j$. In Fig.4.(b), $x'_i = f_{ss}(i)$, and $f_{ss}(i)$ is the suffix sum, i.e., $f_{ss}(i) = \Sigma_{j=n-i+1}^{n} x_j$. Fig.4.(a) is a bucket series of approximate V-optimal histograms; therefore, the size and the absolute error of the last bucket is getting larger and larger with passage of time.

As in the approximate V-optimal histogram, the goals for IH are also the minimal number of buckets and the guaranteed relative error in each bucket. The only difference is that the stream could be regarded as $x'_i = f_{ss}(i)$. The stream x' can be partitioned into B intervals(buckets), $(b_1^a, b_1^b), ..., (b_B^a, b_B^b)$, where $b_i(1 \leq i \leq B)$ is the i-th bucket; b_i^a and b_i^b are the minimum and maximum value within bucket b_i, respectively. In some cases, it's possible that $b_i^a = b_i^b$. Here, we analyze the bound of the relative error δ in each bucket. The maximum relative error in b_i is



(a) Normal Buckets Order



(b) Inverted Buckets Order

Figure 4. Buckets Orders of Approximate V-optimal Histogram and IH

Algorithm 3 updateIH(x_n)

```
1:  increase bucket number $B$ by 1;
2:  $j \leftarrow B$;
3:  create a new bucket and put $x_n$ into it;
4:  for $i = B - 1$ to 1 do
5:      add $x_n$ to both $b_i^a$ and $b_i^b$;
6:      if $b_i^b \leq (1+\delta)b_j^a$ then
7:          $b_j^b \leftarrow b_i^b$;
8:          add width of $b_i$ to width of $b_j$;
9:          delete $b_i$;
10:         decrease bucket number $B$ by 1;
11:     end if
12:     decrease $j$ by 1;
13: end for
```

$\delta = \frac{b_i^b - b_i^a}{b_i^a}$. In our approach, each bucket should maintain a pair b_i^a, b_i^b and the number of values in it, i.e., its width. It is known that the buckets are disjoint and contain $x'_1..x'_n$. Therefore, $b_1^a = x'_1$ and $b_B^b = x'_n$, and $b_i^b \leq (1+\delta)b_i^a$ is always tenable in the process of formation. The construction of IH is described in Algorithm 3.

The only input parameter of Algorithm 3 is x_i, and it has no output. When a new value x_n comes, all the $O(\frac{\log n + \log R}{\log(1+\delta)})$ buckets are updated, as illustrated by statements from Line 4 to 13. First, Line 3 creates a new bucket for x_n and puts the bucket last. Then, Line 5 updates the maximum and minimum values of all the buckets (from the newly created one to the oldest one) by adding x_n into them. In the process of updating, from Line 6 to 11, the algorithm merges two consecutive buckets b_i and b_j when $b_i^b \leq (1+\delta)b_j^a$, with $b_j^b = b_i^b$. Then, the maximum relative error in each bucket of IH can be bounded and the space and time cost are kept low. So, we have the following theorem.

Theorem 2 *Algorithm 3 can construct an IH with $O(\frac{\log n + \log R}{\log(1+\delta)})$ space in $O(\frac{n(\log n + \log R)}{\log(1+\delta)})$ time, and the relative error in each bucket is at most δ.*

As mentioned previously, ComputeAGG(w_i) is invoked to retrieve $F(w_i)$ from IH with false positive guaranteed or false negative guaranteed when needed. The relative error of its return value is bounded by δ. The precision of the approximate aggregate computation can be improved by considering the values within a bucket to be equidistant. To guarantee false positive or false negative detection, $maxD$ and $minD$ need to be maintained within each bucket. $maxD$ is the maximum distance between two consecutive points within the bucket; $minD$ is the minimum distance between two consecutive points within the bucket. Provided that the real value of w_i is within bucket b_j, the false positive aggregate value returned by ComputeAGG(w_i) is $min(b_j^b, b_j^a + maxD(i - \Sigma_{k=1}^{i-1} Wid(b_k) - 1))$, and the false negative aggregate value returned by ComputeAGG(w_i) is $b_j^a + minD(i - \Sigma_{k=1}^{i-1} Wid(b_k) - 1)$.

Up to now, it has been shown that IH can be used to monitor distributive aggregates. Here, the monitoring task with algebraic aggregate, say, $variance$, will be discussed. The $variance$ in a w-length window is $\frac{1}{w}\sum_{i=1}^{w}(x_i - \mu)^2$, where $\mu = \frac{1}{w}\sum_{i=1}^{w} x_i$. It can be transformed to $\frac{1}{w}\sum_{i=1}^{w} x_i^2 - \mu^2$. As we know, $\sum_{i=1}^{w} x_i$ can be obtained from IH. With a new variable y_i and $y_i = x_i^2$, $\sum_{i=1}^{w} x_i^2$ can also be computed by using IH with guaranteed error. Consequently, for monitoring $variance$ on w_i, we need two histograms. One is for computing $\sum_{i=1}^{w} x_i$, the other is for computing $\sum_{i=1}^{w} x_i^2$. Both of them are bounded by maximum relative error δ.

4.2 Error Bound Analysis

This section provides a theoretical analysis for the final results of aggregate monitoring, and presents some analysis on the resulting error for using IH.

Suppose w_1 is a monitored window with a threshold $T(w_1)$. The process of answering aggregate monitoring query on window w_1 is to detect whether $F(w_1)$ is larger than or equal to $T(w_i)$ or not. The probability of false alarm with our method can be evaluated as $P(Err) = 1 - (1 - err_{MS})(1 - err_{FA})(1 - err_{IH})$, where err_{MS}, err_{FA} and err_{IH} are the error probabilities of missorting, fractal approximation and estimated value of IH, respectively. They are independent one another. err_{MS} and err_{FA} have been discussed in section 4. Since the accurate monotonic search space can always be maintained, we could have $err_{MS} = 0$. Thus, $P(Err) = err_{FA} + err_{IH} - err_{FA}err_{IH}$, where err_{FA} is bounded by the deviation of a given threshold to the expected value of aggregate result $\overline{F(w_i)}$ on window w_i. That is to say, $err_{FA} = P(x \geq \lambda)$, where $x \sim Norm(0,1)$ and $\lambda = \frac{T(w_i) - E(F(w_i))}{D(F(w_i))}$.

With the false positive or false negative oriented approaches, the error bound err_{IH} for aggregate monitoring query answering could be analyzed as follows.

Assuming that $x_1 = F(w_1)$ and the estimated value of $F(w_1)$ with IH is x'_1. With the maximum relative error bounded by δ, $F(w_1)$ can be estimated by using IH to be in $[(1-\delta)F(w_1),(1+\delta)F(w_1)]$. For false positive aggregate monitoring, the error probability is

$$err_{IH} = p \cdot \frac{x_1(1+\delta) - T(w_1)}{x_1\delta} \tag{3}$$

The error probability of false negative oriented approach is

$$err_{IH} = (1-p) \cdot \frac{T(w_1) - x_1(1-\delta)}{x_1\delta} \tag{4}$$

Notice that the parameter p measures the occurrence of burst in a data stream. Smaller p implies that more bursts occur in the data stream. Accordingly, the false positive method is suitable for monitoring fluctuant data stream, while the false negative method is suitable for relatively dormant data streams.

Suppose $R(T(w_i))$ is a set of real results of aggregate monitoring query, which could be obtained accurately with threshold $T(w_i)$ on window w_i over the stream x, and we do not consider the error err_{FA} induced by fractal approximation, for err_{FA} and err_{IH} are independent each other. Similarly, suppose $FP(T(w_i))$ is a set of results of aggregate monitoring query, which are computed approximately by using our false positive oriented method with threshold $T(w_i)$ on the window of the same length. The false positive method can provide a $(1+\epsilon)$-approximation ($\epsilon \in (0,1)$) aggregate monitoring query processing. It means that given a threshold $T(w_i)$, the number of query results detected by the false positive method with $T(w_i)$ are at most $1+\epsilon$ times of the query results detected by the accurate method with $T(w_i)$. Such a guarantee can be stated as, $\|R(T(w_i))\| \leq \|FP(T(w_i))\| \leq (1+\epsilon)\|R(T(w_i))\|$. If we set $\frac{T(w_1)}{x_1} = 1+\epsilon$. The equation (3) can be transformed to $err_{IH} = p\frac{1+\delta-\frac{T(w_1)}{x_1}}{\delta} = p\frac{\delta-\epsilon}{\delta}$.

Lemma 1 *Given ϵ, when $\delta > \epsilon$ the false positive method can provide a $(1+\epsilon)$-approximation aggregate monitoring query processing with at most the false alarm probability $p\frac{\delta-\epsilon}{\delta}$.*

Lemma 2 *Given ϵ, when $\delta \leq \epsilon$ the false positive method can provide a $(1+\epsilon)$-approximation aggregate monitoring querying processing with no false alarm.*

Theorem 3 *Given ϵ, λ and δ, the error probability of the proposed method for answering aggregate monitoring query on window w_i can be bounded by $P(x \geq \lambda) + max(p\frac{\delta-\epsilon}{\delta}, 0) - P(x \geq \lambda) \cdot max(p\frac{\delta-\epsilon}{\delta}, 0)$ with false positive oriented IH, where $x \sim Norm(0,1)$.*

The similar conclusions can be achieved for the false negative oriented method.

5. Performance Evaluation

The algorithms proposed in this paper are implemented with Microsoft Visual C++ Version 6.0. All experiments are conducted on a Windows 2000 platform with 2.4GHz CPU and 512MB main memory. The algorithms are run with a variety of data sets. Due to the limitation of space, only the results for two representative data sets are reported here. The two data sets are:

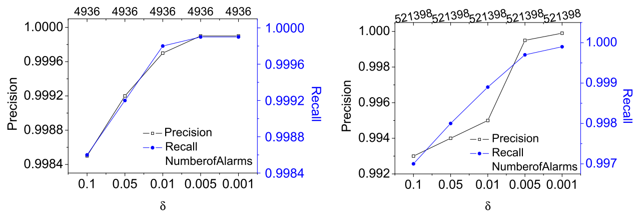
- Network Traffic (Real): The data set BC-Oct89Ext4, called D1 here, is a network traffic tracing data set obtained from the Internet Traffic Archive [1].
- Network Traffic (Synthetic): This synthetic data set is for testing the scalability of our method. It is generated by setting the burst arrival of a data stream with a pareto[9] distribution, as used in simulating network traffic where packets are sent according to ON OFF periods. The density function of pareto distribution is $P(x) = \frac{ab^a}{x^{a+1}}$, where $b \geq x$ and a is the shape parameter. The expected burst count, $E(x)$, is $\frac{ab}{a-1}$. The tuple arrival rate λ_1 is driven by an exponential distribution, and the interval λ_2 between signals is also generated with exponential distribution. In this data set, the expected value $E(\lambda_1) = 400 tuples/s, E(\lambda_2) = 500 tuples, a = 1.5, b = 1$. The size of this time series data set is $n = 100,000,000s$. The whole data set is called D2 here.

In the experiments, two accuracy metrics are used, recall and precision. Recall is the ratio of true alarms raised to the total true alarms which should be raised. Precision is the ratio of true alarms raised to the total alarms raised.

To set the threshold for $F(w_i)$ on i-length window, we compute moving $F(w_i)$ over some training data. The training data is the foremost 10% part of each data set. It forms another time series data set, called y. The absolute thresholds are set to be $T(w_i) = \mu_y + \lambda\sigma_y$, where μ_y and σ_y are the mean and standard deviation, respectively. The threshold can be tuned by varying the prefactor λ of standard deviation. The length of windows are 4, 8, ..., $4 * NW$ time units, where NW is the number of windows. NW varies from 50 to 1000. For no latency, the time unit is one data point in the data sets.

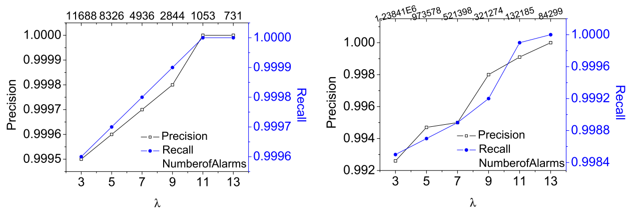
5.1. Experimental results

We evaluate the precision and the recall of our aggregate monitoring method. It is supposed that $F(w_i)$ is computed false positively by IH, and F is the distributive aggregate sum. The experiments are conducted on both the aforementioned two data sets. First, the relationship between accuracy and maximum relative error of IH is studied. In this experiment, we set number of monitored windows $NW = 100$ and prefactor $\lambda = 7$. It can be seen from Fig.5 that under all the setting of δ the precision and recall of our method are at least 99%. With the decreasing of relative error, IH can provide better approximation. Therefore,



(a) On D1(Real) (b) On D2(Synthetic)

Figure 5. Precision and recall on varying maximum relative error δ of IH, with $NW = 100$ and $\lambda = 7$



(a) On D1(Real) (b) On D2(Synthetic)

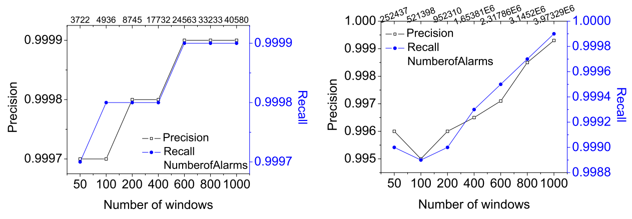
Figure 6. Precision and recall on varying prefactor λ, with $NW = 100$ and $\delta = 0.01$

our method can give highly accurate answers for aggregate monitoring queries based on IH over data streams.

Second, the relationship between accuracy and threshold of aggregation is studied. In this experiment, we set number of monitored windows $NW = 100$ and maximum relative error $\delta = 0.01$ of IH. It can be seen from Fig.6.(a) and Fig.6.(b) that under any setting of λ the precision and recall of our method are always above 99%. With the increasing of λ, our method can guarantee better accuracy, for fractal approximation error err_{FA} is decreasing.

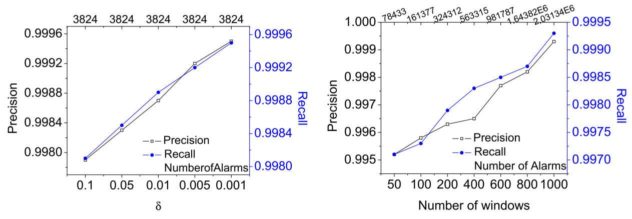
Third, the relationship between accuracy and number of monitored windows is studied. In this experiment, we set $\lambda = 7$ and $\delta = 0.01$. It can be seen from Fig.7 that with NW increasing, the precision and recall of our method are still above 99%. The accuracy are getting better and better with the increasing of NW. That is because the average size of monitored windows is getting larger with the increasing of NW. The fractal approximation is more accurate on large time scales. So, the error err_{FA} is decreasing with the increasing of NW. It can be concluded that our method can monitor large amounts of windows with high accuracy over data streams.

Now, we evaluate the precision and recall when applying the proposed method in algebraic aggregate $variance$. Because of the limitation of space, only a part of results are presented here. As shown by the first experiment, the accuracy of monitoring $variance$ is also very high when based on IH. The settings are $NW = 100$ and $\lambda = 7$. It can be



(a) On D1(Real) (b) On D2(Synthetic)

Figure 7. Precision and recall on varying number of windows, with $\lambda = 7$ and $\delta = 0.01$

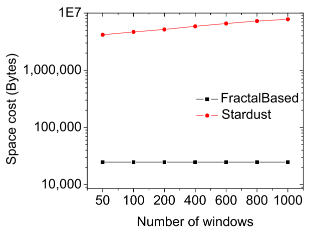


(a) varying δ on D1 (b) varying NW on D2

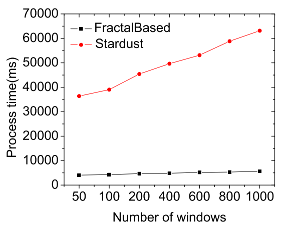
Figure 8. Precision and recall on monitoring algebraic aggregate $variance$

seen from Fig.8.(a) that the precision and recall are always above 99% under any setting of δ. The second experiment is conducted with the setting $NW = 100$ and $\delta = 0.01$. It could be observed from Fig.8.(b) that the increase of λ can reduce the fractal approximation error correspondingly. So, the accuracy of monitoring $variance$ is getting better. In the third experiment, the settings are $\lambda = 7$ and $\delta = 0.01$. The accuracy are getting better with the increase of NW, as shown in Fig.8.(b). It can be concluded again that our method can monitor large amounts of windows over data stream with high accuracy; moreover, both the distributive and algebraic aggregates can be monitored using the proposed method.

Here, some experiments are designed to compare our method with the most recently appeared work *Stardust* [7] in the aspects of space and time overhead. Although the *Stardust* can monitor aggregate exactly, while our method just do it approximately, our method need only very little memory and time to answer aggregate monitoring queries if a very small and acceptable error, say 0.1%, is allowed, which is general in real applications. In comparison with the accurate method, our method could run more than tens of times fast, and it makes sense in real-time monitoring. Thanks to the generous help from the author of [7], we got the original code package of *Stardust*. Then, it makes our comprehensive comparison with *Stardust* possible. Because *Stardust* is written in JAVA, for the sake of fairness, all our algorithms are re-implemented in JAVA. Nevertheless, all parameters setting is kept the same as introduced hereinbefore. That is to say, we set basic window length



(a) Space comparing On D1



(b) Time comparing On D1

Figure 9. Space and Time cost comparing with Stardust.

$W = 1$, smallest monitored window length $K = 4$, and the range of the number of monitored windows is varying in $\{50, 100, 200, 400, 600, 800, 1000\}$ for Stardust. The other setting, F is sum, $\lambda = 7$, and $\delta = 0.01$. A special setting for Stardust is the box capacity $c = 2$.

From Fig.9.(a), it could be seen that the proposed method is space efficient. The space cost of the method is the same as that of IH, which is only depends on the stream length but bears no relationship with the number of monitored windows. However, Stardust has to maintain all the monitored data points and associated index structure at the same time. It can be seen that the space saved by IH is getting larger and larger as NW and stream length increase. Thus, the proposed method is more adaptable to monitor multi-granularity windows over data streams, compared with previous ones. Also, it is time efficient. This is shown in Fig.9.(b). With search space enlarging, the processing time saved by using monotonic search space is getting larger.

6 Discussion and Conclusions

We present a novel method for monitoring aggregates with multi-granularity windows over a data stream. Fractal analysis is employed to transform the original data stream to its intermediate form. It is because the application of fractal techniques that we can not only process algebraic aggregates but also build a monotonic search space, with which the access to the synopsis is dramatically optimized. Inverted histogram is adopted as the core synopsis for the monitoring tasks. It expends limited storage space and computation cost for providing enough accurate monitoring result. Both theoretical and empirical result show the accuracy and efficiency of the proposed method.

The future research could be focused on the following three points. First, the proposed method could be extended to automatically determine the most appropriate windows to be monitored. Second, the method could be generalized, so that it can handle not only the absolute thresholds but the relative ones. The last but not the least, some extension could be done to this method, the goal is to make it possible to process more complicated queries, such as iceberg queries.

Acknowledgement

The authors would like to thank Dr. Ahmet Bulut for providing the code of Stardust. This work is partially supported by the NSFC under grant No.60503034, 60496325, and 60496327 and Shanghai Rising-Star Program under grant No.04QMX1404.

References

[1] Internet traffic archive. http://ita.ee.lbl.gov/.
[2] A. Arasu and J. Widom. Resource sharing in continuous sliding-window aggregates. In *Proc. of VLDB*, 2004.
[3] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proc. of ACM PODS*, 2003.
[4] M. Barnsley. *Fractals Everywhere*. Academic Press, New York, 1988.
[5] S. Ben-David, J. Gehrke, and D. Kifer. Detecting change in data streams. In *Proc. of VLDB*, 2004.
[6] P. Borgnat, P. Flandrin, and P. O. Amblard. Stochastic discrete scale invariance. *IEEE Signal Processing Letters*, 9(6):181 – 184, June 2002.
[7] A. Bulut and A. K. Singh. A unified framework for monitoring data streams in real time. In *Proc. of ICDE*, 2005.
[8] G. Cormode and S. Muthukrishnan. What's new: Finding significant differences in network data streams. In *Proc. of INFOCOM*, 2004.
[9] M. E. Crovella, M. S. Taqqu, and A. Bestavros. Heavy-tailed probability distributions in the world wide web. *A practical guide to heavy tails: STATISTICAL TECHNIQUES AND APPLICATIONS*, pages 3–26, 1998.
[10] S. Guha, N. Koudas, and K. Shim. Datastreams and histograms. In *Proc. of STOC*, 2001.
[11] J. C. Hart. Fractal image compression and recurrent iterated function systems. *IEEE Computer Graphics and Applications*, 16(4):25–33, July 1996.
[12] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *Proc. of CIKM*, 2003.
[13] J. Kleinberg. Bursty and hierarchical structure in streams. In *Proc. of SIGKDD*, 2002.
[14] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proc. of IMC*, 2003.
[15] B. B. Mandlebrot. *The fractal geometry of nature*. Freeman, New York, 1982.
[16] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. of VLDB*, 2002.
[17] D. S. Mazel and M. H. Hayes. Using iterated function systems to model discrete sequences. *IEEE Transactions on Signal Processing*, 40(7):1724–1734, July 1992.
[18] X. Wu and D. Barbara. Using fractals to compress real data sets: Is it feasible? In *Proc. of SIGKDD*, 2003.
[19] J. X. Yu, Z. Chong, H. Lu, and A. Zhou. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In *Proc. of VLDB*, 2004.
[20] A. Zhou, S. Qin, and W. Qian. Adaptively detecting aggregation bursts in data streams. In *Proc. of DASFAA*, 2005.
[21] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *Proc. of SIGKDD*, 2003.