A Geometric Approach to Monitoring Threshold Functions Over Distributed Data Streams[*]

Izchak Sharfman
Faculty of Computer Science
Technion
Haifa, Israel
tsachis@cs.technion.ac.il

Assaf Schuster
Faculty of Computer Science
Technion
Haifa, Israel
assaf@cs.technion.ac.il

Daniel Keren
Department of Computer Science
Haifa University
Haifa, Israel
dkeren@cs.haifa.ac.il

ABSTRACT

Monitoring data streams in a distributed system is the focus of much research in recent years. Most of the proposed schemes, however, deal with monitoring simple aggregated values, such as the frequency of appearance of items in the streams. More involved challenges, such as the important task of feature selection (e.g., by monitoring the information gain of various features), still require very high communication overhead using naive, centralized algorithms.

We present a novel geometric approach by which an arbitrary global monitoring task can be split into a set of constraints applied locally on each of the streams. The constraints are used to locally filter out data increments that do not affect the monitoring outcome, thus avoiding unnecessary communication. As a result, our approach enables monitoring of arbitrary threshold functions over distributed data streams in an efficient manner.

We present experimental results on real-world data which demonstrate that our algorithms are highly scalable, and considerably reduce communication load in comparison to centralized algorithms.

1. INTRODUCTION

A common requirement in many emerging applications is the ability to process, in real time, a continuous high volume stream of data. Examples of such applications are sensor networks [18], real-time analysis of financial data [26, 27], and intrusion detection. These applications are commonly referred to as *data stream systems* [3]. The real-time nature of data stream systems and the vast amounts of data they are required to process introduce new fundamental problems that are not addressed by traditional Database Management Systems (DBMS). Traditional DBMS are based on a *pull* paradigm, where users issue queries regarding data stored by the system, and the system processes these queries as they are issued and returns results. Data stream systems [8, 10, 17, 18, 22] are based on a *push* paradigm, where the users issue *continuous queries* [5, 25] that specify the required processing of the data, which the system processes as it arrives, continuously providing the user with updated results.

Various types of continuous queries have been studied in the past, including continuous versions of selection and join queries [19], various types of aggregation queries [21, 1], and monitoring queries [8]. While most previous work regarding data stream systems considers sequential setups (the data is processed by a single processor), many data stream applications are inherently distributed: examples include sensor networks [18], network monitoring [13], and distributed intrusion detection.

A useful class of queries in the context of distributed data streams are monitoring queries. Previous work in the context of monitoring distributed data streams considered monitoring simple aggregates, such as detecting when the sum of a distributed set of variables exceeds a predetermined threshold [13], or finding frequently occurring items in a set of distributed streams [20]. Some work has been done on monitoring more complex constructs derived from distributed streams, but the proposed solutions are customized for the problem at hand. Examples include [7] which presents an algorithm for detecting similar sets of streams among a large set of distributed streams, and [11], which presents an algorithm for approximating quantiles over distributed streams.

A useful, more general type of monitoring query can be defined as follows: let X_1,X_2,...,X_d be frequency counts for d items over a set of streams. Let $f(X_1,X_2,...,X_d)$ be an arbitrary function over the frequency counts. We are interested in detecting when the value of $f(X_1,X_2,...,X_d)$ rises above, or falls below, a predetermined threshold value. We refer to this query as a threshold function query.

There is a fundamental difference between the cases of linear and non-linear f, which can be demonstrated even for the case of one-dimensional data. Let x_1 and x_2 be values stored in two distinct nodes, and let $f(x) = 6x - x^2$. Suppose one needs to determine whether $f(\frac{x_1+x_2}{2}) > 1$. If f was linear, the solution would have been simple, since in that case $f(\frac{x_1+x_2}{2}) = \frac{f(x_1)+f(x_2)}{2}$. Suppose that initially the value at each node is < 1; then a simple distributed algorithm for monitoring whether $f(\frac{x_1+x_2}{2}) > 1$ is for each

[*]**Honorable mention for best paper award (ranked second/third paper in SIGMOD06).** Partially supported by the Israeli Ministry of Science.

node i to remain silent as long as $f(x_i) < 1$. However, even for the simple non-linear function above, *it is impossible to determine from the values of f at the nodes whether its value at the average is above* 1 *or not.* For example, if $x_1 = 0, x_2 = 6$, then f's value in each node is below 1, but its value in the average of x_1 and x_2 is 9. But if $x_1 = 10, x_2 = 20$, the value at both x_i and their average is below 1. So, nothing can be deduced about the location of $f(\frac{x_1+x_2}{2})$ vis-a-vis the threshold given the locations of $f(x_i)$ vis-a-vis it.

In this trivial example, the cost of sending the data stored in the nodes is the same as sending the value, but in data mining applications the data can be of very high dimensionality. This necessitates a distributed algorithm for locally determining whether f's value at the average data vector is above the threshold.

Following is a more practical example of a threshold function query: consider a classifier built over data extracted from a set of streams, for example a distributed spam mail filtering system. Such a system is comprised of agents installed on several dispersed mail servers. Users mark spam mail they have received as such, providing each server with a continuous stream of positive and negative samples. These samples serve as a basis for building a classifier. Since the vocabulary comprising these samples may be very large, an important task in such a setup is determining which words, or features, should be used for performing the classification. This task is known as feature selection. Feature selection is typically performed by calculating, for every feature, a non-linear scoring function, such as Information Gain or χ^2, over statistics collected from all the streams. All the features scoring above a certain threshold are chosen as parameters for the classification task. Since the characteristics of spam mail may vary over time, one may wish to monitor the features in order to determine if selected features remain prominent, or if any of the features not selected have become prominent. Determining whether a certain feature should be selected at a given time can be viewed as a threshold function query.

Threshold function queries can be implemented by collecting all the mail items to a central location, but such a solution is very costly in terms of communications load. We are interested in algorithms that implement threshold function queries in a more efficient manner. We achieve this by defining numerical constraints on the data collected at each node. As data arrives on the streams, every node verifies that the constraint on its stream has not been violated. We will show that as long as none of these constraints have been violated, the query result is guaranteed to remain unchanged, and thus no communication is required.

In this paper we present two algorithms for efficiently performing threshold function queries. The algorithms are based on a geometric analysis of the problem. Upon initialization, the algorithms collect frequency counts from all the streams, and calculate the initial result of the query. In addition, a numerical constraint on the data received on each individual stream is defined. As data arrives on the streams, each node verifies that the constraint on its stream has not been violated. The geometric analysis of the problem guarantees that as long as the constraints on all the streams are upheld, the result of the query remains unchanged, and thus no communication is required. If a constraint on one of the streams is violated, new data is gathered from the streams, the query is reevaluated, and new constraints are set on the streams.

The first algorithm is a decentralized algorithm, designed for a closely coupled environment, where nodes can efficiently broadcast messages. The second algorithm is designed for loosely coupled environments, where the cost of broadcasting a message is high. These algorithms are, to the best of our knowledge, the first to enable efficient monitoring of arbitrary threshold functions over distributed data streams.

1.1 Detailed Example

Following is a detailed description of the spam filtering example given above. We will use this example to demonstrate the concepts we present. Let $p_1, p_2, ..., p_n$ be n agents installed on n different mail servers. Let $M_i = \{m_{i,1}, m_{i,2}, ..., m_{i,k}\}$ be the last k mail items received at the mail server installed on p_i, and let M denote the union of the last k mail items received at each one of the n mail servers, $M = \bigcup_{i=1}^{n} M_i$. Let X denote a set of mail items, let $Spam(X)$ be the set of mail items in X labeled as spam, and let $\overline{Spam}(X)$ be the set of mail items in X not labeled as spam. Let $Cont(X, f)$ be the set of mail items in X that contain the feature f, and let $\overline{Cont}(X, f)$ be the set of mail items in X that do not contain the feature f. Let the contingency table $C_{f,X}$ for the feature f over the set of mail items X be a 2×2 matrix, $C_{f,X} = \{c_{i,j}\}$, such that $c_{1,1} = \frac{|Cont(X,f) \cap Spam(X)|}{|X|}$, $c_{1,2} = \frac{|Cont(X,f) \cap \overline{Spam}(X)|}{|X|}$, $c_{2,1} = \frac{|\overline{Cont}(X,f) \cap Spam(X)|}{|X|}$, and $c_{2,2} = \frac{|\overline{Cont}(X,f) \cap \overline{Spam}(X)|}{|X|}$. C_{f,M_i} is called the local contingency table for the node p_i, and $C_{f,M}$ is called the global contingency table. Note that $C_{f,M} = \frac{\sum_{i=1}^{n} C_{f,M_i}}{n}$. We are interested in determining, for each feature f, whether the Information Gain over its global contingency table, denoted by $G(C_{f,M})$, is above or below a predetermined threshold r. The formula for Information Gain is given below[1]

$$G(C_{f,X}) = \sum_{i \in \{1,2\}} \sum_{j \in \{1,2\}} c_{i,j} \cdot \log \frac{c_{i,j}}{(c_{i,1} + c_{i,2}) \cdot (c_{1,j} + c_{2,j})}$$

Note that the answer to the threshold function query *cannot* be derived from the value of the monitored function on data from each individual stream. Consider, for example, a spam filtering system consisting of two streams, with a threshold value of 0.5. The first node may hold a contingency table $C_{f,M_1} = \begin{matrix} 1 & 0 \\ 0 & 0 \end{matrix}$, resulting in $G(C_{f,M_1}) = 0$, and the second node may hold a contingency table $C_{f,M_2} = \begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix}$, resulting in $G(C_{f,M_2}) = 0$. As we can see, the gain calculated on each individual stream is 0, and thus below the threshold value, but the gain on the global contingency table for f, $C_{f,M_1 \cup M_2} = \frac{C_{f,M_1} + C_{f,M_2}}{2} = \begin{matrix} 0.5 & 0 \\ 0 & 0.5 \end{matrix}$, is $G(C_{f,M_1 \cup M_2}) = 1$, and thus above the threshold value. Note that this behaviour does not occur when monitoring frequencies of occurrence of items over distributed streams, i.e., if the frequency of occurrence of a certain item in all the streams is below a predetermined threshold, then the

[1]If $c_{i,j} = 0$ then $c_{i,j} \cdot \log \frac{c_{i,j}}{(c_{i,1}+c_{i,2}) \cdot (c_{1,j}+c_{2,j})}$ is defined as 0.

frequency of occurrence of that item over the union of the streams is below the threshold as well.

2. RELATED WORK

A well studied problem is the monitoring of frequency counts over a single data stream [1, 2, 9, 21], however these works do not address distributed environments.

Algorithms proposed in [13] enable detecting when the sum of a distributed set of variables exceeds a predetermined threshold. However, the algorithms proposed in [13] concentrate on monitoring the sum of a set of variables, whereas our algorithm enable monitoring arbitrary threshold functions over such variables. More recently, [12] presented algorithms that adapt local thresholds when monitoring the sum of a set of variables, so that the communication cost is minimized. An interesting avenue for future work is making use of the techniques presented in [12] when monitor arbitrary threshold functions.

The algorithm proposed in [23] enables a central coordinator to answer continuous queries designed to track the sum, average, or minimum of a distributed set of variables within a certain predetermined error margin. That work focuses on minimizing the communications required for performing several concurrent monitoring tasks, whereas our work proposes algorithms for monitoring arbitrary threshold functions.

Babcock and Olston propose an algorithm for finding the k largest aggregated values (for example the k largest frequency counters) over a set of distributed streams [4]. Their algorithm employs a coordinator, which determines the initial set of the k largest aggregates, and sends each node a set of numerical constraints. Each node checks that the data received on its stream does not violate this constraint. As long as all the constraints are upheld, the list of top k values is guaranteed to remain unchanged. In case a constraint is violated at one of the nodes, it notifies the coordinator, which performs a resolution process. The goal of the resolution process is to check if the list of the k largest values has changed, and to update the constraints at the nodes. The coordinator-based algorithm proposed in our work is similar to that algorithm in its use of numerical constraints, but its purpose is to monitor an arbitrary threshold function.

Algorithms proposed in [14, 15] enable estimating certain functions over a set of distributed streams, for example the number of distinct elements in the streams, but this work does not address the monitoring of threshold functions.

3. COMPUTATIONAL MODEL

Let S=$\{s_1,s_2,...s_n\}$, be a set of n data streams, collected at nodes P=$\{p_1,p_2,...,p_n\}$. Let $\vec{v}_1(t)$,$\vec{v}_2(t)$,...,$\vec{v}_n(t)$ be d-dimensional real vectors derived from the streams (the value of these vectors varies over time). These vectors are called *local statistics vector*s. Let w_1,w_2,...,w_n be positive weights assigned to the streams.

The weight w_i assigned to the node p_i usually corresponds to the number of data items its local statistics vector is derived from. Assume, for example, that we would like to determine whether the frequency of occurrence of a certain data item in a set of streams is above a certain threshold value. In this case, the weight we assign to each node at time t is the number of data items received on the stream at time t (and $\vec{v}_i(t)$ is a scalar holding the frequency of occurrence of the item in the stream s_i). In this setup weights change over time. A variant of the problem stated above is for each node to maintain the frequency of occurrence of the item in the recent N_i data items received on the stream (this is known as working with a sliding window of size N_i). In that case, the weight assigned to each node is the size of its sliding window. In this setup weights do not change over time. For the sake of clarity, we assume at first that the weights are fixed, and that they are known to all nodes. Later, we modify our algorithms to handle weights that vary with time.

Let $\vec{v}(t) = \frac{\sum_{i=1}^{n} w_i \vec{v}_i(t)}{\sum_{i=1}^{n} w_i}$. $\vec{v}(t)$ is called the *global statistics vector*. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be an arbitrary function from the space of d-dimensional vectors to the reals. f is called the monitored function. We are interested in determining at any given time, t, whether or not $f(\vec{v}(t)) > r$, where r is a predetermined threshold value.

We present algorithms for two settings, a decentralized setting and a coordinator-based setting. Algorithms in both settings construct a vector called the estimate vector, denoted by $\vec{e}(t)$. The estimate vector is constructed from the local statistics vectors collected from the nodes at certain times, as dictated by the algorithms. The last statistics vector collected from the node p_i is denoted by $\vec{v'}_i$. Each node remembers the last statistics vector collected from it. The estimate vector is the weighted average of the latest statistics vectors collected from the nodes, i.e., $\vec{e}(t) = \frac{\sum_{i=1}^{n} w_i \vec{v'}_i}{\sum_{i=1}^{n} w_i}$. From time to time, as dictated by the algorithm, an updated statistics vector is collected from one or more nodes, and the estimate vector is updated. At any given time the estimate vector is known to all nodes.

In the decentralized setting, when the algorithm dictates that a statistics vector should be collected from a node, the node broadcasts the statistics vector to the rest of the nodes. Each node keeps track of the last statistics vector broadcast by every node, and locally calculates the estimate vector. In the coordinator-based setting, we designate a coordinator node and denote it by p_1. The coordinator is responsible for collecting local statistics vectors from the nodes, calculating the estimate vector, and distributing it to the nodes.

In both settings, each node p_i maintains a parameter called the statistics delta vector. This vector is denoted by $\Delta\vec{v}_i(t)$. The statistics delta vector held by the node p_i is the difference between the current local statistics vector and the last statistics vector collected from the node, i.e., $\Delta\vec{v}_i(t) = \vec{v}_i(t) - \vec{v'}_i$.

In both settings, each node p_i also maintains a parameter called the *drift vector*. This vector is denoted by $\vec{u}_i(t)$. The drift vector is calculated differently in each setting. In the decentralized setting the drift vector is a displacement of $\Delta\vec{v}_i(t)$ in relation to the estimate vector,

$$\vec{u}_i(t) = \vec{e}(t) + \Delta\vec{v}_i(t) \qquad (1)$$

The coordinator-based algorithm employs a mechanism for balancing the local statistics vectors of a subset of the nodes. Consider the case where at a certain time t the statistics delta vector in two equally weighted nodes, p_i and p_j, cancel each other out: that is $\Delta\vec{v}_i(t) = -\Delta\vec{v}_j(t)$. We will see that balancing the local statistics vectors held by p_i and p_j can improve the efficiency of the algorithm. The coordinator facilitates this balancing by sending each node a *slack vector*, denoted by $\vec{\delta}_i$. The sum of the slack vectors sent to the nodes is $\vec{0}$. The drift vector held by each node is calculated as follows:

$$\vec{u}_i(t) = \vec{e}(t) + \Delta\vec{v}_i(t) + \frac{\vec{\delta}_i}{w_i} \tag{2}$$

In the decentralized algorithm, nodes communicate by broadcasting messages. The cost of performing a broadcast varies according to the networking infrastructure at hand. In the worst case broadcasting a message to n nodes requires sending n point to point messages. While the decentralized algorithm remains highly efficient even in those settings, in practice, the cost of broadcasting a message is significantly lower. Some networking infrastructures, such as wireless networks and Ethernet based networks, support broadcasting at the cost of sending a single message. In other cases efficient broadcasting schemes have been developed that significantly reduce the cost of broadcasting.

We assume that communication links are reliable, i.e., no messages are lost (otherwise standard methods for implementing reliability can be employed).

4. GEOMETRIC INTERPRETATION

At the heart of our approach is the ability to decompose the monitoring task into local constraints on streams. As data arrives on the streams, each node verifies that the local constraint on its stream has not been violated. We will show that as long as none of these constraints have been violated, the query result is guaranteed to remain unchanged, and thus no communication is required. As demonstrated in Section 1, this cannot be done solely by observing the value of the monitored function on each stream. Therefore, an estimated global statistics vector, called the estimate vector, is known to all nodes. The estimate vector is said to be correct at a given time if the value of the monitored function on the estimate vector and the value of the monitored function on the global statistics vector at that time (this value is unknown to any singe node) are on the same side of the threshold. Given an initially correct estimate vector, our goal is to set local constraints on each stream such that as long as no constraints have been violated, the estimate vector remains correct, and thus no communication is required. The method for decomposing the monitoring task is based on the following, easily verifiable observation—at any given time the weighted average of the drift vectors held by the nodes is equal to the global statistics vector,

$$\frac{\sum_{i=1}^{n} w_i \vec{u}_i(t)}{\sum_{i=1}^{n} w_i} = \vec{v}(t) \tag{3}$$

We refer to Property (3) as the convexity property of the drift vectors. The geometric interpretation of Property (3) is that the global statistics vector is in the convex hull of the drift vectors held by the nodes,

$$\vec{v}(t) \in \mathrm{Conv}(\vec{u}_1(t), \vec{u}_2(t), ..., \vec{u}_n(t)) \tag{4}$$

This observation enables us to take advantage of Theorem 1 in order to decompose the monitoring task.

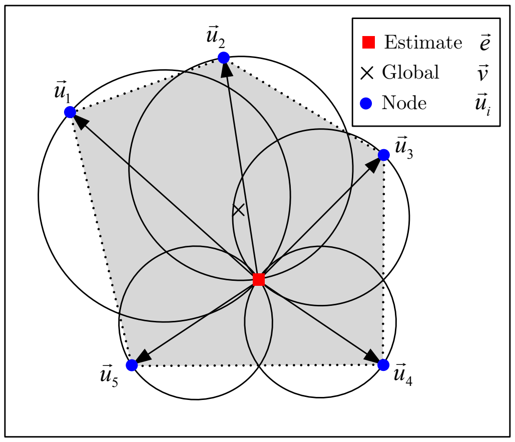


Figure 1: Illustration of Theorem 1. The drift vectors held by 5 nodes and the balls constructed by them are depicted. The convex hull of the drift vectors is highlighted in gray. As stated by the theorem, the union of the balls bounds the convex hull.

THEOREM 1. *Let $\vec{x}, \vec{y}_1, \vec{y}_2, ..., \vec{y}_n \in \mathbb{R}^d$ be a set of vectors in $\mathbb{R}^d$. Let $Conv(\vec{x}, \vec{y}_1, \vec{y}_2, ..., \vec{y}_n)$ be the convex hull of $\vec{x}, \vec{y}_1, \vec{y}_2, ..., \vec{y}_n$. Let $B(\vec{x}, \vec{y}_i)$ be a ball centered at $\frac{\vec{x}+\vec{y}_i}{2}$ and with a radius of $\left\|\frac{\vec{x}-\vec{y}_i}{2}\right\|_2$ i.e., $B(\vec{x}, \vec{y}_i) = \left\{\vec{z} \;\middle|\; \left\|\vec{z} - \frac{\vec{x}+\vec{y}_i}{2}\right\|_2 \leq \left\|\frac{\vec{x}-\vec{y}_i}{2}\right\|_2\right\}$, then $Conv(\vec{x}, \vec{y}_1, \vec{y}_2, ..., \vec{y}_n) \subset \bigcup_{i=1}^{n} B(\vec{x}, \vec{y}_i)$.*

Theorem 1 is used to bound the convex hull of $n+1$ vectors in $\mathbb{R}^d$ by the union of n d-dimensional balls. In our case it is used to bound the convex hull of the estimate vector and the drift vectors i.e., $\mathrm{Conv}(\vec{e}(t), \vec{u}_1(t), \vec{u}_2(t), ..., \vec{u}_n(t))$, by a set of n balls, where each ball is constructed independently by one of the nodes. Each node, p_i, constructs a ball $B(\vec{e}(t), \vec{u}_i(t))$, which is centered at $\frac{\vec{e}(t)+\vec{u}_i(t)}{2}$, and has a radius of $\left\|\frac{\vec{e}(t)-\vec{u}_i(t)}{2}\right\|$. Note that at any given time each node has all the information required to independently construct its ball. Theorem 1 states that $\mathrm{Conv}(\vec{e}(t), \vec{u}_1(t), \vec{u}_2(t), ..., \vec{u}_n(t)) \subset \bigcup_i B(\vec{e}(t), \vec{u}_i(t))$.

The application of Theorem 1 is illustrated in Figure 1, which depicts a setup comprised of 5 nodes, each holding a statistics vector, $\vec{v}_i(t) \in \mathbb{R}^2$. The drift vectors held by the nodes $(\vec{u}_1(t)..., \vec{u}_5(t))$, the global statistics vector $\vec{v}(t)$, and the estimate vector $\vec{e}(t)$ are depicted, as are the balls constructed by the nodes. The convex hull of the drift vectors is highlighted in gray, and one can see that, as the theorem states, the area defined by the convex hull is bounded by the set of balls.

4.1 Local Constraints

The local constraint on each stream is set as follows: the monitored function f and threshold r can be seen as inducing a coloring over $\mathbb{R}^d$. The vectors $\{\vec{x} | f(\vec{x}) > r\}$ are said to be green, while the vectors $\{\vec{y} | f(\vec{y}) \leq r\}$ are said to be red. The local constraint each node maintains is to check whether the ball $B(\vec{e}(t), \vec{u}_i(t))$ (the ball centered at $\frac{\vec{e}(t)+\vec{u}_i(t)}{2}$, and having a radius of $\left\|\frac{\vec{e}(t)-\vec{u}_i(t)}{2}\right\|$) is monochromatic, i.e., whether all the vectors contained in the ball have the same color. Testing for monochromicity is done by finding the maximal and minimal values of f in the ball. This is done locally at each node hence has no effect on the communication load.

If all the local constraints are upheld, the estimate vector is correct: because all the balls contain the estimate vector, and all the balls are monochromatic, the set of vectors defined by the union of all the balls is monochromatic as well. Since the union of all the balls contains the convex hull of the drift vectors and the estimate vector ($\mathrm{Conv}(\vec{e}(t), \vec{u}_1(t), \vec{u}_2(t), ..., \vec{u}_n(t))$), and according to Equation (4) the global statistics vector is contained in the convex hull of the drift vectors, the estimate vector and the global statistics vector have the same color. Therefore, they are on the same side of the threshold, i.e., the estimate vector is correct.

4.2 Proof of Theorem 1

Following is the proof of Theorem 1. The proof uses the following variant of Carathéodory's theorem [6]:

THEOREM 2. *Let $\vec{x}, \vec{y}_1, \vec{y}_2, ..., \vec{y}_n \in \mathbb{R}^d$ be a set of vectors in $\mathbb{R}^d$. Let $Conv(\vec{x}, \vec{y}_1, \vec{y}_2, ..., \vec{y}_n)$ be the convex hull of $\vec{x}, \vec{y}_1, \vec{y}_2, ..., \vec{y}_n$. Any vector $\vec{z} \in Conv(\vec{x}, \vec{y}_1, \vec{y}_2, ..., \vec{y}_n)$ can be expressed as a convex combination of $\vec{x}$, and at most d members of $\{\vec{y}_1, \vec{y}_2, ..., \vec{y}_n\}$.*

Proof of Theorem 1: The theorem is proven by induction over d. The base of the induction is $d = 1$. Proving the base of the induction is trivial ($\vec{x}, \vec{y}_1, \vec{y}_2, ..., \vec{y}_n$ are real numbers and both $\mathrm{Conv}(\vec{x}, \vec{y}_i)$ and $B(\vec{x}, \vec{y}_i)$ are the segment $[x, y_i]$).

The induction step is proven as follows: according to Theorem 2, since any vector $\vec{z} \in \mathrm{Conv}(\vec{x}, \vec{y}_1, \vec{y}_2, ..., \vec{y}_n)$ can be expressed as a convex combination of $\vec{x}$ and a set of at most d members of $\{\vec{y}_1, \vec{y}_2, ..., \vec{y}_n\}$, it is sufficient to show that for an arbitrary set of vectors, $\{\vec{x}, \vec{v}_1, \vec{v}_2, ..., \vec{v}_d\} \in \mathbb{R}^d$,

$$\mathrm{Conv}(\vec{x}, \vec{v}_1, \vec{v}_2, ..., \vec{v}_d) \subset \bigcup_{i=1}^{d} B(\vec{x}, \vec{v}_i) \quad (5)$$

Furthermore, we observe that any vector $\vec{z} \in \mathrm{Conv}(\vec{x}, \vec{v}_1, \vec{v}_2, ..., \vec{v}_d)$ is a convex combination of some vector $\vec{z'} \in \mathrm{Conv}(\vec{v}_1, \vec{v}_2, ..., \vec{v}_d)$ and $\vec{x}$. Therefore, it is sufficient to show that:

$$\mathrm{Conv}(\vec{v}_1, \vec{v}_2, ..., \vec{v}_d) \subset \bigcup_{i=1}^{d} B(\vec{x}, \vec{v}_i) \quad (6)$$

Equation (6) entails Equation (5) since $\vec{z}$ is a convex combination of $\vec{z'} \in \mathrm{Conv}(\vec{v}_1, \vec{v}_2, ..., \vec{v}_d)$ and $\vec{x}$, and since, according to Equation (6), $\mathrm{Conv}(\vec{v}_1, \vec{v}_2, ..., \vec{v}_d) \subset \bigcup_{i=1}^{d} B(\vec{x}, \vec{v}_i)$, $\vec{z'}$ belongs to at least one of the balls $B(\vec{x}, \vec{v}_1)$,$B(\vec{x}, \vec{v}_2)$,...,$B(\vec{x}, \vec{v}_d)$, say $B(\vec{x}, \vec{v}_j)$. Since by definition $\vec{x}$ also belongs to $B(\vec{x}, \vec{v}_j)$, $\vec{z}$ also belongs to $B(\vec{x}, \vec{v}_j)$. This is because $\vec{z}$ is a convex combination of $\vec{z'}$ and $\vec{x}$, which both belong to $B(\vec{x}, \vec{v}_j)$, which is a convex set.

Since for any rotation matrix R and vectors $\{q_1...q_n\}$, $\mathrm{Conv}\{R(q_1)...R(q_n)\} = R(\mathrm{Conv}\{q_1...q_n\})$, we assume without loss of generality that the vectors $\vec{v}_1, \vec{v}_2, ..., \vec{v}_d$ lie on a $d-1$ dimensional hyperplane, P, which consists of vectors whose last coordinate is constant. It is easy to show that the intersection of $B(\vec{x}, \vec{v}_1)$,$B(\vec{x}, \vec{v}_2)$,...,$B(\vec{x}, \vec{v}_d)$ with P yields the set of $d-1$ dimensional balls, $B(\vec{x}_p, \vec{v}_1)$,$B(\vec{x}_p, \vec{v}_2)$,...,$B(\vec{x}_p, \vec{v}_d)$, where $\vec{x}_p$ is the projection of $\vec{x}$ on the plane P. Note that for every i, the $d-1$ dimensional ball $B(\vec{x}_p, \vec{v}_i)$ is contained in the d dimensional ball $B(\vec{x}, \vec{v}_i)$ ($B(\vec{x}_p, \vec{v}_i) \subset B(\vec{x}, \vec{v}_i)$). Since the vectors $\vec{x}_p, \vec{v}_1, \vec{v}_2, ..., \vec{v}_d$ lie on the $d-1$ dimensional plane P, then, according to the induction hypothesis, $\mathrm{Conv}(\vec{v}_1, \vec{v}_2, ..., \vec{v}_d) \subset \bigcup_{i=1}^{d} B(\vec{x}_p, \vec{v}_i)$. Since for every i, $B(\vec{x}_p, \vec{v}_i)$ is contained in $B(\vec{x}, \vec{v}_i)$, then $\mathrm{Conv}(\vec{v}_1, \vec{v}_2, ..., \vec{v}_d) \subset \bigcup_{i=1}^{d} B(\vec{x}, \vec{v}_i)$. We have proved Equation (6), and thus concluded the proof. $\square$

5. DISTRIBUTED MONITORING

In this section we present algorithms that are based on the geometric method for decomposing the monitoring task into local constraints on the streams. After presenting the algorithms, we describe how they can be tuned to relax performance requirements in favour of reducing communication load, and how they can be modified to support time-varying weights.

5.1 The Decentralized Algorithm

Following is a simple, broadcast-based algorithm for monitoring threshold functions: each node maintains a copy of the last statistics vector sent by each of the nodes. The initialization stage consists of every node broadcasting its initial statistics vector. Upon receipt of all the initialization messages, each node calculates the estimate vector, $\vec{e}(t)$. Then, as more data arrives on the stream, each node can check its local constraint according to the estimate vector and its drift vector. If a local constraint is violated at a node, p_i, the node broadcasts a message of the form $< i, \vec{v}_i(t) >$, containing its identifier and its local statistics vector at the time. The broadcasting node updates its $\vec{v'}_i$ parameter and recalculates the estimate vector. Upon receiving a broadcast message from a node, p_i, each node updates its $\vec{v'}_i$ parameter and recalculates the estimate vector.

If all the local constraints are upheld, Theorem 1 guarantees the correctness of the estimate vector (enabling every node to locally calculate the value of the threshold function). After a node broadcasts a message, its local constraint is upheld (because the ball it constructs has a radius of 0, and therefore is monochromatic). If a local constraint has been violated, at worst all n nodes (but possibly fewer) will broadcast a message before all the local constraints are upheld again.

A formal description can be found in Algorithm 1.

5.2 The Coordinator-Based Algorithm

Local constraints are also used in the coordinator-based algorithm, but the coordinator is responsible for calculating the estimate vector, maintaining its correctness, and distributing it to the other nodes. In the decentralized algorithm the violation of a constraint on one node requires communicating with all the rest of the nodes (a broadcast message is sent). While this may be a good solution in setups where the nodes are closely coupled, in other cases we can further reduce the communication load by introducing a coordinator. The presence of a coordinator enables us to resolve a violation at a node by communicating with only a subset of the nodes, as opposed to communicating with all the nodes as required in the decentralized algorithm. Consider, for example, a set of equally weighted nodes monitoring the function $f(x) = (x - 5)^2$ (a function over a single dimensional statistics vector), and a threshold value of $r = 9$. Say

Algorithm 1 The decentralized algorithm

Initialization: at a node p_i

- Broadcast a message containing the initial statistics vector and update $\vec{v'}_i$ to hold the initial statistics vector. Upon receipt of messages from all the nodes, calculate the estimate vector ($\vec{e}(t)$).

Processing Stage at a node p_i:

- Upon arrival of new data on the local stream, recalculate $\vec{v}_i(t)$, and $\vec{u}_i(t)$, and check if $B(\vec{e}(t), \vec{u}_i(t))$ remains monochromatic. If not, broadcast the message $<i,\vec{v}_i(t)>$ and update $\vec{v'}_i$ to hold $\vec{v}_i(t)$.
- Upon receipt of a new message $<j,\vec{v}_j(t)>$, update $\vec{v'}_j$ to hold $\vec{v}_j(t)$, recalculate $\vec{e}(t)$, and check if $B(\vec{e}(t), \vec{u}_i(t))$ is monochromatic. If $B(\vec{e}(t), \vec{u}_i(t))$ is not monochromatic, broadcast the message $<i,\vec{v}_i(t)>$ and update $\vec{v'}_i$ to hold $\vec{v}_i(t)$.

that at time t the estimate vector is $\vec{e}(t) = 5$. Note that since $f(\vec{e}(t)) = 0 < r$, any drift vector in the range $[2, 8]$ satisfies the local constraint at the node. Let us assume that the drift vector at the coordinator, p_1, is $\vec{u}_1(t) = 4$, and the constraints at all n nodes are satisfied except for p_2, which holds the drift vector $\vec{u}_2(t) = 1$. In the decentralized algorithm, since the constraint at p_2 has been violated, it would have broadcast its statistics vector to all n nodes. However, the constraint violation at p_2 can be resolved by setting the drift vector at both p_1 and p_2 to the average of the drift vectors on both nodes, i.e., by setting $\vec{u}_1(t) = \vec{u}_2(t) = \frac{\vec{u}_1(t)+\vec{u}_2(t)}{2} = 2.5$. After this averaging operation, drift vectors on both p_1 and p_2 are within the range $[2, 8]$, and thus all the local constraints are upheld. Note that this action preserves the convexity property of the drift vector (Property 3). The act of averaging out a subset of drift vectors in order to resolve a violated constraint is called a balancing process.

In order to facilitate the balancing of vectors, every node p_i holds a slack vector denoted by $\vec{\delta}_i$, as defined in Section 3. The slack vector is first normalized by dividing it by the weight assigned to the node. Then it is added to the drift vector as specified in Equation (2). The coordinator is responsible for ensuring that the sum of all slack vectors is $\vec{0}$, thus maintaining the convexity property of the drift vectors (Property 3).

To initialize the algorithm, each node sends its initial statistics vector to the coordinator. Initially the slack vector held by each node is set to $\vec{0}$. The coordinator calculates the estimate vector and sends it to the rest of the nodes. As more data arrives on a node's stream, the node checks its local constraint. If a local constraint is violated at one of the nodes, it notifies the coordinator by sending it a message containing its current drift vector and its current statistics vector. The coordinator first tries to resolve the constraint violation by executing a balancing process.

During the balancing process the coordinator tries to establish a group of nodes (called the balancing group and denoted by P'), such that the average of the drift vectors held by the nodes in the balancing group (called the balanced vector and denoted by $\vec{b}$), creates a monochromatic ball with the estimate vector i.e., such that $B(\vec{e}(t), \vec{b})$ is monochromatic. The balanced vector is calculated as follows:

$$\vec{b} = \frac{\sum_{p_i \in P'} w_i \vec{u}_i(t)}{\sum_{p_i \in P'} w_i} \tag{7}$$

The balancing process proceeds as follows: when a node p_i notifies the coordinator that its local constraint has been violated, it appends its drift vector and its current statistics vector to the message. The coordinator constructs a balancing group consisting of p_i and itself. It then checks if the ball defined by the balanced vector, $B(\vec{e}(t), \vec{b})$, is monochromatic. If $B(\vec{e}(t), \vec{b})$ is not monochromatic, the coordinator randomly selects a node that is not in the balancing group, and requests it to send its drift vector and local statistics vector. Then it adds that new node to the balancing group and rechecks $B(\vec{e}(t), \vec{b})$. The process is performed iteratively until either $B(\vec{e}(t), \vec{b})$ is monochromatic, or the balancing group contains all the nodes. If the coordinator established a balancing group such that $B(\vec{e}(t), \vec{b})$ is monochromatic, the balancing process is said to have succeeded. In this case the coordinator sends each node in the balancing group an adjustment to its slack vector. This causes the drift vectors held by all nodes in the balancing group to be equal to $\vec{b}$. The adjustment to the slack vector sent to each node $p_i \in P'$ is denoted by $\Delta\vec{\delta}_i$, and is calculated as follows:

$$\Delta\vec{\delta}_i = w_i\vec{b} - w_i\vec{u}_i(t)$$

After receiving the slack vector adjustment, each node simply adds the adjustment to the current value, i.e., $\vec{\delta}_i \leftarrow \vec{\delta}_i + \Delta\vec{\delta}_i$. One can easily verify that after a successful balancing process the sum of all slack vectors remains $\vec{0}$, and the drift vector held by each node in the balancing group is $\vec{b}$, thus resolving the original constraint violation.

If the balancing process has failed (i.e., the balancing group contains all the nodes, and $B(\vec{e}(t), \vec{b})$ is not monochromatic), the coordinator calculates a new estimate vector (according to the updated statistics vectors sent by the nodes) and sends it to all the nodes. Upon receipt of the new estimate vector, the nodes set their slack vectors to $\vec{0}$ and modify their $\vec{v'}_i$ parameter to hold the value of the statistics vector they sent to the coordinator during the balancing process, thus resolving the original constraint violation.

In order to implement the algorithm, the following messages must be defined:

$<$INIT,$\vec{v}_i>$	Used by nodes to report their initial statistics vector to the coordinator in the initialization stage.
$<$REQ$>$	Used by the coordinator during the balancing process to request that a node send its statistics vector and drift vector.
$<$REP,$\vec{v}_i$,$\vec{u}_i>$	Used by nodes to report information to the coordinator when a local constraint has been violated, or when the coordinator requests information from the node.

<ADJ-SLK,$\Delta\vec{\delta}_i$> Used by the coordinator to report slack vector adjustments to nodes after a successful balancing process.

<NEW-EST,$\vec{e}$> Used by the coordinator to report to the nodes a new estimate vector.

A formal description is given in Algorithm 2.

5.3 Relaxing the Precision Requirements

A desired trade-off when monitoring threshold functions is between accuracy and communication load. In some cases an approximate value of the threshold function is sufficient, that is, the correct value of the threshold function is required only if the value of the monitored function is significantly far from the threshold. In other words, if ε is a predetermined error margin, and if $f(\vec{v}(t)) > r+\varepsilon$ or $f(\vec{v}(t)) \le r-\varepsilon$, we require that the estimate vector, $\vec{e}(t)$, be correct, but we do not require it if $r-\varepsilon < f(\vec{v}(t)) \le r+\varepsilon$.

Consider the feature monitoring example given in Section 1. Say we would like to select all the features whose information gain is above 0.05. Obviously, it is important to select a feature whose information gain is significantly high, and not to select a feature whose information gain is significantly low. For example, it is important to select a feature whose information gain score is 0.1, and not to select a feature whose information gain score is 0.01. Including or excluding features whose information gain score is very close to the threshold value, for example a feature whose information gain score is 0.048, will probably not have a significant effect on the quality of the selected feature set, while the cost of monitoring such features is expected to be high, since their information gain is expected to fluctuate around the threshold value. Therefore we can significantly improve the efficiency of our monitoring algorithms if we set some error margin, say 0.005. In other words, features that are currently selected will be removed from the set of selected features only when their information gain falls below 0.045, and features that are currently not selected will be added to the set of selected feature only if their information gain rises above 0.055.

Our algorithm can be easily tuned to relax the precision requirements by an error margin of ε as follows: instead of working with a single coloring, induced by the monitored function f and the threshold value r, two sets of coloring are defined, one induced by the monitored function f and the threshold value $r+\varepsilon$, and a second induced by the monitored function f and the threshold value $r-\varepsilon$. Whenever the original algorithm checks whether a ball is monochromatic, then, if $f(\vec{v}(t)) \le r$, the modified algorithm will check whether the ball is monochromatic according to the first coloring (the one induced by f and $r+\varepsilon$). If $f(\vec{v}(t)) > r$, the modified algorithm will check whether the ball is monochromatic according to the second coloring (the one induced by f and $r-\varepsilon$). This ensures that if all the balls are in the range defined by $\{\vec{x} | r-\varepsilon < f(\vec{x}) \le r+\varepsilon\}$, no messages are transmitted.

5.4 Handling Time-Varying Weights

Up to this point we have assumed that the weights assigned to nodes are fixed, such as when the weights are the size of sliding windows used for collecting data from the streams. We now address cases where weights assigned to nodes may vary with time, as when a node's weight at a

Algorithm 2 The coordinator-based algorithm

Initialization:

- Send an INIT message to the coordinator, set $\vec{v}'$ to hold the initial statistics vector, and set the slack vector to $\vec{0}$. Upon receipt of messages from all nodes, the coordinator calculates the estimate vector and informs the nodes via a NEW-EST message.

Processing Stage at an Ordinary Node p_i:

- Upon arrival of new data on a node's local stream, recalculate $\vec{v}_i(t)$ and $\vec{u}_i(t)$, and check if $B(\vec{e}(t), \vec{u}_i(t))$ remains monochromatic. If not, send a <REP,$\vec{v}_i(t)$,$\vec{u}_i(t)$> message to the coordinator, and wait for either a NEW-EST or an ADJ-SLK message.
- Upon receipt of a REQ message, send a <REP,$\vec{v}_i(t)$,$\vec{u}_i(t)$> message to the coordinator and wait for either a NEW-EST or ADJ-SLK message.
- Upon receipt of a NEW-EST message, update the estimate vector ($\vec{e}(t)$) to the value specified in the message, set the value of $\vec{v}'$ to the statistics vector sent to the coordinator, and set the slack vector to $\vec{0}$.
- Upon receipt of an ADJ-SLK message, add the value specified in the message to the value of the slack vector ($\vec{\delta}_i \leftarrow \vec{\delta}_i + \Delta\vec{\delta}_i$).

Processing Stage at the Coordinator:

- Upon arrival of new data on the local stream, recalculate $\vec{v}_1(t)$ and $\vec{u}_1(t)$, and check if $B(\vec{e}(t), \vec{u}_1(t))$ remains monochromatic. If not, initiate a balancing process, setting the balancing group to $P' = \{< 1, \vec{v}_1(t), \vec{u}_1(t) >\}$.
- Upon receipt of a REP message from the node p_i, initiate a balancing process, setting the balancing group to $P' = \{< 1, \vec{v}_1(t), \vec{u}_1(t) >, < i, \vec{v}_i, \vec{u}_i >\}$.

Balancing Process at the Coordinator:

1. Calculate balanced vector, $\vec{b}$, according to Equation (7). If the ball $B(\vec{e}(t), \vec{b})$ is monochromatic goto (2), otherwise goto (3).
2. For each item in the balancing group, $< i, \vec{v}_i, \vec{u}_i >$, calculate the slack vector adjustment, $\Delta\vec{\delta}_i = w_i\vec{b} - w_i\vec{u}_i(t)$, send p_i a <ADJ-SLK,$\Delta\vec{\delta}_i$> message, and then exit the Balancing Process.
3. If there are nodes not contained in the balancing group, select one of these nodes at random, and send it a REQ message. Upon receipt of the REP message, add the node to the Balancing Group and goto (1). Otherwise calculate a new estimate vector (based on the $\vec{v}_i$ values received from all the nodes), send a NEW-EST message to all nodes, and exit the Balancing Process.

given time is the number of data items received on its stream so far.

We next describe the required modifications to the algorithms in order to ensure their correctness in a setup where weights vary with time. In such a setup we denote the weight assigned to the node p_i at time t by $w_i(t)$. Each message in the original algorithms is modified by appending $w_i(t)$ to it. Along with the last vector broadcast by each of the other nodes ($\vec{v'}_i$), the nodes in the decentralized algorithm keep track of the last broadcast weight, denoted by w'_i. Nodes calculate $\vec{e}_i(t)$, $\Delta\vec{v}_i(t)$, and $\vec{u}_i(t)$ as follows:

$$\vec{e}_i(t) = \frac{\sum_{i=1}^{n} w'_i \vec{v'}_i}{\sum_{i=1}^{n} w'_i}$$

$$\Delta\vec{v}_i(t) = \frac{w_i(t)\vec{v}_i(t) - w'_i\vec{v'}_i(t) - (w_i(t) - w'_i)\vec{e}_i(t)}{w_i(t)}$$

In the decentralized algorithm the drift vector is calculated by

$$\vec{u}_i(t) = \vec{e}_i(t) + \Delta\vec{v}_i(t)$$

and in the coordinator-based algorithm, by

$$\vec{u}_i(t) = \vec{e}(t) + \Delta\vec{v}_i(t) + \frac{\vec{\delta}_i}{w_i(t)}$$

In the coordinator-based algorithm, the balanced vector and the slack vector adjustments are calculated according to the weights appended to the messages:

$$\vec{b} = \frac{\sum_{i=1}^{k} w_i(t)\vec{u}_i(t)}{\sum_{i=1}^{k} w_i(t)}$$

$$\Delta\vec{\delta}_i = w_i(t)\vec{b} - w_i(t)\vec{u}_i(t)$$

Note that if the weights are fixed, $\vec{e}_i(t)$, $\Delta\vec{v}_i(t)$, and $\vec{u}_i(t)$ hold the same values they hold in the original algorithms. Furthermore, one can easily verify that the new definitions of these parameters maintain Equation (3) i.e., $\vec{v}(t) = \frac{\sum_{i=1}^{n} w_i(t)\vec{u}_i(t)}{\sum_{i=1}^{n} w_i(t)}$, and thus maintain the correctness of the algorithm.

6. PERFORMANCE ANALYSIS

We would like to determine how the various parameters of the monitoring problem affect the communication load generated by the proposed algorithms. In order to do so we present a simplified model of our algorithm and analyze the probability that a constraint violation will occur at a node. Since a constraint violation is the trigger for communications in both algorithms, this analysis should provide indications regarding the generated communication load.

Generally speaking, the dominant factor affecting the performance of the algorithms is the average distance of the estimate vector from the set of vectors for which the value of the monitoring function equals the threshold value. More formally, let the threshold set defined by the monitoring function f and threshold value r be the set of vectors for which the value of the threshold function equals the threshold value. Let the threshold set be denoted by $T(f, r)$, i.e.,

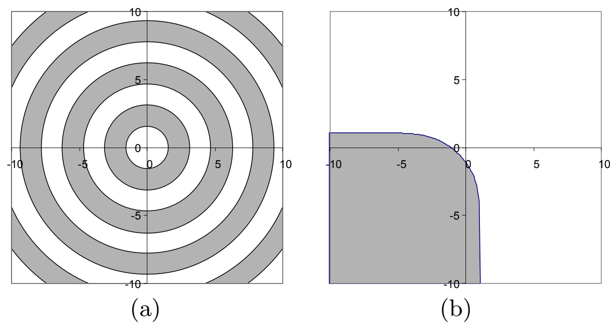


Figure 2: The colorings induced by two sets of monitored functions and threshold values. (a) depicts the coloring induced by $f_1 = \sin(2\sqrt{x^2+y^2}) \geq 0$, and (b) depicts the coloring induced by $f_2 = \frac{1}{1+e^{-x}} + \frac{1}{1+e^{-y}} \geq 0.75$.

$$T(f, r) = \{\vec{x} | f(\vec{x}) = r\}$$

Let the distance of a vector $\vec{x}$ from the threshold set $T(f, r)$, denoted as $dist(\vec{x}, f, r)$, be the minimum distance of $\vec{x}$ from any point in $T(f, r)$ i.e.,

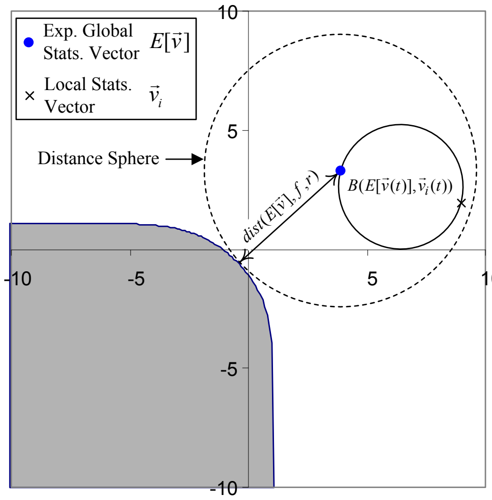
$$dist(\vec{x}, f, r) = \min(\|\vec{y} - \vec{x}\| \,|\, f(\vec{y}) = r)$$

The farther the estimate vector is, at a given time, from the threshold set, the more the local statistics vectors can change without violating local constraints. Therefore, a greater average distance of the estimate vector from the threshold set will result in a greater reduction in communications.

The average distance of the estimate vector from the threshold set is affected by many parameters. To begin with, it is affected by the coloring induced by the monitored function and the threshold value. Figure 2 illustrates the coloring induced by two sets of a monitored function and a threshold value. Figure 2(a) illustrates the coloring induced by the function $f_1(x, y) = \sin(2\sqrt{(x^2 + y^2)})$ and the threshold value 0, and Figure 2(b) illustrates the coloring induced by the function $f_2(x, y) = \frac{1}{1+e^{-x}} + \frac{1}{1+e^{-y}}$ and the threshold value 0.75 ($f_2(x, y)$ is a simple two-layer neural net).

It is clear that the distance of any point in $\mathbb{R}^2$ from the threshold set defined by f_1 and 0 cannot be greater than $\frac{\pi}{4} \approx 0.785$. Therefore, the average distance of the estimate vector from the threshold set in this case is bounded from above by 0.785, thus yielding a relatively low reduction in communications when monitored by our algorithms. However, the maximum distance of a point in $\mathbb{R}^2$ from the threshold set defined by f_2 and 0.75 is unbounded, and the dominating factor affecting the performance of our algorithms in this case is the nature of the data received on the streams.

In order to analyze how our algorithms are affected by the nature of the data on the streams, we consider periods during which this data is stationary (this fact, however, is



Exp. Global Stats. Vector $E[\vec{v}]$
Local Stats. Vector $\vec{v}_i$
Distance Sphere
$dist(E[\vec{v}], f, r)$
$B(E[\vec{v}(t)], \vec{v}_i(t))$
10
5
-10
-5
5
10
-5
-10

Figure 3: Depicts the coloring induced by the function $f_2 = \frac{1}{1+e^{-x}} + \frac{1}{1+e^{-y}}$ and the threshold value 0.75, together with the expected global statistics vector, its distance from the threshold set, and a local statistics vector, that is contained within the distance sphere. One can see that the ball $B(E[\vec{v}(t)], \vec{v}_i(t))$ is fully contained in the distance sphere.

not known to any of the nodes). More formally, we assume that each stream item is a d-dimensional vector, where the j^{th} component is independently drawn from a random variable denoted by X_j with a defined expectancy and variance, denoted by $E[X_j]$ and $V[X_j]$ respectively. We assume the system consists of n nodes, and that each node holds a sliding window of N items. We denote the last N items received on the stream monitored by p_i as $\vec{v}_{i,1}$,$\vec{v}_{i,2}$,...,$\vec{v}_{i,N}$, and the components of a vector as follows: $\vec{v}_{i,k} = (v_{i,k}^{(1)}, v_{i,k}^{(2)}, ..., v_{i,k}^{(d)})$. The local statistics vector held by a node is the average of the items contained in its sliding window, and the global statistics vector is the average of the items contained in the sliding windows held by all the nodes, i.e.,

$$\vec{v}_i(t) = \frac{1}{N}\sum_{k=1}^{N} \vec{v}_{i,k} \ \ ; \ \ \vec{v}(t) = \frac{\sum_{i=1}^{n}\sum_{k=1}^{N} \vec{v}_{i,k}}{N \cdot n}$$

It is easy to see that the expected value for the global statistics vector and each local statistics vector is $E[\vec{v}(t)] = E[\vec{v}_i(t)] = (E[X_1], E[X_2], ..., E[X_d])$.

Figure 3 depicts the coloring induced by f_2 and the threshold value 0.75, the expected global statistics vector, and the distance of the expected global statistics vector from the threshold set. The expected global statistics vector and its distance from the threshold set define a sphere called the distance sphere. We denote the distance of the expected global statistics vector from the threshold set by D_{global} i.e., $D_{global} = dist(E[\vec{v}(t)], f, r)$.

We present the following simplified model of our algorithms: we assume that the estimate vector holds the value of the expected global statistics vector, i.e., $\vec{e}(t) = E[\vec{v}(t)] = (E[X_1], E[X_2], ..., E[X_d])$. Furthermore, we assume that data on each stream arrives in blocks of N items. We would like to bound $\Pr_{violation}$, the probability that the arrival of a new block of data items on a stream will cause a constraint violation at the node monitoring the stream.

We assume that the estimate vector is the expected global statistics vector. Consequently, as long as the local statistics vector held by a node is contained within the distance sphere, the constraint checked by the node is guaranteed not to be violated. That is, if $\|E[\vec{v}(t)] - \vec{v}_i(t)\| < D_{global}$, then $B(E[\vec{v}(t)], \vec{v}_i(t))$ is fully contained in the distance sphere (see Figure 3) .

Therefore, the probability that a constraint violation will occur at a node is less than the probability that the local statistics vector held by the node will not be contained in the distance sphere. Using the Markov Inequality we obtain:

$$\Pr_{violation} \ \ \leq \ \ \frac{\sum_{i=1}^{d} V[X_i]}{N \cdot (D_{global})^2}$$

The proof is omitted due to lack of space. If the components of the data vectors are bounded between 0 and 1 – as happens in the important case in which they represent probabilities of terms to appear in a document – the Hoeffding bound can be used, to show that:

$$\Pr_{violation} \leq \exp\left(-2\left(D_{global}^2 - \frac{\sum_{i=1}^{d} V[X_i]}{N}\right)^2 / d\right)$$

Both bounds decrease quickly when D_{global} increases. This suggests that for data mining applications, features with a small information gain will not cause many constraint violations at any node, since their D_{global} is large. This is practically important, since usually most of the candidate features have a rather small information gain, and thus the proposed algorithm will considerably reduce communication. This is supported by the experimental results presented in the next section.

7. EXPERIMENTAL RESULTS

We performed several experiments with the decentralized algorithm. We tested the algorithm in a distributed feature selection setup. We used the Reuters Corpus (RCV1-v2) [24] in order to generate a set of data streams. RCV1-v2 consists of 804414 news stories, produced by Reuters between August 20, 1996, and August 19, 1997. Each news story, which we refer to as a document, has been categorized according to its content, and identified by a unique document id.

RCV1-v2 has been processed by Lewis, Yank, Rose, and Li [16]. Features were extracted from the documents, and indexed. A total of 47236 features were extracted. Each document is represented as a vector of the features it contains. We refer to these vectors as feature vectors. We simulate n streams by arranging the feature vectors in ascending order (according to their document id), and selecting feature vectors for the streams in a round robin fashion.

In the original corpus each document may be labeled as belonging to several categories. The most frequent category documents are labeled with is “CCAT” (the “CORPORATE/INDUSTRIAL” category). In the experiments our goal is to select features that are most relevant to the

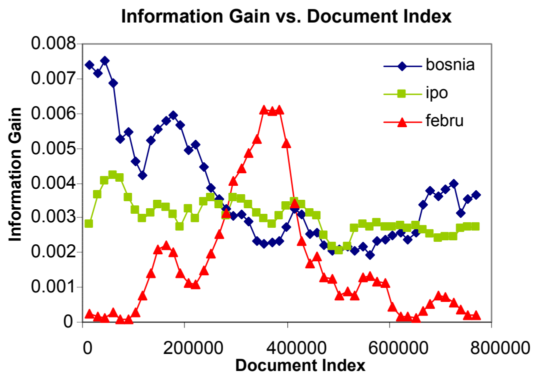


Figure 4: Information gain for the features "bosnia", "ipo", and "febru" as it evolves over the streams. The information gain for the feature "bosnia" displays a declining trend as the stream evolves. The information gain for the feature "ipo" remains relatively steady, while the information gain for the feature "febru" peaks about halfway through the stream.

"CCAT" category, therefore each vector is labeled as positive if it is categorized as belonging to "CCAT", and negative otherwise.

Unless specified otherwise, each experiment was performed with 10 nodes, where each node holds a sliding window containing the last 6700 documents it received. In each experiment we used the decentralized algorithm in order to detect for each feature, at any given time, whether its information gain in above or below a given threshold value. At any given time the information gain of a feature is based on the documents contained at the time in the sliding windows of all the nodes.

The experiments were designed to explore several properties of the algorithm. We were interested in determining how various parameters of the monitoring task affect the performance of the algorithm. The parameters of the monitoring task can be divided into characteristics of the monitoring task, and tunable parameters. The characteristics of the monitoring task include the number of streams to be monitored, and the desired threshold value. Tunable parameters include the size of the sliding window used by each node, and the permitted error margin. In addition we were interested in examining the behaviour of the algorithm when used for simultaneously monitoring several features.

In order to examine the effect of the various parameters on the performance of the algorithm, we chose three features that display different characteristic behaviour. The chosen features are "bosnia", "ipo", and "febru". Figure 4 depicts how the information gain for each feature evolves over the streams. The information gain for the feature "bosnia" displays a declining trend as the stream evolve. The information gain for the feature "ipo" remains relatively steady, while the information gain for the feature "febru" peaks about halfway through the stream.

We start by examining the influence of the characteristics of the monitoring task on the performance of the algorithm. Figure 5 show the number of broadcasts produced when monitoring each one of the features for threshold values ranging from 0.00025 to 0.006. In addition the cost

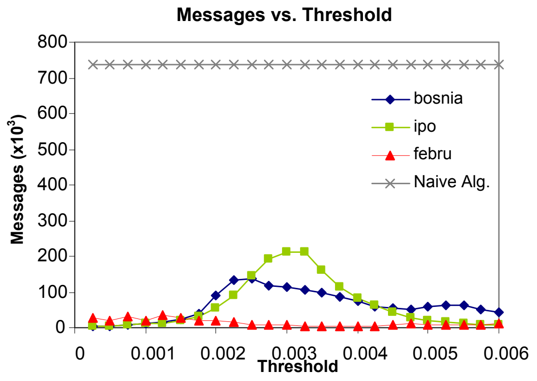


Figure 5: Number of broadcasts produced in order to monitor each feature as a function of the threshold value. In addition, the cost incurred by monitoring a feature by a naive algorithm is plotted. Even for adverse threshold values our algorithm performs significantly better than the naive algorithm.

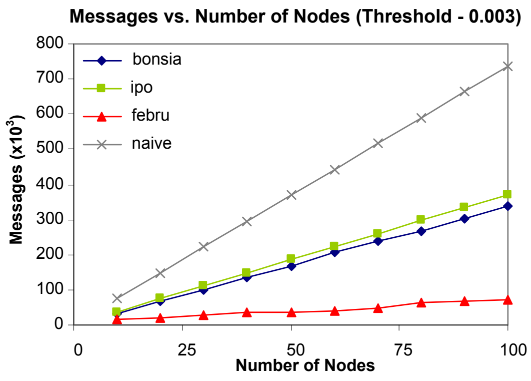
incurred by the naive algorithm is plotted, i.e., the number of messages required for collecting all the data to a central location. One can notice that even for adverse threshold values, the algorithm incurs a significantly lower communication cost than the cost incurred by the naive algorithm.

In order to check the effect the number of nodes has on the performance of the algorithm, we performed the following experiment: the stream of documents was divided in advance into 100 sub-streams in a round robin fashion. Simulations were run with the number of nodes ranging from 10 to 100. In a simulation consisting of n nodes, the first n sub-streams were used. This methodology ensures that the characteristics of the streams remain similar when simulating different numbers of nodes. Each node held a sliding window of 670 items.

Obviously, increasing the number of nodes will increase the number of broadcasts required in order to perform the monitoring task. Since the nodes in our experiment receive streams with similar characteristics, we expect that the number of broadcasts will increase linearly.

Two sets of simulation were run, the first with a threshold value of 0.003, and the second with a threshold value of 0.006. The results are plotted in Figure 6. Both graphs show that the number of broadcasts increases linearly as more nodes are added. Comparing the two graphs reveals that the number of broadcasts increases more moderately when using a threshold value of 0.006. This is due to the fact that as indicated in Figure 4, the average information gain on the monitored features is closer to 0.003.

Next we performed two experiments in order to evaluate the effect of tunable parameters on the performance of the algorithm. We performed the following experiments on the three features: for each feature we chose the threshold value that incurred the highest communication cost (0.0025 for "bosnia", 0.003 for "ipo", and 0.00125 for "febru"). We ran a set of simulations on each feature, using error margin values ranging from 0 to 50 percent of the threshold value. Then we ran an additional set of simulations for each feature, setting the size of the sliding window used by each node to values ranging from 6700 items to 13400 items. The results of these experiments are plotted in Figure 7. The results



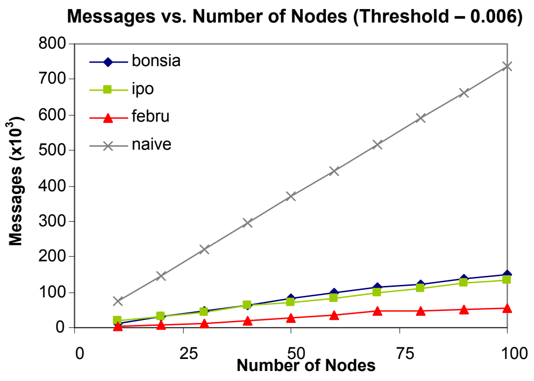
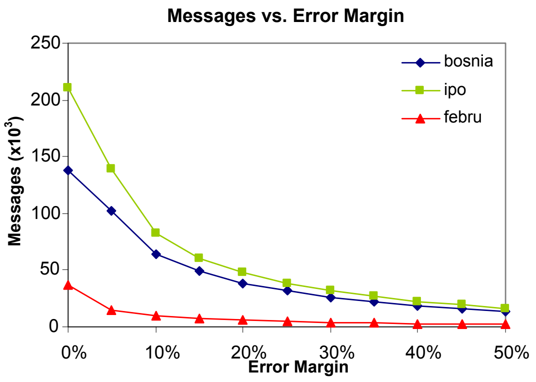


Figure 6: Number of messages produced in relation to the number of nodes. The number of messages increases linearly as the number of nodes increases, indicating that the algorithm scales well. Since the average information gain of all the features is closer to 0.003 than to 0.006, the number of messages increases more moderately when using a threshold value of 0.006.



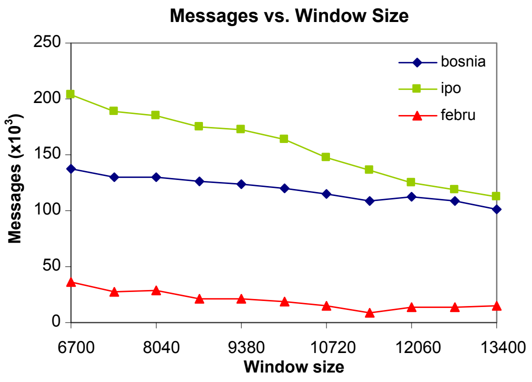


Figure 7: The influence of tunable parameters on performance. Increasing the error margin is more effective in reducing the communication load than increasing the window size. Using an error margin as small as 5 percent significantly reduces the communication load.

indicate that increasing the error margin is very effective in reducing the communication load. Using an error margin as small as 5 percent significantly reduces the communication load. Increasing the window size also reduces the communications load. The effect of increasing the window size is most evident for the feature “ipo”, which incurs the highest communication cost among the three features. In general, increasing the window size has a greater effect the closer the information gain of feature is to the threshold value.

Finally, we checked the performance of the algorithm when simultaneously monitoring multiple features. As the number of features that are monitored simultaneously increases, the probability that a constraint on one of the features will be violated when a new data item is received increases as well. Furthermore, a constraint violation can cause a cascading effect. A constraint violation for a feature at one of the nodes causes all the nodes to calculate a new estimate vector for the feature. Since the value of the estimate vector for the feature has changed, the constraint for the feature may be violated at additional nodes, causing these nodes to broadcast. The purpose of this experiment is to determine the number of simultaneous features the algorithm can monitor while remaining efficient, i.e., incurring a cost that is lower than the cost incurred by the naive algorithm. The experiment consisted of a series of simulations, using a threshold value of 0.001. In each simulation a number of features were selected randomly. Simulations were run with the number of features ranging from 1 to 5000. The results of this experiment are plotted in Figure 8. In addition the cost incurred by the naive algorithm is plotted.

The results indicate the algorithm remains efficient when simultaneously monitoring several thousands of features, but is inefficient when simultaneously monitoring more than about 4500 features.

8. CONCLUSION

Monitoring streams over distributed systems is an important challenge which has a wide range of applications. Scalability and efficiency of proposed solutions strongly depend on the volume and frequency of communication operations. However, despite the amount of work that was invested in this direction, most of the efficient solutions found in the literature can only be applied to simple aggregations or to linear functions. Most probably the reason is that when the function is non linear, effects seen in one – or only a few – of the streams may often turn out to be misleading with regards to the global picture.

In this work we proposed a solution through a general framework for monitoring arbitrary threshold functions over

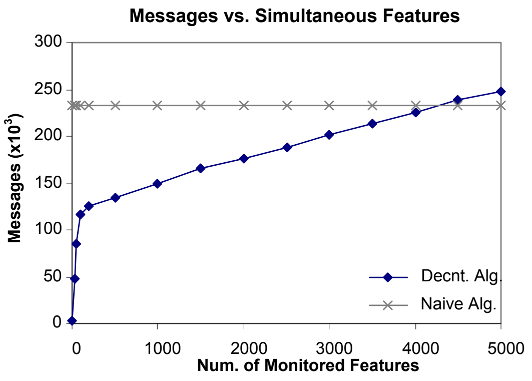


Figure 8: Number of messages in relation to the number of simultaneously monitored features. Our algorithm remains efficient when simultaneously monitoring up to about 4500 features.

a system of distributed streams. The evaluation of this approach using real-life data, applied to the information gain function, reveals that it is highly effective in reducing communication frequency.

Immediate future work will concentrate on developing methods to fine-tune various parameters (window size, error margin, threshold) in order to find an optimal trade-off between communication load and accuracy. We also plan to try and characterize families of functions for which the algorithm is more efficient.

9. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC '96*, pages 20–29, New York, NY, USA, 1996. ACM Press.

[2] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS '04*, pages 286–296, New York, NY, USA, 2004. ACM Press.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS '02*, pages 1–16, New York, NY, USA, 2002. ACM Press.

[4] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD '03*, pages 28–39, New York, NY, USA, 2003. ACM Press.

[5] S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, 2001.

[6] L. Berkovitz. *Convexity and Optimization in Rn.* Wiley, 2002.

[7] A. Bulut, A. K. Singh, and R. Vitenberg. Distributed data streams indexing using content-based routing paradigm. In *IPDPS*. IEEE Computer Society, 2005.

[8] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring streams - a new class of data management applications. In *VLDB*, 2002.

[9] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *ICALP '02*, pages 693–703, London, UK, 2002. Springer-Verlag.

[10] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable Distributed Stream Processing. In *CIDR 2003*, Asilomar, CA, January 2003.

[11] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. In *SIGMOD '05*, pages 25–36, New York, NY, USA, 2005. ACM Press.

[12] G. Cormode, R. Keralapura, and J. Ramimirtham. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD '06*, 2006.

[13] M. Dilman and D. Raz. Efficient reactive monitoring. In *INFOCOM*, pages 1012–1019, 2001.

[14] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *SPAA '01*, pages 281–291, New York, NY, USA, 2001. ACM Press.

[15] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA '02*, pages 63–72, New York, NY, USA, 2002. ACM Press.

[16] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[17] L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):610–628, 1999.

[18] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *ICDE '02*, page 555, Washington, DC, USA, 2002. IEEE Computer Society.

[19] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *SIGMOD '02*, pages 49–60, New York, NY, USA, 2002. ACM Press.

[20] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *ICDE '05*, pages 767–778, Washington, DC, USA, 2005. IEEE Computer Society.

[21] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.

[22] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *CIDR*, pages 245–256, Asilomar, California, Jan. 2003.

[23] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD '03*, pages 563–574, New York, NY, USA, 2003. ACM Press.

[24] T. Rose, M. Stevenson, and M. Whitehead. The Reuters Corpus Volume 1 - from Yesterday's News to Tomorrow's Language Resources. In *LREC-02*, Las Palmas de Gran Canaria, May 2002.

[25] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In *SIGMOD '92*, New York, NY, USA, 1992. ACM Press.

[26] B.-K. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for

co-evolving time sequences. In *ICDE '00*, page 13, Washington, DC, USA, 2000. IEEE Computer Society.

[27] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002.