Efficient Detection of Distributed Constraint Violations

Shipra Agrawal[†] Supratim Deb[‡] K. V. M. Naidu[‡] Rajeev Rastogi[‡]

[†]Dept. of Computer Science
Stanford University
Stanford, CA, USA
shipra@cs.stanford.edu

[‡]Bell Labs Research
Bangalore, INDIA
{supratim, naidukvm,
rastogi}@lucent.com

Abstract

In many distributed environments, the primary function of monitoring software is to detect *anomalies*, that is, instances when system behavior deviates substantially from the norm. Existing approaches for detecting such abnormal behavior record system state at all times, even during normal operation, and thus incur wasteful communication overhead. In this paper, we propose communication-efficient schemes for the anomaly detection problem, which we model as one of detecting the violation of global constraints defined over distributed system variables. Our approach eliminates the need to continuously track the global system state by decomposing global constraints into local constraints that can be checked efficiently at each site. Only in the occasional event that a local constraint is violated, do we resort to more expensive global constraint checking. We formulate the problem of selecting local constraints as an optimization problem that takes into account the frequency distribution of individual system variables, and whose objective is to minimize communication costs. After showing the problem to be NP-hard, we propose approximation algorithms for computing *provably near-optimal* (in terms of the number of messages) local constraints. In our experiments with real-life network traffic data sets, we found that our techniques for detecting global constraint violations can reduce message communication overhead by as much as 70% compared to existing data distribution-agnostic approaches.

1 Introduction

With the proliferation of large-scale distributed systems (e.g., peer-to-peer systems [22, 26], sensor networks [11, 23, 25]), *monitoring applications* are increasingly required to handle hundreds of thousands of nodes with dynamically changing states. The research community has responded to these new requirements by developing algorithms for continuously tracking a wide range of statistics over distributed streams of values – these include sums and averages of numeric values [20], top-k values [3], set-expression cardinality [7], number of distinct values [12], quantiles [6], and joins [5].

International Conference on Management of Data
COMAD 2006, Delhi, India, December 14–16, 2006

Unfortunately, the above-mentioned algorithms are ill-suited for a vast majority of monitoring tasks. This is because a key goal of monitoring is to ensure smooth system operation by quickly identifying abnormal system behavior (e.g., overload conditions, DDoS attacks). For detecting abnormalities, the above algorithms that continuously record global system state at all times are an overkill, and lead to unnecessary communication even when all is well [14]. Ideally, we would like algorithms that incur very little or no communication when the system is operating normally, and only in the rare instances when system parameters get close to abnormal regions do they trigger message communication.

Anomaly detection problem. At a very high level, anomaly detection involves the identification of system states that deviate substantially from the norm. The norm is typically captured using a global constraint $\mathcal{G}$ defined over system variables at the geographically distributed sites. As long as $\mathcal{G}$ holds, the system is considered to be in a normal state. Thus, the anomaly detection problem can be stated as follows: *Identify all the instances when system variable values violate the global constraint* $\mathcal{G}$. We focus on solving this problem in the paper.

The ability to detect global constraint violations is a critical requirement for monitoring software in IP networks, P2P systems, sensor networks, and the Web. For instance, the following global constraint can be used to detect DDoS attacks in an IP network: the total TCP SYN packet rate for a destination observed across the network's edge routers does not exceed a specified limit. Similarly, for an Enterprise that is connected to the Internet via multiple links, if the cumulative traffic on the links exceeds a threshold, then this could be used to trigger actions like activating backup links or requesting additional link capacity (from the Service Provider). And for a Voice over IP call, QoS can be ensured using a global constraint that specifies that the sum of link delays observed at routers along the call path is at most 200 msec.

In P2P systems, if files beyond a certain size limit are exchanged between peer nodes, then system parameters may need to be modified to optimize overall performance. Similarly, in sensor networks (e.g., for environmental monitor-

ing), collecting every individual reading (e.g., of temperature or humidity), besides being too expensive, may also be unnecessary; only extreme sensor readings that are either too low or too high may be of interest. And finally, for Web sites that are replicated at geographically distributed servers, knowledge of popular pages (for whom the overall number of hits exceeds a threshold) can be crucial for load balancing, charging for advertisements, etc.

The above-mentioned applications impose the following two requirements on our constraint violation detection algorithms:

1. *Real-time detection.* For effective problem resolution, global constraint transgressions should be detected in a timely manner.

2. *Low communication overhead.* In general, it is desirable to keep monitoring traffic as low as possible, since this opens up more bandwidth for user applications. This is especially important in wireless networks, where links typically have low bandwidths, and in sensor networks, where nodes have limited battery lives. Ideally, our schemes should transmit messages only when global constraints are in danger of being breached.

Brute force solutions. Traditional monitoring systems are based on *polling*. A central coordinator periodically polls system variables distributed across the various sites, and checks to see if $\mathcal{G}$ is violated. The problem with such a polling-based approach is that timely detection of global constraint violations will require frequent polling at short intervals, which in turn will lead to high message overhead. An alternative to polling is to continuously track the statistics for $\mathcal{G}$ using recently proposed data streaming algorithms for distributed environments. However, these too can result in a large number of message transmissions.

A major drawback of the above brute force schemes for checking the validity of global constraint $\mathcal{G}$ is that they are oblivious of the system state with respect to $\mathcal{G}$. Essentially, the schemes incur the same communication overhead irrespective of how close the system variables are to violating $\mathcal{G}$.

Our approach. Our approach is based on the following two key observations:

1. *A local constraint involving only system variables at a site can be checked locally and efficiently at the site without any message communication.*

2. *A large class of global constraints $\mathcal{G}$ that occur in practice can be decomposed into local constraints $\mathcal{L}_1, \ldots, \mathcal{L}_n$ such that if $\mathcal{G}$ is violated, then one of $\mathcal{L}_1, \ldots, \mathcal{L}_n$ must also be violated.*

For example, the global constraint $X_1 + X_2 \leq 5$ defined over system variables X_1 and X_2 can be decomposed into the following two local constraints: $\mathcal{L}_1 = X_1 \leq 2$ and $\mathcal{L}_2 = X_1 \leq 3$. It is easy to see that $(\mathcal{L}_1 \wedge \mathcal{L}_2) \rightarrow \mathcal{G}$.

The implication here is that as long as the local constraints $\mathcal{L}_i$ are satisfied, there is no need to check $\mathcal{G}$ since it must hold as well. Only in the event of an occasional violation of a local constraint $\mathcal{L}_i$ does the validity of $\mathcal{G}$ need to be ascertained. Thus, in our scheme, each site is responsible for continuously and locally monitoring its own local constraint $\mathcal{L}_i$. In case a site detects a local constraint violation, it triggers global constraint checking using one of the brute force solutions (based on either polling or continuous tracking) described earlier.

The bottom line here is that most of the time, system variables will be well within the bounds specified by $\mathcal{G}$, and so local constraints will hold and there will be no need for message transmissions to check $\mathcal{G}$. Only in the rare cases when the system state takes on extreme values, will local constraints be violated, causing additional communication to check $\mathcal{G}$. Thus, our approach can lead to dramatically lower message overhead compared to brute force solutions.

Our contributions. A primary contribution of this paper is a general framework for detecting global constraint violations. While the idea of using local constraints to detect global constraint violations is not new [9, 14, 16, 24], many of the existing schemes [9, 14, 16] consider very specific forms of global and local constraints. In contrast, our framework allows for arbitrary global and local constraints, and only specifies the relationship that must hold between them. In addition, we make the following contributions.

- In much of the prior work [9, 14, 16], with the exception of [24], global constraints are restricted to be simple linear expressions over system variables. In contrast, we permit more general constraints containing aggregate operators MIN and MAX, as well as boolean conjunctions and disjunctions, in this paper.
- There can be multiple possible local constraints for a given global constraint. We propose algorithms for selecting the "best" among these that minimize communication costs by taking into account the frequency distribution of individual system variables.

We show that it is possible to formulate the problem of selecting the best local constraints (that minimize communication costs) as an optimization problem whose objective function aims to maximize the frequency of occurrence of system states covered by the local constraints. We estimate the state occurrence frequencies using the frequency distributions of individual system variables (assuming variable independence), and show that the corresponding optimization problem is NP-hard. After proposing a pseudo-polynomial time algorithm for optimally computing local constraints, we go on to devise an FPTAS[1] that yields *provably near-optimal* local constraints.

- Our experimental results with a real-life network traffic data set demonstrate the effectiveness of our local constraint selection algorithms. Compared to naive approaches that assume a uniform data distribution, our techniques that take into account the frequency distribution of system variables reduce message communication overhead (due to local threshold violations) by as much as 70%.

[1] A *Fully Polynomial-Time Approximation Scheme* (FPTAS) is an approximation algorithm which (1) for a given $\epsilon > 0$, returns a solution whose cost is within $(1 \pm \epsilon)$ of the optimal cost, and (2) has a running time that is polynomial in the input size and $1/\epsilon$.

Organization of the paper. Section 2 compares our proposed schemes with related work. In Section 3, we describe the local constraint selection problem that is tackled in this paper. We first present schemes for computing local constraints when global constraints are simple linear expressions in Section 4. We then describe extensions to our schemes for handling more complex constraints containing MIN, MAX, conjunctions and disjunctions in Section 5. In Section 6, we present the results of our experiments with real-life data sets. And finally, Section 7 contains concluding remarks.

2 Related work

Aggregate statistics computation over distributed data streams has attracted considerable attention in the recent past. Algorithms have been proposed for tackling a broad array of problems, including computing the top-k set [3], sums and counts [20], set expressions [7], number of distinct values [12], quantiles [6], and joins [5]. However, these data streaming algorithms track system state continuously, and are thus too expensive for detecting constraint violations.

There has been some previous work [10, 15, 19, 29] on intelligent polling-based solutions for monitoring global anomaly conditions – these try to reduce unnecessary polling by exploiting past statistics. But a shortcoming of pure polling-based approaches is that they may miss constraint violations unless the polling interval is set to be small enough. On the other hand, our approach combines local-event-reporting or 'triggering' with polling to ensure guaranteed detection of every alarm condition.

Constraint checking is performed today by implementations of embedded *triggers* or predicates in database query engines. The topic has also been extensively researched in the context of so-called "active databases"[28]. However, the focus of research on distributed active databases has thus far been primarily on how to decompose the triggering rules and distribute them across sites so that they can be evaluated correctly. To the best of our knowledge, none of the prior work in this field addresses the problem of evaluating a distributed triggering condition with minimal communication overhead.

The works most closely related to our approach are those of [9, 14, 16, 24]. The pioneering work of [9] proposed the idea of using local constraints for monitoring global constraints, and presented a simple solution for selecting local constraints assuming uniform data distributions for system variables. In [14], the authors discuss the research challenges in building a distributed triggering mechanism to maintain system-wide invariants. And more recently, Keralapura et al. [16] studied the problem of monitoring *thresholded counts* with bounded error, but only when their value exceeds a pre-specified threshold. To detect instances when the distributed count breaches the threshold (so that accurate tracking of it can be begun), they propose an adaptive algorithm that dynamically adjusts local constraints each time there is a violation.

The recent work of Sharfman et al. [24] represents the state-of-the-art in detecting distributed constraint violations. In [24], global constraints are permitted to contain an arbitrary function over weighted sums of system variables. A geometric approach is proposed which allows each site to constrain the possible values for the weighted sums based on their locally observed values. Thus, each site can locally determine if the global constraint is potentially violated, and if so, initiate local constraint adjustment by balancing local variable values across the sites. A key difference with our work is that [24] does not take into account the data distribution of system variables.

In summary, our effort represents the first attempt at formulating local constraint selection as a combinatorial optimization problem that (1) incorporates the knowledge of system variable data distributions, and (2) seeks to minimize communication costs.

3 Problem Definition

3.1 System Model

Our distributed system consists of n remote sites $1, \ldots, n$ and a central coordinator site 0. Each remote site i contains a variable X_i which takes non-negative integer values from the domain $[0, M_i]$. Intuitively, the integer values taken by X_i reflect a measurement of some aspect of site i's state; depending on the application, X_i could be any of the following: the SYN packet rate (packets/sec) for a specific destination seen at router i, the traffic (in bytes) on network link i, or the number of times a particular Web page is accessed at site i.

Global constraints. A global constraint $\mathcal{G}$ defined over variables X_i specifies the system states that are considered to be normal. Thus, anomalies correspond to violations of $\mathcal{G}$. In its full generality, $\mathcal{G}$ is an arbitrary boolean expression over *atomic conditions* connected using conjunctions ($\wedge$) and disjunctions ($\vee$). Each atomic condition has the form $agg\_exp\ op\ T$, where agg_exp is an expression involving X_i and zero or more aggregate operators MAX, MIN and SUM, op is one of $\geq$ or $\leq$, and T is an arbitrary integer constant. The aggregate expression agg_exp is defined recursively as follows: (1) each term A_iX_i is an aggregate expression – here A_i is an arbitrary integer constant, (2) if a_1 and a_2 are aggregate expressions, then so are SUM$\{a_1, a_2\}$ (or $a_1 + a_2$) and MIN/MAX$\{a_1, a_2\}$.

Note that the syntax of our global constraint $\mathcal{G}$ is powerful enough to express anomaly conditions for most practical applications (including those mentioned in Section 1). Furthermore, it can also capture SQL queries containing the standard aggregate operators like SUM, COUNT, MAX, MIN and AVG. An example global constraint is: $((3X_1 + X_2 \geq 1) \vee (\text{MIN}\{X_1, 2X_3 - X_2\} \leq 5)) \wedge (X_1 + \text{MAX}\{3X_2, X_3\} \geq 4)$.

For simplicity, in the remainder of the paper, we will assume that A_i and T are positive integers, and that op

is $\leq$. Our proposed techniques can be extended to handle scenarios when this assumption may not hold. Furthermore, in the remainder of this Section (and the next two Sections), we will only consider simple constraints of the form $\sum_i A_i X_i \leq T$. In Section 5, we will show how our schemes for handling these simple types of constraints can be extended to deal with general, more complex global constraints (containing MIN, MAX, conjunctions and disjunctions).

Local constraints. Our goal is to devise solutions that would enable the coordinator site to detect global constraint violations in a timely manner and with minimal message overhead. As described earlier in Section 1, we achieve this by installing at each site i a local constraint $\mathcal{L}_i$ of the form $X_i \leq T_i$, where T_i is a local threshold value. Each site i locally and continuously checks constraint $\mathcal{L}_i$, and sends an alarm to the coordinator every time it detects a violation of the local constraint. On receiving the alarm, the coordinator starts tracking the quantity $\sum_i A_i X_i$ to check if the global constraint $\mathcal{G}$ is also violated. It does this using either continuous polling or the algorithms of Olston et al. [20] for estimating $\sum_i A_i X_i$ with a very small relative error ϵ.

The coordinator continuously tracks $\sum_i A_i X_i$ only as long as at least one of the local constraints is violated. Thus, in order to ensure that no global constraint violation goes undetected, we require the local constraints to satisfy the following *covering property*.

$$(\mathcal{L}_1 \wedge \mathcal{L}_2 \wedge \cdots \mathcal{L}_n) \rightarrow \mathcal{G}$$

The covering property ensures that if $\mathcal{G}$ is violated, then some $\mathcal{L}_i$ will be violated as well, thus causing the global constraint to be checked at the coordinator, and its violation to be detected.

3.2 Local Threshold Selection Problem

A key question that we need to answer is: what values should we choose for the local thresholds T_i? We focus on answering this question for a given global threshold T.

For the covering property to be satisfied, we require the thresholds T_i to satisfy $\sum_i A_i T_i \leq T$. Furthermore, it is straightforward to see that if $\sum_i A_i T_i \leq T$, then the covering property is indeed satisfied.

Now, for a given T, there may be many choices for T_i that satisfy the covering property. For example, consider the global constraint $\mathcal{G} = X_1 + 2X_2 \leq 5$. Then, both $(T_1, T_2) = (1, 2)$ and $(T_1, T_2) = (3, 1)$ satisfy the covering property. So the question is which local threshold values should we choose? Is there a way to determine if one set of threshold values is better than another?

The answer to the above questions lies in the observation that not all local constraint violations correspond to global constraint violations. Basically, there can be *false alarms* where a local constraint is violated, but the global constraint still holds. For example, for the $\mathcal{G}$ above, and local thresholds $(T_1, T_2) = (1, 2)$, if the system enters a global state in which $(X_1, X_2) = (2, 1)$, then the local constraint $X_1 \leq 1$ at site 1 will be violated even though (X_1, X_2) does not violate the global constraint $X_1 + 2X_2 \leq 5$.

Every time a local constraint is violated, irrespective of whether or not it is a false alarm, the coordinator and remote sites exchange messages to track the quantity $\sum_i A_i X_i$. Thus, to minimize communication overhead, we should select local thresholds such that the local constraint violation frequency is minimized. (Note that, due to the covering property, this is equivalent to minimizing the number of false alarms. Although it may be impossible to completely eliminate false alarms since each site only knows the value of its own local variable X_i.)

Let $\mathcal{L} = \wedge_i \mathcal{L}_i$. Thus, we are looking to minimize the number of instances when $\mathcal{L} = false$, or alternately, to maximize $P(\mathcal{L} = true)$, the probability that local constraints hold. To compute $P(\mathcal{L} = true)$, we need to introduce additional notation. Let $\mathbf{v} = [v_1, \ldots, v_n]$ denote the vector of values taken by variables $X_1, \ldots, X_n$. Further, let f denote the joint frequency distribution of variables X_i; thus, for $\mathbf{v} = [v_1, \ldots, v_n]$, $f(\mathbf{v})$ is the frequency with which the variables $X_1, \ldots, X_n$ take the values $v_1, \ldots, v_n$, respectively. We will say that $\mathbf{v}$ satisfies $\mathcal{L}$ if $\mathcal{L}$ evaluates to $true$ when v_i is substituted for X_i. Then $P(\mathcal{L} = true) = \frac{\sum_{\mathbf{v} \text{ satisfies } \mathcal{L}} f(\mathbf{v})}{\sum_{\mathbf{v}} f(\mathbf{v})}$.

Unfortunately, the joint frequency distribution f may not be known. We could try to compute a multi-dimensional histogram for the X_i variables based on past observations and use this to estimate f – however, constructing multi-dimensional histograms is in general a difficult problem [21, 27], and may require too much communication in a distributed setting.

So instead, we consider a more practical alternative from a computation perspective in which we estimate the frequency distribution f_i of each X_i. We then use this in conjunction with the assumption that the variables X_i are independent to derive an estimate for $P(\mathcal{L} = true)$. Now, let F_i be the cumulative frequency distribution for X_i, that is, $F_i(j) = \sum_{k \leq j} f_i(k)$. Then, for a vector $\mathbf{v}$, assuming variable independence, $\frac{f(\mathbf{v})}{\sum_{\mathbf{v}} f(\mathbf{v})} = \prod_{i=1}^{n} \frac{f_i(v_i)}{F_i(M_i)}$. Thus, we get that

$$P(\mathcal{L} = true) = \prod_{i=1}^{n} \frac{F_i(T_i)}{F_i(M_i)}$$

Thus, since $F_i(M_i)$s are constant, our local threshold selection problem is the following.

Problem statement: Given positive integers A_i and T, a cumulative frequency distribution function F_i for each variable X_i, compute positive integer values $T_1, \ldots, T_n$ such that

$$\prod_{i=1}^{n} F_i(T_i) \text{ is maximum, and}$$
$$\sum_{i=1}^{n} A_i T_i \leq T \qquad \square$$

Note that we can estimate each f_i by maintaining concise histograms of X_i values locally. Since each histogram

is for one-dimensional values at a single site, it can be constructed efficiently for a sequence of X_i values using centralized streaming algorithms described in [13], and for a recent window of values using the techniques of [8, 18].

Now, observe that the frequency distribution f_i may not be static and may vary with time. To deal with this, we can use the change detection algorithms of [17] to identify the points in time when the underlying distribution f_i has changed. Once a change is detected, we immediately trigger the recomputation of histograms for the stream of X_i values, and then use these to compute new local thresholds T_i. Thus, we can ensure that the threshold values T_i are always based on the most current data distribution f_i.

We should point out here that we do not expect threshold recomputations to occur very frequently, since in most applications, shifts in data distributions usually happen gradually rather than abruptly. For instance, in our experiments with real-life network traffic data sets, we found link traffic distributions to be fairly stable from week to week; essentially, link traffic histograms for one week were good predictors of link traffic distributions for the following weeks.

4 Threshold Selection

In this section, we present schemes for computing thresholds for local variables $X_1, X_2, \ldots, X_n$ assuming that each variable is statistically independent of the others. Recall from the previous section that the problem here is to compute integer threshold values $T_1, T_2 \ldots, T_n$ such that $\sum_{i=1}^{n} A_i T_i \leq T$ and $\prod_{i=1}^{n} F_i(T_i)$ is maximum.

We can devise a simple pseudo-polynomial time dynamic programming algorithm to optimally solve the above problem. Let $V_i(S)$ be defined as follows:

$$V_i(S) = \max\{\prod_{k=1}^{i} F_k(T_k) \ : \ \sum_{k=1}^{i} A_k T_k \leq S\} \qquad (1)$$

Then, $V_n(T)$ yields the optimal thresholds for our problem. The value of $V_n(T)$ can be computed recursively using the following relation:

$$V_i(S) = \max\{F_i(j) V_{i-1}(S - A_i j) : \ j \in [1, T/A_i]\} \qquad (2)$$

One can compute the value of $V_n(T)$ in $O(nT^2)$ operations, as there are $O(nT)$ values of $V_i(S)$ to be computed, and each computation involves computing a maximum over at most T/A_i elements. A problem here, however, is that the running time of the algorithm $O(nT^2)$ is not polynomial in the input size. Consequently, since T can be large in practice, such a pseudo-polynomial time algorithm may not really be practical.

Unfortunately, the following theorem implies that it may be difficult to devise an efficient polynomial time algorithm for the threshold selection problem. (Due to space constraints, we omit the proof of NP-hardness here. It can be found in [2].)

Theorem 1 *The local threshold selection problem is NP-hard.* □

In the following, we present an FPTAS for the problem, that is, an approximation algorithm that runs in time polynomial in the input size and approximates the optimal solution to within ϵ relative error for an arbitrarily small ϵ. Thus, our results are the *best possible* in terms of approximation guarantees.

We will like to make a brief comment here regarding the histograms for variables X_i that are used to estimate the distributions F_i. Although M_i, the size of X_i's domain, can be quite large in practice, the histograms themselves are rather coarse with very few buckets (much smaller than M_i). Thus, the input size is much smaller than M_i, and in order for the running times of algorithms to be polynomial in the input size, their time complexities will need to have a logarithmic as opposed to a polynomial dependence on M_i.

4.1 FPTAS

Intuition

We begin by exploring the basic intuition under the following simplistic assumption: for all i, j pairs, values of $F_i(j)$ are *integral* powers of a known $\alpha > 1$. Thus, $F_i(T_i)$ will correspond to α^{r_i} for some integer r_i, and maximizing $\prod_{i=1}^{n} F_i(T_i)$ will correspond to maximizing $\alpha^{\sum_{i=1}^{n} r_i}$. Let $I_i(r)$ denote the smallest value such that $F_i(I_i(r)) = \alpha^r$, if such a value exists, or is set to infinity otherwise. Replacing T_i by $I_i(r_i)$, the problem can now be reformulated as one of finding r_i's such that

$$\sum_{i=1}^{n} r_i \text{ is maximum, and}$$

$$\sum_{i=1}^{n} A_i I_i(r_i) \leq T$$

Note that this is essentially a variant of the knapsack problem, and can be solved using dynamic programming [4]. The running time of the algorithm will depend on the total number of r_i's possible, which is polynomial in the input size, thus making it a polynomial time algorithm.

Now, if the values of F_is are not integral powers of α, then we will simply round down each $F_i(j)$ to the largest integral power of α smaller than $F_i(j)$, and then solve the problem. By doing so, intuitively, we will introduce an approximation factor of α for each i because our new F_i values are within an α factor of the original F_i value. This will introduce an approximation factor of α^n in the final solution. By choosing α appropriately, we can get as close to the optimal solution as desired.

Overview of the Algorithm

Let us assume that we have chosen an α to work with. The algorithm essentially has two phases, computing $I_i(r)$s and dynamic programming.

1. **Computing $I_i(r)$s:** $I_i(r)$ is computed as the smallest j such that $\alpha^r \leq F_i(j) \leq \alpha^{r+1}$. We choose the minimum such j because we are looking to minimize the weighted sum of T_is and such a choice is the best possible one. If no

such j exists, then we do not want the corresponding r to appear in the solution and hence we set $I_i(r)$ to infinity.

2. **Dynamic Programming:** Let us first introduce some notation. Let $\bar{P} = \prod_{i=1}^{n} F_i(M_i)$ denote the maximum possible value of the product of F_is. Let l denote the smallest power of α that is greater than or equal to $\bar{P}$ (thus, $l = \lceil \frac{\log(\bar{P})}{\log(\alpha)} \rceil$). Our dynamic programming algorithm builds a table with n rows and l columns whose entries are denoted by $S(i,p)$. Each $S(i,p)$ corresponds to the minimum value for $\sum_{j=1}^{i} A_j I_j(r_j)$ such that $\sum_{j=1}^{i} r_j = p$. Thus, we are interested in the largest p such that $S(n,p) \leq T$. Now, the table of $S(i,p)$ values can be constructed as follows:

1. $S(1,p) = A_1 I_1(p)$
2. $S(i+1,p) = \min_r \{A_{i+1} I_{i+1}(r) + S(i,p-r)\}$

To compute the solution to our problem, we first identify the largest p such that $S(n,p) \leq T$. The r_i values corresponding to this p value then give us our desired threshold values $T_i = I_i(r_i)$.

Analysis of the Algorithm

First, let us analyze the running time of the algorithm. Let $\bar{M} = \max\{M_1, \ldots, M_n\}$. In the first phase, a table of nl $I_i(r)$s is constructed. Each $I_i(r)$ value can be computed using binary search over the domain of X_i in $\log M_i$ steps. Thus, the running time for the first phase is $O(nl \log \bar{M})$. The second phase, dynamic programming, populates the nl $S(i,p)$ table entries. Computing each entry will take at most l computations because there are at most l possible values of r in step 2 above. Thus, the running time of the second phase is $O(nl^2)$. So the overall running time of the algorithm is $O(nl(l + \log \bar{M}))$.

Before analyzing the approximation guarantee provided by the algorithm, let us introduce some more notation. Let $T_1^*, T_2^*, ..., T_n^*$ denote optimal thresholds, and $r_1, r_2, \ldots, r_n$ denote the r_i values returned by our algorithm (recall that $I_i(r_i)$ are the threshold values T_i in our solution). Also, let r_i^* denote the largest r such that $I_i(r) \leq T_i^*$. By definition, this implies that $\alpha^{r_i^*} \leq F_i(T_i^*) \leq \alpha^{r_i^*+1}$. Now, the cost of our solution is $\prod_{i=1}^{n} F_i(I_i(r_i))$ and the cost of the optimal solution is $\prod_{i=1}^{n} F_i(T_i^*)$.

Any r_i will satisfy $\alpha^{r_i} \leq F_i(I_i(r_i)) \leq \alpha^{r_i+1}$, by definition. Using this, it is easy to see that cost of our solution, $\prod_{i=1}^{n} F_i(I_i(r_i)) \geq \prod_{i=1}^{n} \alpha^{r_i}$. Also, since $\alpha^{r_i^*} \leq F_i(T_i^*) \leq \alpha^{r_i^*+1}$, we have $\alpha^{r_i^*} \geq \frac{F_i(T_i^*)}{\alpha}$, which implies that $\prod_{i=1}^{n} \alpha^{r_i^*} \geq \frac{\prod_{i=1}^{n} F_i(T_i^*)}{\alpha^n}$.

Note that $\sum_{i=1}^{n} A_i I_i(r_i^*) \leq \sum_{i=1}^{n} A_i T_i^* \leq T$. Among all the r_is satisfying $\sum_{i=1}^{n} A_i I_i(r_i) \leq T$, our solution has the maximum $\sum_{i=1}^{n} r_i$, which implies that $\sum_{i=1}^{n} r_i \geq \sum_{i=1}^{n} r_i^*$. From these, we finally derive the following:

$$\prod_{i=1}^{n} F_i(I_i(r_i)) \geq \prod_{i=1}^{n} \alpha^{r_i} \geq \prod_{i=1}^{n} \alpha^{r_i^*} \geq \frac{\prod_{i=1}^{n} F_i(T_i^*)}{\alpha^n}$$

.

Since, T_i^* is the optimal solution, it follows that our algorithm approximates the optimal solution by a factor of α^n. Clearly, this gives an algorithm that runs in time $O(n\lceil \frac{\log(\bar{P})}{\log(\alpha)} \rceil (\lceil \frac{\log(\bar{P})}{\log(\alpha)} \rceil + \log \bar{M}))$ and provides an approximation guarantee of α^n.

Theorem 2 (FPTAS) *Let $\bar{P}$ denote the maximum possible product of F_i's and $\bar{M}$ denote the maximum domain size of X_i's. Then for any $\epsilon > 0$, our dynamic programming algorithm returns a $1+\epsilon$ approximation and has a running time of $O(\frac{n^2}{\epsilon} \log(\bar{P})(\frac{n \log(\bar{P})}{\epsilon} + \log \bar{M}))$.*

Proof: Choose $\alpha = 1 + \frac{\epsilon}{2n}$. Then the approximation factor of our algorithm becomes $(1+\frac{\epsilon}{2n})^n \leq 1 + \sum_{i=1}^{n} n^i (\frac{\epsilon}{2n})^i \leq 1 + \sum_{i=1}^{n} (\frac{\epsilon}{2})^i \leq \frac{1}{1-\frac{\epsilon}{2}} \leq 1 + \epsilon$.

Also, the running time of our algorithm is $O(nl(l + \log \bar{M}))$, where $l = \lceil \frac{\log(\bar{P})}{\log \alpha} \rceil$. Thus, substituting for $\log(\alpha) = log(1 + \frac{\epsilon}{2n}) \approx \frac{\epsilon}{2n}$ proves the theorem. □

5 Constraints Containing Boolean Conjunctions and Disjunctions

Let $E(X_1, \ldots, X_n)$ denote a linear expression of the form $\sum_i A_i X_i$. So far, we have considered simple global constraints of the form $E \leq T$ containing only a single linear expression. In this Section, we will consider more general global constraints that are conjunctions and disjunctions of the simpler constraints $E \leq T$. Specifically, we will consider global constraints $\mathcal{G}$ of the form $\bigwedge_j (\bigvee_k E_{j,k} \leq \hat{T}_{j,k})$, where $E_{j,k}$ is a linear expression over variables $X_1, \ldots, X_n$, and $\hat{T}_{j,k}$ is an integer threshold value. We will refer to these richer constraints as *boolean* constraints.

In the following, we will first show that our boolean constraints subsume the class of constraints constructed using the SUM, MIN, and MAX operators. We will then present an FPTAS for finding local threshold values T_i for global constraints containing only disjunctions. (Each local constraint $\mathcal{L}_i$ is still of the form $X_i \leq T_i$ as before.) For global constraints containing only conjunctions, we will prove that finding good approximations for threshold values T_i is NP-hard, which implies that the most general form we are considering is hard to approximate, too. Finally, we will propose heuristics to solve the general problem.

5.1 Constraints containing MIN/MAX

Consider any constraint of the form $agg\_exp \leq T$, where the aggregate expression agg_exp is either a linear expression (SUM) or constructed recursively using the operators MIN, MAX, or SUM (that is, if a_1 and a_2 are aggregate expressions, then so are $a_1 + a_2$ and MIN/MAX$\{a_1, a_2\}$). We show that these aggregate constraints can be expressed as boolean constraints (that is, all occurrences of MIN/MAX can be replaced with conjunctions and disjunctions). First, note that we can push all the SUM operators inside MIN/MAX. For instance, consider the constraint $(A + \text{MIN}\{B, C\}) \leq T$. This is equivalent to

MIN$\{A+B, A+C\} \leq T$. Thus, we now have constraints with MIN/MAX as the outermost operators. These can be replaced with conjunctions and disjunctions as follows. For instance, MIN$\{A, B\} \leq T$ can be rewritten as the boolean constraint $(A \leq T) \vee (B \leq T)$, and MAX$\{A, B\} \leq T$ can be transformed to $(A \leq T) \wedge (B \leq T)$. Note that these transformations can blow up the input constraint size exponentially, and so may not be feasible for constraints containing a large number of MIN/MAX operators.

5.2 Constraints containing only disjunctions

Now let us consider boolean constraints containing only disjunctions, that is, constraints of the form $\mathcal{G} = \bigvee_j (E_j \leq \hat{T}_j)$. We want to compute local thresholds T_i for each X_i such that the covering property is satisfied, that is, whenever $\mathcal{G}$ is violated, one of the $\mathcal{L}_i$s is also violated. This covering property implies that it is necessary and sufficient for the T_is to satisfy the constraint $\bigvee_j (E_j(T_1, \ldots, T_n) \leq \hat{T}_j)$. So the problem is to find threshold values T_i such that

$$\prod_{i=1}^{n} F_i(T_i) \text{ is maximum, and}$$
$$\bigvee_j (E_j(T_1, \ldots, T_n) \leq \hat{T}_j)$$

To solve the above problem, we compute thresholds for each disjunct $E_j \leq \hat{T}_j$ separately using our FPTAS (from Section 4) with approximation factor $1+\epsilon$. Let $T_{i,j}$ denote the ith threshold (for variable X_i) in our solution for disjunct $E_j \leq \hat{T}_j$. Further, let j^* be the disjunct j for which $\prod_i F_i(T_{i,j})$ is maximum. Then we choose the local threshold values $T_i = T_{i,j^*}$. In the following, we show that our solution is within a factor of $1+\epsilon$ of the optimal solution.

Lemma 3 *The thresholds $T_i = T_{i,j^*}$ are a solution to our threshold computation problem for constraints with only disjunctions. Moreover, if the optimal threshold values are T_i^*, then*

$$\frac{\prod_{i=1}^{n} F_i(T_i^*)}{1+\epsilon} \leq \prod_{i=1}^{n} F_i(T_{i,j^*}) \leq \prod_{i=1}^{n} F_i(T_i^*)$$

Proof: It is easy to see that T_{i,j^*} is a feasible solution to the problem at hand since it satisfies the disjunct $E_{j^*} \leq \hat{T}_{j^*}$. Also, $\prod_{i=1}^{n} F_i(T_{i,j^*}) \leq \prod_{i=1}^{n} F_i(T_i^*)$ is true because T_i^* is the optimal solution.

Next, we show that the second inequality also holds. First, observe that the optimal thresholds T_i^* must satisfy one of the disjuncts $E_j \leq \hat{T}_j$. Let this disjunct be j'. Also, let $T_{i,j'}^*$ denote the optimal solution for disjunct $E_{j'} \leq \hat{T}_{j'}$. This means that among all the thresholds for disjunct j', the thresholds $T_{i,j'}^*$ maximize the product of the F_is. Hence, $\prod_{i=1}^{n} F_i(T_{i,j'}^*) \geq \prod_{i=1}^{n} F_i(T_i^*)$ (in fact, the products are equal since T_i^* is optimal, too). Thus, we get that

$$\prod_{i=1}^{n} F_i(T_{i,j^*}) \geq \prod_{i=1}^{n} F_i(T_{i,j'}) \geq \frac{\prod_{i=1}^{n} F_i(T_{i,j'}^*)}{1+\epsilon} \geq \frac{\prod_{i=1}^{n} F_i(T_i^*)}{1+\epsilon}$$

The first inequality follows because such a product is maximum for disjunct $j = j^*$, the second inequality is true because our solution is a $1+\epsilon$ approximation to the optimal solution, and the third inequality holds because the optimal thresholds satisfy disjunct j'. This proves the lemma. □

Theorem 4 *There is an FPTAS for the problem of computing local thresholds for a global constraint containing only disjunctions over linear inequalities.*

Proof: From Lemma 3, it follows that our algorithm that returns thresholds $T_i = T_{i,j^*}$ gives a $1+\epsilon$ approximation to the optimal solution. Further, it simply runs the FPTAS (from Section 4) for each disjunct which amounts to m runs (m is the number of disjuncts) of the FPTAS whose time complexity is given in Theorem 2. Thus, its running time is polynomial in the input size and $\frac{1}{\epsilon}$. □

5.3 Constraints containing only conjunctions

Next, let us consider boolean constraints containing only conjunctions, that is, constraints of the form $\mathcal{G} = \bigwedge (E_j \leq \hat{T}_j)$. Our problem is to compute local thresholds T_i for each X_i such that

$$\prod_{i=1}^{n} F_i(T_i) \text{ is maximum, and}$$
$$\bigwedge_j (E_j(T_1, \ldots, T_n) \leq \hat{T}_j)$$

In [2], we prove the following theorem regarding the hardness of obtaining good approximations to the above problem.

Theorem 5 *For an arbitrary constant $p > 0$, the problem of finding local thresholds for a global constraint that is a conjunction of linear inequalities is hard to approximate within a factor of n^p unless P = NP.* □

Thus, it is hard to achieve a satisfactory approximation for our local threshold computation problem when the global constraint is a conjunction of linear inequalities. A simple heuristic that employs our FPTAS (from Section 4) is as follows. First, we compute thresholds for each conjunct $E_j \leq \hat{T}_j$ separately using our FPTAS. Let $T_{i,j}$ denote the ith threshold in our solution for conjunct $E_j \leq \hat{T}_j$. Now, we choose the local threshold values $T_i = \min_j \{T_{i,j}\}$.

It is easy to see that our threshold values will satisfy every conjunct $E_j \leq \hat{T}_j$, since we have chosen $T_i \leq T_{i,j}$ and the $T_{i,j}$s are a feasible solution for conjunct j. Also, note that there may still be scope for improving the thresholds since increasing some of the threshold values may still satisfy all the linear inequalities. One option here is to uniformly increase all the threshold values as long as no linear inequality is violated.

5.4 Handling general boolean constraints

Now, we are in a position to show how our threshold selection techniques from the previous two subsections can be used to handle general boolean constraints of the form $\bigwedge_j(\bigvee_k E_{j,k} \leq \hat{T}_{j,k})$. Note that since this is a generalization of the conjunction case discussed in Section 5.3, it is hard to approximate. Here, we describe a two-step heuristic for computing local thresholds T_i that utilizes our schemes from the previous two subsections.

• **Compute thresholds for disjunctions:** Each conjunct j is a disjunction of the form $\bigvee_k(E_{j,k} \leq \hat{T}_{j,k})$, and thus thresholds $T_{i,j}$ for each such disjunction can be computed using the method described in Section 5.2.

• **Compute thresholds for conjunction:** Next, we use the method from Section 5.3 to compute the final local thresholds T_i using the threshold values $T_{i,j}$ obtained in Step 1 for each conjunct j. Basically, for each i, choose $T_i = \min_j\{T_{i,j}\}$, and then increase the thresholds as long as they satisfy $\bigwedge_j(\bigvee_k E_{j,k} \leq \hat{T}_{j,k})$.

6 Performance Evaluation

In this Section, we quantitatively assess the utility of our FPTAS for selecting local threshold values using a real-life data set. Our findings indicate that, compared to simple heuristics for computing thresholds, our FPTAS consistently reduces message communication costs (due to local constraint violations) by a factor of 2-3.

6.1 Schemes Considered

Our global constraint $\mathcal{G}$ is of the form $\sum_i X_i \leq T$. Each local constraint $\mathcal{L}_i$ is of the form $X_i \leq T_i$. Whenever a local variable X_i exceeds its threshold T_i, the corresponding remote site triggers an alarm to the central coordinator, and the central coordinator issues a global poll. In addition to our FPTAS, we consider the following heuristics for selecting the local thresholds T_i.

FPTAS. We implement the FPTAS described in Section 4.1 with an approximation factor of 1.05 ($\epsilon = 0.05$). The cumulative frequency distribution F_i for each X_i is estimated using histograms constructed on past data observed at the site.

Equal-Value.[2] This scheme is based on a straightforward partitioning of the global threshold equally among the individual sites, i.e., the scheme sets each local threshold $T_i = \frac{T}{n}$. This heuristic is oblivious of the data distribution at the various sites, and thus the selected threshold values can be expected to be good only if the data distribution at each remote site is uniform.

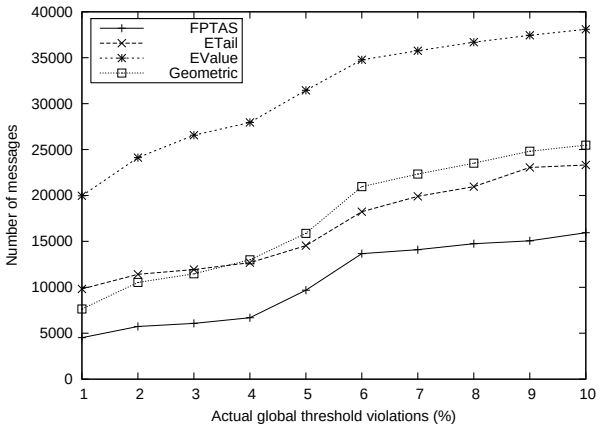

Figure 1: Experimental Results (Nov 17 - Dec 12, 2003)

Equal-Tail. A more intelligent heuristic makes use of the knowledge of the frequency distributions (estimated using histograms) for the various remote sites. We know, based on earlier discussion, that the probability of violation of a local threshold T_i is given by the right tail distribution $(1 - \frac{F_i(T_i)}{F_i(M_i)})$. This heuristic tries to minimize this tail for each X_i, and sets the local thresholds T_i such that the right tail is equal and the minimum possible across all the sites (while satisfying $\sum_i A_i T_i \leq T$).

A drawback of the Equal-Tail heuristic is that it individually minimizes the violation probability for each local constraint rather than minimizing the overall joint probability $(1 - \prod_i \frac{F_i(T_i)}{F_i(M_i)})$ that one or more local constraints are violated (as is done by our FPTAS). In the optimal solution, the local threshold violation probabilities $(1-\frac{F_i(T_i)}{F_i(M_i)})$ for the various X_is may not be equal; rather, they may be vastly different depending on the frequency distribution F_i for each X_i.

Geometric. Unlike the above schemes, the Geometric scheme [24] dynamically adjusts local thresholds T_i each time a remote site reports a local constraint violation. First, the coordinator polls local variables X_i to determine their most recent values, and then it distributes the slack $(T-\sum_i X_i)$ equally among the sites. Thus, the new threshold T_i at site i is set to $X_i + \frac{T-\sum_i X_i}{n}$. Observe that with the Geometric scheme, each local constraint violation leads to two rounds of message exchanges between the coordinator and remote sites, one to collect X_i values and a second to set the local threshold values[3].

6.2 Performance Metric

In order to compare the various schemes, we use as the performance metric the number of messages exchanged be-

[2] This scheme was called *Simple-Value* in [9]. We do not consider the *Improved-Value* algorithm of [9] in our experiments because like Equal-Value, it also assumes uniform data distribution and sets all local thresholds to be equal. Furthermore, in [9], the authors do not really address the issue of computing local thresholds for Improved-Value. Instead, they propose a naive scheme which reduces to Simple-Value when domain sizes M_i are large, and comparable to the threshold value T, which is the case for our traffic data sets.

[3] In [24], collection of values and threshold adjustment is initially done for a small subset of sites – this set is randomly increased as long the global constraint remains unsatisfied.

tween the coordinator and remote sites when one or more local thresholds are violated. This includes alarm messages from remote sites and global poll messages from the coordinator. Recall that every local constraint violation may not correspond to a global constraint violation – as a result, schemes with more local constraint violations lead to higher unnecessary polling of remote sites by the central site.

6.3 Real-life Data Sets

We use a real-life SNMP network usage data set in our experiments. This is obtained from the Dartmouth Wireless Network Trace Archive [1]. It contains data sets from polling around 500 Wireless Access Points installed in the Dartmouth University campus every five minutes over the period of Fall 2003/2004; each poll returns the number of bytes transmitted and received during roughly five-minute intervals. We use data sets for 10 Access Points, with one variable X_i per Access Point, whose value is equal to the number of bytes transmitted by the Access Point per five-minute interval. Thus, $\sum_i X_i$ is the total bytes transmitted by the 10 Access Points per five-minute interval, and we require this to be below the global threshold T.

6.4 Experimental Results

Our results with SNMP network usage data for four consecutive weeks[4](Nov 17 - Dec 12, 2003) are shown in Figure 1. We plot the number of messages exchanged due to local threshold violations for the various schemes - *FPTAS*, *Equal-Value*, *Equal-Tail* and *Geometric* - as the global threshold T is varied. In the figures, the x-axis shows the fraction (%) of observations for which the sum exceeded the chosen global threshold T, and the y-axis plots the number of transmitted messages.

For FPTAS, we use one week's (Nov 10-14) data (containing 1435 observations) to obtain an initial estimate of the frequency distributions F_i for the X_i variables, and construct equi-depth histograms (100 buckets) on the transmitted bytes in five-minute intervals. We use these histograms to set the local thresholds for the following weeks. When a change in the underlying distribution is detected locally at a site, we trigger the recomputation of histograms for the stream of X_i values, and then use these to compute new local thresholds T_i. In our experiments on four weeks (Nov 17-Dec 12) data, the recomputation was triggered only once, for the week of Nov 24-28. The thresholds were recomputed using this week's histograms and used for the following two weeks.

From Figure 1, it is easy to see that FPTAS consistently outperforms *Equal-Value*, *Equal-Tail* and *Geometric*. FPTAS generally results in 70% fewer communication messages compared to *Equal-Value* and 50% fewer messages compared to *Geometric* and *Equal-Tail*. The superior performance of FPTAS can be attributed to the fact that it takes into account the frequency distributions of variables when computing local thresholds, and also selects thresholds such that the joint probability of one or more local constraints being violated is minimized.

Further, observe that even though F_i is estimated using data from a previous week, FPTAS is able to reduce local threshold violations on the following weeks by a factor of 2-3 compared to the other schemes. This goes on to show that, in practice, histograms constructed using past observations yield accurate enough F_i estimates, and so the thresholds computed by FPTAS using past histograms should be quite effective in reducing communication overhead in real-world environments.

[4]Since network usage patterns are very different for weekdays and weekends, we restrict ourselves to data only for weekdays.

7 Conclusions

In this paper, we presented a novel scheme for detecting global constraint violations while incurring minimal communication overhead. Instead of continuously polling sites to check global constraints, our scheme decomposes global constraints into local constraints that can be checked efficiently at each site. Only in the rare instances that a local constraint is violated, is global polling triggered. We showed that by taking into account the data distributions at remote sites, local constraint thresholds can be set intelligently so that the overall probability of local constraint violations (and thus, global polling) is minimized. We developed an FPTAS that computes *near-optimal* local constraint threshold values, and via empirical evaluation on a real-life network traffic data set, showed that it can reduce communication costs by a factor of 2-3 compared to simple heuristics.

We are pursuing several promising research directions. Our constraint specification language currently allows the SQL aggregate operators SUM, MIN and MAX over scalar variables. A major research challenge is to permit richer operators like quantiles, top-k, etc., and even SQL Join queries over relational tuples. Also, instead of a single local constraint threshold at each site, it may be possible to further reduce global polling overhead (albeit, at the expense of increased complexity) by maintaining multiple local thresholds per site and tracking each threshold violation locally. We need to explore further the trade-off here between the additional traffic because of more threshold violations and the savings due to reduced polling.

References

[1] Dartmouth wireless network traces. http://crawdad.cs.dartmouth.edu/data/dartmouth.html.

[2] S. Agrawal, S. Deb, K. V. M. Naidu, and R. Rastogi. Efficient detection of distributed constraint violations. Technical Report ITD-06-46857G, Bell Labs Technical Memorandum, 2006.

[3] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, 2003.

[4] T. Corman, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[5] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *VLDB*, pages 13–24, 2005.

[6] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD*, 2005.

[7] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. Distributed set-expression cardinality estimation. In *VLDB*, 2004.

[8] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *SODA*, 2002.

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

[10] P. Dini, G. V. Bochmann, T. Koch, and B. Kramer. Agent based management of distributed systems with variable polling frequency policies. In *Integrated Network Management Symposium (IM 97)*, San Diego, May 1997.

[11] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *MOBICOMM*, 1999.

[12] P. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA*, pages 63–72, 2002.

[13] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, 2001.

[14] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall. A wakeup call for internet monitoring systems: The case for distributed triggers. In *HotNets-III*, November 2004.

[15] J. Jiao, S. Naqvi, D. Raz, and B. Sugla. Toward efficient monitoring. In *IEEE Jr. on Selected Areas in Comm.*, volume 18(5), pages 723–732, May 2000.

[16] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *SIGMOD*, 2006.

[17] D. Kifer, B. Shai, and J. Gehrke. Detecting change in data streams. In *VLDB*, 2004.

[18] L. Lee and H. Ting. Maintaining significant stream statistics over sliding windows. In *SODA '06*, pages 724–732, 2006.

[19] P. Moghe and M. Evangelista. An adaptive polling algorithm. In *IEEE/IFIT NOMS*, February 1998.

[20] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD*, 2003.

[21] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, pages 486–495, 1997.

[22] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[23] L. Schwiebert, S. Gupta, and J. Weinmann. Research challenges in wireless networks of biomedical sensors. In *MOBICOMM*, pages 151–161, 2001.

[24] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring thresholding functions over distributed data streams. In *SIGMOD*, 2006.

[25] D. Steere, A. Baptista, D. McNamee, C. Pu, and J. Walpole. Research challenges in environmental observation and forecasting systems. In *MOBICOMM*, pages 292–299, 2000.

[26] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, 2001.

[27] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *SIGMOD*, 2002.

[28] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.

[29] K. Yoshihara, K. Sugiyama, H. Horiuchi, and S. Obana. Dynamic polling scheme based on time variation of network management information values. In *Integrated Network Management Symposium (IM 99)*, pages 141–154, May 1999.