

# Efficient Detection of Distributed Constraint Violations

Shipra Agrawal  
Stanford University, Stanford, CA  
shipra@cs.stanford.edu

Supratim Deb, K. V. M. Naidu, Rajeev Rastogi  
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, i.e., instances when system behavior deviates substantially from the norm. 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 show that the problem of selecting the local constraints, based on frequency distribution of individual system variables, so as to minimize the communication cost is NP-hard. We propose approximation algorithms for computing provably near-optimal (in terms of the number of messages) local constraints. Experimental results with real-life network traffic data sets demonstrate that our technique 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., P2P systems, sensor networks), *monitoring applications* are increasingly required to handle hundreds of thousands of nodes with dynamically changing states. For detecting abnormalities, distributed algorithms like sums and averages of numeric values [8] or top-k values [3], that continuously record global system state at all times are an overkill, and lead to unnecessary communication even when all is well [7]. 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, ensuring that the total TCP SYN packet rate to a destination across many edge routers is within a threshold, can be a global constraint for detecting DDoS attacks. Another example of a global constraint can be ensuring that files beyond a certain size limit are not exchanged between nodes in a P2P system.

Most applications of global constraints 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 the monitoring traffic minimal. Ideally, our schemes should transmit messages only when global constraints are in danger of being breached.

**Prior Art.** 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. However, such a scheme is oblivious of the system state with respect to  $\mathcal{G}$  and thus, it may incur unnecessary communication overhead. While intelligent *polling* based schemes have been proposed in [11, 9, 15, 10], but shortcoming of pure polling-based approaches is that they may miss constraint violations unless the polling interval is set to be small enough. Our approach in this paper is to combine local-event-reporting or ‘triggering’ with polling to ensure guaranteed detection of every alarm condition.

The works most closely related to our approach are those of [6, 7, 12, 16]. The pioneering work of [6] 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. However, none of these take the data distribution

of the variables into account.

**Our contributions.** Our main contributions are as follows:

1. We propose a general framework for detecting global constraint violations using local constraints. While the idea of using local constraints to detect global constraint violations is not new [6, 7, 12, 16], many of these schemes consider very specific forms of global and local constraints, whereas, we permit more general constraints containing aggregate operators MIN and MAX, as well as boolean conjunctions and disjunctions.
2. We formulate the problem of selecting the best local constraints, which depends on past observation of state-occurrence frequencies of the state variables, as an optimization problem whose objective function aims to maximize the frequency of occurrence of system states covered by the local constraints. We show that the problem is NP-hard and we also develop an FPTAS<sup>1</sup> that yields *provably near-optimal* local constraints.
3. Experimental results with a real-life network traffic data set demonstrate that our technique reduces message communication overhead (due to local threshold violations) by as much as 70% as compared to naive approaches that do not account for the frequency distribution of the system variables.

## 2. Problem Definition

### 2.1. System Model

Our distributed system consists of  $n$  remote sites and a central coordinator. The  $i^{\text{th}}$  site 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  $A_i X_i$  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  $MIN/MAX\{a_1, a_2\}$ .

<sup>1</sup>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  $1/\epsilon$ .

In the remainder of the paper, we will assume that  $A_i$  and  $T$  are positive integers, and that  $op$  is  $\leq$ . This assumption can be easily relaxed. 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 4, 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. [8] for estimating  $\sum_i A_i X_i$  with a very small relative error  $\epsilon$ .

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*.

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

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.

### 2.2. Local Threshold Selection Problem

In this section we focus on appropriate choice of the local thresholds  $T_i$ 's.

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.

Note that, there can be *false alarms* where a local constraint is violated, but the global constraint still holds. For example, for the  $\mathcal{G} = X_1 + 2X_2 - 2 \leq 5$ , 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  $X_1 + 2X_2 \leq 5$ . Thus, we wish to minimize the false alarms due to violation of local thresholds when  $\mathcal{G}$  holds. Let  $\mathcal{L} = \bigwedge_i \mathcal{L}_i$ . Our goal is to minimize  $\Pr(\mathcal{L} = \text{false})$ , or equivalently maximize  $\Pr(\mathcal{L} = \text{true})$  when  $\mathcal{G}$  holds.

To compute  $P(\mathcal{L} = \text{true})$ , we need to introduce additional notation. Let  $\mathbf{v} = [v_1, \dots, v_n]$  denote the vector of values taken by variables  $X_1, \dots, X_n$ . Further, let  $f$  denote the joint frequency distribution of variables  $X_i$ ; thus,

for  $\mathbf{v} = [v_1, \dots, v_n]$ ,  $f(\mathbf{v})$  is the frequency with which the variables  $X_1, \dots, X_n$  take the values  $v_1, \dots, v_n$ , 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  $P(\mathcal{L} = \text{true}) = \frac{\sum_{\mathbf{v} \text{ satisfies } \mathcal{L}} f(\mathbf{v})}{\sum_{\mathbf{v}} f(\mathbf{v})}$ . Since, typically the joint frequency distribution  $f$  are not available, we consider a more practical alternative from a computation perspective in which we estimate the frequency distribution  $f_i$  of each  $X_i$ . Then, assuming independence of the variables  $X_i$ 's,

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

where  $F_i$  be the cumulative frequency distribution for  $X_i$ , i.e.,  $F_i(j) = \sum_{k \leq j} f_i(k)$ .

**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  $T_1, \dots, T_n$  such that

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

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 [5, 14].

Since the  $f_i$ 's can change with time, we will assume that the threshold values  $T_i$  are always based on the most current data distribution  $f_i$ . We point out 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.

### 3. Threshold Selection

In this section, we present schemes for solving (1). 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:

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

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:

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

It is straight-forward to see that one can compute the value of  $V_n(T)$  in  $O(nT^2)$  operations. 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. In fact, we can show that devising an efficient polynomial time algorithm for the

threshold selection problem is NP-hard. (Due to space constraints, we omit the proof of NP-hardness here. It can be found in [2].)

In the following, we present an FPTAS for the problem.

### 3.1. FPTAS

We now give an algorithm for computing an  $(1 + \epsilon)$  approximation to the problem given by (1) that has a complexity polynomial in  $1/\epsilon$ ,  $n$ , and  $\log T$ .

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

#### Overview of the Algorithm

Let us assume that we have chosen an  $\alpha$  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  $T_i$ s 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  $F_i$ s. Let  $l$  denote the smallest power of  $\alpha$  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

We will first analyze the running time of the algorithm. Let  $\bar{M} = \max\{M_1, \dots, M_n\}$ . 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  $l$  (clearly,  $l \leq \log \bar{P} / \log \alpha$ ) 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  $O(nl(l + \log \bar{M}))$ . Also, since each  $F_i$  is quantized in steps of  $\alpha$ , it can be argued that the overall approximation factor of the cost  $\alpha^n$ . For an approximation guarantee of  $1 + \epsilon$ , we equate  $\alpha^n$  to  $1 + \epsilon$  to get the complexity of the algorithm in terms of  $1/\epsilon$ ,  $n$ , and  $\log T$ . We thus have the following main result:

**Theorem 1** [2] *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  $O(\frac{n^2}{\epsilon} \log(\bar{P}))(\frac{n \log(\bar{P})}{\epsilon} + \log \bar{M})$ .*

## 4. General Boolean Constraints

Let  $E(X_1, \dots, X_n)$  denote a linear expression of the form  $\sum_i A_i X_i$ . We now consider more general *boolean* global constraints that are conjunctions and disjunctions of the simpler constraints  $E \leq T$ , i.e., 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  $X_i$ 's, and  $\hat{T}_{j,k}$  is an integer threshold value.

**Constraints containing MIN/MAX:** We will explain this through an example. Suppose, we have a constraint of the form  $(A + \text{MIN}\{B, C\}) \leq T$ . Since this is equivalent to  $\text{MIN}\{A + B, A + C\} \leq T$ , we can rewrite the constraint as  $(A + B \leq T) \vee (A + C \leq T)$ , which is a disjunction of two linear constraints. In fact, we can argue that 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/SUM, can be expressed as boolean constraints with conjunctions and disjunctions.

**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  $T_i$ s to satisfy the constraint  $\bigvee_j (E_j(T_1, \dots, T_n) \leq \hat{T}_j)$ . So the problem is to find threshold values  $T_i$  such that

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

In [2], we have proved the following.

**Theorem 2** [2] *There is an FPTAS for the problem of computing local thresholds for a global constraint containing only disjunctions over linear inequalities.*  $\square$

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

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

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

**Theorem 3** [2] *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  $P = NP$ .*  $\square$

A simple heuristic that employs our FPTAS (from Section 3) 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  $i$ th 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  $E_j \leq \hat{T}_j$ .

**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 4, 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.

**1. 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 the threshold for the  $i$ th site for the disjunction in the  $j$ th conjunct,  $T_{i,j}$ , can be computed using the FPTAS mentioned in Theorem 2.

**2. Compute thresholds for conjunction:** Now, for each  $i$ , choose  $T_i = \min_j \{T_{i,j}\}$ , and then increase the thresholds as long as they satisfy  $\bigwedge_j (\bigvee_k E_{j,k} \leq \hat{T}_{j,k})$ .

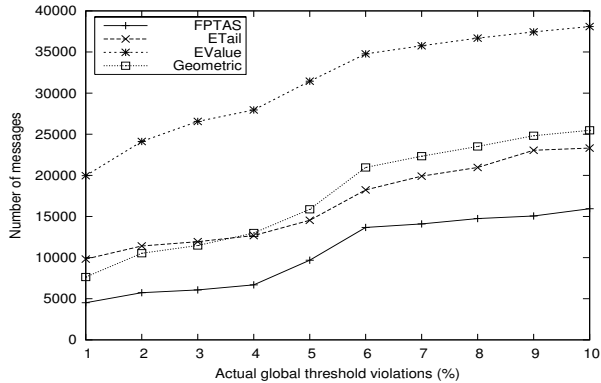


Figure 1. Experimental Results (Nov 17 - Dec 12, 2003)

## 5. Performance Evaluation

In this Section, we quantitatively assess the utility of our FPTAS for selecting local threshold values using a real-life data set. 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 certain 500 Wireless Access Points; 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  $T$ .

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  $X_i \leq T_i$ . We compare the following schemes.

**FPTAS:** This is the FPTAS described in Section 3.1. The frequency distribution of the previous week is used to compute the threshold for the subsequent week.

**Equal-Value:** This is a scheme proposed in [6] where all the thresholds are set to be equal to  $T_i = T/n$ .

**Equal-Tail:** We know 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  $\sum_i A_i T_i \leq T$ ).

**Geometric:** The Geometric scheme [16] 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  $X_i + \frac{T - \sum_i X_i}{n}$ .

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.

## 6. Conclusions

In this paper, we presented a novel scheme for detecting global constraint violations using local constraints. The local thresholds are chosen based on frequency distribution in a manner so as to minimize the messages exchanged. We presented an FPTAS for the problem, and demonstrated the huge gains of the scheme through experiments with real data sets.

## 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] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *SODA*, 2002.
- [6] M. Dilman and D. Raz. Efficient reactive monitoring. In *INFOCOM*, pages 1012–1019, 2001.
- [7] A. J. et. al. A wakeup call for internet monitoring systems: The case for dist. triggers. In *HotNets-III*, Nov 2004.
- [8] C. O. et. al. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD*, 2003.
- [9] J. J. et. al. Toward efficient monitoring. In *IEEE JSAC*, volume 18(5), 2000.
- [10] K. Y. et. al. Dynamic polling scheme based on time variation of network management information values. In *Int. Network Mgmt. Symp.*, 1999.
- [11] P. D. et. al. Agent based management of distributed systems with variable polling frequency policies. In *Int. Network Mgmt Symp.*, San Diego, May 1997.
- [12] R. K. et. al. Communication-efficient dist. monitoring of thresholded counts. In *SIGMOD*, 2006.
- [13] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, 2001.
- [14] L. Lee and H. Ting. Maintaining significant stream statistics over sliding windows. In *SODA '06*, pages 724–732, 2006.
- [15] P. Moghe and M. Evangelista. An adaptive polling algorithm. In *IEEE/IFIT NOMS*, February 1998.
- [16] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring thresholding functions over distributed data streams. In *SIGMOD*, 2006.