Efficient Constraint Monitoring Using Adaptive Thresholds

Srinivas Kashyap *, Jeyashankher Ramamirtham **, Rajeev Rastogi #, Pushpraj Shukla †

*IBM T.J. Watson Research Center, Hawthorne, NY. srkashya@us.ibm.com

**Netcore Solutions, India. jeyashankher@gmail.com

#Bell Laboratories, Bangalore, India. rastogi@alcatel-lucent.com

†CS Department, University of Texas, Austin, TX. pushpraj@cs.utexas.edu

***Abstract*— Detecting constraint violations in large-scale distributed systems has recently attracted plenty of attention from the research community due to its varied applications (security, network monitoring, etc.). Communication efficiency of these systems is a critical concern and determines their practicality.**

In this paper, we introduce a new set of methods called *non-zero slack* schemes to implement distributed SUM queries efficiently. We show, both analytically and empirically, that these methods can lead to a considerable reduction in the amount of communication. We propose three adaptive non-zero slack schemes that adapt to changing data distributions; our best scheme is a lightweight reactive scheme that probabilistically adjusts local constraints based on the occurrence of certain events (using only a periodic probability estimation). We conduct an extensive experimental study using real-life and synthetic data sets, and show that our non-zero slack schemes incur significantly less communication overhead compared to the state of the art zero slack scheme (over a 60% savings).

I. Introduction

Monitoring emerging large-scale, distributed systems (like peer to peer systems, server clusters, IP networks and sensor networks) poses several interesting challenges. Sensor networks place various constraints on the communication and processing capabilities of the nodes. Network monitoring systems need to process large volumes of data in (near) real-time from a widely distributed set of sources, and it is nearly impossible to store and process the entire information in an on-demand fashion. Further, many queries in these systems are continuous and are executed for the lifetime of the system. For example, consider a system that monitors a large network for distributed denial of service (DDoS) attacks. This system needs to process data from several routers at a rate of several gigabits per second. Also, the system needs to detect attacks as soon as they happen (with minimal latency) to enable networks operators to take expedient countermeasures in order to mitigate the effect of these attacks. The research community has looked at developing algorithms for computing and tracking a wide range of aggregate statistics over distributed data streams [3], [2]. These apply to a general class of continuous monitoring applications, where the goal is to optimize the operational resource usage of these algorithms and still guarantee that the estimate of the aggregate function is always within specified error bounds.

Communication efficiency is a critical concern for these distributed data management systems. In sensor networks, transmitting messages from sensor nodes consumes valuable battery resources that determine the lifetime of these networks. In network data monitoring systems, each node in the network receives voluminous amounts of data. Transmitting an equivalent amount of data across the network in order to perform distributed computations is impractical. Thus, the communication efficiency of the distributed computation algorithms determines the practicality of these systems. [8], [14], [6], [7] describe distributed constraints monitoring or distributed triggers as a mechanism of reducing the amount of communication. These methods filter out "uninteresting" events and do not require communication across the network for these events; thus, reducing the communication needed to perform the computations.

In this paper, we introduce a new set of methods that we call *non-zero slack* schemes. We study these methods for an important class of queries called distributed SUM constraints or distributed triggers, that are used to track anomalous behavior. We quantify the benefits of non-zero slack schemes for distributed SUM constraints monitoring, both analytically and empirically, and show that these methods can considerably reduce the number of communication messages in the network (by 60% in our experiments). Implementing non-zero slack schemes for the simple distributed SUM constraint queries presents a number of challenges and we address these challenges in this paper. We believe that non-zero slack schemes can be applied to a wide range of other distributed queries to make them communication efficient and thus, practical. We leave the study of non-zero schemes for other queries as future work.

Distributed SUM constraints: The distributed constraints that we focus on are of the form $\sum_{i=1}^{n} x_i \leq T$, where n is the number of nodes in the system, x_i is the value of a variable that is being monitored at node i, and T is the constraint's threshold. This constraint can be decomposed into a set of local thresholds, T_i at each remote site i, such that $\sum_{i=1}^{n} T_i \leq T$. If at all sites, $x_i \leq T_i$ then $\sum_{i=1}^{n} x_i \leq \sum_{i=1}^{n} T_i \leq T$. Thus, if none of the thresholds at the nodes are violated, the global constraint is satisfied. In effect, the local thresholds act

as filters that help in reducing the amount of communication messages in the system.

Consider an example application for detecting service quality degradations of VoIP sessions in a network. Let us suppose that VoIP requires the end-to-end delay to be within 200 milliseconds and the loss probability to be within 1%. Consider a path through the network with n network elements (routers, switches) $s_1, s_2, \ldots, s_n$. To monitor loss probabilities through the network, each network element has an estimate of its local loss probability, say $l_i, i \in [1, n]$. The loss probability of the path through these network elements is given by $L = 1 - (1 - l_1)(1 - l_2)\ldots(1 - l_n)$. This yields $\log(1 - L) = \log(1 - l_1) + \log(1 - l_2) + \cdots + \log(1 - l_n)$. If we need $L \leq 0.01$, we need $\log(1-L) \geq \log(0.99)$. Inverting the sign on both sides, this transforms into the constraint $\sum_{i=1}^{n} (-\log(1 - l_i)) \leq -\log(0.99)$.

Thus, the problem of monitoring losses in a network can be addressed using distributed constraints monitoring. Delays can be monitored similarly using distributed SUM constraints.

Non-zero slack schemes: Algorithms to determine local constraints can be classified into two categories: zero slack schemes and non-zero slack schemes.

- Zero slack schemes: These algorithms assign local constraints that do not have any *slack* in the system. Specifically, the local threshold values, T_i are determined such that $\sum_{i=1}^{n} T_i = T$. Most prior work uses this method of *tight* allocation of threshold values at the nodes.
- Non-zero slack schemes: These algorithms determine local constraints that retain some slack in the system; i.e. $\sum_{i=1}^{n} T_i \leq T$ and the slack in the system is given by $S = T - \sum_{i=1}^{n} T_i$.

Zero slack schemes perform well if the values at the nodes do not exceed their threshold values, i.e. $x_i \leq T_i$. However, this is seldom the case and the values can exceed this value frequently even when the global sum is less than T. When a node's value exceeds this threshold value, we can no longer say with certainty if the global constraint is satisfied *or not*. This requires polling the values at all nodes in the network to determine whether the global constraint is still satisfied, causing a flurry of communication messages.

Non-zero slack schemes, on the other hand, retain some slack, and thus allow nodes to exceed their local thresholds by that amount and are still able to guarantee satisfaction of the global constraint without polling for values at all nodes. We demonstrate that the reduction in the messages is considerable both analytically and through experiments for typical data.

Our contributions: In this paper, we undertake a comprehensive study of communication efficient monitoring of distributed SUM constraints using non-zero slack schemes.

1) We show both analytically and empirically that non-zero slack schemes outperform the state-of-the-art zero slack scheme for different data distributions.
2) We develop adaptive algorithms for setting threshold values at remote sites in the presence of non-zero slack (for changing data distributions).
3) Finally, we present the results of a thorough and detailed set of experiments using both synthetically generated data and real world data, and show that our adaptive non-zero slack algorithms can result in significant savings in the amount of communication.

To the best of our knowledge, our's is the first work to systematically study non-zero slack schemes for detecting distributed constraint violations.

II. Related Work

Monitoring data streams in a distributed environment has been an important focus area of research in recent years. Algorithms have been proposed for continuous monitoring of top-k items [3], sums and counts [13], quantiles [2], joins and Max values. These papers address problems that are different from ours. For instance, Olston *et al.* [13] tackle the problem of *continuously tracking* multiple SUM queries with different error bounds which is very different from our problem which aims to detect if the result of a single SUM query exceeds a given threshold.

Most recent work on the problem of distributed constraint monitoring propose zero-slack schemes and one prior work describes a non-zero slack scheme for implementing distributed constraint monitoring.

Zero slack schemes: Jain *et al.* [8] discuss the challenges in implementing distributed triggering mechanisms for network monitoring. They use a zero slack scheme that uses local constraints of T/n to detect constraint violations.

The recent work of Sharfman *et al.* [14] represents the state of the art in detecting distributed constraint violations. For SUM constraints over variables x_i, their scheme reduces to a zero slack adaptive scheme that always maintains the invariant $\sum_i T_i = T$. Each time a local constraint $x_i \leq T_i$ is violated, the x_i values are polled and the slack $S = T - \sum_i x_i$ is distributed among the sites, that is, each T_i is set to $T_i + S/n$. We compare our algorithms against this scheme, which we also refer to as the "Geometric Scheme" in later sections.

Agrawal *et al.* [5] present a zero slack scheme that formulates the problem of selecting local constraints as an optimization problem whose objective function aims to minimize the probability of global polls given the individual frequency distributions of variables x_i.

Keralapura *et al.* [9] propose static and adaptive algorithms to monitor distributed SUM constraints. We do not study static methods in this paper as they result in much more communication than adaptive ones. Their adaptive schemes use similar methods to the ones proposed in [14] and are essentially zero slack schemes.

Non-zero slack schemes: The scheme that is perhaps closest to our approach is that of Dilman and Raz [6]. In addition to the Simple-Value scheme that sets each local threshold

T_i to T/n, they also propose an *Improved Value* scheme in which local thresholds T_i's (same for all i) are set to lower values than T/n, observing that the *Improved Value* scheme can outperform the simple value scheme. This translates to the improvement of non-zero slack schemes over zero slack schemes in our work. Their paper however does not address the following issues that we study in this paper - (1) They do not show how to set local thresholds, which is critical for achieving good performance. (2) They do not show how to adapt local threshold values for changing data distributions.

Huang *et al.* [10] consider a novel variant of the instantaneous tracking problem where they track constraint violations that *persist over time*. [10] uses queuing theory as an analytical tool to compute local threshold values that meet user-specified false alarm and missed detection rates. Their analysis makes two assumptions which may not be true in our setting: (1) All local threshold values are equal, and (2) Local site values follow a Normal $N(0, \sigma^2)$ distribution (e.g., network link delay values have been found to follow a Weibull distribution). Also, it is unclear if the computed local threshold settings in [10] optimize total communication costs - this is because false alarms correspond to global polls and are only one component of communication costs (the second being local alarms). Note that [10] allows missed detections, but we do not.

We propose a non-zero slack scheme called the reactive scheme that is inspired by the probabilistic scheme presented in [12]. However, our problem setting is significantly more complex. Please refer to the end of Section IV-D for a detailed discussion of the differences between our work and theirs as well as novel contributions of our work in this context.

III. Problem Definition

A. System Architecture

We consider a distributed monitoring system consisting of n remote sites $s_1, \ldots, s_n$ and a central coordinator site s_0. Each site s_i observes a continuous stream of updates, which it records as a constantly changing value of its local variable x_i. x_i's can take non-negative real values in the domain $[0, \infty)$ [1]. The remote sites can communicate with the coordinator to send or receive messages. We assume that this communication can happen without any loss or delay. In addition, time is assumed to be slotted and synchronized across sites. At the end of each time slot t, site s_i observes value $x_i(t)$ [2]. We shall sometimes refer to a slot as a 'round' for the system. Note that the values x_i can increase or decrease with time. This system architecture is in line with previous work [9].

B. Detection of distributed constraints violations

We are looking to devise schemes that can detect violation of global constraints defined over distributed system variables of the form $\sum_{i=1}^{n} x_i \leq T$. Each site s_i is assigned a local threshold T_i such that $\sum_{i=1}^{n} T_i \leq T$. A remote site sends a *local alarm* to the coordinator whenever $x_i > T_i$ and remains silent otherwise. If $\forall i \in [1, n], x_i \leq T_i$ and $\sum_{i=1}^{n} T_i \leq T$, then $\sum_{i=1}^{n} x_i \leq T$. Thus, if none of the sites report local alarms, the global constraint is not violated.

If some site violates the local constraint, then the global constraint could have been violated. Let us assume that site j violates the local constraint and sends the value at the site x_j to the coordinator. The coordinator now verifies if $x_j + \sum_{i \neq j} T_i \leq T$ is satisfied. If this condition is satisfied, then the global constraint is not violated. However, if the constraint is not satisfied, the coordinator polls the remote sites for their exact values to determine if the global constraint is still satisfied. We refer to this as a *global poll*.

A global poll can be performed by polling for the exact value at *all* sites. Alternatively, the coordinator site can poll a subset of sites S and determine if the global constraint is not violated by checking if $\sum_{i \in S} x_i + \sum_{i \notin S} T_i \leq T$ is satisfied. If this constraint is satisfied, then the global constraint is not violated. The coordinator need not poll the rest of the sites and hence this method reduces the communication overhead. However, if the constraint is still violated, the coordinator polls another larger subset of sites and performs the same procedure. This procedure is continued until either the coordinator detects that the global constraint is not violated or all sites have been polled. While this method reduces the communication required, it introduces undesired latency in detecting constraint violations which might be unacceptable for many applications. Hence, we prefer to restrict the number of rounds of polling to 1 or 2.

C. Cost model

We now present the model that we use to estimate the communication cost of a distributed constraints violation detection scheme. The coordinator keeps an estimate Y_i of each x_i such that $Y_i \geq x_i$ at all times. Local alarms from a remote site s_i are used to update Y_i. Once a remote site exceeds its local threshold, it sends any change in its local value to the coordinator [3].

$$\begin{aligned} Y_i &= x_i \text{ for each } s_i \text{ that reports a local alarm} \\ &= T_i \text{ for each } s_i \text{ that has not reported anything} \end{aligned}$$

Whenever this estimate $\sum_i Y_i > T$, the coordinator initiates a global poll to know the x_i values and check if the constraint is actually violated.

The communication cost of the system comes from local alarms and global polls. Define

- $P_l(i)$: The probability of a local alarm at site i = $\Pr(x_i > T_i)$ - i.e. The probability that the value at remote site s_i is greater than its threshold T_i.
- P_g : The probability of a global poll = $\Pr(\sum_i Y_i > T)$

Also, let C_l be the cost of transmitting a message from a

[1] Negative values can be handled as well. We omit it for clarity of discussion.

[2] If time is not important, we refer to the value at the site i as simply x_i.

[3] It is easy to extend our schemes to an approximate scheme where a remote site sends updates to the coordinator on "significant" changes in its value only (for example, if the value changes by Δv, where v is the previous value sent to the coordinator). The coordinator estimate of the global sum is approximate by a factor of Δ.

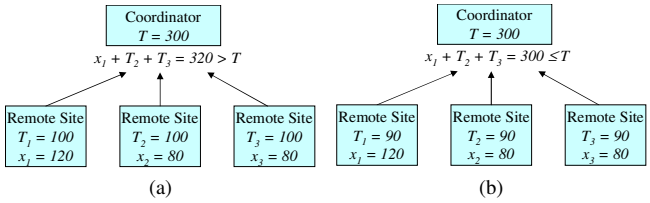


Fig. 1: (a) Assigning local thresholds of T/n results in a global poll, (b) Assigning lower local thresholds absorbs transient spikes in values requiring no global polls

remote site s_i to the coordinator on a local alarm [4] and C_g be the cost of the global poll. Typically C_l is $O(1)$ and C_g is $O(n)$. The communication cost of our scheme is then given by

$$C = P_g C_g + \sum_{i=1}^{n} P_l(i) C_l \quad (1)$$

Given a value of T_i at a remote site, $P_l(i)$ depends entirely on the distribution of observed values at the remote site and is not affected by changes in behavior of the other sites (assuming that observed values are independent across sites). P_g, however, depends on the observed values as well as the threshold values at the other remote sites. Hence, a change in the threshold value at some site or a change in distribution of any site's observed value affects the P_g across all sites.

D. Local thresholds assignment problem

A simple method to set the local thresholds at the remote sites is to assign $T_i = T/n, \forall i \in [1, n]$. This is referred to as the *simple value* scheme in Reference [6]. Note that if some site s_j violates its local constraint ($x_j > T/n$), then the coordinator site needs to perform a global poll. This is because $x_j + \sum_{i \neq j} T_i > T$. This method is an example of a zero slack scheme, i.e. $\sum_{i=1}^{n} T_i = T$. Note that in general for all zero slack schemes, a local alarm results in a global poll because the estimate of the constraint would exceed T on a local alarm.

On the other hand, setting local thresholds to values less than T/n provides some *slack* at the coordinator that can be used to avoid expensive global polls. The slack at the coordinator is given by $T - \sum_{i=1}^{n} T_i$ and any site can exceed its local threshold value by this value without needing a global poll. This slack can thus be used to absorb temporary spikes (e.g., flash crowds) in the values at the remote sites, whereas setting all local thresholds at T/n would have resulted in global polls. Thus, a non-zero slack scheme can result in lower communication costs. Setting the local thresholds to very low values however results in frequent local alarms and hence high communication overhead. At an extreme, we can set all local thresholds to 0. This is equivalent to the remote sites sending every change in local values to the coordinator and hence, the coordinator tracks the exact values at the sites, requiring no global polls. However, the number of updates sent from remote sites to the coordinator is clearly unacceptably large.

We illustrate this with an example in Figure 1 that has 3 remote sites and has a threshold of $T = 300$. Figure 1a shows the case of a zero slack threshold assignment, where each local threshold is set to T/n ($= 100$). The first site's value increases to 120, causing it to send a local alarm to the coordinator. The coordinator's site estimate ($x_1 + T_2 + T_3$) is greater than T causing it to initiate a global poll. Figure 1b shows a non-zero slack assignment where the local thresholds are assigned lower than T/n at 90. The slack in the system is 30 and since the first site's value is not more than this slack's value above its local threshold value, the coordinator does not need to initiate a global poll. This reduces the amount of communication required to track the distributed constraint. Note that if the local thresholds were set to 75 instead, this would have resulted in local alarms at all sites and the amount of communication would be equivalent to a global poll. Thus, it is vital to identify the optimum value of the local thresholds that results in low overall communication overhead. Reference [6] makes the same observation in their *Improved Value* scheme.

While smaller values of local thresholds results in higher probability of local alarms $P_l(i)$ and lower global poll probability P_g, larger values of local thresholds result in lower probability of local alarms and higher global polls probability. The optimum local threshold assignment balances these two costs to minimize the cost given by C in Equation 1.

We now define the local threshold assignment problem.

Problem Statement 1: Given a threshold T and n remote sites with values x_i at site $i \in [1, n]$, determine the threshold values, T_i at each site such the total cost of communication C, given by Equation (1), is minimized.

E. Zipf case

We now present a more concrete example that illustrates the cost benefits of identifying optimal local thresholds. Consider n remote sites, each having values in the range $[0, T]$ that follow a Zipf distribution (with Zipf exponent 1) in this example. The probability that site s_i takes on a value v is given by $Pr(x_i = v) = \frac{1}{H_T} \cdot \frac{1}{v}$, where H_i is the i^{th} Harmonic Number defined by $H_i = \sum_{k=1}^{i} \frac{1}{k}$. Let the global threshold value be T, where $2^n < T < 2^{2^n}$. We assume that $C_l = 1$ for all sites and $C_g = n$.

Theorem 2: For this example, if $C_{T/n}$ is the cost of the system when the local thresholds at all the sites are T/n and $C_{T/log(T)}$ is the cost of the system when the local thresholds at the sites are $T/log(T)$, then the gain derived by using a threshold value of $T/log(T)$ at the remote sites instead of T/n, given by $g = \frac{C_{T/n}}{C_{T/log(T)}}$, is $\Omega(log(n))$.

The proof of Theorem 2 can be found in the full version of our paper [1]. We assume large values of T ($> 2^n$) because violation of this threshold must be a rare event and hence the probability of this threshold being violated should be low. Note that since $T > 2^n$, the slack in the system is $T - \frac{nT}{log(T)} > 0$.

[4] Our model can be easily extended to systems where the costs of sending messages to the coordinator are different for the remote sites. For simplicity of exposition, we assume that this cost is the same for all remote sites.

This slack absorbs the large values of the Zipf distribution at the sites and hence results in the cost gains.

This example shows that using a threshold value of $T/log(T)$ results in a reduction in the communication overhead of tracking distributed constraint violations for the case where all sites' values follow a Zipf distribution. In reality, the values at the sites need not follow this distribution. Further, each site's values can follow a different distribution and the distribution of values can change over time. Thus, a threshold assignment algorithm should adapt to these changes when they happen.

IV. Adaptive threshold assignment

We now present our algorithms for the problem of determining optimal local threshold values at the remote sites. We present three schemes to assign threshold values T_i at the remote sites.

- **Brute force algorithm:** The first algorithm uses the coordinator to assist the remote sites to determine optimal threshold values. This algorithm performs well in our experiments. However, it requires each remote site to periodically send their histograms to the coordinator, and the coordinator performs a complex computation to determine the local thresholds for each remote site. This makes the algorithm relatively expensive and less desirable to use in large scale deployments.
- **Markov based algorithm:** This algorithm uses Markov's inequality to approximate the global poll probability and this results in a decentralized algorithm. The advantage of this algorithm is that it is decentralized except for a few messages to ensure correctness. It, however, performs relatively poorly in our experiments.
- **Reactive algorithm:** The third algorithm uses local alarm and global poll events in the system to assign the local thresholds at the remote sites. This algorithm does not require as much communication as the brute force algorithm but still results in comparable performance.

Before describing our algorithms, we present the cost of the geometric scheme [14] as applied to our problem. We consider this algorithm to be state of the art and compare the performance of our algorithms with this method. For all algorithms, a local alarm is a single message from the remote site to the coordinator and each global poll requires n messages assuming that the coordinator polls every remote site.

A. Geometric approach

The geometric approach is an adaptive algorithm that we have described in Section II. The communication costs of this scheme are as follows.

Communication costs: The cost for this scheme comprises of global polls and *control messages*. Following every global poll, the scheme sends n *control messages*, one for each remote site to set new threshold values. We can ignore the cost due to local alarms since this is a zero-slack scheme and therefore the coordinator needs to perform a global poll for every local alarm.

B. Brute force algorithm

This algorithm uses information from all remote sites at the coordinator to compute the local threshold values. It determines the local thresholds by computing $P_l(i)$ and P_g, and then selecting the local threshold that minimizes the total cost C (given by Equation 1).

Each site maintains a histogram of the values that it sees over time as $H_i(v), \forall v \in [0, T]$, where $H_i(v)$ is the probability of site s_i taking the value v. We use equi-depth histograms at each monitoring site to keep track of the data distribution. We also employ exponential aging to ensure that the histogram reflects recently seen values more prominently than older ones.

The probability of a local alarm is entirely independent on the state of the remote sites and each remote site can independently calculate $P_l(i)$ at a given value of T_i and is given by $P_l(i) = 1 - \sum_{j=0}^{T_i} H_i(j)$.

P_g however is dependent on the state of all remote sites. In order to compute P_g, each remote site sends its local histogram to the coordinator periodically. We call this period the *recompute interval*. The coordinator uses the histograms to compute P_g (where $P_g = \Pr(Y > T) = 1 - \sum_{v=0}^{T} \Pr(Y = v)$).

$\Pr(Y = v)$ can be computed at the coordinator using a dynamic programming algorithm with pseudo-polynomial time complexity of $O(nT^2)$ (see the full paper [1] for details).

In order to determine the optimal threshold values at each site that result in minimum cost, we can do a naive exhaustive enumeration of all T^n possible sets of local threshold values. For each combination of threshold values, we compute the $P_l(i)$ values at each site and the P_g value to determine the cost. Thus, this naive enumeration has a running time of $O(nT^{n+2})$. This is clearly not scalable for large values of T and n. One optimization that we use to reduce the running time is to only consider local threshold values in the range $[T_i - \delta, T_i + \delta]$ for a small constant δ (see the full paper [1] for details).

Communication costs: Apart from local alarms and global polls, each remote site sends a histogram update (see full paper [1] for details) of its histogram values every recompute interval and the coordinator recomputes the threshold values. Thus, there are $2n$ control messages in the system every recompute interval.

Note that we count the message to send histogram data from a remote site to the coordinator as one control message. This message is however larger in terms of size than a control message used to send new threshold values to remote sites. Thus, our estimate in terms of the number of messages (and not the size of messages) is an optimistic estimate of the control overhead of this algorithm.

C. Markov-based algorithm

The brute force algorithm requires remote sites to send their histograms every recompute interval and requires the coordinator to perform the above computation to determine the local threshold values. This is not very desirable when the

number of remote sites is large. Our Markov-based algorithm decentralizes the computation of P_g thus enabling each site to independently determine the local threshold values.

Using Markov's inequality,

$$P_g = \Pr(Y > T) \leq \frac{E[Y]}{T} = \frac{E[\sum_{i=1}^{n} Y_i]}{T} = \frac{\sum_{i=1}^{n} E[Y_i]}{T}$$

Thus, the cost of the system is given by

$$C = \sum_{i=1}^{n} C_l P_l(i) + C_g P_g \leq \sum_{i=1}^{n} C_l P_l(i) + \frac{C_g}{T} \sum_{i=1}^{n} E[Y_i]$$

$$C \leq \sum_{i=1}^{n} \left(C_l P_l(i) + \frac{C_g}{T} E[Y_i] \right)$$

$E[Y_i]$ can be computed at the local sites as follows.

$$E[Y_i] = \sum_{v=0}^{T} Y_i \Pr(Y_i = v) = \sum_{v=0}^{T_i} T_i H_i(v) + \sum_{v=T_i+1}^{T} v H_i(v)$$

Note that each site can independently determine the local threshold value that minimizes its contribution to the total cost, $C_l P_l(i) + \frac{C_g}{T} E[Y_i]$, thus requiring no assistance from the coordinator. The global poll probability P_g using the Markov inequality is an upper bound on the actual probability and this estimate grows to 1 very quickly with increasing T_i values. Hence, this algorithm assigns local threshold values that are much smaller than the optimum threshold values to minimize this estimated cost. This results in a large number of local alarms and hence, higher cost [5]. We demonstrate this in our experiments in Section V.

Each remote site computes its optimal threshold value every recompute interval. The optimal local thresholds are computed by performing a linear search in the range 0 to T. This takes $O(T)$ running time. We can reduce the running time to $O(\delta)$ by searching for the optimal threshold value in a small range $[T_i - \delta, T_i + \delta]$ in each round.

Ensuring correctness: If each remote site is allowed to independently decide on their local threshold values, we will not be able to ensure correctness; i.e. $\sum_{i=1}^{n} T_i \leq T$ cannot be guaranteed. A simple method to ensure correctness is to restrict each remote site's local to a maximum of T/n. This however can result in poor performance in cases where one site's value is very high on average compared to other sites.

In order to ensure that the sum of the threshold values is bounded by T, each remote site sends the computed optimal local threshold value, T_i, to the coordinator every recompute interval. The coordinator determines if $\sum_{i=1}^{n} T_i \leq T$. If not, it reduces each threshold value T_j by $\frac{T_j}{\sum_{i=1}^{n} T_i} (\sum_{i=1}^{n} T_i - T)$. This ensures that $\sum_{i=1}^{n} T_i \leq T$.

Communication costs: Apart from local alarms and global polls, the Markov based algorithm sends $2n$ control messages every recompute interval to ensure correctness. Each remote site sends its calculated threshold value and the coordinator sends either modified threshold values or validates the threshold values calculated by the remote sites. These control messages are very light weight as compared to the control messages sent by the remote sites in the brute force algorithm.

[5]Other estimates using Hoeffding and Chebyshev bounds do not yield better results.

LOCALALARMACTION$(i, T, n, \alpha, \rho_i)$

1: With probability $\min(1, \frac{1}{\rho_i})$, $T_i \leftarrow \alpha \times T_i$

GLOBALPOLLACTION$(i, T, n, \alpha, \rho_i)$

1: With probability $\min(1, \rho_i)$, $T_i \leftarrow \frac{T_i}{\alpha}$

Fig. 2: Reactive threshold assignment algorithm

D. Reactive algorithm

The reactive algorithm adjusts local threshold values at the remote sites based on local alarm and global poll events that occur in the system. Each local alarm from a remote site indicates that the threshold value at the remote site is possibly lower than optimum and each global poll indicates that the threshold value is higher than optimum. The basic reactive scheme at each remote sites adapts the thresholds at the remote sites based on these events using the algorithm shown in Figure 2.

Whenever there is a local alarm, the site increases the threshold value by a factor α with a probability $1/\rho_i$ (or 1, if $1/\rho_i$ is greater than 1), where α and ρ_i are parameters of the system greater than 0. Whenever there is a global poll each remote site reduces the threshold value by a factor of α with a probability ρ_i (or 1, if ρ_i is greater than 1). α is a constant that determines the rate of convergence and can typically take values in the range $(1, 1.2]$. Choosing an α value that is too small leads to bad performance since it does not converge fast enough, while choosing a large α leads to large oscillations in threshold values. We choose $\alpha = 1.1$ in our experiments (which we found to be the best).

If we want the remote site to take the optimal local threshold T_i^{opt}, then setting $\rho_i = \frac{P_l(T_i^{opt})}{P_g^{opt}}$ will achieve this - here $P_l(T_i^{opt})$ is the probability of a local alarm when the local threshold is T_i^{opt} and P_g^{opt} is the probability of a global poll when all remote sites take the optimal threshold values. The reason for this is that if the system is not at T_i^{opt} at all sites, then we can show that at some site either (1) current threshold $T_i' > T_i^{opt}$, $P_l(T_i') < P_l(T_i^{opt})$ and $P_g(T_i') > P_g(T_i^{opt})$, or (2) current threshold $T_i' < T_i^{opt}$, $P_l(T_i') > P_l(T_i^{opt})$ and $P_g(T_i') < P_g(T_i^{opt})$. Let us look at the first case; if at some site $T_i' > T_i^{opt}$, $P_l(T_i') < P_l(T_i^{opt})$ and $P_g(T_i') > P_g(T_i^{opt})$:

$$\frac{P_l(T_i')}{P_g(T_i')} < \frac{P_l(T_i^{opt})}{P_g(T_i^{opt})} \text{ and } P_l(T_i') < \rho_i P_g(T_i')$$

Hence, the average number of observed local alarms is lesser than ρ_i times the average number of observed global polls. Thus, the threshold value decreases over time from T_i'.

We can similarly argue that the threshold value will increase if the threshold is lesser than T_i^{opt}. So, the stable state of the system is around T_i^{opt} using the reactive algorithm. This argument ignores other interactions in the system such as other sites varying their thresholds, thus affecting the observed P_g. We conjecture that the system converges to the desired value of T_i even in the presence of these interactions and our experiments corroborate this. Observe that, we introduce randomness by increasing and reducing thresholds probabilistically and this desynchronizes threshold changes at the remote sites and helps in convergence.

Since determining the optimum T_i^{opt} and the corresponding P_g values is not feasible, we propose to use the Markov-based scheme to identify the threshold value that gives the minimum cost estimate and use this value to compute the contribution of the remote site to P_g. Every recompute interval, the remote site then sends this component of P_g to the coordinator. The coordinator sums the components of P_g it receives from the remote sites and computes the P_g value. The coordinator sends this value of P_g to the remote sites. Each remote site uses this value of P_g to compute the value of ρ_i for the reactive scheme. Note that the P_l used in the computation of ρ_i is for the threshold value that gives the minimum cost according to the Markov-based scheme.

Comparison with Markov algorithm: The Markov algorithm does not perform well because it sets the local thresholds to very low values. However, in the reactive scheme, the remote sites see far less global polls than is estimated by the Markov scheme and hence, sets higher threshold values than the Markov scheme. Thus, the reactive scheme is able to perform much better than the Markov scheme in our experiments.

Let T_i^{est} be the threshold value at remote site i determined by the Markov scheme and T_i^{real} be the threshold value where the system actually converges. Let $est(P_g)$ be the Markov estimate of the global poll probability, $P_g(T_i^{real})$ be the real global poll probability observed in the system at T_i^{real} and $P_g(T_i^{est})$ be the real global poll probability at T_i^{est}. Also, we have that $P_g(T_i^{est}) \leq est(P_g)$ since Markov overestimates P_g. By definition,

$$\rho_i = \frac{P_l(T_i^{est})}{est(P_g)} = \frac{P_l(T_i^{real})}{P_g(T_i^{real})}$$

If $T_i^{real} < T_i^{est}$, we have:

$$P_l(T_i^{real}) > P_l(T_i^{est}) \text{ and } P_g(T_i^{real}) < P_g(T_i^{est}) \leq est(P_g)$$

Thus,

$$\frac{P_l(T_i^{est})}{est(P_g)} < \frac{P_l(T_i^{real})}{P_g(T_i^{real})}$$

Hence, we have a contradiction. Thus, $T_i^{real} \geq T_i^{est}$. In reality, Markov's estimate of P_g is much higher than the real P_g observed in the system and hence, the system converges to a threshold T_i^{real} that is significantly higher than the threshold T_i^{est} determined by the Markov-based algorithm.

Ensuring correctness: The coordinator is always aware of the latest threshold value at each remote site - this is because every time there is a local alarm, the remote site informs the coordinator of the value that caused the local alarm along with the new threshold value at that remote site. Therefore, whenever local alarms cause $\sum_{i=1}^{n} T_i > T$ at the coordinator, the resulting global polls reduce thresholds until $\sum_{i=1}^{n} T_i \leq T$.

Communication costs: Apart from local alarms and global polls, every recompute interval the scheme sends $2n$ control messages. Each site communicates its contribution to the global poll probability at its estimate of the optimum based on the Markov-estimate, the coordinator then adds up all these estimates from the remote sites and broadcasts this value to all the remote sites which then use it to compute the local ρ value.

Comparison with the scheme presented by Olston et al [12]: While our adaptive algorithm is inspired by the probabilistic scheme in [12], our problem setting is significantly more complex.

- [12] tackles the problem of setting the precision of interval approximations for a single cached numeric value. Setting too narrow an interval leads to frequent value refreshes (whenever the modified value exceed the interval), while too wide an interval leads to query-initiated refreshes due to query precision requirements not being met. The authors make simplifying assumptions (value follows a random walk, query precision is random) to derive clean formulas for the refresh probabilities in terms of the interval width, and use this to compute the optimal interval width that minimizes communication cost.
- In contrast, in our distributed constraint setting, system behavior (e.g., global polls) is determined by multiple values at the various sites, and not a single value as in [12]. While the local alarm probability $P_l(i)$ depends only on the local value distribution and threshold T_i at site i, the global poll probability P_g is an extremely complex function of the (different) local values and thresholds at the different sites. The interdependency among the multiple values makes our local threshold selection problem a lot more challenging. For e.g., computing optimal local threshold values requires exhaustive enumeration with complexity $O(T^n)$.
- A key and novel technical contribution of ours is the use of Markov's Inequality to obtain a simple upper bound that expresses P_g as the sum of n independent components, one per site involving the local variable plus threshold at the site. Thus, optimal threshold values (that minimize communication costs) can be computed locally and independently by the sites without assistance from the coordinator. This is novel and specific to our distributed constraint setting - it not required in [12] because there refresh probabilities depend on a single interval width, and not multiple values.

E. Computational Overhead

The computation overhead has two components - the cost incurred at each remote site and the cost incurred at the central coordinator. If we maintain full histograms at each remote site, then the computational cost for each scheme is as follows (for each recompute interval):

- Markov based scheme: $O(T)$ cost at each remote site, $O(n)$ cost at the coordinator.
- Brute force scheme: $O(T)$ cost at each remote site, $O(nT^{n+2})$ at the coordinator. With the optimizations we suggest the computational overhead at the coordinator reduces to $O(n^2T^2 + n\delta)$ (where threshold values are allowed to vary in a O(δ) range). Since we expect O(δ) to be small, the cost at the coordinator for this scheme is $O(n^2T^2)$.
- Reactive scheme: $O(T)$ cost at each remote site, $O(n)$ cost at the coordinator.

As expected, the brute force scheme has a huge computational overhead and is intractable for asymptotic n and T. The other schemes are less computationally intensive. Note that although the cost depends on T since this comparison assumes we use full-size histograms; in practice however, we will only use approximate histograms.

V. Experiments

We performed extensive experiments, using multiple real-world traces and also using synthetic data, to evaluate the performance of our non-zero slack schemes and to explore properties of our algorithms. When using our schemes to set thresholds at monitors for both real-world data and synthetic data, we observed significant savings (40% to 90%) in the number of messages, over using the state of the art zero-slack geometric scheme in [14]. We also found that the savings in communication overhead when using our non-zero slack schemes increases as the number of monitoring nodes in the system increases. Our experimental results indicate that amongst the non-zero slack schemes that we suggest, the reactive scheme is the best in terms of performance across different datasets and also in terms of scalability. Due to space constraints, we only present a small subset of our experimental results. Please see the full paper [1] for the complete set of experimental results.

A. Experimental Setup

For our experiments, we consider a monitoring application that monitors the total amount of traffic flowing into a service provider network. Our monitoring setup consists of getting information about the ingress traffic of the network. This information can be derived by deploying passive monitors at each link or by collecting flow information (e.g. Netflow records) from the ingress routers. Each monitor aggregates the information (packet level or flow level) to derive the total amount of traffic (in bytes in this experiment) coming into the network through that ingress point. The distributed constraint monitors the total amount of traffic flowing into the network across all ingress links and throws an alarm if this amount exceeds a certain threshold.

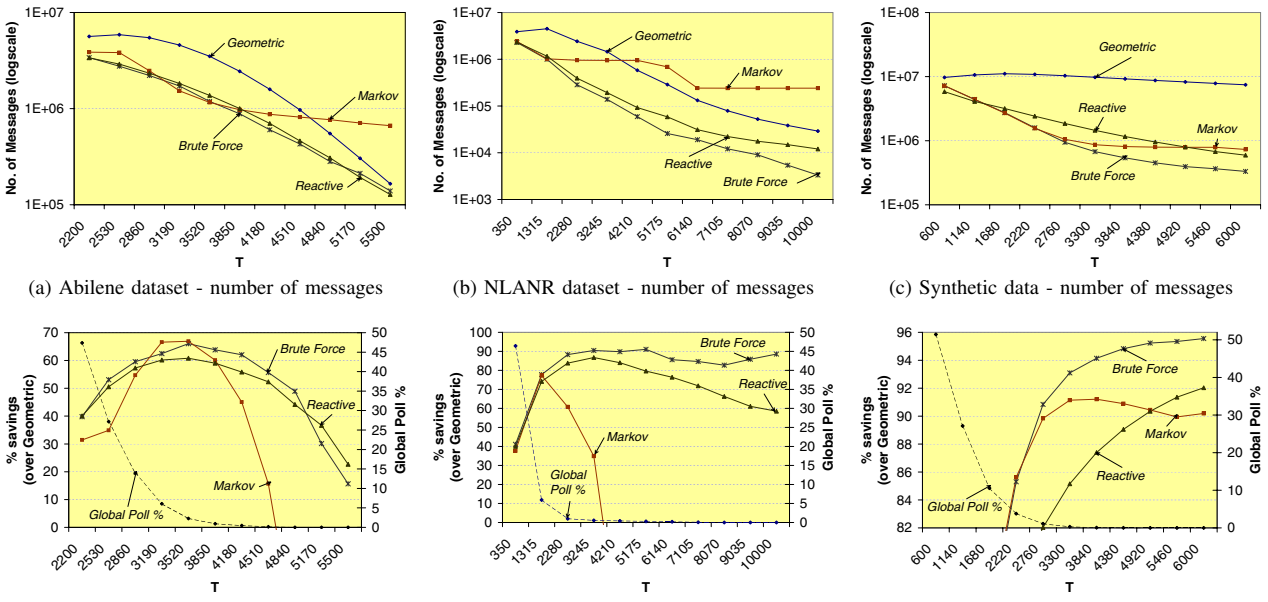
In our experiments, we use the number of messages sent by each scheme to track the constraint as the performance metric. We used a recompute interval of 1000 rounds for each scheme. For the bruteforce scheme, we used $\delta = 10$. For the reactive scheme, we used $\alpha = 1.1$. For all schemes we used histograms that represent 5% of their original range.

B. Datasets

Abilene: Our first set of Netflow traces were obtained from the Abilene network, which is an Internet2 high-performance backbone network with 11 routers located across the US [4]. We used traces from +1000hrs to +1100hrs UTC on August 15th, 2006. The Netflow records show a total of 73.3 million packets during this period. These packets were seen across the 11 Abilene router nodes; the Chicago location saw the most packets (9.6 million) while the Seattle location saw the least packets (2.2 million). Therefore the Abilene dataset gave us a real-life, large scale and *naturally* distributed dataset. We scaled down all packet sizes by a factor of 100. Although the results we present are for this selected time frame, we performed several experiments by looking at data that spanned weekend/weekday transitions, different days of the week as well as different times of day. We obtained very similar results for all these cases.

NLANR: We also use publicly available link traces from NLANR [11] as input to our distributed monitoring system. This trace was collected with an NLANR PMA OC192MON located on SDSC's TeraGrid Cluster, from +0000hrs UTC to +0100hrs UTC on February 18th, 2004. The trace contained a total of 21 million packets. These traces are for a single ingress link, and we transform this data for our distributed system by assigning a probability distribution for distributing packets randomly to the various monitors. By using different probability distributions, we can simulate various scenarios that can occur in real networks. A uniform distribution implies that any packet is equally likely to go to any of the monitoring nodes. A skewed distribution distributes packets unevenly and a few nodes receive more packets than others. For ease of presentation and also because we have a dataset that is naturally separated in Abilene, we only present the case where data is uniformly distributed across the different routers in this case. We also present results for experiments where we vary the number of monitoring nodes (from 10 to 160) across which this data is distributed. We scaled down all packet sizes by a factor of 10.

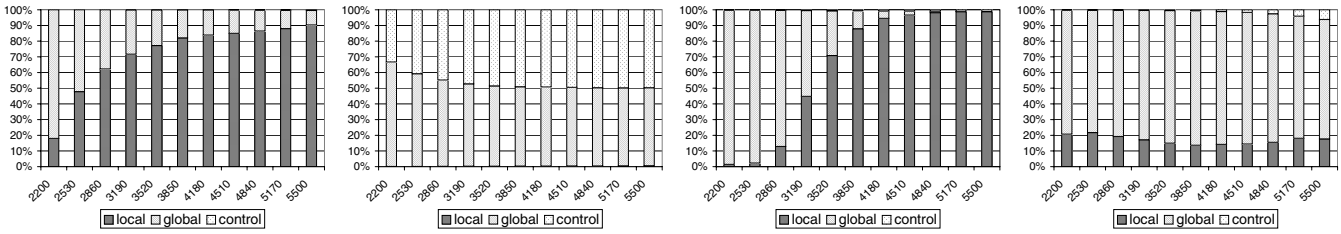
Synthetic: Finally, we also used synthetic datasets to evaluate parameters that could not be controlled using real-world datasets. For instance, to study the effect of changing the number of monitors (from 10 to 160) on the observed savings. Values at each monitor were generated independently at random in the range $[0, 1500]$ using a Zipf distribution with a randomly chosen Zipf exponent between 1.0 and 2.0. We ran experiments where all the monitors had the same



(a) Abilene dataset - number of messages (b) NLANR dataset - number of messages (c) Synthetic data - number of messages

(d) Abilene dataset - percentage savings (e) NLANR dataset - percentage savings (f) Synthetic data - percentage savings

Fig. 3: Number of messages sent in tracking events at various value of T and Percentage Savings in number of messages over the geometric scheme.



(a) Brute Force (b) Geometric (c) Markov (d) Reactive

Fig. 4: Bar charts showing the breakup of cost for the various schemes while tracking various events for the Abilene dataset.

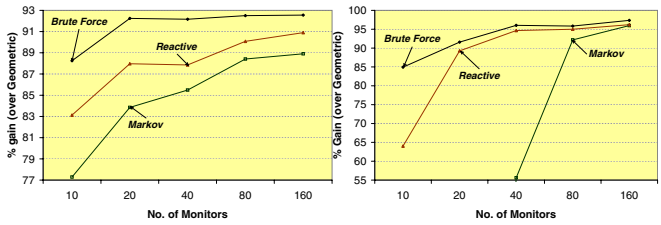
behavior as well as when they all had different behavior. Unless specifically mentioned, we used $n = 20$ monitors.

C. Results

Comparison of message overhead: Figure 3a compares the number of messages sent by the various threshold setting mechanisms, while Figure 3d compares the percentage gain in number of messages of the various schemes over using the geometric scheme for the Abilene dataset. In each plot we vary the threshold values (T) on the x-axis in a way that we have the true global poll percentage (i.e. the percentage of the number of true global constraint violations) vary from 50% to 0%. Notice that we get a 40%-90% improvement over the geometric scheme using our reactive scheme. Also notice that the reactive scheme performs close to the bruteforce approach throughout the range of T values. Notice that as the event we are tracking becomes rare, the performance of the Markov scheme degrades rapidly and it actually performs much worse than the geometric scheme. Our reactive scheme does not suffer from these drawbacks. This is because Markov overestimates global poll probability and sets lower thresholds, while (as argued earlier) in the reactive case, thresholds converge to a higher value. These savings in plots *include* the various costs incurred by the schemes in computing the thresholds and communicating them to the monitors. One explanation for the drop off in gain while tracking rare events is that the control overhead of schemes begin to dominate the overall cost since the number of local alarms and global polls in these cases is small. Figures 3b and 3e show similar plots for the NLANR dataset. Figures 3c and 3f show similar results for the synthetic dataset.

As we mentioned earlier, while the bruteforce approach shows very good performance it involves periodically shipping site-histograms and therefore may not be very practical. We use its performance more as an indicator of the optimum performance (that a histogram-based scheme might achieve).

Breakdown of message overhead: Figure 4 shows the breakup of cost for the various schemes while tracking events in the Abilene dataset. Each stacked chart breaks down the message overhead in terms of local alarms, global polls and control overhead. In general, if a scheme sets lower thresholds than another, then it will cause more local alarms than global polls compared to the other scheme. The Markov scheme (Figure 4c) (especially at higher values of T - i.e. when tracking rare events), overestimates the global poll probability and sets very low thresholds. As a result, the number of global polls is very low. However, the scheme



(a) Zipf, T = 50n (b) NLANR, T = 9035

Fig. 5: Percentage gain over the geometric scheme as we vary the number of monitoring sites.

suffers due to the large number of local alarms. The reactive scheme (Figure 4d) and the bruteforce scheme (Figure 4a) are able to strike a good balance here and this is reflected in their performance - where these two schemes consistently perform the best in the entire range of T values (Figure 3a). The cost of the geometric scheme (Figure 4b) is comprised entirely of global polls and control overhead - this is because it maintains zero slack and consequently every local alarm results in a global poll. The bruteforce scheme has relatively low control overhead because we employ KL-distance based histogram shipping. Another interesting observation is that at higher values T, control overhead begins to look significant due to the smaller number of local alarms and global polls at these values.

Effect of scale - varying the number of monitoring nodes: Our analytical result from Theorem 2 suggests that the benefit from using a non-zero slack scheme over using a zero-slack scheme should increase as the number of monitoring nodes in the system increases. The theorem of course makes assumptions that might not always hold in practice. We did experiments to determine the savings in communication using non-zero slack schemes with increasing number of monitoring nodes. Our first experiment was with the synthetic dataset, where we varied the number of monitoring nodes from 10 to 160, and each site chose a random Zipf exponent in the range of 1.0 to 2.0. Our second experiment was with the monolithic NLANR dataset, where we varied the number of monitoring nodes from 10 to 160 and we distributed the packets in the trace by randomly assigning them to the monitors. In both experiments, we were interested in tracking an event that happens at most once during the entire trace. Figure 5a shows the results for the synthetic experiment and Figure 5b shows the results for the NLANR experiment. Notice that in both cases there is an increase in percentage gain over the geometric scheme as the number of monitoring nodes increases. The most interesting result from these experiments was that our Markov-based scheme starts performing very well for the NLANR dataset as the number of monitoring nodes increases. Notice that in 5b, the Markov-based scheme goes from performing much worse than the geometric scheme (this portion of the Markov plot is cut off) when $n = 10$, to showing a 95% savings in communication cost when $n = 160$. This is because in the NLANR dataset we keep the T value constant even as we increase n - since the event we are tracking remains the same in this case. Markov's estimate of the optimum threshold and the actual optimum threshold are actually close by because the range of good threshold values shrinks as n increases and T remains the same. We point out that while the bruteforce approach shows good results, its computation took a long time at high values of n.

VI. Future Work

In this paper, we have studied the implementation of a simple distributed constraint, sum of variables. It would be interesting to generalize the observation that non-zero slack methods can result in better performance for general functions (like join sizes, quantiles etc.) using the framework proposed by [14]. Reference [10] suggested a novel tracking problem called cumulative triggers and it would be interesting to see how our methods perform when applied to their problem. In typical networks, nodes can be organized in a hierarchical structure that can be exploited to further reduce communication required in implementing distributed constraints. Studying non-zero slack algorithms for such structured networks presents an interesting area of future research too.

References

[1] S. Kashyap, J. Ramamirtham, R. Rastogi, and P. Shukla. Efficient Constraint Monitoring Using Adaptive Thresholds. Bell Labs Technical Memorandum, ITD-06-47318H, 2006.

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

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

[4] Abilene Observatory. http://abilene.internet2.edu/.

[5] S. Agrawal, S. Deb, KVM Naidu, and R. Rastogi. Efficient detection of distributed constraint violations. In *ICDE 2007*.

[6] M. Dilman and D. Raz. Efficient reactive monitoring. In *INFOCOM 2001*.

[7] L. Huang, M. Garofalakis, J. Hellerstein, A. Joseph, and N. Taft. Toward sophisticated detection with distributed triggers. In *MineNet 2006*.

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

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

[10] A. D. Joseph, L. Huang, M. Garofalakis and N. Taft. Communication-efficient tracking of distributed cumulative triggers. ICDCS 2007.

[11] National Laboratory for Applied Network Research. http://www.nlanr.net/.

[12] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *SIGMOD 2001*.

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

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