State Monitoring in Cloud Datacenters

Shicong Meng, *Student Member*, *IEEE*, Ling Liu, *Senior Member*, *IEEE*, and Ting Wang, *Student Member*, *IEEE*

Abstract—Monitoring global states of a distributed cloud application is a critical functionality for cloud datacenter management. State monitoring requires meeting two demanding objectives: high level of correctness, which ensures zero or low error rate, and high communication efficiency, which demands minimal communication cost in detecting state updates. Most existing work follows an instantaneous model which triggers state alerts whenever a constraint is violated. This model may cause frequent and unnecessary alerts due to momentary value bursts and outliers. Countermeasures of such alerts may further cause problematic operations. In this paper, we present a WIndow-based StatE monitoring (WISE) framework for efficiently managing cloud applications. Window-based state monitoring reports alerts only when state violation is continuous within a time window. We show that it is not only more resilient to value bursts and outliers, but also able to save considerable communication when implemented in a distributed manner based on four technical contributions. First, we present the architectural design and deployment options for window-based state monitoring with centralized parameter tuning. Second, we develop a new distributed parameter tuning scheme enabling WISE to scale to much more monitoring nodes as each node tunes its monitoring parameters reactively without global information. Third, we introduce two optimization techniques, including their design rationale, correctness and usage model, to further reduce the communication cost. Finally, we provide an in-depth empirical study of the scalability of WISE, and evaluate the improvement brought by the distributed tuning scheme and the two performance optimizations. Our results show that WISE reduces communication by 50-90 percent compared with instantaneous monitoring approaches, and the improved WISE gains a clear scalability advantage over its centralized version.

Index Terms—State monitoring, datacenter, cloud, distributed, aggregation, tuning.

1 Introduction

Cloud datacenters represent the new generation of datacenters that promote on-demand provisioning of computing resources and services. Amazon's Elastic Computer Cloud (EC2) [1] is an example of such cloud datacenters. A typical cloud application in such cloud datacenters may spread over a large number of computing nodes. Serving cloud applications over multiple networked nodes also provides other attractive features, such as flexibility, reliability, and cost-effectiveness. Thus, state monitoring becomes an indispensable capability for achieving on-demand resource provisioning in cloud datacenters. However, the scale of cloud datacenters and the diversity of application specific metrics pose significant challenges on both system and data aspects of datacenter monitoring for a number of reasons.

First, the tremendous amount of events, limited resources and system failures often raise a number of system-level issues in datacenter monitoring:

- *Event capturing*. Applications, OS, servers, network devices can generate formidable amount of events, which makes directly storing and searching these events infeasible. To address this issue, Bhatia et al. [2] proposed Chopstix, a tool that uses approximate data collection techniques to efficiently collect a rich set of system-wide data in large-scale production systems.
- *Resource consumption*. Servers usually have limited resources available for monitoring. Assigning monitoring tasks and organizing monitoring overlays without considering this fact may lead to unreliable monitoring results. Jain et al. [3] proposed a self-tuning monitoring overlay to trade precision and workload. Mengy et al. [4] studied the problem of monitoring network construction for multiple monitoring tasks without overloading member hosts.
- *Reliability*. Failures of server, network links can lead to inconsistent monitoring results. Jain et al. [5] introduced and implemented a new consistency metric for large-scale monitoring. The new metric indicates the precision of monitoring results, and thus, can identify inconsistent results caused by system failures.

Second, large-scale monitoring often involves processing large amount of monitoring data in a distributed manner. Such computing paradigm also introduces several challenges at the data management level:

- *Distributed aggregation*. The ability of summarizing information from voluminous distributed monitored values is critical for datacenter monitoring. Previous work proposed several efficient algorithms for different aggregation over distributed stream values. Babcock and Olston [6] studied the problem

- *S. Meng is with the College of Computing, Georgia Institute of Technology, 37975 GT Station, Atlanta, GA 30332. E-mail: smeng@cc.gatech.edu.*
- *L. Liu is with the College of Computing, Georgia Institute of Technology, KACB, room 3340, Georgia Tech, 266 Ferst Dr., Atlanta, GA 30332-0765. E-mail: lingliu@cc.gatech.edu.*
- *T. Wang is with the College of Computing, Georgia Institute of Technology, 329544 Georgia Tech Station, Atlanta, GA 30332-1275. E-mail: twang@cc.gatech.edu.*

Manuscript received 30 Oct. 2009; revised 31 Jan. 2011; accepted 23 Feb. 2011; published online 18 Mar. 2011.
Recommended for acceptance by D. Lomet.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-2009-10-0749.
Digital Object Identifier no. 10.1109/TKDE.2011.70.

of monitoring top-k items over physically distributed streams. Olston et al. [7] introduced an efficient algorithm for computing sums and counts of items over distributed streams. As its distinct feature, the proposed algorithm can achieve efficiency by trading precision for communication overhead. Cormode et al. [8] proposed an approach for approximate quantile summaries with provable approximation guarantees over distributed streams.

- *Shared aggregation*. Different monitoring tasks may share some similarities. Running similar tasks in an isolated manner may lead to unnecessary resource consumption. Krishnamurthy et al. [9] developed techniques for binding commonalities among monitoring queries and sharing work between them.

In this paper, we study state monitoring at cloud datacenters, which can be viewed as a cloud state management issue, as it mainly involves collecting local state information and evaluating aggregated distributed values against predefined monitoring criteria. A key challenge for efficient state monitoring is meeting the two demanding objectives: high level of correctness, which ensures zero or very low error rate, and high communication efficiency, which requires minimal communication cost in detecting critical state violation.

1.1 State Monitoring

Despite the distributed nature of cloud-hosted applications, application owners often need to monitor the global state of deployed applications for various purposes. For instance, Amazon's CloudWatch [10] enables users to monitor the overall request rate on a web application deployed over multiple server instances. Users can receive a state alert when the overall request rate exceeds a threshold, e.g., the capacity limit of provisioned server instances. In this case, users can deploy the web application on more server instances to increase throughput.

As another example, service providers who offer software-as-a-service to organizations often need to perform distributed rate limiting (DRL) to restrict each organization to use the software within its purchased level (e.g., 100 simultaneous sessions). Because software services are usually deployed over distributed servers in one or multiple datacenters, they require DRL to check if the total number of running sessions from one organization at all servers is within a certain threshold.

We refer to this type of monitoring as *state monitoring*, which continuously evaluates if a certain aspect of the distributed application, e.g., the overall request rate, deviates from a normal state. State monitoring is widely used in many applications. Examples also include:

Example 1. Traffic engineering: monitoring the overall traffic from an organization's subnetwork (consists of distributed hosts) to the Internet.

Example 2. Quality of service: monitoring and adjusting the total delay of a flow which is the sum of the actual delay in each router on its path.

Example 3. Fighting DoS attack: detecting DoS attack by counting SYN packets arriving at different hosts within a subnetwork.

Example 4. Botnet detection: tracking the overall simultaneous TCP connections from a set of hosts to a given destination.

State monitoring in datacenters poses two fundamental requirements. First, given the serious outcome of incorrect monitoring results, state monitoring must deliver correct monitoring results [11]. A false state alert in the previous CloudWatch example would cause provisioning of new server instances which is clearly unnecessary and expensive. Missing a state alert is even worse as the application gets overloaded without new server instances, which eventually causes potential customers to give up the application due to poor performance. This correctness requirement still holds even if monitored values contain momentary bursts and outliers.

Second, communication related to state monitoring should be as little as possible [12], [13], [14]. Data centers usually run a large number of state monitoring tasks for application and infrastructure management [1]. As monitoring communication consumes both bandwidth and considerable CPU cycles [4], state monitoring should minimize communication. This is especially important for infrastructure services such as EC2, as computing resources directly generate revenues.

One intuitive state monitoring approach is the instantaneous state monitoring, which triggers a state alert whenever a predefined threshold is violated. This approach, though makes algorithm design easy, idealizes real world monitoring scenarios. As unpredictable short-term bursts in monitored values are very common for Internet applications [15], [16], [17], instantaneous state monitoring may cause frequent and unnecessary state alerts. In the previous example, momentary HTTP request bursts trigger unnecessary state alerts whenever their rates exceed the threshold. Furthermore, since state alerts usually invoke expensive countermeasures, e.g., allocating and deploying new web server instances, unnecessary state alerts may cause significant resource loss. Surprisingly, we find most of the existing work to date [18], [7], [19], [20], [21], [22] deals only with this type of state monitoring.

1.2 Overview of Our Approach

In this paper, we introduce the concept of window-based state monitoring and devise a distributed *WIndow-based StatE monitoring* (WISE) framework for cloud datacenters. Window-based state monitoring triggers state alerts only when observing *continuous* state violation within a specified time window. It is developed based on the widely recognized observation that state violation within a short period may simply indicate the dynamics of the runtime system and it does not necessarily trigger a global state violation. Thus, with the persistence checking window, window-based state monitoring gains immunity to momentary monitoring value bursts and unpredictable outliers.

In addition to filtering unnecessary alerts, window-based state monitoring explores monitoring time windows at distributed nodes to yield significant communication savings. Although the window-based state monitoring approach was first introduced in [23], the focus of our earlier results was mainly on the basic approach to

window-based state monitoring with centralized parameter tuning to demonstrate and evaluate its advantage in monitoring cost saving compared to instantaneous state monitoring. In this paper, we identify that this basic approach to window-based state monitoring may not scale well in the presence of lager number of monitoring nodes. We present an improved window based monitoring approach that improves our basic approach along several dimensions. First, we present the architectural design of the WISE system and its deployment options (Section 3.1). Second, to address the scalability issue of the basic WISE, we develop a distributed parameter tuning scheme to support large scale distributed monitoring (Section 5.4). This distributed scheme enables each monitoring node to search and tune its monitoring parameters in a reactive manner based on its observations of state update events occurred, without requiring global information. It enables WISE to scale to a much larger number of nodes compared with the centralized scheme. Third, we design two concrete optimization techniques, aiming at minimizing the communication cost between a coordinator and its monitoring nodes. The first optimization is dedicated to enhance the effectiveness of the global pull procedure at the coordinator by reducing the communication cost for global pulls, while ensuring the correctness of the monitoring algorithm. The second optimization aims at reducing unnecessary global polls by reporting more information of local violations at monitoring nodes (Section 6). Finally, we have conducted extensive empirical studies on the scalability of the distributed parameter tuning scheme compared to the centralized scheme appeared first in [23], and evaluated the effectiveness of both the distributed WISE solution and the two optimization techniques, compared to the basic WISE approach (Section 7.2).

In summary, this paper makes three unique contributions. First, WISE employs a novel distributed state monitoring algorithm that deploys time windows for message filtering and achieves communication efficiency by intelligently avoiding collecting global information. More importantly, it also guarantees monitoring correctness. Second, WISE uses a distributed parameter tuning scheme to tune local monitoring parameters at each distributed node and uses a sophisticated cost model to carefully choose parameters that can minimize the communication cost. As a result, this scheme scales much better than the centralized scheme presented in [23]. Last but not the least, we develop a set of optimization techniques to optimize the performance of the fully distributed WISE.

We conducted extensive experiments over both real world and synthetic monitoring traces, and show that WISE incurs a communication reduction from 50 to 90 percent compared with existing instantaneous monitoring approaches and simple alternative window based schemes. We also compare the original WISE with the improved WISE on various aspects. Our results suggest that the improved WISE is more desirable for large-scale datacenter monitoring.

1.3 Outline

The rest of this paper is organized as follows: Section 2 introduces the preliminaries and defines the problem of window-based state monitoring. Section 3 gives an overview of our approach. Section 4 presents the detail of the WISE monitoring algorithm. Section 5 describes our scalable parameter setting scheme. We discuss optimization techniques to further improve the performance of WISE in Section 6. Section 7 presents the experimental evaluation. Section 8 discusses the related work. We conclude the paper in Section 9 with a summary and an outline of our future work.

2 Preliminaries

We consider a state monitoring task involving a set N of nodes where $|N| = n$. Among these n nodes, one is selected to be a coordinator which performs global operations such as collecting monitored values from other nodes and triggering state alerts. For a given monitoring task, node i locally observes a variable v_i which is continuously updated at each time unit. The value of v_i at time unit t is $v_i(t)$ and we assume $v_i(t)$ is correctly observed. When necessary, each monitor node can communicate with the coordinator by sending or receiving messages. We consider that communication is reliable and its delay is negligible in the context of datacenter state monitoring. As communication cost is of concern, we are interested in the total number of messages caused by monitoring. We also consider the size of messages in our experiment.

A state monitoring task continuously evaluates a certain monitored state is normal or abnormal. Similar to previous work [18], [7], [19], [20], [21], [22], we distinguish states based on sum aggregate of monitored values. For instance, we determine whether a web application is overloaded based on the sum of HTTP request rates at different hosts. We use sum aggregates because they are widely applied and also simplify our discussion, although our approach supports any aggregate that linearly combines values from nodes.

2.1 The Instantaneous State Monitoring

The instantaneous state monitoring model [18], [7], [19], [20], [21], [22] detects state alerts by comparing the current aggregate value with a global threshold. Specifically, given $v_i(t), i \in [1, n]$ and the global threshold T, it considers the state at time t to be abnormal and triggers a state alert if $\sum_{i=1}^{n} v_i(t) > T$, which we refer to as *global violation*.

To perform instantaneous state monitoring, the line of existing work decomposes the global threshold T into a set of local thresholds T_i for each monitor node i such that $\sum_{i=1}^{n} T_i \leq T$. As a result, as long as $v_i(t) \leq T_i, \forall i \in [1, n]$, i.e., the monitored value at any node is lower or equal to its local threshold, the global threshold is satisfied because $\sum_{i=1}^{n} v_i(t) \leq \sum_{i=1}^{n} T_i \leq T$. Clearly, no communication is necessary in this case. When $v_i(t) > T_i$ on node i, it is possible that $\sum_{i=1}^{n} v_i(t) > T$ (global violation). In this case, node i sends a message to the coordinator to report *local violation* with the value $v_i(t)$. The coordinator, after receiving the local violation report, invokes a *global poll* procedure where it notifies other nodes to report their local values, and then determines whether $\sum_{i=1}^{n} v_i(t) \leq T$. The focus of existing work is to find optimal local threshold values that minimize the overall communication cost.

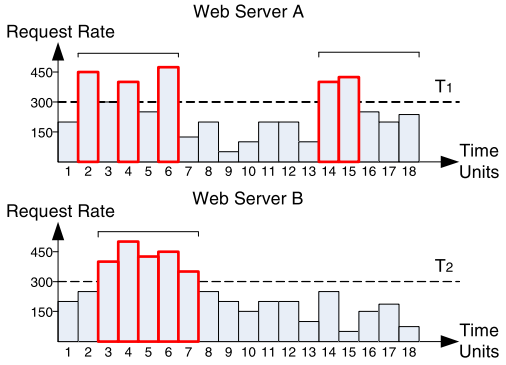


Fig. 1. A motivating example.

2.2 The Window-Based State Monitoring

As monitored values often contain momentary bursts and outliers, instantaneous state monitoring [16] is subject to cause frequent and unnecessary state alerts, which could further lead to unnecessary countermeasures. Since short periods of state violation are often well acceptable, a more practical monitoring model should tolerate momentary state violation and capture only continuous one. Therefore, we introduce window-based state monitoring which triggers state alerts only when the normal state is *continuously* violated for L time units.

We study window-based state monitoring instead of other possible forms of state monitoring for two reasons. First, we believe continuous violation is the fundamental sign of established abnormality. Second, window-based state monitoring tasks are easy to configure, because the window size L is essentially the tolerable time of abnormal state, e.g., degraded service quality, which is known to service providers.

2.3 Problem Definition

Our study focuses on finding efficient ways to perform distributed window-based state monitoring, as this problem is difficult to solve and, to the best of our knowledge, has not been addressed before. Formally, we define the distributed window-based state monitoring problem as follows:

Problem Statement 1. *Given the threshold T, the size L of the monitoring window, and n monitor nodes with values $v_i(t), i \in [1, n]$ at time t, devise an algorithm that triggers state alerts only when $\sum_{i=1}^{n} v_i(t-j) > T, \forall j \in [0, L-1]$ at any t while minimizing the associated communication cost.*

Solving this problem, however, is challenging, as it requires careful handling of monitoring windows at distributed nodes to ensure both communication efficiency and monitoring correctness. Simple solutions such as applying modified instantaneous monitoring approaches either fail to minimize communication or miss state alerts. We next present a motivating example to show the reason as well as some insights into the solution.

Fig. 1 shows a snippet of HTTP request rate traces collected from two web servers in a geographically distributed server farm [24], where time is slotted into 5-second units. Let us first consider an instantaneous monitoring task which triggers state alerts when the sum of request rates at two servers exceeds $T = 600$. For simplicity, we assume server A and B have the same local thresholds $T_1 = T_2 = T/2 = 300$, as indicated by dashed lines. A local violation happens when a bar raises above a dashed line, as indicated by bars with red borders.

In the example, servers A and B report local violation, respectively, at time unit 2, 4, 6, 14, 15, and time unit 3-7, which generates 10 messages. When receives local violation reports, the coordinator invokes global polls at time unit 2, 3, 5, 7, 14, 15 to collect values from the server that did not report local violation. No global poll is necessary at time unit 4 and 6 as the coordinator knows local values of both servers from their local violation reports. Each global poll includes one message for notification and one message for sending back a local value, and all global polls generate $6 \times 2 = 12$ messages. Thus, the total message number is $10 + 12 = 22$.

2.3.1 Applying Instantaneous Monitoring

Now we perform window-based state monitoring to determine whether there exists continuous global violation against T lasting for $L = 8$ time units. We start with the most intuitive approach, applying the instantaneous monitoring algorithm. Specifically, a monitor node i still evaluates whether $v_i(t) > T_i$ and reports local violation to the coordinator if it is true. The coordinator then invokes a global poll to determine if $\sum v_i(t) > T$. The only difference is that the coordinator triggers state alerts only when observing continuous global violation of 8 time units. As a result, the communication cost is the same as before, 22 messages. Note that 22 messages are generated for only two monitor nodes and all messages have to be processed by the coordinator. Our experiment suggests that the total message number in this scheme grows quickly with increasing monitor nodes. This can cause significant bandwidth and CPU cycle consumption at the coordinator, which limits the scalability of monitoring.

2.3.2 Saving Communication at the Coordinator

In fact, invoking a global poll for every local violation is not necessary. Since state alerts require continuous global violation, the coordinator can delay global polls unless it observes eight continuous time units with local violation. When it observes a time unit t with no local violation, it can clear all pending global polls, as the violation is not continuous, and thus, avoids unnecessary communication. This modified scheme avoids all six global polls, as no local violation exists at time units 8 and 16. Therefore, by avoiding unnecessary communication at the coordinator side, the total message number is reduced to 10.

2.3.3 Saving Communication at Monitor Nodes

Reducing communication at monitor nodes is relatively more difficult. One may propose to let each node report the beginning and the end of a continuous local violation period, instead of reporting for each time unit with local violation. This scheme, which we refer to as *double-reporting*, saves three messages on server B by reporting at times 3 and 7, but performs poorly (eight messages) on server A as each violation period costs two messages, even when it is short (e.g., time units 2, 4, and 6). The total message number is still 10. One may also suggest monitor nodes to report only the end of violation period for less communication. This *end-reporting* scheme, however, fails to ensure monitoring correctness. Assume server A observes local violation

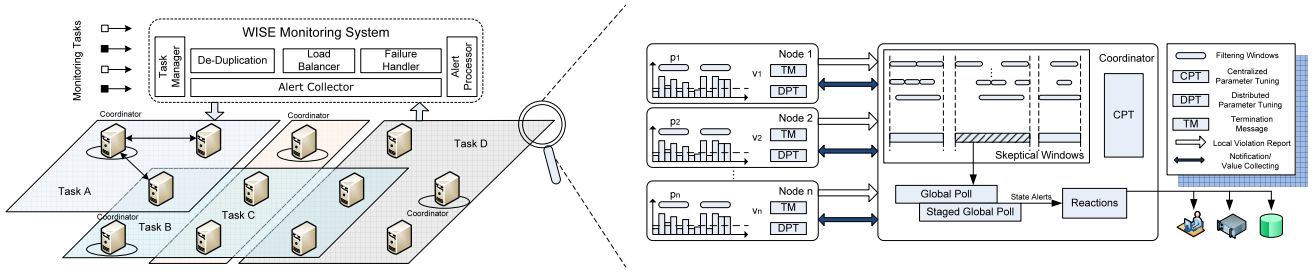


Fig. 2. WISE monitoring system.

throughout time units 2-10 and $\sum v_i(t) > T, \forall t \in [2, 10]$. The coordinator inevitably fails to trigger a state alert at time unit 9 without knowing that server A has started to observe local violation at time unit 2.

2.3.4 Insights and Challenges

One solution is to lower the granularity of local violation reporting, as approximate information on local violation is often adequate to rule out state alerts. Monitor nodes, after reporting one local violation, can employ message filtering time windows with predefined lengths to suppress subsequent local violation reporting messages. For instance, assume both servers A and B use 5-time-unit filtering windows. Server A reports local violation at time unit 2, and then enters a filtering window, during which it avoids to report at time units 4 and 6. Similarly, it reports at time 14 and server B reports once at time 3. At the coordinator side, as filtering windows span 5 time units, the worst case that one reported local violation could imply is a local violation period of 5 time units. Thus, the worst case scenario indicated by the three reports is global violation in time units 2-7 and 14-18, which suggests no state alert exists. The resulting message number is 3, a 86.36 percent communication reduction over 22 messages.

While the above approach seems promising, devising a complete solution requires answers to several fundamental questions. Example questions include how to process reported local violation and filtering windows at the coordinator side to guarantee monitoring correctness? how to tune monitoring parameters, e.g., local threshold and filtering window size, at each node to achieve minimum communication? and how to optimize different subroutines (e.g., global poll) to further reduce communication cost? In addition, datacenter monitoring often requires many tasks, and each task could potentially involve hundreds, even thousands, of monitor nodes. Thus, it is also important to address questions such as what architecture should WISE employ to support such deployment, and how to achieve high scalability for tasks with many monitor nodes? In the subsequent sections, we present the design and development of WISE, a system that performs accurate and efficient window-based state monitoring over a network of distributed monitor nodes.

3 WISE Monitoring System

We present an overview of the WISE monitoring system in this section. We first introduce the architecture and deployment of WISE, and then, describe important components of WISE.

3.1 Architecture and Deployment

The WISE monitoring system takes the description of window-based monitoring tasks as input, continuously watches the state changes over the nodes being monitored, and triggers alerts when the state change meets the specified threshold. The description of a window-based monitoring task specifies the following five conditions:

1. the metric to be monitored at a node (e.g., incoming HTTP request rates),
2. the set of nodes associated with the monitoring task (N),
3. the global value threshold (T),
4. the monitoring time window (L), and
5. the countermeasures to take when a state alert is triggered.

The left side of Fig. 2 illustrates a sketch of the architectural design of the WISE system and a deployment example of monitoring tasks. Given a set of monitoring tasks, the system first scans for identical monitoring tasks and removes duplicated ones. It then deploys monitoring tasks on their associated nodes. During the monitoring process, the system collects reported state alerts from deployed monitoring tasks and processes these alerts according to specified countermeasures. It also watches machine failures that may impact deployed monitoring tasks. For instance, if one machine becomes unavailable, it identifies monitoring tasks involved with the machine and marks the corresponding monitoring results as unreliable to prevent false positive or negative results.

The deployment example in Fig. 2 shows four monitoring tasks running over 12 hosts. One host may be involved with multiple monitoring tasks. The deployment may involve load balancing and monitoring network construction [4]. For example, the system may choose hosts involved with few monitoring tasks to be coordinators as coordinators consume more CPU and bandwidth resources compared with monitor nodes. In the rest of this paper, we focus on developing efficient schemes for a single monitoring task. We leave other problems such as multitask optimization as our future work.

3.2 WISE Monitoring Approach

We now focus on the three technical developments that form the core of the WISE monitoring approach: the WISE monitoring algorithm, the monitoring parameter tuning schemes, and performance optimization techniques. The right side of Fig. 2 shows a high level view of the WISE monitoring approach.

3.2.1 The Monitoring Algorithm

The idea behind the WISE monitoring algorithm is to report partial information on local violation series at the monitor node side to save communication cost. The coordinator then uses such partial information to determine whether it is possible to detect state alerts. The coordinator collects further information only when the possibility of detecting state alerts cannot be ruled out.

Specifically, the monitor node side algorithm employs two monitoring parameters, the local threshold T_i and the filtering window size p_i. When detects local violation ($v_i(t) > T_i$), a monitor node i sends a local violation report and starts a filtering window with size p_i during which it only records monitored values and does not send violation reports.

The coordinator considers a reported local violation at node i as possible continuous local violation spanning p_i time units, since it does not know the complete violation information within the corresponding filtering window. It then "merges" possible continuous local violation reported from different nodes into a potential global continuous violation against T, namely *skeptical window*. The skeptical window holds a nature invariant that no state alert is necessary as long as the length of the skeptical window does not exceed L. The coordinator continuously maintains the skeptical window and tries to rule out the possibility of state alerts based on this invariant. It invokes a global poll to collect complete violation information only when the length of the skeptical window exceeds L.

Intuition. The WISE monitoring algorithm makes two effects to achieve communication efficiency. One is to avoid unnecessary global polls by optimistically delaying global polls, because later observed time units with no local violation indicate that previous global violation is not continuous. The other is to avoid frequent local violation reporting with monitor node side filtering windows. Filtering windows, when their sizes are properly tuned (Section 5), can save significant communication from frequently reporting local violation without noticeably diminishing the chance of ruling out state alerts and avoiding global polls. In addition, it ensures monitoring correctness as it always considers the worst case based on received partial information.

3.2.2 Scalable Parameter Tuning

State monitoring environments are usually heavily diversified. They may involve monitoring tasks with very different monitoring threshold T and time window L, as well as heterogeneous monitored value distributions across different nodes. As a result, monitoring parameters, i.e., T_i and p_i, should be properly tuned toward the given monitoring task and monitored value patterns for the best communication efficiency. For instance, if a given state monitoring task tries to capture a very rare event, monitor nodes should employ large filtering windows to deliver coarse information to maximally save communication. As another example, if a node often observes higher monitored values compared with other nodes, it should be assigned with relatively higher T_i accordingly.

To provide such flexibility, we proposed a centralized parameter tuning scheme in the original conference paper. The centralized tuning scheme runs at the coordinator and setting the parameters for all monitor nodes based on collected information on monitored value distribution. The centralized parameter tuning scheme has one drawback that it requires collecting of global information and performs intensive computation on the coordinator. Given the scale of datacenter monitoring and the exponential increasing nature of search space, the centralized tuning scheme may cause significant resource consumption on the coordinator and fail to find good parameters.

To address this issue, we develop a distributed parameter tuning scheme that avoids centralized information collecting and parameter searching. The distributed scheme runs at each monitor node. Each node tunes its local monitoring parameters based on observed events in a reactive manner. This scheme may produce slightly less efficient parameters compared with those generated by the centralized scheme because it tunes parameters based on local information. Nevertheless, its features such as avoiding searching the entire solution space and limited inter-node communication make it a desirable parameter tuning scheme for large-scale monitoring tasks.

3.2.3 Performance Optimization

In addition to improve the basic WISE approach with distributed parameter tuning, we also devise two novel performance optimization techniques, *the staged global poll* and *the termination message*, to further minimize the communication cost between a coordinator node and its monitoring nodes.

The staged global poll optimization divides the original global poll process into several stages. Each stage tries to rule out or confirm state alerts based on a fraction of monitored values that would be collected by the original global poll. Since later stages can be avoided if a previous stage can decide whether a state alert exists, the staged global poll reduces considerable communication. The termination message based optimization deals with "over-reported" local violation periods, which only contain little local violation and may increase the chance of invoking global poll. It tries to remove "over-reported" local violation periods by sending an extra message at the end of a filtering window to indicate real local violation.

In this paper, we not only provide the algorithmic design but also provide correctness analysis and usage model for both techniques.

4 The Monitoring Algorithm

We present the detail of WISE monitoring algorithm in this section. In addition, we also explain why WISE monitoring algorithm guarantees monitoring correctness and theoretically analyze its communication efficiency.

4.1 Algorithm Description

WISE monitoring algorithm consists of two parts, the monitor node side algorithm and the coordinator side algorithm:

4.1.1 The Monitor Node Side

A monitor node i reports partial information of local violation based on two monitoring parameters, local threshold T_i and filtering window size p_i. Local thresholds of different nodes satisfy $\sum_{i=1}^{n} T_i \leq T$. This restriction ensures the sum of monitored values at all nodes does not

exceed T if each value is smaller than its corresponding local threshold.

The filtering window size is the time length of a filtering time window and is defined over $[0, L]$. Specifically, filtering windows are defined as follows:

Definition 1. *A filtering window τ of node i is p_i continuous time units during which node i does not send local violation reports even if it observes $v_i(t) > T_i$ where t is a time unit within τ. In addition, we use $|\tau|$ to represent the remaining length of a filtering window τ, $t_s(\tau)$, and $t_e(\tau)$ to denote the start time and the end time of τ. If $p_i = 0$, $t_s(\tau) = t_e(\tau)$, and $|\tau| = 0$.*

When a node i detects $v_i(t) > T_i$ at time unit t and if it is currently not in a filtering window ($|\tau| = 0$), it sends a local violation report to the coordinator, and then enters a filtering window by setting $|\tau| = p_i$. During a filtering window ($|\tau| > 0$), it does not report local violation and decreases $|\tau|$ by 1 in every time unit. Node i starts to detect and report violation again only after $|\tau| = 0$. For now, we assume T_i and p_i are given for each node. We will introduce techniques for selecting proper values for T_i and p_i later.

4.1.2 The Coordinator Side

The coordinator side algorithm "reassembles" potential periods of local violation indicated by local violation reports into a potential period of continuous global violation, which we refer to as the *skeptical window*. The skeptical window essentially measures the length of the most recent continuous global violation in the worst case. The coordinator considers reported local violation from node i as continuous local violation lasting p_i time units, i.e., assuming filtering windows fully filled with local violation. It concatenates reported filtering windows that overlap in time into the skeptical window, which is defined as follows:

Definition 2. *A skeptical window κ is a period of time consisting of most recent overlapped filtering windows related to reported local violation since last global poll. Initially, the size of a skeptical window $|\kappa|$ is 0. Given a set of filtering windows $\mathbb{T} = \{\tau_i | i \in [1, n]\}$ observed at time t, κ can be updated as follows:*

$$t_s(\kappa') \leftarrow \begin{cases} t_s(\kappa), & t_e(\kappa) \geq t, \\ t, & otherwise, \end{cases} \tag{1}$$

$$t_e(\kappa') \leftarrow \begin{cases} t + \max\limits_{\forall \tau_i \in \mathbb{T}} \{t_e(\kappa) - t, |\tau_i|\}, & t_e(\kappa) \geq t, \\ \max\limits_{\forall \tau_i \in \mathbb{T}} \{t, t_e(\tau_i)\}, & otherwise, \end{cases} \tag{2}$$

where κ' is the updated skeptical window, $t_s(\cdot)$ and $t_e(\cdot)$ are the start and the end time of a window. In addition, $|\kappa| = t_e(\kappa) - t_s(\kappa) + 1$. In our motivating example, servers A and B with $p_A = p_B = 5$ report local violation at times 2 and 3, respectively. The corresponding skeptical window covers both filtering windows as they overlap, and thus, spans from times 2 to 7. Fig. 3 shows an illustrative example of skeptical windows.

When $t - t_s(\kappa) = L$, it indicates that there may exist continuous local violation for the last L time units (which could lead to continuous global violation of L time units). Thus, the coordinator invokes a global poll to determine whether a state alert exists. The coordinator first notifies all nodes about the global poll, and then, each node sends its buffered $v_i(t-j), j \in [0, t']$, where $0 < t' \leq L$, to the coordinator in one message. Here, t' depends on how many past values are known to the coordinator, as previous global polls and local violation also provides past v_i values. After a global poll, if the coordinator detects continuous global violation of L time units, i.e., $\sum_{i=1}^{n} v_i(t-j) > T, \forall j \in [0, L-1]$, it triggers a state alert and set $|\kappa| = 0$ before continuing. Otherwise, it updates κ according to received v_i. Clearly, the computation cost of both monitor node and coordinator algorithms is trivial.

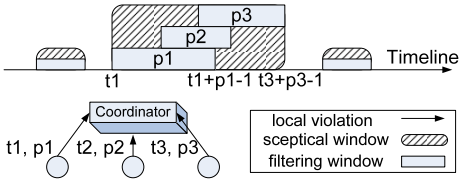


Fig. 3. Filtering windows and skeptical windows.

Filtering windows greatly reduce communication on local violation reporting, but may also cause overestimated local violation periods at the coordinator when filtering windows cover time units with no local violation. This, however, rarely leads to less chance of ruling out global polls and noteworthy increased cost in global polls. First, state alerts are usually rare events. With filtering windows, the coordinator still finds enough "gaps," i.e., time units with no local violation, between reported filtering windows before skeptical window size grows to L. Second, the parameter tuning schemes we introduce later set proper filtering window sizes so that the saving in local violation reporting always exceeds the loss in global polls. Last but not the least, we also develop a staged global poll procedure in Section 6 which significantly reduces communication cost in global polls.

4.2 Correctness

The WISE monitoring algorithm guarantees monitoring correctness because of two reasons. First, the coordinator never misses state alerts (false negative), as the skeptical window represents the worst case scenario of continuous global violation. Second, the coordinator never triggers false state alerts (false positive) as it triggers state alerts only after examining the complete local violation information. Theorem 1 presents the correctness guarantee of the WISE algorithm.

Theorem 1. *Given a monitoring task (T, L, N), the WISE algorithm triggers a state alert at time unit t if and only if $\sum_{i=1}^{n} v_i(t-j) > T, \forall j \in [0, L-1]$.*

Proof. In a filtering window of a node i, there may exist multiple periods of continuous local violation. We use $p_i^{'1}, p_i^{'2}, \ldots, p_i^{'k}$ to denote these periods of local violation where $k \in [1, p_i]$. Let $p_i^{'max} = \max\{p_i^{'1}, p_i^{'2}, \ldots, p_i^{'k}\}$ be the longest local violation period. Clearly, the filtering window $\tau_i (|\tau_i| = p_i)$ contains $p_i^{'max}$, i.e., τ_i starts at least as early as $p_i^{'max}$ and ends at least as late as $p_i^{'max}$ does. We denote this inclusion relation as $p_i \succeq p_i^{'max}$.

If constraints on T and L are violated, then there exists at least one series of local violation periods which overlap with each other and the total length of the

overlapped period is L. For any one of such series $p_i^{'*}$, consider any one of its local violation periods p_i'. If p_i' is within one filtering window of node i, we have $p_i \succeq p_i'$. If p_i' spans multiple filtering windows, denoted as p_i^*, it is not hard to see $p_i^* \succeq p_i'$. Since it is the same for all p_i', all associated filtering windows, P_i^*, must satisfy $P_i^* \succeq p_i^{'*}$. As a result, a global poll is invoked no later than the state alert. The global poll sets κ to the length of observed $p_i^{'*}$. Similarly, subsequent global polls will keep increasing κ to the length of observed $p_i^{'*}$ until the last global poll which triggers the state alert at time t. The other direction can be proved in a similar way. □

4.3 Communication Efficiency

Consider a state monitoring task with $n(n > 1)$ monitor nodes. Assume each T_i is perfectly tuned in the sense that one local violation occurs *if and only if* a global violation exists. Clearly, this is almost impossible in reality, as local violation does not always lead to global violation and global violation may correspond to multiple local violation. We use these "perfectly" tuned T_i to obtain the optimal performance of the instantaneous monitoring algorithm, so that we can study the lower bound of communication saving of the WISE algorithm. In addition, as Zipf distribution is often observed in distributed monitoring values [25], we assume the number of continuous local violation across nodes follows a Zipf distribution. Specifically, the probability of detecting continuous local violation of i time units is $Pr(x = i) = \frac{1}{H_{L+1}} \frac{1}{i+1}$, where H_{L+1} is the $(L+1)$th Harmonic number defined by $H_{L+1} = \sum_{j=1}^{L+1} \frac{1}{j}$. Using Zipf distribution here is to simplify our analysis. In reality, continuous local violation needs not to follow this distribution. Furthermore, let the communication cost of local violation be 1 and that of global polls be n.

Theorem 2. *Given the above settings, let C_I be the communication cost of running the instantaneous monitoring algorithm with perfectly tuned T_i, and let C_W be the communication cost of running the WISE algorithm, which uses the same T_i and simply sets $p_i = 1$. The resulting gain in communication cost, given by $gain = \frac{C_I}{C_W}$, is $n(1 - \frac{1}{\log(L+1)})/(1 + \frac{n-2}{\log(L+1)})$.*

Proof. Since each local violation causes one global poll in the instantaneous triggering algorithm, we have $C_I = n \cdot \sum_{i=1}^{L} Pr(x = i) = n - \frac{n}{\log(L+1)}$. The communication cost of WISE consists of two parts, one is local violation, the other is global poll. Thus, $C_W = \sum_{i=1}^{L} Pr(x = i) + L \cdot (n-1) \cdot Pr(x = L)$. Therefore, the gain of using WISE is

$$gain = \frac{C_I}{C_W} \geq \frac{n - \frac{n}{\log(L+1)}}{1 + \frac{n-2}{\log(L+1)}} \in [1, n).$$

□

The above theorem suggests that WISE yields more gain given larger L and n. For instance, when $L = 15$, $gain \geq \frac{3n}{n+3}$, the gain approximates to 3 when n is large enough. This implies that WISE scales well, which is confirmed by our experiment results. Furthermore, $gain$ is a theoretical bound derived with the unoptimized WISE algorithm. The actual gain is generally better (50 to 90 percent reduction in communication cost) with parameter tuning and optimized subroutines.

5 Scalable Parameter Tuning

The performance of WISE monitoring algorithm also depends on the setting of local monitoring parameters, i.e., T_i and p_i. To achieve the best communication efficiency, local monitoring parameters need to be tuned according to the given monitoring task and monitored value distributions. In the original conference paper, we proposed a centralized parameter tuning scheme which searches for the best parameters based on a sophisticated cost model. This scheme works well when the number of monitor nodes is moderate. However, datacenter environments often involve monitoring tasks running on a large number of nodes. The centralized scheme suffers from scalability issues in such large-scale monitoring tasks. First of all, the parameter space increases exponentially when the number of monitor nodes increases. As a result, the searching process of the centralized scheme may take considerable time to complete. Second, the centralized scheme requires the coordinator to collect monitored value distribution from all monitor nodes, which puts heavy burden on the coordinator node, especially with large-scale monitoring tasks.

To address these issues, we propose a scalable parameter tuning scheme which runs distributedly at each monitor node, and avoids searching in the entire parameter space and centralized data collection. In the rest of the section, we present detail of this distributed parameter tuning scheme.

5.1 Modeling Communication Cost

To begin with, we first introduce a cost model which can predict the communication cost of WISE monitoring algorithm given a set of monitoring parameters and the monitored value distribution. This model is frequently used for the development of our parameter tuning schemes.

5.1.1 Cost Analysis

Communication in the WISE algorithm consists of local violation reporting and global polls. We use C_l and $P_l(i)$ to denote the communication cost of sending a local violation report and the probability of sending it at one time unit on node i. Since a local violation report is of fixed size, we set C_l to 1. $P_l(i)$ is the probability of $v_i(t) > T_i$ and no local violation occurs during last p_i time units, because otherwise node i is in a filtering window during which it suppresses all violation reports.

Estimating the communication overhead for global polls is relatively complicated. To ease our discussion, we first define independent and continuous global polls:

Definition 3. *Given a global poll g occurring at time t, if there is no other global poll that occurs during time $[t - L + 1, t - 1]$, we say this global poll is an* independent global poll. *Otherwise, let g' be a global poll that happens during $[t - L + 1, t - 1]$, we say g' and g* overlap *with each other, denoted as $g \rightleftharpoons g'$. In addition, given a set of global polls, $\mathcal{G} = g_1, g_2, \ldots, g_k$, we refer $\mathcal{G}$ as a* continuous global poll *if $\forall g \in \mathcal{G}, \exists g' \in \mathcal{G}, g \rightleftharpoons g'$ and $\forall g'$ that $g' \rightleftharpoons g, g' \in \mathcal{G}$.*

Fig. 4 shows an example of independent and continuous global polls. Intuitively, independent global polls are separated global polls which collect v_i values for L time units. Continuous global polls are adjoined global polls that each may collect v_i values for less than L time units, except

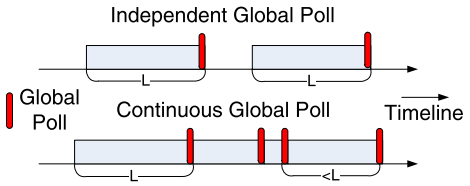


Fig. 4. Independent and continuous global polls.

the first one. In the following discussion, we refer a global poll which collects values for j time units as a j-windowed global poll. Clearly, $j = L$ for a independent global poll and $j > L$ for a continuous global poll. We use C_g^j to represent the cost associated with a j-windowed global poll. Since a j-windowed global poll requires all nodes to upload their v_i values of previous j time units, $C_g^j = n \cdot j$. In addition, we define P_g^j be the probability of a j-windowed global poll, since the probability of a global poll is also related to j.

Given the above settings, the communication cost of the WISE algorithm can be estimated as follows:

$$C = \sum_{i=1}^{n} C_l P_l(i) + \sum_{j=L}^{\infty} C_g^j P_g^j. \tag{3}$$

Note that C is essentially the expectation of communication cost for any time unit. Based on this cost function, we now define our parameter tuning problem as follows:

Problem Statement 2. *Given the global threshold T, monitoring window size L, and n monitor nodes, determine the values of $T_i, p_i, \forall i \in [1, n]$ so that the total communication cost C, given by (3), is minimized.*

5.1.2 Determining Event Probabilities

We next present further detail on predicting the communication cost of WISE algorithm based on the cost function given by (3). Clearly, we need to determine the probability of local violation events, $P_l(i)$, and the probability of global poll violation events, P_g^j, in order to compute C. Recall that $P_l(i)$ is the probability that a local violation occurs at time t and no local violation occurs during last p_i time units. Let V_i^t be the event of a violation on node i at time t, and correspondingly, $\overline{V_i^t}$ be the event of no violation on node i at time t. We have

$$P_l(i) = P\left[\bigcap_{k=1}^{p_i} \overline{V_i^{t-k}}\right] \cdot P\left[V_i^t\right]. \tag{4}$$

Compared with $C_l P_l(i)$, computing the cost for global polls is more complicated, as it depends on the states of monitor nodes. P_g^j is the probability that the size of a skeptical window equals to j. It is also the probability that at least one filtering window exists for each of the past j time units. Let W^t represents the event of at least one filtering window existing at t. Since W^t is independent among different t , we have, $P_g^j = P[\bigcap_{k=0}^{j-1} W^{t-k}] = (P[W^t])^j$. Denoting $P[W^t]$ by P_w, the cost of global polls is $\sum_{j=L}^{\infty} C_g^j P_g^j = n \sum_{j=L}^{\infty} j \cdot P_w^j$.

The sum part of the result is a variant of infinite geometric series, which can be solved via Taylor expansion. By solving this series, we have

$$C_g(P_w) = \sum_{j=L}^{\infty} C_g^j P_g^j = n \frac{L P_w^L - (L-1) P_w^{L+1}}{(1 - P_w)^2}.$$

As the cost for global polls can be considered as a function of P_w, we use $C_g(P_w)$ to denote the cost of global polls. The value of P_w can be computed as

$$P_w = 1 - P\left[\bigcap_{i=1}^{n} \bigcap_{k=1}^{p_i} \overline{V_i^{t-k}}\right]. \tag{5}$$

This is because the probability of W^t is the probability of at least one node existing in its filtering window at time t. Up to this point, the only thing left unknown in both (4) and (5) is the probability of $\overline{V_i^t}$, which depends on values of T_i, p_i and the distribution of v_i. To further compute $P_l(i)$ and P_w, we need to distinguish two types of stream values v_i. One is *time independent* values where v_i observed at the current moment is independent from those observed previously. The other type, *time dependent* values means v_i observed at the current moment is dependent from previous values. We next discuss the computation of $P_l(i)$ and P_w in both cases.

Time independent v_i assumes v_i in different time units is i.i.d. In this case, $P[V_i^t]$ and $P[V_i^{t-1}]$ are independent. Thus, (4) can be written as

$$P_l(i) = (P[v_i \le T_i])^{p_i}(1 - P[v_i \le T_i]). \tag{6}$$

Similarly, (5) now can be written as

$$P_w = 1 - \prod_{i=1}^{n} (P[v_i \le T_i])^{p_i}. \tag{7}$$

Based on (6) and (7), we only need the value of $P[v_i \le T_i]$ to compute $P_l(i)$ and P_w. To obtain $P[v_i \le T_i]$, each node maintains a histogram of the values that it sees over time as $H_i(x), x \in [0, T]$, where $H_i(x)$ is the probability of node i observing $v_i = x$. Given $H_i(x)$, $P[v_i \le T_i] = \sum_{x=0}^{T_i} H_i(x)$.

Time dependent v_i. We choose discrete-time Markov process, i.e., Markov chain, for modeling time dependent values, since it is simple and has been proved to be applicable to various real world stream data. Under this model, the values of future v_i and past v_i are independent, given the present v_i value. Formally, $P[v_i(t+1) = x | v_i(t) = x_t, \ldots, v_i(1) = x_1] = P[v_i(t+1) = x | v_i(t) = x_t]$. For simplicity, we use v_i and v_i' to denote the present value and the value of the previous time unit, respectively. Assuming v_i is time dependent, (4) and (5) can be written as

$$\begin{aligned} P_l(i) = P[v_i \le T_i](P[v_i \le T_i | v_i' \le T_i])^{p_i - 1} \\ P[v_i > T_i | v_i' \le T_i], \end{aligned} \tag{8}$$

$$P_w = 1 - \prod_{i=1}^{n} P[v_i \le T_i](P[v_i \le T_i | v_i' \le T_i])^{p_i - 1}. \tag{9}$$

To compute $P_l(i)$ and P_w, each monitor node maintains a set of transition probabilities $P[v_i = x | v_i' = x']$ where $x \in [0, T]$. Given these transition probabilities,

$$P[v_i \leq T_i] = \sum_{y=0}^{T} \sum_{x=0}^{T_i} P[v_i = x | v_i' = y],$$

$$P[v_i \leq T_i | v_i' \leq T_i] = \sum_{y=0}^{T_i} \sum_{x=0}^{T_i} P[v_i = x | v_i' = y], \text{ and}$$

$$P[v_i > T_i | v_i' \leq T_i] = 1 - P[v_i \leq T_i | v_i' \leq T_i].$$

Interestingly, looking for the best values for T_i and p_i is essentially finding the best trade-off between local violation and global polls which leads to the minimal communication cost. When increasing (decreasing) T_i, we reduce (increase) $P_l(i)$ which causes local violation to reduce (increase). However, larger (smaller) T_i also leads to larger (smaller) P_w which in turn increases (decreases) $C_g(P_w)$. It is also the same case for increasing or decreasing p_i.

5.2 Centralized Parameter Tuning

The centralized parameter tuning scheme is an intuitive development based on the above cost model. To determine best values for T_i and p_i, the centralized scheme adopts an EM-style local search scheme which iteratively looks for values leading to less cost. This scheme starts with two sets of initial values for T_i and p_i. Iteratively, it fixes one set of parameters and performs hill climbing to optimize the other set of parameters until reaching local minimum. It then fixes the optimized set and tunes the other one. It repeats this process until no better solution is found. To avoid local minimum, we run the scheme multiple times with different initial T_i and p_i values, and choose the best results.

5.3 Drawbacks of Centralized Tuning

The centralized scheme can find good local monitoring parameter values which minimizes communication given small number of monitor nodes. However, we find that this scheme suffers from scalability issues when this condition is not met.

First, as the centralized scheme holistically setting parameter for all nodes, the search space grows exponentially as the number of monitor nodes increases. Consequently, the search time of the centralized scheme also grows tremendously. Furthermore, when the number of monitor nodes is large, the search process causes significant consumption of CPU cycles at the coordinator, which could interfere with other jobs running on the coordinator node. One trade-off technique we apply to lower computation complexity is to reduce the search space by increasing the step size while performing hill climbing. This technique enables the centralized scheme to work with relatively large-scale monitoring tasks at the cost of less efficient parameters.

Second, the coordinator running the centralized scheme needs to collect the information of monitored value distribution, i.e., histograms in the time independent case and transition probabilities in the time dependent case, from all monitor nodes. This type of global information collecting is clearly not scalable and may consume considerable resources at the coordinator side. To address these issues, we propose a distributed parameter tuning scheme which allows each node to locally tune its monitoring parameters with minimal internode communication.

5.4 Distributed Parameter Tuning

The distributed parameter tuning scheme relieves the coordinator of the computation and communication burden by letting each node tune its monitoring parameters in a reactive manner based on events it observes. The main challenge in distributed parameter tuning is to effectively search for the best parameters at each monitor node without acquiring global information. We next describe detail of this scheme.

For ease of discussion, we use X_i to denote the probability of not having local violation at node i for both time dependent and independent v_i by defining X_i as following:

$$X_i = \begin{cases} P[v_i \leq T_i | v_i' \leq T_i] & \text{if time dependent,} \\ P[v_i \leq T_i] & \text{if time independent.} \end{cases}$$

By introducing X_i, (3) can be written as follows:

$$C = \sum_{i=1}^{n} C_l X_i^{p_i - 1}(1 - X_i) + C_g \left(1 - \prod_{i=1}^{n} X_i^{p_i}\right).$$

Since X_i is the probability of no local violation, we assume $X_i \geq (1 - X_i)$ for reasonable monitoring applications. Thus,

$$C \leq \sum_{i=1}^{n} C_l X_i^{p_i} + C_g \left(1 - \prod_{i=1}^{n} X_i^{p_i}\right).$$

Furthermore, Let $\alpha X_i \leq \min\{X_i | \forall i \in [1, n]\}$, where α can be predefined by user based on observed distribution, we have

$$C \leq \sum_{i=1}^{n} C_l X_i^{p_i} + C_g (1 - (\alpha X_i)^{np_i}).$$

Let $Y_i = X_i^{p_i}$ and $\beta_i = \alpha^{np_i}$, this relation can be written as

$$C \leq \sum_{i=1}^{n} C_l Y_i + C_g \left(1 - \beta_i Y_i^n\right). \tag{10}$$

Thus, instead of directly tuning values for T_i and p_i, we can optimize values for Y_i. In fact, Y_i can be considered as the area of a 2D "suppression window" at node i. The height of the window is controlled by X_i, which is determined by T_i, and the length of the window is controlled by p_i.

The distributed scheme adjusts Y_i at the monitor nodes based on their observed local violation reports and global poll events. Each local violation report from node i indicates that the area of the suppression window of node i is possibly lower than the optimum. Similarly, each global poll suggests the area of the suppression window is possibly higher than the optimum. Algorithm 1 shows the detail of the reactive scheme.

Algorithm 1. The Distributed Reactive Scheme

$*$ Invoked whenever received an event E

1: **if** $E =$ local violation **then**
2: $Y_i \leftarrow \alpha Y_i$ with probability $\min(1, \frac{1}{\rho_i})$
3: **else** $\{E =$ global poll$\}$
4: $Y_i \leftarrow \frac{Y_i}{\alpha}$ with probability $\min(1, \rho_i)$
5: **end if**

Choosing a proper value for ρ_i is critical for the reactive scheme to converge. Similar to the observation made in [22], the key point to achieve convergence is to make the scheme moves toward the optimal Y_i and stays at the optimal Y_i values once it reaches them. Assume the value of Y_i is not optimal, then either $Y_i < Y_i^{opt}$, which leads to

$P_l(Y_i) > P_l(Y_i^{opt})$ and $P_w(Y) < P_w(Y^{opt})$, or $Y_i > Y_i^{opt}$, which leads to $P_l(Y_i) < P_l(Y_i^{opt})$ and $P_w(Y) > P_w(Y^{opt})$, where Y_i^{opt} is the optimal Y_i, Y and Y^{opt} stand for all Y_i and all Y_i^{opt}, respectively. In the first case, we have

$$\frac{P_l(Y_i)}{P_w(Y)} > \frac{P_l(Y_i^{opt})}{P_w(Y^{opt})}.$$

By setting $\rho_i = \frac{P_w(Y^{opt})}{P_l(Y_i^{opt})}$, we have $\rho_i P_l(Y_i) > P_w(Y)$, which means the value of Y_i decreases. Similarly, we can see that the value of Y_i increases when $Y_i > Y_i^{opt}$. Thus, the reactive scheme reaches stable state when

$$\frac{P_l(Y_i)}{P_w(Y)} = \frac{P_l(Y_i^{opt})}{P_w(Y^{opt})}.$$

While estimating the exact Y_i^{opt} is infeasible, we can still approximate this value by minimizing the upper bound of C based on (10). More importantly, such computation can be done distributedly at each monitor node, as the right hand side of the equation can be divided into n items and each is only related to node i itself. Once each monitor node obtains its Y_i^{opt}, it sends this value to the coordinator. The coordinator gathers Y_i^{opt} for all nodes and sends these values to all nodes. Each node then can compute its ρ_i based on the received Y_i values.

One remaining question is which component, T_i or p_i, to change when Y_i is updated. We develop the following heuristics to handle this problem. When Y_i is updated, node i first computes the new $T_i'(p_i')$ for the updated Y_i by using old $p_i(T_i)$. With probability $\min\{1, \Delta_{T_i}^{max} \frac{1}{T_i' - T_i}\}$, where $\Delta_{T_i}^{max}$ is the maximum step length for updating T_i, it updates p_i if $p_i' \leq L$. If p_i is not updated, it updates T_i if $T_i' \leq T$. The rationale is that T_i is restricted by the global threshold T, and thus, is updated only when the change is small.

To ensure correctness, when node i updates T_i, it sends T_i' and p_i' to the coordinator. If $T_i' < T_i$, the coordinator updates its slack $S \leftarrow T_i - T_i'$. Otherwise, the coordinator approves the update if $S \geq (T_i' - T_i)$. When $S < (T_i' - T_i)$, it notifies the node to update its T_i' to S if $S > 0$. If $S = 0$, it notifies the node to update p_i instead. Note that the above messages sent from monitor nodes can be combined with local violation reports or global poll messages, as an update is necessary only when a local violation or a global poll occurs.

6 Performance Optimization

The performance of WISE can be further optimized by improving the implementation of its major subroutines. In this section, we describe two interesting optimization techniques of this kind, one for enhancing the global poll procedure at the coordinator side and the other for improving local violation reporting procedure at the monitor node side.

6.1 Staged Global Polls

In the global poll procedure we introduced earlier, each node i sends its buffered $v_i(t-j)$ values, where $j \in [0, L]$, to the coordinator for state alert verifying. However, as the coordinator, more often than not, does not need all buffered values from all nodes to determine whether a state alert exists, such a global poll procedure usually causes unnecessary communication.

To further reduce the communication cost for global polls while still ensure the correctness of the monitoring algorithm, we propose a novel *staged global poll* procedure as an optimization technique. The staged global poll procedure divides the original global poll process into three stages. In each stage, only part of the $v_i(t-j), j \in [0, L]$ values are transmitted. In addition, if an early stage already rules out or triggers a state alert, then the rest of the stages can be avoided. Even if all stages are required, the new procedure transmits the same amount of v_i data as the original one.

Stage one. Node i only sends those $v_i(t-j)$ values that satisfies $v_i(t-j) \leq T_i$. Once received all the data, the coordinator tries to rule out the state alert by looking for a time unit t' in which $v_i(t') \leq T_i, \forall i \in [1, n]$. If such a time unit is found, it suggests that there exists at least one gap, i.e., a slot without violations, between local violations, and thus, the state alert can be ruled out.

Stage two. If such gaps are not found, the global poll process enters the second stage, where it tries to confirm the existence of a state alert without invoking further communication. Specifically, the coordinator computes a partial slack $S'(t) = \sum_{i \in G(t)} T_i - v_i(t)$, where $G(t) = \{i | v_i(t) < T_i\}$ for all time units associated with the global poll. This partial slack $S'(t)$ is essentially the sum of "space," $T_i - v_i(t)$ at nodes not having local violation at time t. In addition, let $O = \{t | S'(t) \geq |N - G(t)|\}$ where N is the set of all monitor nodes and $N - G(t)$ is the set of nodes having local violations at time t. Note that $|N - G(t)|$ is the lower bound of the sum of "overflow," $v_i(t) - T_i$ at nodes having local violation at time t. The coordinator then triggers a state alert if $O = \emptyset$, because a state alert must exist if the sum of "space" is smaller than the sum of "overflow" for all time units associated with the global poll.

Final stage. If the second stage does not trigger a state alert, the third, also the last, stage begins, in which the coordinator notifies all nodes that detected local violation at time units $t \in O$ (can be inferred based on data received in the first stage) to send the rest of their unsent v_i data. Once the data are received, the coordinator triggers a state alert if $S'(t) < \Delta(t)$ for all t associated with the global poll, where $\Delta(t) = \sum_{i \in N - G(t)} v_i(t) - T_i$, it terminates the global poll procedure otherwise.

The correctness proof of the staged global poll is straightforward. The first stage *rules out* state alerts according to a sufficient condition of *not having* state alerts, i.e., the existence of "gaps." The second stage *triggers* state alerts according to a sufficient condition of *having* state alerts, i.e., the sum of "space" is smaller than the sum of "overflow" at all times units. Finally, the last stage collects all buffered values from all nodes, and thus, can always make correct decisions.

The staged global poll reduces significant amount of communication cost compared with the original global poll, as witnessed by our experiment results. The first and the second stages has the capability of ruling out or triggering state alerts with only a subset of buffered values in most cases. In the worse case, the staged global poll transmits the same amount of buffered values as the original global poll does, since the last stage only collects previously uncollected values.

6.2 Termination Messages

In the original WISE algorithm, when a monitor node enters a filtering window, it no longer reports any local violation

until it leaves the window. Although the coordinator assumes the node is experiencing local violations throughout its filtering window, it may not be true, as v_i may drop below T_i before the filtering window ends. While this scheme is very communication efficient, it also reduces the chance of ruling out global polls, as filtering windows may "exaggerate" the length of real local violation period and overlap with each other undesirably.

Based on this observation, we optimize the violation reporting procedure by letting node send a termination message which contains the information of unviolated time units at the end of a filtering window when it helps the coordinator to avoid global polls. While this modified scheme introduces extra communication at the end of filtering windows, the corresponding termination message, when properly generated, may avoid unnecessary global polls, and thus, reduces the total communication cost.

Specifically, a termination message of node i contains sequence numbers of time units during which $v_i(t) \leq T_i$. The sequence number here is associated with the previous violation report sent by node i, and thus, is defined over $(0, p_i - 1]$. When receiving a termination message m_t of a filtering window τ, the coordinator first updates τ by removing the time units contained in m_t, and then, use the updated filtering window to calculate its skeptical window.

Due to lack of global information, it is difficult for a monitor node to locally determine whether a termination message would help the coordinator to discover gaps between filtering windows. Clearly, always sending termination messages at the end of filtering window is not efficient. A termination message is beneficial only when the corresponding filtering window contains sufficient unviolated time units, because the more unviolated time units one termination message contains, the more likely the coordinator can avoid global polls. Therefore, we use $\frac{|m_t|}{p_i}$, where $|m_t|$ is the number of time units in the termination message m_t, to measure the likeliness of m_t can reveal gaps in the skeptical window. In our experiment, we restrict that a node i sends a termination message only when $\frac{|m_t|}{p_i} \geq 0.75$. By introducing this restriction, only nodes that observe adequate number of unviolated time units within its filtering windows send out termination messages.

7 Experimental Evaluation

We performed extensive experiments over both real world and synthetic traces to evaluate WISE. First, we evaluate the basic WISE with centralized parameter tuning. Our empirical study shows several important observations:

- WISE achieves a reduction from 50 to 90 percent in communication cost compared with instantaneous monitoring algorithm [22] and simple alternative schemes.
- The centralized parameter tuning scheme effectively improves the communication efficiency.
- The optimization techniques further improve the communication efficiency of WISE.

Second, we evaluate the scalability of the WISE system with respect to different design choices and optimizations, especially we compare WISE equipped with the two optimizations with the basic WISE, and compare the improved WISE, powered by the distributed parameter tuning scheme, with the basic WISE using centralized tuning. We highlight the experimental results we observed as follows:

- WISE scales better than the instantaneous algorithm in terms of communication overhead. It scales even better with the distributed parameter tuning scheme.
- While the distributed parameter tuning scheme performs slightly worse than the centralized scheme, it scales better, and thus, is suitable for large scale distributed systems.
- The two optimization techniques continue to contribute additional communication saving when running with the distributed parameter tuning scheme.

7.1 Experiment Settings

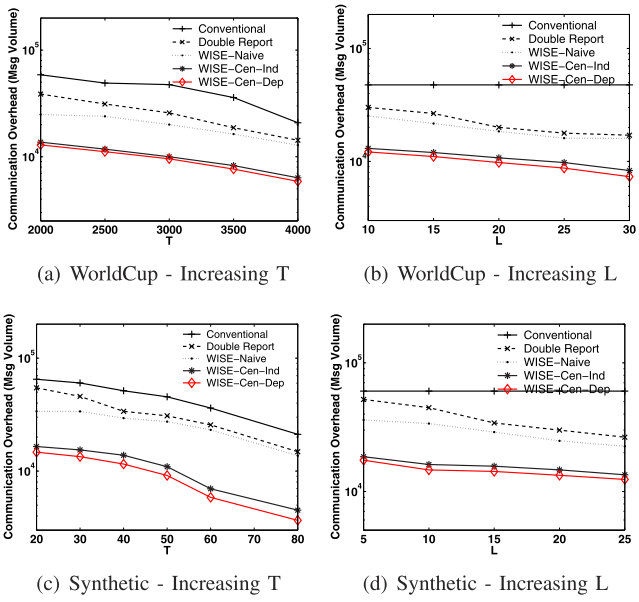
We consider our simulation scenario as detecting DDoS attacks for a set of distributed web servers. Each server is equipped with a monitoring probe. In addition, a centralized monitoring server watches the total number of HTTP requests received at different web servers. When the total number of requests continuously stays above a predefined threshold T for L time units, the monitoring server triggers a state alert.

We compare communication efficiency and scalability of the WISE algorithm, the instantaneous monitoring algorithm, and simple alternative window-based schemes. We choose the nonzero slack instantaneous monitoring algorithm [22] for comparison, as it is the most recent instantaneous monitoring approach and is reported to achieve significant communication reduction ($\leq$60%) over previous approaches. The nonzero slack algorithm employs a local value threshold at each monitor node and reports local violation whenever the local value exceeds the local value threshold. It uses a set of optimization techniques to set optimal local threshold values so that $\sum T_i - T > 0$, a.k.a the slack may be positive.

We also use several simple alternative window based state monitoring schemes as evaluation baseline. These schemes include the aforementioned "double reporting" scheme and WISE with naive parameter setting. Monitor nodes running the double reporting scheme report both the beginning and the end of a local violation period and the coordinator delays the global poll until necessary. The naive parameter setting simply sets $T_i = \frac{T}{n}$ and $p_i = \max\{\frac{L}{n}, 2\}$ for node i.

We measure communication cost by message volume (the total size of all messages) and message number. By default, we use message volume for comparison. In addition, we categorize messages into data messages and control messages. Data messages are those containing monitored values, e.g., local violation reports. The size of data message is m where m is the number of values encapsulated in the message. Control messages refer to all the other messages and their size is 1.

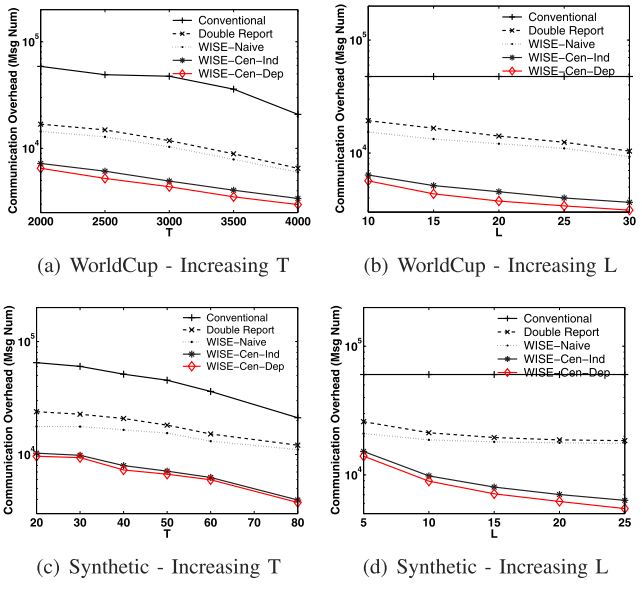
Our trace-driven simulator runs over two data sets. One data set, *WorldCup*, contains real world traces of HTTP requests across a set of distributed web servers. The trace data come from the organizers of the 1998 FIFA Soccer World Cup [24] who maintained a popular website that was accessed over 1 billion times between April 30 and July 26, 1998. The website was served to the public by 30 servers



(a) WorldCup - Increasing T (b) WorldCup - Increasing L

(c) Synthetic - Increasing T (d) Synthetic - Increasing L

Fig. 5. Comparison of communication efficiency in terms of message volume.



(a) WorldCup - Increasing T (b) WorldCup - Increasing L

(c) Synthetic - Increasing T (d) Synthetic - Increasing L

Fig. 6. Comparison of communication efficiency in terms of message number.

distributed among four geographic locations around the world. Thus, the traces of WorldCup provide us a real-world, large-scale distributed data set. In our experiments, we used the server log data consisting of 57 million page requests distributed across 26 servers that were active during that period. We set the length of time unit to 1 minute and invoke a reconfiguration every 1,000 time units. Although results presented here are based on a 24-hour time slice (from 22:01:00 June 6th GMT to 22:00:00 June 7th GMT) of the system log data, we conducted a series of experiments over log data that spanned different days and different hours of day and we observed very similar results.

The other data set, *Synthetic*, contains randomly generated traces that give us the freedom of evaluating parameters cannot be controlled in real world traces. For instance, we can increase the number of monitor nodes from 20 to 5,000 for scalability evaluation. We first generate a trace of aggregate values and then distribute values to different nodes based on Uniform or Zipf distributions. Unless otherwise specified, the number of nodes is 20 and Uniform distribution is applied. To track data distribution, we use equidepth histograms at each monitor node and we also employ exponential aging on histograms to make it reflecting recent observed values more prominently than older ones. For both data sets, the parameter reconfiguration interval is 1,000 time units.

7.2 Results

7.2.1 Comparison of Communication Efficiency

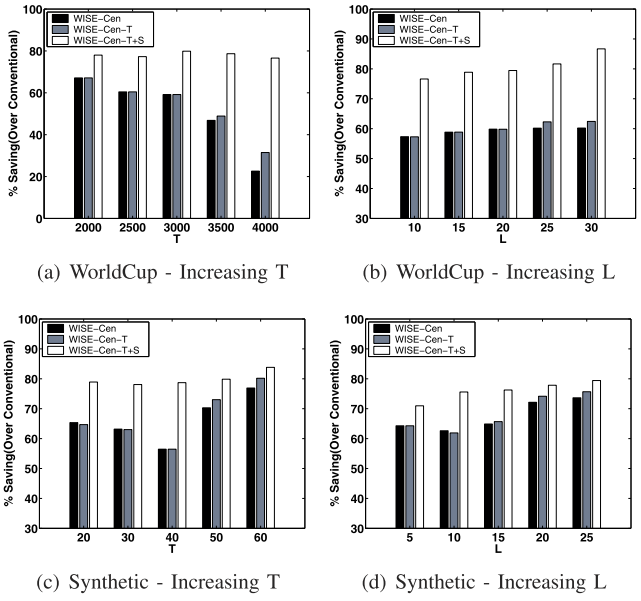
Figs. 5 and 6 compare the communication overhead of WISE enhanced by centralized tuning (WISE-Cen) with that of the instantaneous monitoring algorithm(Instantaneous), the double reporting scheme (Double Report) and WISE with naive parameter setting (WISE-Naive) for the World Cup data set and Synthetic data set. We vary T and L in a way that the total length of global violation takes up from 0 to 50 percent of the total trace length. By default, we set $T = 2{,}500(20)$ and $L = 15(10)$ for the WorldCup (Synthetic) data set.

Fig. 5a shows the total message volume generated by WISE is nearly a magnitude lower than that of the instantaneous approach. Double Report and WISE-Naive, while outperform the instantaneous approach as they delay global polls, generate more traffic compared with WISE. Double Report suffers from frequent reporting for short violation periods, especially when T is small. WISE-Naive fails to achieve better efficiency because it does not explore different value change patterns at different nodes. Note that parameter setting schemes using the time independent model (Ind) performs slightly better than those using time dependent one (Dep). However, as the time dependent model associates higher communication and computation cost, the time independent model is more desirable.

In Fig. 5b, while the instantaneous approach is not benefited from large values of L, the WISE algorithm pays less and less communication overhead as L grows, since nodes increase filtering window sizes and the coordinator rules out more global polls with increasing L. Figs. 5c and 5d show similar results for the Synthetic data set. Furthermore, as Fig. 6 shows, WISE achieves even better efficiency advantage in terms of message *number*, as global polls in WISE collects multiple values, instead of a single value in the instantaneous approach. Thus, WISE is even more favorable when per message payload is insensitive to message sizes.

7.2.2 Effect of Optimization Techniques

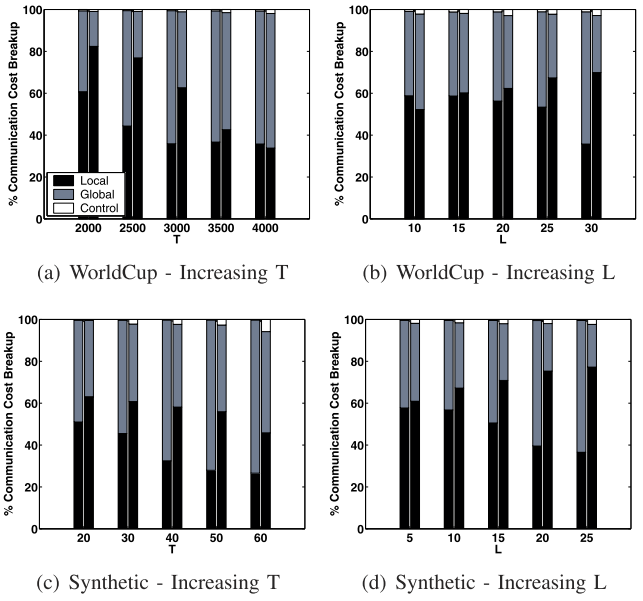
Figs. 7a and 7b show effect of *termination messages (T)* and *staged global polls (S)* in terms of communication reduction, where the Y-axis is the percentage of message volume saved over the instantaneous scheme for the WorldCup data set. In Fig. 7a, the WISE monitoring algorithm achieves 60 to 80 percent saving after optimization. However, the saving of unoptimized ones reduces as T grows because of two reasons. First, the instantaneous scheme also causes less communication as T grows. Second, with growing T, the portion of global poll communication increases (as suggested



(a) WorldCup - Increasing T (b) WorldCup - Increasing L

(c) Synthetic - Increasing T (d) Synthetic - Increasing L

Fig. 7. Effectiveness of optimization techniques in enhanced WISE (message volume).



(a) WorldCup - Increasing T (b) WorldCup - Increasing L

(c) Synthetic - Increasing T (d) Synthetic - Increasing L

Fig. 8. Communication cost breakup of WISE (message volume and message number).

later by Fig. 13a) due to reduced local violation, and the original global poll is very expensive. Termination messages achieve relatively less saving compared with staged global polls. In Fig. 7b, the saving increases when L grows, as larger L leads to larger p_i. Figs. 7c and 7d show similar results for the Synthetic data set.

7.2.3 Communication Cost Breakup Analysis

Fig. 8 shows communication cost breakup of WISE with centralized tuning, where communication overhead is divided into three parts: local violation reporting, global polls, and control. The first part is the overhead for value reporting in local violations, i.e., messages sent by nodes during local violations. The second part is the overhead for value reporting in global polls, which consists of messages with buffered stream values sent by nodes during global polls and notification messages from the coordinator. All the rest of the messages, most of which generated by the parameter tuning scheme, are classified as control messages. Furthermore, the left bar in each figure shows the percentage of different types of communication in message volume, and the right bar measures the percentage in message number.

In Fig. 8a, as T grows, the portion of global poll communication steadily increases, as local violation occurs less frequently. The portion of control communication also increases, due to the reduction of local violation reporting and global polls. Similarly, Fig. 8b observes the growth of the global poll portion along with increasing L, as nodes increase p_i to filter more reports. Figs. 8c and 8d provide similar results for the Synthetic data set.

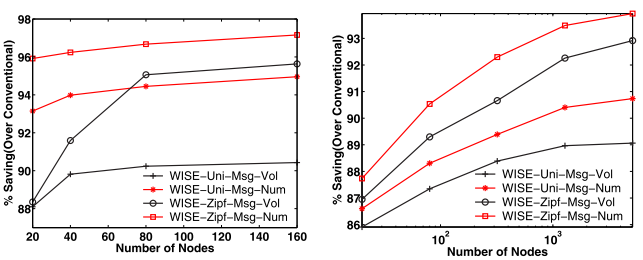
7.2.4 Scalability

Figs. 9a and 9b evaluate the communication saving for WISE with centralized tuning. For World Cup data set, we distributed the aggregated requests randomly to a set of 20 to 160 monitor nodes by Uniform and Zipf distributions. For the Synthetic data set, we increased the number of nodes from 20 to 5,000 nodes. When a uniform distribution was used, every node received a similar amount of requests. When a Zipf distribution was assumed, a small portion of the nodes received most of the requests. For Zipf distribution, we chose a random Zipf exponent in the range 1.0-2.0. Same as before, we measure communication cost in both message volume and message number.

In Figs. 9a and 9b, the saving of WISE increases up to over 90 percent when the node number increases, which indicates that WISE scales better than the instantaneous approach. Interestingly, WISE performs better when Zipf distribution is used, because the parameter tuning scheme can set higher T_i and p_i to nodes observing higher values, which avoids considerable local violation and global polls. Again, WISE achieves higher advantage in communication reduction when we measure communication cost in number of messages generated, as global polls in WISE collect values in multiple time units.

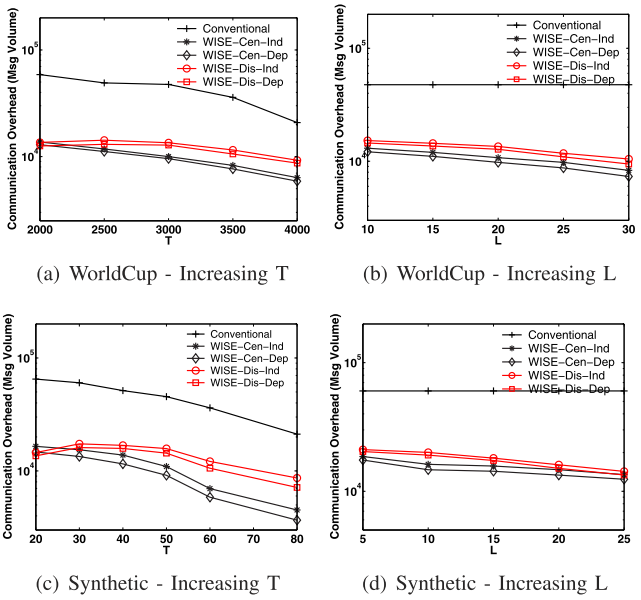
7.2.5 Distributed Tuning versus Centralized Tuning

We now compare distributed tuning and centralized tuning in several different aspects. Fig. 10 compares the communication efficiency achieved by the centralized tuning scheme and the distributed one. The centralized parameter tuning scheme (Cen) generally performs slightly better than the distributed one (Dis) does, as the centralized scheme has the complete value distribution information. Note that the distributed scheme works as



(a) WorldCup - Increasing n (b) Synthetic - Increasing n

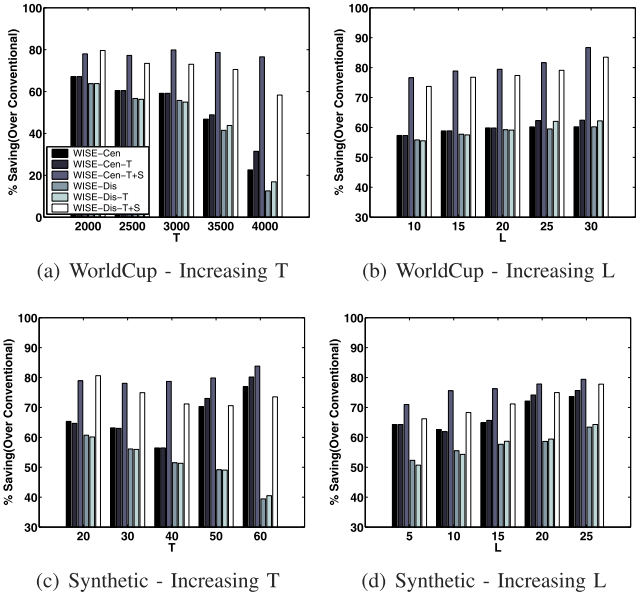
Fig. 9. Scalability (message volume and number).



(a) WorldCup - Increasing T (b) WorldCup - Increasing L

(c) Synthetic - Increasing T (d) Synthetic - Increasing L

Fig. 10. Comparison of communication efficiency in terms of message volume.



(a) WorldCup - Increasing T (b) WorldCup - Increasing L

(c) Synthetic - Increasing T (d) Synthetic - Increasing L

Fig. 11. Effectiveness of optimization techniques in enhanced WISE (message volume).

good as the centralized scheme when T is relatively low, because violation is so frequent that there is little space for parameter optimization. When T increases, the centralized scheme starts to find better parameters than the distributed scheme does.

Another interesting observation is that the distributed scheme actually performs better than the centralized scheme when L is relatively large. As the centralized scheme overestimates the communication overhead for global polls, it tends to assign small values to p_i. The distributed scheme does not suffer from this problem as it can reactively increase p_i when it observes more local violations than global polls. As later proved in Fig. 13b, the centralized scheme pays much more local communication overhead than the distributed scheme does.

Fig. 11 compares the effectiveness of the two optimization techniques when WISE is tuned by the centralized scheme and the distributed one, respectively. In general, staged global poll and termination message work with both tuning schemes, although staged global poll achieves more communication reduction. Furthermore, as the distributed scheme tends to use larger p_i, WISE with distributed tuning encounters more global polls. Consequently, the staged global poll often gains more communication reduction when works with the distributed tuning scheme.

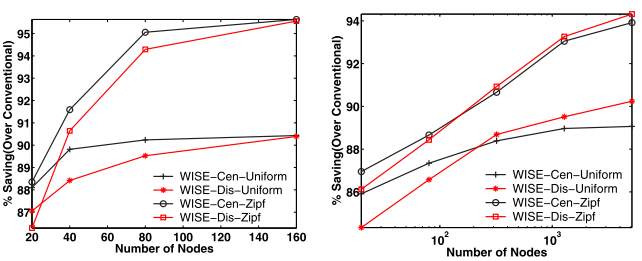
Fig. 13 presents cost break (message volume) with centralized tuning scheme (left bar) and distributed tuning scheme (right bar). In Figs. 13a and 13b, the centralized scheme has a relatively larger portion of local violation overhead, as it overestimates the communication cost for global poll. In both figures, the distributed scheme pays more overhead in control, because it requires communication when adjusting local threshold T_i. Figs. 13c and 13d provide similar results for the Synthetic data set.

Figs. 12a and 12b compare the scalability of centralized and distributed tuning schemes. The distributed scheme performs even better than the centralized scheme when the number of nodes is large. This is because we restrict the computation time used by the centralized scheme given a large number of nodes, which degrades its performance. Note that although computation and communication burden of the coordinator is not taken into account in the above results, it could cause serious workload imbalance between monitor nodes. For instance, in our experiment the CPU time consumed by the coordinator running the centralized scheme is an order of magnitude more than that consumed by a monitor node under typical settings. Therefore, the distributed tuning scheme is a desirable alternative as it provides comparable communication efficiency and better scalability.

8 Related Work

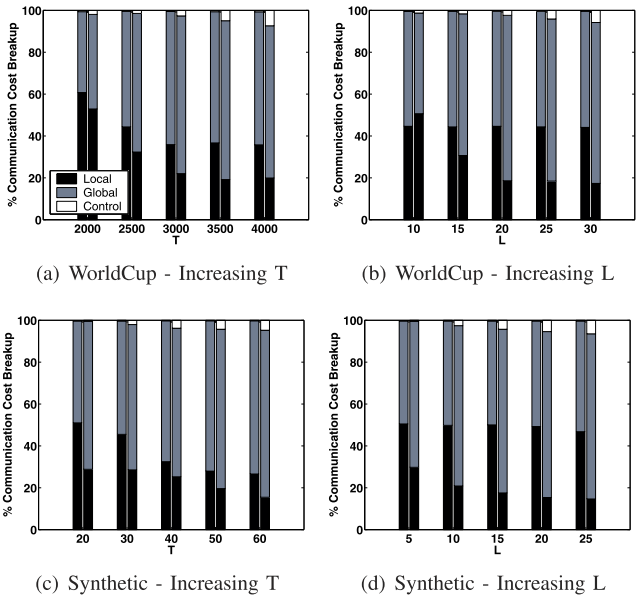
Distributed data stream monitoring has been an active research area in recent years. Researchers have proposed algorithms for the continuous monitoring of top-k items [6], sums and counts [7], and quantiles [8], problems addressed by these work are quite different from ours. While these work study supporting different operators, e.g., top-k and sums, over distributed data streams with guaranteed error bounds, we aim at detecting whether an aggregate of distributed monitored values violates constraints defined in value and time.

Recent work [18], [20], [19], [21], [22] on the problem of distributed constraint monitoring proposed several



(a) WorldCup - Increasing n (b) Synthetic - Increasing n

Fig. 12. Scalability of different parameter tuning schemes.



(a) WorldCup - Increasing T (b) WorldCup - Increasing L

(c) Synthetic - Increasing T (d) Synthetic - Increasing L

Fig. 13. Communication cost breakup between WISE with centralized and distributed tuning scheme.

algorithms for communication efficient detection of constraint violation. They study a different problem by using an instantaneous monitoring model where a state alert is triggered whenever the sum of monitored values exceeds a threshold. By checking persistence, the window-based state monitoring model we study gains immunity to unpredictable bursty behavior and momentary outlier data [15], and provides more space for improving communication efficiency. In addition, the instantaneous model is a special case of ours when $L = 1$.

The early work [18] done by Dilman and Raz propose a Simple Value scheme which sets all T_i to T/n and an Improved Value which sets T_i to a value lower than T/n. Jain et al. [26] discuss the challenges in implementing distributed triggering mechanisms for network monitoring and they use local constraints of T/n to detect violation. The more recent work of Sharfman et al. [20] represents a geometric approach for monitoring threshold functions. Keralapura et al. [19] propose static and adaptive algorithms to monitor distributed sum constraints. Agrawal et al. [21] formulate the problem of selecting local constraints as an optimization problem which minimizes the probability of global polls. Kashyap et al. [22] propose the most recent work in detecting distributed constraint violation. They use a nonzero slack scheme which is close to the idea of Improved Value scheme in [18]. They show how to set local thresholds to achieve good performance. We choose this nonzero slack scheme for comparison purpose.

The work that is perhaps closest to ours is that of Huang et al. [16]. They consider a variant of the instantaneous tracking problem where they track the cumulative amount of "overflows" which is $max\{0, \sum_i v_i - T\}$. This work makes two assumptions which may not be true: 1) All local threshold values are equal, and 2) local values follow a Normal distribution. In addition, it is unclear if the computed local thresholds in [16] optimize total communication costs. WISE employs a sophisticated cost model to estimate the total communication overhead and optimizes parameter setting based on this estimation. Furthermore, while [16] allows missed detections, WISE guarantees monitoring correctness.

Compared with our earlier results reported in [23], this paper makes four new developments. First, we present the architectural design and deployment options for a WISE-enabled monitoring system. Second, we develop a novel distributed parameter tuning scheme that offers considerable performance improvement in terms of the scalability of the WISE framework. Third, we develop two concrete optimization techniques to further reduce the communication cost between a coordinator and its monitoring nodes. We show that both techniques guarantee the correctness of monitoring. Finally, we conduct a series of new experiments with a focus on how the distributed parameter tuning scheme and the two optimization techniques contribute to the enhancement of the scalability of WISE.

9 Conclusions and Future Work

The increasing use of consolidation and virtualization is driving the development of innovative technologies for managing cloud applications and services. We argue that state monitoring is one of the crucial functionalities for on-demand provisioning of resources and services in cloud datacenters. We have presented a distributed approach for WIndow-based StatE monitoring. In contrast to the instantaneous state monitoring model which triggers state alert whenever a constraint is violated, the window-based state monitoring reports alerts only when state violation is continuous within a specified time window. Our formal analysis and experimental evaluation of WISE demonstrate that window-based state monitoring is not only more resilient to temporary value bursts and outliers, but also able to save considerable communication when implemented in a distributed manner. To achieve efficiency, WISE utilizes the window based monitoring algorithm, parameter tuning schemes, and optimized monitoring subroutines to minimize monitoring related communication at three different levels. Experimental results show that WISE achieves significant communication reduction (50-90 percent) compared with instantaneous monitoring approaches and simple alternative schemes.

Our work on application state monitoring in cloud datacenters continues along several directions. In this work, we focus on how to efficiently monitor the window-based state violation for one application running over a collection of distributed computing nodes. Another interesting challenge in state monitoring for cloud datacenters is to study the problem of efficient scheduling of multiple application state monitoring tasks by optimizing resource utilization and minimizing the amount of duplicate processing and messaging. In addition, performing failure-resilient state monitoring is another interesting yet challenging problem.

Acknowledgments

This work is partially supported by grants under US National Science Foundation (NSF) CISE CyberTrust program, NetSE cross cutting program, and IBM SUR grant, IBM faculty award, and a grant from Intel Research council. The authors would like to thank the guest editor and the reviewers for their helpful and constructive comments that help improve the overall presentation of the paper.

REFERENCES

[1] Amazon, "Amazon Elastic Computer Cloud(Amazon ec2)," 2008.

[2] S. Bhatia, A. Kumar, M.E. Fiuczynski, and L.L. Peterson, "Lightweight, High-Resolution Monitoring for Troubleshooting Production Systems," *Proc. Eighth USENIX Conf. Operating Systems Design and Implementation (OSDI)*, pp. 103-116, 2008.

[3] N. Jain, M. Dahlin, Y. Zhang, D. Kit, P. Mahajan, and P. Yalagandula, "Star: Self-Tuning Aggregation for Scalable Monitoring," *Proc. 33rd Int'l Conf. Very Large Data Bases (VLDB)*, 2007.

[4] S. Mengy, S.R. Kashyap, C. Venkatramani, and L. Liu, "Remo: Resource-Aware Application State Monitoring for Large-Scale Distributed Systems," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS)*, 2009.

[5] N. Jain, P. Mahajan, D. Kit, P. Yalagandula, M. Dahlin, and Y. Zhang, "Network Imprecision: A New Consistency Metric for Scalable Monitoring," *Proc. Eighth USENIX Symp. Operating Systems Design and Implementation (OSDI)*, pp. 87-102, 2008.

[6] B. Babcock and C. Olston, "Distributed Topk Monitoring," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2003.

[7] C. Olston, J. Jiang, and J. Widom, "Adaptive Filters for Continuous Queries over Distributed Data Streams," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2003.

[8] G. Cormode, M.N. Garofalakis, S. Muthukrishnan, and R. Rastogi, "Holistic Aggregates in a Networked World: Distributed Tracking of Approximate Quantiles," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 25-36, 2005.

[9] S. Krishnamurthy, C. Wu, and M.J. Franklin, "On-the-Fly Sharing for Streamed Aggregation," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 623-634, 2006.

[10] "Amazon Cloudwatch Beta," http://aws.amazon.com/cloudwatch/, 2011.

[11] A. Bulut and A.K. Singh, "A Unified Framework for Monitoring Data Streams in Real Time," *Proc. 21st Int'l Conf. Data Eng. (ICDE)*, 2005.

[12] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "Tinydb: An Acquisitional Query Processing System for Sensor Networks," *ACM Trans. Database Systems*, vol. 30, no. 1, pp. 122-173, 2005.

[13] A. Manjhi, S. Nath, and P.B. Gibbons, "Tributaries and Deltas: Efficient and Robust Aggregation in Sensor Network Streams," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2005.

[14] M. Li, Y. Liu, and L. Chen, "Non-Threshold Based Event Detection for 3D Environment Monitoring in Sensor Networks," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS '07)*, 2007.

[15] A. Deligiannakis, V. Stoumpos, Y. Kotidis, V. Vassalos, and A. Delis, "Outlier-Aware Data Aggregation in Sensor Networks," *Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE)*, 2008.

[16] L. Huang, M.N. Garofalakis, A.D. Joseph, and N. Taft, "Communication-Efficient Tracking of Distributed Cumulative Triggers," *Proc. 27th Int'l Conf. Distributed Computing Systems (ICDCS)*, p. 54, 2007.

[17] A. Silberstein, K. Munagala, and J. Yang, "Energy-Efficient Monitoring of Extreme Values in Sensor Networks," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2006.

[18] M. Dilman and D. Raz, "Efficient Reactive Monitoring," *Proc. IEEE INFOCOM '01*, 2001.

[19] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-Efficient Distributed Monitoring of Threshold Counts," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2006.

[20] I. Sharfman, A. Schuster, and D. Keren, "A Geometric Approach to Monitor Threshold Functions over Distributed Data Streams," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, 2006.

[21] S. Agrawal, S. Deb, K.V.M. Naidu, and R. Rastogi, "Efficient Detection of Distributed Constraint Violations," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE)*, 2007.

[22] S.R. Kashyap, J. Ramamirtham, R. Rastogi, and P. Shukla, "Efficient Constraint Monitoring Using Adaptive Thresholds," *Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE)*, 2008.

[23] S. Meng, T. Wang, and L. Liu, "Monitoring Continuous State Violation in Datacenters: Exploring the Time Dimension," *Proc. IEEE 26th Int'l Conf. Data Eng. (ICDE)*, 2010.

[24] M. Arlitt and T. Jin, "1998 World Cup Website Access Logs," http://www.acm.org/sigcomm/ITA/, Aug. 1998.

[25] L.A. Adamic and B.A. Huberman, "Zipfs Law and the Internet," *Glottometrics*, 2002.

[26] A. Jain, J.M. Hellerstein, S. Ratnasamy, and D. Wetherall, "The Case for Distributed Triggers," *Proc. ACM Workshop Hot Topics in Networks (HotNets)*, 2004.

Shicong Meng received the BS degree from East China Normal University and the MS degree from Shanghai Jiao Tong University, both in computer science, before he came to Georgia Tech. He is currently working toward the PhD degree in the College of Computing at Georgia Tech. He is affiliated with the Center for Experimental Research in Computer Systems (CERCS) where he works with Professor Ling Liu. His research focuses on performance, scalability, and security issues in large-scale distributed systems such as cloud datacenters. He has recently been working on several projects on cloud datacenter monitoring and management, with a strong emphasis on cloud monitoring services. He is a student member of the IEEE.

Ling Liu is a full professor in the School of Computer Science at Georgia Institute of Technology. There she directs the research programs in Distributed Data Intensive Systems Lab (DiSL), examining various aspects of data intensive systems with the focus on performance, availability, security, privacy, and energy efficiency. She and her students have released a number of open source software tools, including GTMobiSim, PeerCrawl, WebCQ, XWRAPElite. She has published more than 300 International journal and conference articles in the areas of databases, distributed systems, and Internet Computing. Her current research is primarily sponsored by US National Science Foundation (NSF), IBM, and Intel. She is a recipient of the best paper award of ICDCS 2003, WWW 2004, the 2005 Pat Goldberg Memorial Best Paper Award, and 2008 International conference on Software Engineering and Data Engineering. She has served as general chair and PC chairs of numerous IEEE and ACM conferences in data engineering, distributed computing, service computing, and cloud computing fields and is a co-editor-in-chief of the five volume *Encyclopedia of Database Systems* (Springer). She is currently on the editorial board of several international journals, such as *Distributed and Parallel Databases* (DAPD, Springer), *Journal of Parallel and Distributed Computing (JPDC)*, *IEEE Transactions on Service Computing (TSC)*, *ACM Transactions on Web (TWEB)*, and *Wireless Network (WINET, Springer)*. She is a senior member of the IEEE.

Ting Wang received the BS degree from Zhejiang University and MS degree from the University of British Columbia, both in computer science. He is working toward the PhD degree in the School of Computer Science, Georgia Institute of Technology, where he conducts research at the intersection of database and networked computing systems. His research interests primarily lie in the management and mining of large-scale, networked, and possibly private data sets and data streams. He is a student member of the IEEE.