Reliable State Monitoring in Cloud Datacenters

Shicong Meng[†] Arun K. Iyengar[‡] Isabelle M. Rouvellou[‡] Ling Liu[†]
Kisung Lee[†] Balaji Palanisamy[†] Yuzhe Tang[†]
[†]College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA
[‡]IBM Research T.J. Watson, Hawthorne, NY 10532, USA
{smeng@cc., lingliu@cc., kslee@, balaji@cc., yztang@}gatech.edu {aruni, rouvellou}@us.ibm.com

***Abstract*—State monitoring is widely used for detecting critical events and abnormalities of distributed systems. As the scale of such systems grows and the degree of workload consolidation increases in Cloud datacenters, node failures and performance interferences, especially transient ones, become the norm rather than the exception. Hence, distributed state monitoring tasks are often exposed to impaired communication caused by such dynamics on different nodes. Unfortunately, existing distributed state monitoring approaches are often designed under the assumption of always-online distributed monitoring nodes and reliable inter-node communication. As a result, these approaches often produce misleading results which in turn introduce various problems to Cloud users who rely on state monitoring results to perform automatic management tasks such as auto-scaling.**

This paper introduces a new state monitoring approach that tackles this challenge by exposing and handling communication dynamics such as message delay and loss in Cloud monitoring environments. Our approach delivers two distinct features. First, it quantitatively estimates the accuracy of monitoring results to capture uncertainties introduced by messaging dynamics. This feature helps users to distinguish trustworthy monitoring results from ones heavily deviated from the truth, yet significantly improves monitoring utility compared with simple techniques that invalidate all monitoring results generated with the presence of messaging dynamics. Second, our approach also adapts to non-transient messaging issues by reconfiguring distributed monitoring algorithms to minimize monitoring errors. Our experimental results show that, even under severe message loss and delay, our approach consistently improves monitoring accuracy, and when applied to Cloud application auto-scaling, outperforms existing state monitoring techniques in terms of the ability to correctly trigger dynamic provisioning.

I. Introduction

State monitoring is a fundamental building block for many distributed applications and services hosted in Cloud datacenters. It is widely used to determine whether the aggregated state of a distributed application or service meets some predefined conditions [1]. For example, a web application owner may use state monitoring to check if the aggregated access observed at distributed application-hosting servers exceeds a pre-defined level [2]. Table I lists several common applications of state monitoring.

Most existing state monitoring research efforts have been focused on minimizing the cost and the performance impact of state monitoring. For example, a good number of state monitoring techniques developed in this line of works focus on the threshold based state monitoring by carefully partitioning monitoring tasks between local nodes and coordinator nodes such that the overall communication cost is minimized [3][2][4][1]. Studies along this direction often make strong assumptions on monitoring-related communications, such as 100% node availability and instant message delivery.

These assumptions, however, often do not hold in real Cloud deployments. Many Cloud systems and applications utilize hundreds or even thousands of computing nodes to achieve high throughput and scalability. At this level of scale, node/network failures, especially transient ones, are fairly common [5][6]. Furthermore, resource sharing techniques and virtualization in Clouds often introduce performance interferences and degradation, and cause computing nodes to respond slowly or even become temporarily unavailable [7][8]. Such unpredictable dynamics in turn introduce message delay and loss which we refer to as message dynamics. Monitoring approaches designed without considering such messaging dynamics would inevitably produce unreliable results. Even worse, users are left in the dark without knowing that the monitoring output is no longer reliable. For instance, state monitoring techniques assuming 100% node availability or instant message delivery [3][1] would wait for messages from failed nodes indefinitely without notifying users about potential errors in monitoring results. Consequently, actions performed based on such unreliable results can be harmful or even catastrophic [9]. Furthermore, simple error-avoiding techniques such as invalidating monitoring results when messaging dynamics exist do not work well either, as certain issues such as performance interferences can last fairly long and the scale of Cloud monitoring tasks makes failures very common. For example, even if the probability of one node failing is 0.001, the probability of observing messaging dynamics in a task involving 500 nodes would be $1-(1-0.001)^{500} \approx 0.4$. Invalidating monitoring results whenever problems exist would render 40% monitoring results useless.

In this paper, we present a new state monitoring framework that incorporates messaging dynamics in terms of message delay and message losses into monitoring results reporting and distributed monitoring coordination. Our framework provides two fundamental features for state monitoring. First, it estimates the accuracy of monitoring results based on the impact of messaging dynamics, which provides valuable information for users to decide whether monitoring results are trustworthy. Second, it minimizes the impact of dynamics whenever possible by continuously adapting to changes in monitoring communication and striving to produce accurate monitoring results. When combined, these two features shape a reliable state monitoring model that can tolerate communication dynamics and mitigate their impact on monitoring results.

To the best of our knowledge, our approach is the first state monitoring framework that explicitly handles messaging dynamics in large-scale distributed monitoring. We perform extensive experiments, including both trace-driven and real deployment ones. The results show that our approach produces good accuracy estimation

Applications	Description
Content Delivery	Monitoring the total access to a file mirrored at multiple servers to decide if serving capacity is sufficient.
Rate Limiting [2]	Limiting a user's total access towards a cloud service deployed at multiple physical locations.
Traffic Engineering [10]	Monitoring the overall traffic from an organization's sub-network (consists of distributed hosts) to the Internet.
Quality of Service [11]	Monitoring and Adjusting the total delay of a flow which is the sum of the actual delay in each router on its path.
Fighting DoS Attack	Detecting DoS Attack by counting SYN packets arriving at different hosts within a sub-network.
Botnet Detection [12]	Tracking the overall simultaneous TCP connections from a set of hosts to a given destination.

TABLE I: Examples of State Monitoring

and minimizes monitoring errors introduced by messaging dynamics via adaptation. Compared with existing monitoring techniques, our approach significantly reduces problematic monitoring results in performance monitoring for Cloud application auto-scaling [13] with the presence of messaging dynamics, and improves application response time by up to 30%.

The rest of this paper is organized as follows. In Section II, we introduce the problem of reliable state monitoring. Section III presents the details of our approach. We discuss our experimental evaluation in Section IV. Section V summarizes related work, and Section VI concludes this paper.

II. Problem Definition

Most existing state monitoring studies employ an instantaneous state monitoring model, which triggers a state alert whenever a predefined threshold is violated. Specifically, the instantaneous state monitoring model [3][14][15][16][17] detects state alerts by comparing the current aggregate value with a global threshold. Given the monitored value on monitor i at time t, $x_i(t), i \in [1, n]$, where n is the number of monitors involved in the monitoring task, and the global threshold T, it considers the state at time t to be abnormal and triggers a state alert if $\sum_{i=1}^{n} x_i(t) > T$, which we refer to as a *global violation*.

To perform state monitoring, the line of existing works employs a distributed monitoring framework with multiple monitors and one coordinator (Figure 1). The global threshold T is decomposed into a set of local thresholds T_i for each monitor i such that $\sum_{i=1}^{n} T_i \leqslant T$. As a result, as long as $x_i(t) \leqslant T_i, \forall i \in [1, n]$, i.e. the monitored value at any node is lower or equal to its local threshold, the global threshold cannot be exceeded because $\sum_{i=1}^{n} x_i(t) \leqslant \sum_{i=1}^{n} T_i \leqslant T$. In this case, monitors do not need to report their local values to the coordinator. When $x_i(t) > T_i$ on monitor i, it is possible that $\sum_{i=1}^{n} x_i(t) > T$ (global violation). Hence, monitor i sends a message to the coordinator to report a *local violation* with the value $x_i(t)$. The coordinator, after receiving the local violation report, invokes a *global poll* procedure which notifies other monitors to report their local values, and then determines whether $\sum_{i=1}^{n} x_i(t) \leqslant T$. The focus of existing works is to find optimal local threshold values that minimize the overall communication cost. For instance, if a monitor i often observes relatively higher x_i, it may be assigned with a higher T_i so that it does not frequently report local violations to the coordinator and trigger expensive global polls.

A. Reliable State Monitoring and Challenges

Existing state monitoring works [3][14][15][16][17][1] often share the following assumptions: 1) nodes involved in a monitoring task are perfectly reliable in the sense that they are always available and responsive to monitoring requests; 2) a monitoring message can always be reliably and instantly delivered from one node to another. These two assumptions, however, do not always hold in Cloud datacenters. First, Cloud applications and services are often hosted by a massive number of distributed computing nodes. Failures, especially transient ones, are common for nodes of such large-scale distributed systems [5], [6]. Second, Cloud datacenters often employ virtualization techniques to consolidate workloads and provide management flexibilities such as virtual machine cloning and live migration. Despite its benefits, virtualization also introduces a number of challenges such as performance interferences among virtual machines running on the same physical host. Such interferences could introduce serious network performance degradation, including heavy message delays and message drops [7], [8]. Note that reliable data delivery protocols such as TCP cannot prevent monitoring message loss caused by failures of monitoring nodes or networks, nor can it avoid message delay.

To provide robustness against messaging dynamics, Jain et al. [9] proposed to employ a set of coarse network performance metrics to reflect the status of monitoring communication, e.g., the number of nodes contributed to a monitoring task. The intention is to allow users to decide how trustworthy monitoring results are based on values of such metrics. While this approach certainly has its merits in certain monitoring scenarios, it also has some limitations.

First, it considers the status of a monitor as either online or offline, and overlooks situations involving message delays. For instance, a monitor node may appear online, but it may introduce considerable latencies to messages sent to or received from it. Such message delays are as important as message loss caused by offline nodes, because they may also lead to misdetection of anomalies. In fact, evidence exists [18] suggesting that communication latency caused by virtual machine interference in virtualized Cloud datacenters is a common and serious issue.

Second, it is difficult for users to interpret the impact of reported network level issues on monitoring accuracy. If one of the nodes fails to report its local monitoring data, are the corresponding monitoring results still reliable? The scale of distributed Cloud monitoring exacerbates the problem as message delay or loss can be quite common given the number of participating nodes, e.g., hundreds of web servers for large Cloud applications, and even thousands of servers for Hadoop clusters. If we simply invalidate the monitoring results whenever message delay or loss occurs, we would end up with frequent gaps in monitoring data and low monitoring utility. On the contrary, if we choose to use such monitoring results, how should we *assess the accuracy of monitoring results* given the observed message delay and loss?

Figure 1 shows a motivating example where a distributed rate limiting monitoring task involves one coordinator and six monitors. As a Cloud service often runs on distributed servers across multiple datacenters, service providers need to perform distributed rate limiting to ensure that the aggregated access rate of a user does not exceed the purchased level. The task in Figure

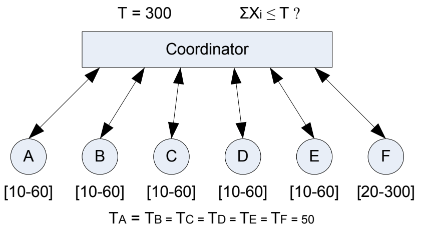


Fig. 1: A Motivating Example

1 continuously checks the access rate of an user (x_i) on all 6 servers (A to F) and triggers a state alert when the sum of the access rates on all 6 sites exceed the global threshold $T = 300$. The numbers under each monitor indicate the range of values observed by the monitor. Such range statistics can be obtained through long-term observations. For simplicity, we assume local thresholds employed by all monitors have the same value 50, i.e. $T_A = T_B = T_C = T_D = T_E = T_F = 50$.

Estimating monitoring accuracy based on messaging dynamics information is difficult. Simply using the scope of message delay or loss to infer accuracy can be misleading. For example, if monitor A, B, and C (50% of total monitors) all fail to respond to a global poll during a transient failure, one may come to the conclusion that the result of global poll should be invalidated as half of the monitors do not contribute to the result. However, as monitor A, B and C observe relatively small monitored values (e.g., most users access server F which is geographically closer), the corresponding global poll results may still be useful. For instance, if the global poll suggests that $x_D + x_E + x_F = 100$, we can conclude that there is no global violation, i.e. $\sum_i^{i=\{A...F\}} x_i \leqslant 300$ with high confidence, because the probability of $\sum_i^{i=\{A...C\}} x_i \leqslant 180$ is fairly high given observed value ranges of A, B and C. On the contrary, if monitor F fails to respond, even though F is only one node, the uncertainty of monitoring results increases significantly. For example, if $\sum_i^{i=\{A...E\}} x_i = 150$, it is hard to tell whether a global violation exists due to the high variance of F's observed values.

An ideal approach should provide users an intuitive accuracy estimation such as "the current monitoring result is correct with a probability of 0.93", instead of simply reporting the statistics of message delay or loss. Such an approach must quantitatively estimate the accuracy of monitoring results. It should also be aware of state monitoring algorithm context as the algorithm has two phases, the local violation reporting phase and global poll phase.

Third, accuracy estimation alone is not enough to provide reliable monitoring and minimize the impact of messaging quality degradation. Resolving node failures may take time. Network performance degradation caused by virtual machine interferences often lasts for a while until one virtual machine is migrated to other hosts. As a result, messaging dynamics can last for some time. Without self-adaptive monitoring to minimize the corresponding accuracy loss, users may lose access to any meaningful monitoring result during a fairly long period, which may not be acceptable for Cloud users who pay for using Cloud monitoring services such as CloudWatch [19]. For instance, if node F continuously experiences message loss, local violation reports sent from F are very likely to be dropped. Consequently, the coordinator does not trigger global polls when it receives no local violation reports. If a true violation exists, e.g., $x_A = 45, x_B = 45, x_C = 45, x_D = 45, x_E = 45, x_F = 110$ and $\sum_i^{i=\{A...F\}} x_i = 335$, the coordinator will mis-detect it.

One possible approach to reduce monitoring errors introduced by such messaging dynamics is to let healthy nodes, i.e. nodes not affected by messaging dynamics, to report their local values at a finer granularity to compensate for the information loss on problem nodes. In the above example, if we reduce local thresholds on nodes A, B, C, D and E to 30. the coordinator will receive local violations from nodes A, B, C, D and E, and trigger a global poll. Even if F also fails to respond to the global poll, the coordinator can find that $\sum_i^{i\in\{A,...,E\}} x_i = 225$. For the sake of the example, suppose x_F is uniformly distributed over $[20, 300]$. The coordinator can infer that the probability of a global violation is high. This is because a global violation exists if $x_F > 75$ which is very likely (> 0.8) given x_F's distribution. Similarly, adaptation can also be used to rule out the possibility of global violations. For instance, if node E is troubled by messaging dynamics, we can increase E's local threshold to 70 so that the probability of detecting local violation on E is trivial. Correspondingly, we also reduce the thresholds on the rest of the nodes to 45 to ensure the correctness of monitoring ($\sum_i T_i \leqslant T$). As a result, as long as $\sum_i^{i\in\{A,...,D,F\}} x_i < 230$, we can infer that there is no global violation with high probability, even though node E is under the impact of messaging dynamics.

While this type of self-adaptation seems promising, designing such a scheme is difficult and relies on answers to a number of fundamental questions: how should we divide the global thresholds when there are multiple problem nodes to minimize the possible error they may introduce, especially when they observe different levels of message and delay? In the rest of this paper, we address these challenges and present details of our reliable state monitoring approach.

III. Reliable State Monitoring

State monitoring continuously checks whether a monitored system enters a critical pre-defined state. Hence, state monitoring tasks usually generate binary results which indicate either "state violation exists"(positive detection) or "no state violation exists"(negative detection). Beyond this basic result, our reliable state monitoring approach also marks the estimated accuracy of a monitoring result in the form of error probabilities. For positive detections, the error probability is the probability of false positives. The error probability is the probability of false negatives for negative detections.

To perform accuracy estimation, we design estimation schemes for both local violation reporting and global poll processes respectively. These schemes leverage the information on messaging dynamics and per-node monitored value distributions to capture uncertainties caused by messaging dynamics. In addition, we also examine the unique problem of out-of-order global polls caused by message delay. The final accuracy estimation results synthesize the uncertainties observed at different stages of the state monitoring algorithm.

Besides accuracy estimation, our approach also minimizes errors caused by non-transient messaging dynamics via two parallel directions of adjustments on distributed monitoring parameters. One tries to minimize the chance that troubled nodes deliver

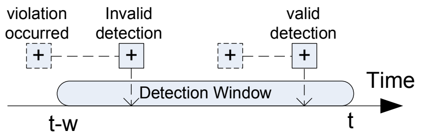


Fig. 2: Detection Window

local violation reports to the coordinator. Since they may fail to deliver these reports, such adjustments essentially minimize the uncertainties caused by them. The other direction of adjustments is to configure healthy nodes to report their local monitored values more often. This allows the coordinator to make better accuracy estimation which in turn helps to detect or rule out a global violation with high confidence.

A. *Messaging Dynamics*

Although a Cloud datacenter may encounter countless types of failures and anomalies at different levels (network/server/OS/etc.), their impact on monitoring related communication can often be characterized by message delay and message loss. For brevity, we use the term *messaging dynamics* to refer to both message delay and loss. Depending on the seriousness of messaging dynamics, the monitoring system may observe different difficulties in inter-node communication, from slight message delay to complete node failure (100% message loss rate or indefinite delay).

The focus of our study is utilizing message delay and loss information to provide reliable state monitoring functionalities via accuracy estimation and accuracy-driven self-adaptation. Our approach obtains message delay and loss information in two ways. One is direct observation in global polls, e.g., the coordinator knows whether it has received a response from a certain monitor on time. The other is utilizing existing techniques such as [9] to collect pair-wise message delay and loss information between a monitor and the coordinator. Note that our approach is orthogonal to the messaging quality measurement techniques, as it takes the output of the measurement to perform accuracy estimation and self-adaptation. Our approach only requires basic messaging dynamics information. For message delay, it requires a histogram that records the distribution of observed message delays. For message loss, it takes the message loss rate as input.

B. *Detection Window*

We introduce the concept of detection window to allow users to define their tolerance level of result delays. Specifically, a detection window is a sliding time window with length w. We consider a global violation V detected at time t a correctly detected one if its actual occurrence time $t_o \in [t-w, t]$. Note that multiple global violations may occur between the current time t and $t-w$ as Figure 2 shows. We do not distinguish different global violations within the current detection window, as users often care about whether there exists a global violation within the detection window instead of exactly how many global violations there are. The concept of detection window is important for capturing the dynamic nature of state monitoring in real world deployment.

C. *Accuracy Estimation*

Recall that the distributed state monitoring algorithm we introduced in Section II has two stages, the local violation reporting stage and the global poll stage. As message delay and loss have an impact on both stages, our analysis on their accuracy impact needs to be conducted separately. When message delay or loss occurs during local violation reporting, the coordinator may fail to receive a local violation report and trigger a global poll in time. Consequently, it may mis-detect a global violation if one does exist, and introduce false negative results. To estimate the monitoring accuracy at this stage, the coordinator continuously updates the estimated probability of failing to receive one or more local violations based on the current messaging dynamics situation and per-monitor value distribution. When message delay or loss occurs during a global poll, the coordinator cannot collect all necessary information on time, which again may cause the coordinator to mis-detect global violation and introduces false negatives. Hence, we estimate the probability of mis-detecting a global violation based on collected values during the global poll and the value distribution of troubled monitors.

Local Violation Reporting. To facilitate the accuracy estimation at the local violation reporting stage, each monitor maintains a local histogram that records the distribution of local monitored values. Much previous research [14][16][20][1] suggests that such distribution statistics of recent monitored values provide good estimation on future values. Specifically, each monitor maintains a histogram of the values that it sees over time as $H_i(x)$ where $H_i(x)$ is the probability of monitor i observing the value x. We use equi-depth histograms to keep track of the data distribution. For generality purposes, we assume that the monitored value distribution is independent of messaging dynamics. To ensure that the histogram reflects recently seen values more prominently than older ones, each monitor continuously updates its histogram with exponential aging. A monitor also periodically sends its local histogram to the coordinator.

We first look at the probability of monitor i failing to report a local violation which can be computed as follows,

$$P(f_i) = P(v_i)P(m_i)$$

where $P(v_i)$ is the probability of detecting a local violation on monitor i, and $P(m_i)$ is the probability of a message sent from monitor i failing to reach the coordinator due to messaging dynamics. $P(v_i) = P(x_i > T_i)$ where x_i and T_i are the monitored value and the local threshold on monitor i respectively. $P(x_i > T_i)$ can be easily computed based on T_i and the distribution of x_i provided by the histogram of monitor i. $P(m_i)$ depends on the situation of message delay and loss. Let $P(p_i)$ be the probability of a message sent from monitor i to the coordinator being dropped. Let $P(d_i)$ be the probability of a reporting message sent from monitor i to the coordinator being delayed beyond users' tolerance, i.e. the local violation report is delayed more than a time length of w (the detection window size) so that the potential global violation associated with the delayed local violation report becomes invalid even if detected. Given $P(p_i)$ and $P(d_i)$, we have

$$P(m_i) = 1 - (1 - P(p_i))(1 - P(d_i))$$

The rational here is that if a local violation report successfully reaches the coordinator, it must not have been dropped or heavily delayed at the same time. Both $P(p_i)$ and $P(d_i)$ can be easily determined based on the measurement output of messaging dynamics. $P(p_i)$ is simply the message loss rate. $P(d_i)$ can be computed as $P(d_i) = P(l_i > w)$ where l_i is the latency of messages sent from monitor i to the coordinator, and $P(l_i > w)$ is easy to obtain given the latency distribution of messages. Clearly, $P(m_i)$ grows with $P(p_i)$ and $P(d_i)$, and $P(m_i) = 0$ when messaging dynamics do not exist.

During the local violation reporting phase, the overall probability of the coordinator failing to receive local violations $P(F)$ depends on all monitors. Therefore, we have

$$P(F) = 1 - \prod_{i}^{n}(1 - P(f_i))$$

where n is the number of monitors and we consider local violations on different monitors are independent for generality. Clearly, $P(F)$ grows with the number of problem monitors. With $P(F)$, the probability of false negatives caused by missing local violation reports P_l can be estimated as $P_l = cP(F)$ where c is referred as the conversion rate between local violations and global violations, i.e., the percentage of local violations leading to true global violations. The coordinator maintains c based on its observations of previous local violations and global violations.

Global Polls. Recall that in the original state monitoring algorithm, when the coordinator receives a local violation report, it initiates the global poll process, where it requests all monitors to report their *current* local monitored values. However, when message delay and loss exist, the coordinator may receive a delayed report about a local violation that actually occurs at an earlier time t. As a result, when the coordinator invokes a global poll, it requests all monitors to report their *previous* local monitored values observed at time t. To support this functionality, monitors locally keep a record of previous monitored values observed within the detection window (a sliding time window with size w). Values observed even earlier are discarded as the corresponding monitoring results are considered as expired.

Once the coordinator initiates the global poll process, our accuracy estimation also enters the second stage, where we estimate the possibility of mis-detecting global violations due to message delay and loss in the global poll process. The estimation starts when the coordinator does not receive all responses on time. Since the coordinator does not report anything until it receives all monitoring data, the probability of detecting a state violation given the set of received monitored values is

$$P(V) = P\{\sum_{i \in K} x_i > T - \sum_{i \in \bar{K}} x_i\} \tag{1}$$

where K is the set of monitors whose responses do not reach the coordinator, and $\bar{K}$ are the rest of the monitors. The right hand side of the equation can be determined based on the value histogram of monitors. At any time point, the probability of detecting global violation is the probability of detecting global violation within the time window of delay tolerance.

Out-of-Order Global Polls. Due to the existence of message delays, local violation reports sent from different monitors may arrive out-of-order. Accordingly, as new global poll processes may start before previous global poll processes finish, the coordinator may be involved in multiple ongoing global poll processes at the same time as Figure 3 shows.

When the coordinator receives local violation reports r, it first checks its timestamp t_r (local violation occurring time) to see if $t_r \geqslant t - w$ where t is the current time (report receiving time) and w is the user-specified detection window size. If true, it ignores the local violation report as the violation report is expired. Otherwise, it initiates a global poll process and uses t_r as its timestamp. As each global poll may take different time to finish (due to message delay or loss), the coordinator continuously checks the lifetime of global polls and removes those with t_r that $t_r \geqslant t - w$.

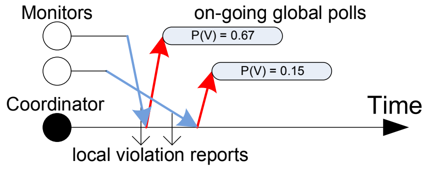


Fig. 3: Out-of-order Global Polls

For accuracy estimation, users are interested in whether there exists one or more global violations within the time interval of $[t - w, t]$. When there are multiple ongoing global polls, it means that there are multiple potential global violations requiring verification. Accordingly, our accuracy estimation should be on whether there exists at least one ongoing global poll leading to global violation.

Let $P_j(V)$ be the probability of triggering global violation in global poll j. $P_j(V)$ can be determined based on Equation 1. The probability P_g of at least one global poll out of M ongoing ones triggering global violation is

$$P_g = 1 - \Pi_{j=1}^{M}(1 - P_j(V))$$

Clearly, P_g increases quickly when the coordinator observes a growing number of ongoing global polls. If P_g is sufficiently high, our monitoring algorithm will report possible state violation. This is particularly useful for situations with a few monitors suffering serious message delay or loss, because no global polls can finish if these nodes cannot send their responses in time and the coordinator can never trigger a global violation if running existing state monitoring algorithms.

Combining Estimations of Both Stages. While we have considered the accuracy estimation problem for local violation reporting and global poll stages separately, a running coordinator often experiences both local violation failures and incomplete global polls at the same time. Hence, combining estimation on both stages is critical for delivering correct accuracy estimation results. The overall probability of false negatives can be computed as $\beta = 1 - (1 - P_l)(1 - P_g)$ where P_l and P_g are the probability of false negatives introduced by failed local violation reporting and global polls respectively. Note that $\beta \neq P_l + P_g$ as the event of miss-detecting a global violation due to failed local violation reporting and the event of miss-detecting a global violation due to failed global polls are not mutually exclusive.

A Balanced State Monitoring Algorithm. The original state monitoring algorithm invokes global polls only when it receives local violation reports, and triggers state alerts only after the coordinator collects responses from all monitors. When messaging dynamics exist, such an algorithm has two issues. First, it may miss opportunities to invoke global polls. Second, it never produces false positive results, but may introduce many false negatives results. We introduce a balanced state monitoring algorithm that minimizes the overall monitoring error. The balanced algorithm is obtained through two revisions on the original algorithm. First, when $P(F)$, the probability of failing to receive local violation reports at the coordinator, is sufficiently large (e.g., $\geqslant 0.95$), the algorithm triggers a global poll. Second, if the estimated false negative probability β in the global poll phase rises above 50%, the monitoring algorithm also reports state violation with a false positive probability $1-\beta$. The balanced algorithm is more likely to detect global violations compared with the original algorithm, especially when β is large.

D. Accuracy-Oriented Adaptation

Sometimes monitors may experience long-lasting message loss and delays. For instance, a Xen-based guest domain continuously generating intensive network IO may cause considerable CPU consumption on `Domain0`, which further leads to constant packet queuing for other guest domains running on the same host [7][8]. As a result, monitor processes running on troubled guest domains would experience continuous messaging dynamics until the performance interference is resolved. Reliable state monitoring should also adapt to such non-transient messaging dynamics and minimize accuracy loss whenever possible.

Recall that the distributed state monitoring algorithm employs local thresholds to minimize the amount of local violation reports sending to the coordinator. This technique, however, introduces extra uncertainties when messaging dynamics exist, because the coordinator cannot distinguish the case where a monitor does not detect local violation from the case where a monitor fails to report a local violation. Our approach minimizes such uncertainties through two simultaneous adjustments of local thresholds. First, it adjusts local thresholds on troubled monitors to reduce its chance of detecting local violations, as the corresponding reports may not reach the coordinator which in turn introduces uncertainties. Second, it also adjusts local thresholds on healthy monitors to increase their local violation reporting frequencies to maximize the information available to the coordinator so that it can provide good accuracy estimation. The adjustment on healthy monitors is also important for monitoring correctness where we ensure $\sum_i^n T_i \leqslant T$.

As the impact of message delay and loss to local violation reporting can be measured by the expected number of failed local violation reports $E(\text{fr})$, we formulate the local threshold adjustment problem as a constrained optimization problem as follows,

$$\begin{array}{ll} \min & E(fr) = \Sigma_i^n P(v_i|T_i)P(m_i) \\ \text{s.t.} & \Sigma_i^n T_i \leqslant T \end{array}$$

where $P(v_i|T_i)$ is the conditional probability of reporting local violation on monitor i given its local threshold T_i and $P(m_i)$ is the probability of failing to send a message to the coordinator. Since we do not have a closed form for $P(v_i|T_i) = P(x_i > T_i)$ (only histograms of x_i), we replace $P(v_i|T_i)$ with its upper bound $\overline{P(v_i|T_i)}$ by applying Markov's inequality (Chebyshev's inequality does not yield a closed form) where $P(|x_i| > T_i) \leqslant \frac{E(|x_i|)}{T_i}$. Since x_i is positive in most scenarios and $E(|x_i|)$ can be obtained through x_i's histograms, applying this approximation and Lagrange multiplier leads us to a closed form solution. We find the resulting adjustments perform well in practice. In addition, we invoke adaptation only when at least one node experiences relatively long-lasting (e.g., 5 minutes) messaging dynamics to avoid frequent adaptation.

IV. Evaluation

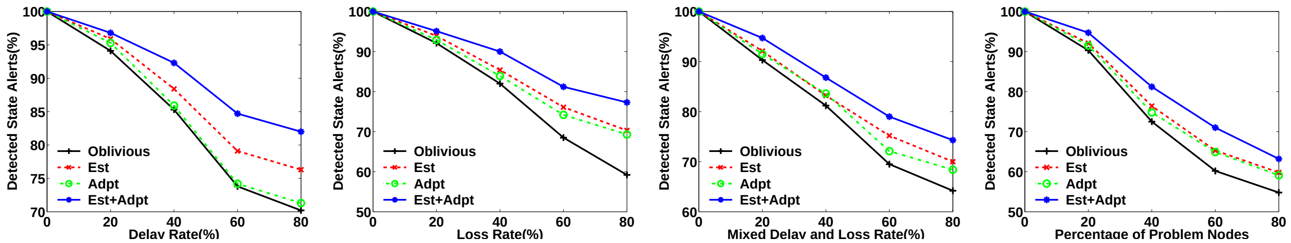
Our experiments consist of both trace-driven simulation and real system evaluation. The trace-driven experiment evaluates the performance of our approach with access traces of the official 1998 WorldCup website hosted by 30 servers distributed across the globe [21]. We used the server log data consisting of 57 million page requests distributed across servers. We evaluate the monitoring accuracy achieved by our approach for a variety of messaging dynamics in this set of experiments. The other part of our experiments leverages our monitoring techniques to support auto-scaling of Cloud applications where server instances can be added to the resource pool of an application dynamically based on the current workload [13]. We deploy a distributed RUBiS [22], an auction web application modeled after eBay.com for performance benchmarking, and use state monitoring to trigger new server instance provisioning. For the real system evaluation, we are interested in the impact of improved monitoring accuracy on real world application performance.

A. Results

Figure 4 shows the state violation detection percentage of different monitoring approaches under different levels and types of messaging quality degradation. Here the y-axis is the percentage of state violations detected by the monitoring algorithm over state violations detected by an oracle which can detect all violations in a given trace. In our comparison, we consider four monitoring algorithms: 1) Oblivious, the existing instantaneous monitoring algorithm which is oblivious to inter-node messaging quality; 2) Est, the instantaneous monitoring algorithm enhanced with our accuracy estimation techniques; 3) Adpt, the instantaneous monitoring algorithm enhanced with our accuracy-oriented adaptation techniques; 4) Est+Adpt, the instantaneous monitoring algorithm enhanced with both estimation and adaptation techniques.

We emulate a distributed rate limiting monitoring task which triggers state violations whenever it detects the overall request rate (the sum of request rates on all monitors) exceeds a global threshold (set to 3000 per second). The task involves 30 monitors, each of which monitors the request rate of one server by reading the corresponding server request trace periodically. Furthermore, we set the detection window size to be 15 seconds, which means a state violation is considered as successfully detected if the time of detection is at most 15 seconds later than the occurrence time of the state violation.

Figure 4(a) illustrates the performance of different algorithms under increasing message delay. Here the x-axis shows the levels of injected message delay. For $k\%$ delay rate, we pick $k\%$ of messages of a problem monitor and inject a delay time randomly chosen from 5 to 60 seconds. By default, we randomly pick 10% of monitors to be problem monitors. While there are many ways to inject message delays, we use the above injection method for the sake of simplicity and interpretation. The detection rate of the oblivious algorithm drops quickly as delay level increases, primarily because its global poll process always waits until messages from all monitors arrive and the resulting delay on the violation reporting often exceeds the delay tolerance interval. The Est algorithm performs much better as it can estimate the probability of a state violation based on incomplete global poll results, which allows the Est scheme to report state violation when



(a) (b) (c) (d)

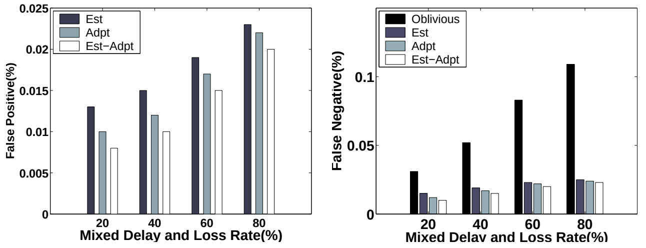
Fig. 4: State Violation Detection Rate (10% problem nodes by default): (a) under increasing message delay rates; (b) under increasing message loss rates; (c)under increasing mixed message delay and loss rates; (d) with increasing number of problem monitors

the estimated probability is high (above 0.9 in our experiment). For instance, when an incomplete global poll yields a total request rate close to the global threshold, it is very likely that a state violation exists even though responses from problem monitors are not available. The Adpt scheme, however, provides limited improvement when used alone. This is because accuracy-oriented adaptation by itself only reduces the chance of a problem monitor reporting local violation. Without accuracy estimation, the Adpt scheme still waits for all responses in global polls. With both accuracy estimation and adaptation, the Est+Adpt scheme achieves significantly higher detection rate.

In Figure 4(b), we use different levels of message loss to evaluate the performance of different algorithms. Similar to the injection of delay, we randomly pick $k\%$ messages of a problem node to drop for a $k\%$ loss rate. The relative performance of the four algorithms is similar to what we observed in Figure 4(a), although the detection rate achieved by each algorithm drops slightly compared with that in Figure 4(a) as delayed messages often still help to detect state violation compared with completely dropped messages.

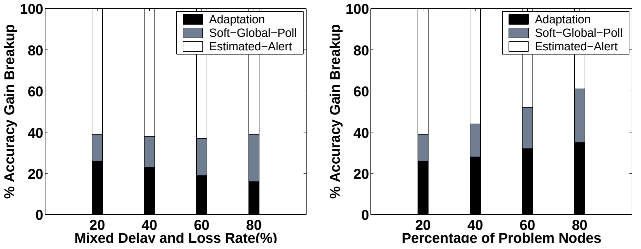
For the rest of the experiments, we inject mixed message delay and loss, instead of message delay or loss alone, for comprehensive reliability evaluation. Similarly, the $k\%$ mixed delay and loss rate means that $k/2\%$ of the messages are randomly chosen to drop and another $k/2\%$ of the messages are randomly chosen to add delays. Figure 4(c) shows the violation detection performance of different algorithms given increasing levels of mixed message delay and loss. We observe similar results in this figure, and the performance achieved by our approach lies between those achieved in the two previous figures. In Figure 4(d), we vary the scope of problem nodes from 20%(the default case) to 80%. The result suggests that our approach consistently improves monitoring accuracy. Nevertheless, when problem monitors becomes dominant, its performance is relatively worse that that in the three previous figures.

Figure 5(a) shows the corresponding percentage of false positives (reporting state violations when none exist) produced. Recall that the original monitoring algorithm (Oblivious) does not produce false positives (as its global polls report state violations only when the completely collected responses confirm state violations), and therefore, is not included in Figure 5(a). Both estimation and adaptation have low false positive rates, and when combined, they achieve even lower false positive rates. Figure 5(b) shows the false negative (reporting no state violation when at least one exists) rates of all schemes. While the Oblivious scheme produces no false positives, it suffers from high false negative rates as it waits for delayed or dropped messages indefinitely. By contrast, all of our three schemes achieve fairly low false negative rates.



(a) (b)

Fig. 5: Errors in State Violation Detection: (a) comparison of false positive; (b) comparison of false negative;

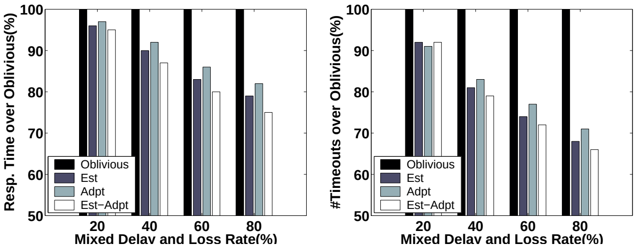


(a) (b)

Fig. 6: Accuracy Improvement Breakup: (a) with increasing message loss and delay levels; (b) with increasing percentage of problem monitors.

Figure 6 illustrates the three key efforts our approach makes to improve monitoring accuracy and the corresponding portion of correctly reported state violations that are missed by the original monitoring algorithm. Here *Adaptation* refers to the effort of reconfiguring local threshold, *Soft-Global-Poll* refers to the effort of triggering global polls when the estimated local violation reporting probability is high (instead of receiving a local violation), and *Estimated-Alert* refers to the effort of reporting state violation when the estimated probability is sufficiently high. Note that multiple efforts may contribute to a correctly reported state violation at the same time. Among the three efforts in both Figure 6(a) and Figure 6(b), *Estimated-Alert* clearly contributes the most as incomplete global polls are the main reason for false negatives in the original monitoring algorithm.

Figure 7 shows the performance difference of RUBiS with auto-scaling enabled by different monitoring schemes. We deploy a PHP version of RUBiS in Emulab [23] where it has a set of web servers and a database backend. Each web server runs in a small footprint XEN-based virtual machine (1 vCPU), and the database



(a)

(b)

Fig. 7: Application on Cloud application auto-scaling: (a) comparison of response time; (b) comparison of timeouts.

runs on a dedicated physical machine. This is to ensure that the database is not the performance bottleneck. We periodically introduce workload bursts to RUBiS, and use state monitoring to check if the total number of timeout requests on all web servers exceeds a given threshold, i.e., one monitor runs on one web server to observe local timeout requests. RUBiS initially runs with 5 web servers. When violations are detected, we gradually add new web servers one by one to absorb workload bursts until no violation is detected (auto-scaling). Similarly, when no violations are detected for 5 minutes, we gradually remove dynamically added web servers one by one.

We introduce messaging delay and loss to monitor-coordinator communication in the same way as that in the trace-driven experiments where 10% of nodes experience a given rate of message loss and delay. The y-axis of Figure 7 shows the average response time and the timeout request number of RUBiS requests which are normalized by those of the oblivious scheme. Clearly, as our enhanced schemes detect more state violations, they can more reliably trigger auto-scaling when there is a workload burst, which in turn reduces response time and request timeout by up to 30%. In addition, accuracy estimation achieves higher detection rates compared with self-adaptation. This is because monitors on load balanced web servers often observe similar timeouts, and accuracy estimation can still confirm global violations based on partial monitoring data.

V. Related Work

Most existing state monitoring works [3][16][15][17][1] study communication efficient detection of constraint violation. These approaches often assume reliable inter-node communication, and are subject to producing misleading results with the presence of messaging dynamics that are common in Cloud monitoring environments. Jain and et al. [9] studies the impact of hierarchical aggregation, arithmetic filtering and temporary batching in an unreliable network. They propose to gauge the degree of inaccuracy based on the number of unreachable monitoring nodes and the number of duplicated monitoring messages caused by DHT overlay maintenance. While this work provides insight for understanding the interplay between monitoring efficiency and accuracy given message losses, it also has several limitations as we mentioned in Section II-A such as not considering delay and difficulties in assessing application level monitoring accuracy. Our work is complementary to [9] as we advance the understanding of monitoring reliability by studying accuracy estimation and self-adaptation in state monitoring.

VI. Conclusions and Future Work

We have presented a reliable state monitoring approach that enables estimation of monitoring accuracy based on observed messaging dynamics and self-adaption to disruptions. We built a prototype system based on this approach and evaluated the system in various settings. The results showed that our approach can effectively invoke web application auto-scaling during workload bursts even with substantial monitoring message dynamics and reduces request timeouts by up to 30% over existing state monitoring techniques. As part of our ongoing work, we are working on safeguarding multiple state monitoring tasks. We are also studying providing reliability features to other types of monitoring.

Acknowledgment

This work was initiated while the first author was interning with IBM T.J. Watson and also supported by an IBM PhD fellowship for the first author. The last three authors are partially supported by grants from NSF NetSE and NSF CyberTrust, an IBM faculty award and a grant from Intel ISTC Cloud Computing.

References

[1] S. Meng, T. Wang, and L. Liu, “State monitoring in cloud datacenters,” *IEEE Transactions on Knowledge and Data Engineering (TKDE), Special Issue on Cloud Data Management*, 2011.

[2] B. Raghavan, K. V. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, “Cloud control with distributed rate limiting,” in *SIGCOMM07*.

[3] M. Dilman and D. Raz, “Efficient reactive monitoring,” in *INFOCOM01*.

[4] S. Meng, S. R. Kashyap, C. Venkatramani, and L. Liu, “Remo: Resource-aware application state monitoring for large-scale distributed systems,” in *ICDCS*, 2009, pp. 248–255.

[5] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” in *OSDI*, 2004, pp. 137–150.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, “Bigtable: A distributed storage system for structured data,” in *OSDI*, 2006.

[7] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, “Enforcing performance isolation across virtual machines in xen,” in *Middleware06*.

[8] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, “Understanding performance interference of i/o workload in virtualized cloud environments,” in *IEEE Cloud*, 2010.

[9] N. Jain, P. Mahajan, D. Kit, P. Yalagandula, M. Dahlin, and Y. Zhang, “Network imprecision: A new consistency metric for scalable monitoring,” in *OSDI*, 2008, pp. 87–102.

[10] L. Gao, M. Wang, and X. S. Wang, “Quality-driven evaluation of trigger conditions on streaming time series,” in *SAC*, 2005.

[11] L. Raschid, H.-F. Wen, A. Gal, and V. Zadorozhny, “Monitoring the performance of wide area applications using latency profiles,” in *WWW03*.

[12] G. Gu, R. Perdisci, J. Zhang, and W. Lee, “Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection,” in *USENIX Security Symposium*, 2008, pp. 139–154.

[13] “Auto scaling,” http://aws.amazon.com/autoscaling/.

[14] C. Olston, J. Jiang, and J. Widom, “Adaptive filters for continuous queries over distributed data streams,” in *SIGMOD Conference*, 2003.

[15] R. Keralapura, G. Cormode, and J. Ramamirtham, “Communication-efficient distributed monitoring of threshold counts,” in *SIGMOD*, 2006.

[16] I. Sharfman, A. Schuster, and D. Keren, “A geometric approach to monitor threshold func. over distrib. data streams,” in *SIGMOD*, 2006.

[17] S. Agrawal, S. Deb, K. V. M. Naidu, and R. Rastogi, “Efficient detection of distributed constraint violations,” in *ICDE*, 2007.

[18] “Visual evidence of amazon ec2 network issues,” https://www.cloudkick.com/blog/2010/jan/12/visual-ec2-latency/, 2010.

[19] “Amazon cloudwatch beta,” http://aws.amazon.com/cloudwatch/.

[20] N. Jain, M. Dahlin, Y. Zhang, D. Kit, P. Mahajan, and P. Yalagandula, “Star: Self-tuning aggregation for scalable monitoring,” in *VLDB*, 2007.

[21] M. Arlitt and T. Jin, “1998 world cup web site access logs,” http://www.acm.org/sigcomm/ITA/, August 1998.

[22] “Rubis,” http://rubis.ow2.org/.

[23] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, “An integrated experimental environment for distributed systems and networks,” in *OSDI*, 2002.