2013 IEEE 33rd International Conference on Distributed Computing Systems

Volley: Violation Likelihood Based State Monitoring for Datacenters

Shicong Meng Arun K. Iyengar
Isabelle M. Rouvellou
IBM T.J. Watson Research Center
Email: {smeng, aruni, rouvellou}@us.ibm.com

Ling Liu
College of Computing
Georgia Institute of Technology
Email: lingliu@cc.gatech.edu

***Abstract*—Distributed state monitoring plays a critical role in Cloud datacenter management. One fundamental problem in distributed state monitoring is to minimize the monitoring cost while maximizing the monitoring accuracy at the same time. In this paper, we present Volley, a violation likelihood based approach for efficient distributed state monitoring in Cloud datacenters. Volley achieves both efficiency and accuracy with a flexible monitoring framework which uses dynamic monitoring intervals determined by the likelihood of detecting state violations. Volley consists of three unique techniques. It utilizes efficient node-level adaptation algorithms that minimize monitoring cost with controlled accuracy. Volley also employs a distributed scheme that coordinates the adaptation on multiple monitor nodes of the same task for optimal task-level efficiency. Furthermore, it enables multi-task level cost reduction by exploring state correlation among monitoring tasks. We perform extensive experiments to evaluate Volley with system, network and application monitoring tasks in a virtualized datacenter environment. Our results show that Volley can reduce considerable monitoring cost and still deliver user specified monitoring accuracy under various scenarios.**

***Keywords*-Distributed, State, Monitoring, Datacenter, Likelihood, Correlation, Adaptation**

I. Introduction

To ensure performance and availability of distributed systems and applications, administrators often run a large number of monitoring tasks to continuously track the global state of a distributed system or application by collecting and aggregating information from distributed nodes. For instance, to provide a secured datacenter environment, administrators may run monitoring tasks that collect and analyze datacenter network traffic data to detect abnormal events such as Distributed Denial of Service (DDoS) attacks [1]. As another example, Cloud applications often rely on monitoring and dynamic provisioning to avoid violations to Service Level Agreements (SLA). SLA monitoring requires collecting of detailed request records from distributed application-hosting servers, and checks if requests are served based on the SLA (e.g., whether the response time for a particular type of requests is less than a given threshold). We refer to this type of monitoring as *distributed state monitoring* [2], [3], [4], [5] which continuously examines if a certain global state (e.g., traffic flowing to an IP) of a distributed system violates a predefined condition. Distributed state monitoring tasks are useful for detecting signs of anomalies and are widely used in scenarios such as resource provisioning [6], distributed rate limiting [2], QoS maintenance [7] and fighting network intrusion [8].

One substantial cost aspect in distributed state monitoring is the overhead of collecting and processing monitoring data, a common procedure which we refer to as sampling. First, sampling operations can be resource-intensive. For example, sampling in the aforementioned DDoS attack monitoring may involve packet logging and deep packet inspection over large amounts of datacenter network traffic data [9]. Even for relatively simple tasks such as SLA monitoring, the cost can still be substantial for running a large number of such tasks for various applications. This is also the primary reason that commercial monitoring services and systems often provide at most 1-minute level periodical sampling granularity, if not at 5-minute level or 15-minute level [10]. Second, users of Cloud monitoring services (e.g., Amazon's CloudWatch) pay for monetary cost proportional to the frequency of sampling (pay-as-you-go). Such monitoring costs can account for up to 18% of total operation cost [10]. Clearly, sampling cost is a major factor for the scalability and effectiveness of datacenter state monitoring.

Many existing datacenter monitoring systems provide periodical sampling as the only available option for state monitoring (e.g., CloudWatch [10]). Periodical sampling performs continuous sampling with a user-specified, fixed interval. It often introduces a cost-accuracy dilemma. On one hand, one wants to enlarge the sampling interval between two consecutive sampling operations to reduce sampling overhead. This, however, also increases the chance of mis-detecting state violations (e.g., mis-detecting a DDoS attack or SLA violation), because state violations may occur without being detected between two consecutive sampling operations with large intervals. In addition, coarse sampling intervals reduce the amount of data available for offline event analysis, e.g., 15-minute sampling is very likely to provide no data at all for the analysis of an event occurred during a sampling interval. On the other hand, while applying small sampling intervals lowers the chance of mis-detection, it can introduce significantly high resource consumption or monitoring service fees. In general, determining the ideal sampling interval with the best cost and accuracy trade-off for periodical sampling is difficult without studying task-

1063-6927/13 $26.00 © 2013 IEEE
DOI 10.1109/ICDCS.2013.61

specific characteristics such as monitored value distributions, violation definitions and accuracy requirements. It is even harder when such task-specific characteristics change over time, or when a task performs sampling over a distributed set of nodes.

We argue that one useful alternative to periodical sampling is a flexible monitoring framework where the sampling interval can be dynamically adjusted based on how likely a state violation will be detected. This flexible framework allows us to perform intensive monitoring with small sampling intervals when the chance of a state violation occurring is high, while still maintaining overall low monitoring overhead by using large intervals when the chance of a state violation occurring is low. As state violations (e.g., DDoS attacks and SLA violations) are relatively rare during the lifetime of many state monitoring tasks, this framework can potentially save considerable monitoring overhead, which is essential for Cloud state monitoring to achieve efficiency and high scalability.

In this paper, we propose Volley, a violation likelihood based approach for distributed state monitoring. Volley addresses three key challenges in violation likelihood based state monitoring. First, we devise a sampling interval adaptation technique that maintains a user-specified monitoring accuracy while continuously adjusting the sampling interval to minimize monitoring overhead. This interface makes Volley easy to configure as users can specify an intuitive targeted monitoring accuracy (e.g., $< 1\%$ miss-detection), rather than choosing sampling intervals that depend on monitoring tasks and data. It also employs a set of low-cost estimation methods to ensure adaptation efficiency. Second, for tasks that perform sampling over distributed nodes, we develop a distributed coordination scheme that not only safeguards the global task-level monitoring accuracy, but also minimizes the total cost. Finally, Volley also leverages state-correlation to further reduce monitoring cost across related tasks in a datacenter.

To the best of our knowledge, Volley is the first violation-likelihood based state monitoring approach that achieves both efficiency and controlled accuracy. We perform extensive experiments to evaluate the effectiveness of Volley with real world monitoring tasks and data in a testbed virtualized Cloud datacenter environment running 800 virtual machines. Our results indicate that Volley reduces monitoring workloads by up to 90% and at the same time, achieves accuracy that is better or close to the user-specified level. For virtual network monitoring, Volley reduces the CPU utilization of `Dom0` from 20-34% to a mere 5%, which in turn minimizes the interference with application workloads and improves monitoring scalability.

II. Problem Definition

A distributed state monitoring task continuously tracks a certain global state of a distributed system and raises a state alert if the monitored global state violates a given condition. The global state is often an aggregate result of monitored values collected from distributed nodes. For example, assume there are n web servers and the i-th server observes a timeout request rate v_i (the number of timeout requests per unit time). A task may check if the total timeout request rate over all servers exceeds a given level T, i.e., $\sum v_i > T$. The general setting of a distributed state monitoring task includes a set of monitor nodes (e.g., the n web servers) and a coordinator. Each monitor can observe the current value of a variable (v_i) through sampling, and the coordinator aggregates local monitor values to determine if a violation condition is met (e.g., whether $\sum v_i > T$). We say a *state violation* occurs if the violation condition is met.

We define states based on the monitored state value and a user-defined threshold, which is commonly used in many different monitoring scenarios. The monitored state value can be a single metric value (e.g., CPU utilization). It can also be a scalar output of a function taking vector-like inputs, e.g., a DDoS detection function may parse network packets to estimate the likelihood of an attack. We assume that all distributed nodes have a synchronized wall clock time which can be achieved with the Network Time Protocol (NTP) at an accuracy of 200 microseconds (local area network) or 10 milliseconds (Internet) [11]. Hence, the global state can be determined based on synchronized monitored values collected from distributed nodes.

Existing works on distributed state monitoring focus on dividing a distributed state monitoring task into local tasks that can be efficiently executed on distributed nodes with minimum inter-node communication [3], [5]. As a common assumption, a monitoring task employs a user specified sampling interval that is fixed across distributed nodes and over the entire task lifetime (more details in Section VI). In this paper, we address a different yet important problem in distributed state monitoring, how to achieve efficient and accurate monitoring through dynamic sampling intervals.

A. A Motivating Example

DDoS attacks bring serious threats to applications and services running in datacenters. To detect DDoS attacks, some techniques [9] leverage the fact that most existing DDoS attacks lead to a growing difference between incoming and outgoing traffic volumes of the same protocol. For example, a SYN flooding attack often causes an increasing asymmetry of incoming TCP packets with SYN flags set and outgoing TCP packets with SYN and ACK flags set [9]. State monitoring based on such techniques watches the incoming rate of packets with SYN flags set P_i observed on a certain IP address, and the outgoing rate of packets with SYN and ACK flags set P_o observed on the same IP address. It then checks whether the traffic difference $\rho = P_i - P_o$ exceeds a given threshold, and if true, it reports a state alert. Such monitoring tasks collect network packets and perform deep

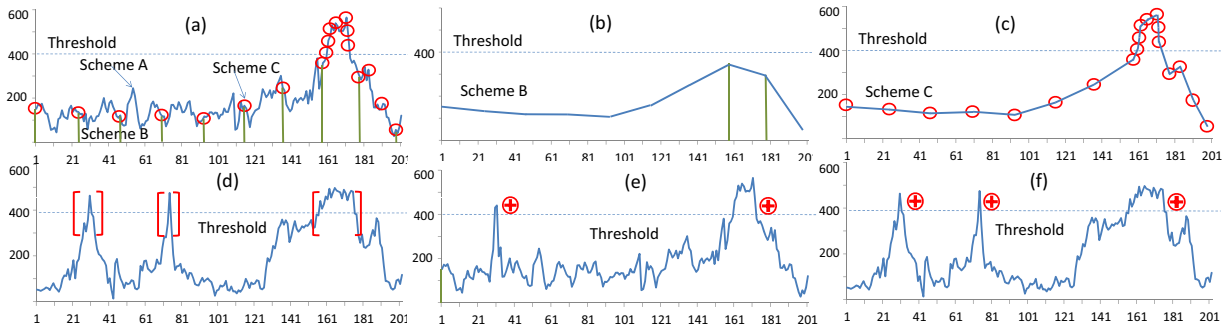


Figure 1. A Motivating Example

packet inspection [12] in repeated monitoring cycles. For brevity, we refer to each cycle of packets collection and processing as sampling.

Periodical Sampling versus Dynamic Sampling. Sampling frequency plays a critical role in this type of DDoS attack monitoring. First, the sampling cost in the DDoS attack monitoring case is non-trivial. As we show later in the evaluation section, frequent collecting and analyzing packets flowing to or from virtual machines running on a server leads to an average of 20-34% server CPU utilization. As a result, reducing the sampling frequency is important for monitoring efficiency and scalability. Second, sampling frequency also determines the accuracy of monitoring. Figure 1 shows an example of a task which monitors the traffic difference between incoming and outgoing traffic for a given IP. The x-axis shows the time where each time point denotes 5 seconds. The y-axis shows the traffic difference ρ. The dashed line indicates the threshold. We first employ high frequency sampling which we refer to as scheme A and show the corresponding trace with the curve in Chart (a). Clearly, scheme A records details of the value changes and can detect the state violation in the later portion of the trace where ρ exceeds the threshold. Nevertheless, the high sampling frequency also introduces high monitoring cost, and most of the earlier sampling yields little useful information (no violation). One may sample less frequently to reduce monitoring cost. For example, scheme B (bar) reduces considerable monitoring cost by using a relatively large monitoring interval. Consequently, as the curve in Chart (b) shows, scheme B also misses many details in the monitored values, and worst of all, it fails to detect the state violation (between the gap of two consecutive samples of scheme B). In general, it is difficult to find a fixed sampling frequency for a monitoring task that achieves both accuracy and efficiency.

One possible way to avoid this issue is to use dynamic sampling schemes where the sampling frequency is continuously adjusted on the fly based on the importance of the results. Scheme C (circle) in Chart (c) gives an example of dynamic sampling. It uses a low sampling frequency at first, but switches to high frequency sampling when a violation is likely to happen.

While such an approach seems promising, it also involves several fundamental obstacles. First, we must find a way to *measure* and *estimate* violation likelihood before using it to adjust sampling intervals. The estimation should also be efficient. Second, since sampling introduces a trade-off between cost and accuracy, a dynamic sampling scheme should provide accuracy control by meeting a user-specified accuracy goal, e.g., "I can tolerate at most 1% state alerts being missed".

Distributed State Monitoring. When the monitored object is a set of servers hosting the same application, DDoS attack monitoring requires collecting traffic data of distributed servers. For example, suppose the traces in Chart (a) and (d) show the traffic difference on two distributed servers. The overall traffic difference on the two servers is the sum of trace values (denoted as v_1 and v_2) in Chart (a) and (d), and the monitoring task now checks if the overall traffic difference exceeds a global threshold T. For the sake of this example, we assume $T = 800$. While one can collect all packets on both servers (monitors) and send them to a coordinator which parses the packets to see if $v_1 + v_2 > T$, a more efficient way is to divide the task into local ones running on each monitor to avoid frequent communication. For example, we can assign monitors in Chart (a) and Chart (d) with local threshold $T_1 = 400$ and $T_2 = 400$ respectively. As a result, as long as $v_1 < T_1$ and $v_2 < T_2$, $v_1 + v_2 < T_1 + T_2 = T$ and no violation is possible. Hence, each server can perform local monitoring and no communication is necessary. We say a local violation occurs when a local threshold is exceeded, e.g., $v_1 > T_1$. Clearly, the coordinator only needs to collect values from both monitors to see if $v_1 + v_2 > T$ (referred to as a global poll) when a local violation happens.

The above local task based approach is employed in most existing state monitoring works [3], [5]. By dividing a global task into local ones, it also introduces new challenges for dynamic sampling. First, dynamic sampling schemes tune sampling intervals on individual monitors for a monitor-level accuracy requirement. For a task involving distributed monitors, how should we coordinate interval adjusting on

each monitor to meet a task-level accuracy requirement? Second, suppose such coordination is possible. What principle should we follow to minimize the total monitoring cost? For instance, the trace in Chart (d) causes more local violations than Chart (a). Should we enforce the same level of accuracy requirement on both monitors?

State Correlation. Datacenter management relies on a large set of distributed state monitoring tasks. The states of different tasks are often related. For example, suppose the trace in Chart (e) shows the request response time on a server and the trace in Chart (f) shows the traffic difference on the same server. If we observe growing traffic difference in (f), we are also very likely to observe increasing response time in (e) due to workloads introduced by possible DDoS attacks. Based on state correlation, we can selectively perform sampling on some tasks only when their correlated ones suggest high violation likelihood. For instance, since increasing response time is a necessary condition of a successful DDoS attack, we can trigger high frequency sampling for DDoS attack monitoring only when the response time is high to reduce monitoring cost. Designing such a state-correlation based approach also introduces challenges. How to detect state correlation automatically? How to efficiently generate a correlation based monitoring plan to maximize cost reduction and minimize accuracy loss? These are all important problems deserving careful study.

B. Volley Overview

Volley consists of dynamic sampling techniques at three different levels. **Monitor Level Sampling** dynamically adjusts the sampling interval based on its estimation of violation likelihood. The algorithm achieves controlled accuracy by choosing a sampling interval that makes the probability of mis-detecting violations lower than a user specified error allowance. Furthermore, we devise violation likelihood estimation methods with negligible overhead. **Task Level Coordination** is a lightweight distributed scheme that adjusts error allowance allocated to individual nodes in a way that both satisfies the global error allowance specified by the user, and minimizes the total monitoring cost. **Multi-Task Level State Correlation** based scheme leverages the state correlation between different tasks to avoid sampling operations that are least likely to detect violations across all tasks. It automatically detects state correlation between tasks and schedules sampling for different tasks at the datacenter level considering both cost factors and degree of state correlation. Due to space limitation, we discuss the first two techniques in this paper and leave details of the state-correlation based monitoring to our technical report [13].

III. Accuracy-Driven Dynamic Sampling

A dynamic sampling scheme has several requirements. First, it needs a method to estimate violation likelihood in a timely manner. Second, a connection between the sampling interval and the mis-detection rate should be established, so that the dynamic scheme can strive to maintain a certain level of error allowance specified by users. Third, the estimation method should be efficient because it is invoked frequently to quickly adapt to changes in monitoring data. We next address these requirements and present the details of our approach.

A. Violation Likelihood Estimation

The specification of a monitoring task includes a *default sampling interval* I_d, which is the smallest sampling interval necessary for the task. Since I_d is the smallest sampling interval necessary, the mis-detection rate of violations is negligible when I_d is used. In addition, we also use I_d as our guideline for evaluating accuracy. The specification of a monitoring task also includes an *error allowance* which is an acceptable probability of mis-detecting violations (compared with periodical sampling using I_d as the sampling interval). We use err to denote this error allowance. Note that $err \in [0, 1]$. For example, $err = 0.01$ means at most 1 percent of violations (that would be detected when using periodical sampling with the default sampling interval I_d) can be missed.

Recall that state monitoring reports state alerts whenever $v > T$ where v is the monitored value and T is a given threshold. Hence, violation likelihood is defined naturally as follows,

Definition 1: Violation likelihood at time t is defined by $P[v(t) > T]$ where $v(t)$ is the monitored metric value observed at time t.

Before deriving an estimation method for violation likelihood, we need to know 1) *when* to estimate and 2) for *which* time period the estimation should be made. For 1), because we want the dynamic sampling scheme to react to changes in monitored values as quickly as possible, estimation should be performed right after a new monitored value becomes available. For 2), note that uncertainties are introduced by unseen monitoring values between the current sampling and the next sampling (inclusive). Hence, estimation should be made for the likelihood of detecting violations within this period.

Violation likelihood for the next (future) sampled value is determined by two factors: the current sampled value and changes between two sampled values. When the current sampled value is low, a violation is less likely to occur before the next sampling time, and vice versa. Similarly, when the change between two continuously sampled values is large, a violation is more likely to occur before the next sampling time. As a result, estimation should also be based on the *current sampled value* and *changes between sampled values*. Let $v(t_1)$ denote the current sampled value, and $v(t_2)$ denote the next sampled value (under current sampling frequencies). Let δ be the difference between the two continuously sampled values when the default sampling

interval is used. We consider δ as a time-independent[1] random variable. Hence, the violation likelihood for a value $v(t_2)$ that is sampled i default sampling intervals after $v(t_1)$ is,

$$P[v(t_2) > T] = P[v(t_1) + i\delta > T]$$

To efficiently estimate this probability, we apply Chebyshev's Inequality [14] to obtain its upper bound. The one-sided Chebyshev's inequality has the form $P(X - \mu \geqslant k\sigma) \leqslant \frac{1}{1+k^2}$, where X is a random variable, μ and σ are the mean and the variance of X, and $k > 0$. The inequality provides an upper bound for the probability of a random variable "digressing" from its mean by a certain degree, regardless of the distribution of X. To apply Chebyshev's inequality, we have

$$P[v(t_1) + i\delta > T] = P[\delta > \frac{T - v(t_1)}{i}]$$

Let $k\sigma + \mu = \frac{T - v(t_1)}{i}$ where μ and σ are the mean and variance of δ; we obtain $k = \frac{T - v(t_1) - i\mu}{i\sigma}$. According to Chebyshev's inequality, we have

$$P[\delta > \frac{T - v(t_1)}{i}] \leqslant 1/(1 + (\frac{T - v(t_1) - i\mu}{i\sigma})^2) \quad (1)$$

When selecting a good sampling interval, we are interested in the corresponding probability of mis-detecting a violation during the gap between two continuous samples. Therefore, we define the mis-detection rate for a given sampling interval I as follows,

Definition 2: Mis-detection rate $\beta(I)$ for a sampling interval I is defined as $P\{v(t_1 + \Delta_t) > T, \Delta_t \in [1, I]\}$ where $I(\geqslant 1)$ is measured by the number of default sampling intervals.

Furthermore, according to the definition of $\beta(I)$, we have

$$\beta(I) = 1 - P\{\bigcap_{i \in [1,I]} (v(t_1) + i\delta \leqslant T)\}$$

Because $\delta \in \mathbb{R}$ is time-independent, we have

$$\beta(I) = 1 - \prod_{i \in [1,I]} (1 - P(v(t_1) + i\delta > T)) \quad (2)$$

According to Inequality 1, we have

$$\beta(I) \leqslant 1 - \prod_{i \in [1,I]} \frac{(\frac{T - v(t_1) - i\mu}{i\sigma})^2}{1 + (\frac{T - v(t_1) - i\mu}{i\sigma})^2} \quad (3)$$

Inequality 3 provides the method to estimate the probability of mis-detecting a violation for a given sampling interval I.

[1]We capture the time-dependent factor with online statistics update which is described in Section III-B.

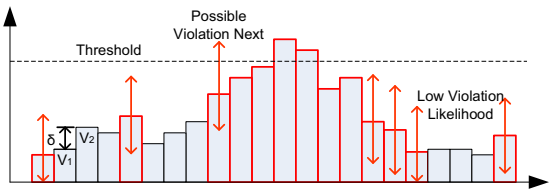


Figure 2. Violation Likelihood Based Adaptation

B. Violation Likelihood Based Adaptation

Figure 2 illustrates an example of violation likelihood based dynamic sampling. The dynamic sampling algorithm adjusts the sampling interval each time when it completes a sampling operation. Once a sampled value is available, it computes the upper bound of the mis-detection rate $\beta(I)$ according to inequality 3. We denote this upper bound with $\overline{\beta(I)}$. As long as $\beta(I) \leqslant \overline{\beta(I)} \leqslant err$ where err is the user-specified error allowance, the mis-detection rate is acceptable. To reduce sampling cost, the algorithm checks if $\overline{\beta(I)} \leqslant (1 - \gamma)err$ for p continuous times, where γ is a constant ratio referred as the slack ratio. If true, the algorithm increases the current sampling interval by 1 (1 default sampling interval), i.e. $I \leftarrow I + 1$. The slack ratio γ is used to avoid risky interval increasing. Without γ, the algorithm could increase the sampling interval even when $\overline{\beta(I)} = err$, which is almost certain to cause $\overline{\beta(I+1)} > err$. Through empirical observation, we find that setting $\gamma = 0.2, p = 20$ is a good practice. The sampling algorithm starts with the default sampling interval I_d, which is also the smallest possible interval. In addition, users can specify the maximum sampling interval denoted as I_m, and the dynamic sampling algorithm would never use a sampling interval $I > I_m$. If it detects $\overline{\beta(I)} > err$, it switches the sampling interval to the default one immediately. This is to minimize the chance of mis-detecting violations when the distribution of δ changes abruptly.

Because we use the upper bound of violation likelihood to adjust sampling intervals and the Chebyshev bound is quite loose, the dynamic sampling scheme is conservative on employing large sampling intervals unless the task has very stable δ distribution or the monitored values are consistently far away from the threshold. As sampling cost reduces sub-linearly with increasing intervals ($1 \rightarrow \frac{1}{2} \rightarrow \frac{1}{3} \cdots$), being conservative on using large intervals does not noticeably hurt the cost reduction performance, but reduces the chance of mis-detecting important changes between sampling.

Since computing inequality 3 relies on the mean and the variance of δ, the algorithm also maintains these two statistics based on observed sampled values. To update these statistics efficiently, we employ an online updating scheme[15]. Specifically, let n be the number of samples used for computing the statistics of δ, μ_{n-1} denote the current mean of δ and μ_n denote the updated mean of δ. When the sampling operation returns a new sampled value

$v(t)$, we first obtain $\delta = v(t) - v(t-1)$. We then update the mean by $\mu_n = \mu_{n-1} + \frac{\delta - \mu_{n-1}}{n}$. Similarly, let σ_{n-1} be the current variance of δ and σ_n be the updated variance of δ; we update the variance by $\sigma_n^2 = \frac{(n-1)\sigma_{n-1}^2 + (\delta - \mu_n)(\delta - \mu_{n-1})}{n}$. Both updating equations are derived from the definition of mean and variance respectively. The use of online statistics updating allows us to efficiently update μ and σ without repeatedly scanning previous sampled values. Note that sampling is often performed with sampling intervals larger than the default one. In this case, we estimate $\hat{\delta}$ with $\hat{\delta} = (v(t) - v(t-I))/I$, where I is the current sampling interval and $v(t)$ is the sampled value at time t, and we use $\hat{\delta}$ to update the statistics. Furthermore, to ensure the statistics represent the most recent δ distribution, the algorithm periodically restarts the statistics updating by setting $n = 0$ when $n > 1000$.

The main cost of the dynamic sampling algorithm consists of two parts, the sampling cost and the computation cost of violation likelihood estimation. Sampling operations are usually much more expensive than violation likelihood estimation, as distributed sampling often involves scheduling of sampling operations, executing local sampling instructions, polling sampling data from the monitored system to the monitor, sampling data persistence and etc. Our experiment results show that the algorithm saves substantial monitoring overhead for a wide rage of monitoring tasks.

IV. Distributed Sampling Coordination

A distributed state monitoring task performs sampling operations on multiple monitors to monitor the global state. For dynamic sampling, it is important to maintain the user-specified task-level accuracy while adjusting sampling intervals on distributed monitors.

A. Task-Level Monitoring Accuracy

Recall that a distributed state monitoring task involves multiple monitors and a coordinator. Each monitor performs local sampling and checks if a local condition is violated. If true, it reports the local violation to the coordinator which then collects all monitored values from all monitors to check if the global condition is violated.

The local violation reporting scheme between monitors and the coordinator determines the relation between local monitoring accuracy and global monitoring accuracy. When a monitor mis-detects a local violation, the coordinator may miss a global violation if the mis-detected local violation is indeed part of a global violation. Let β_i denote the mis-detection rate of monitor i and β_c denote the mis-detection rate of the coordinator. Clearly, $\beta_c \leqslant \sum_{i \in N} \beta_i$ where N is the set of all monitors. Therefore, as long as we limit the sum of monitor mis-detection rates to stay below the specified error allowance err, we can achieve $\beta_c \leqslant \sum_{i \in N} \beta_i \leqslant err$.

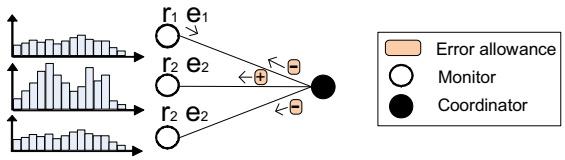


Figure 3. Distributed Sampling Coordination

B. Optimizing Monitoring Cost

For a given error allowance err, there are different ways to distribute err among monitors, each of which may lead to different monitoring cost. For example, suppose trace (e) and (f) in Figure 1 show values monitored by monitor 1 and 2. As (f) is more likely to cause violations than (e), when evenly dividing err among monitor 1 and 2, one possible result is that $I_1 = 4$ (interval on monitor 1) and $I_2 = 1$ (interval on monitor 2). The total cost reduction is $(1 - 1/4) + (1 - 1) = 3/4$. When assigning more err to monitor 2 to absorb frequent violations, we may get $I_1 = 3, I_2 = 2$ and the corresponding cost reduction is $2/3 + 1/2 = 7/6 > 3/4$. Therefore, finding the optimal way to assign err is critical to reduce monitoring cost.

Nevertheless, finding the optimal assignment is difficult. Brute force search is impractical ($O(n^m)$ where m is the number of monitors and n is the number of minimum assignable units in err). Furthermore, the optimal assignment may also change over time when characteristics of monitored values on monitors vary. Therefore, we develop an iterative scheme that gradually tunes the assignment across monitors by moving error allowance from monitors with low cost reduction yield (per assigned err) to those with high cost reduction yield.

Figure 3 illustrates the process of distributed sampling coordination. The coordinator first divides err evenly across all monitors of a task. Each monitor then adjusts its local sampling interval according to the adaptation scheme we introduced in Section III to minimize local sampling cost. Each monitor i locally maintains two statistics: 1) r_i, potential cost reduction if its interval increased by 1 which is calculated as $r_i = 1 - \frac{1}{I_i + 1}$; 2) e_i, error allowance needed to increase its interval by 1 which is calculated as $e_i = \frac{\beta(I_i)}{1 - \gamma}$ (derived from the adaptation rule in Section III-B).

Periodically, the coordinator collects both r_i and e_i from each monitor i, and computes the cost reduction yield $y_i = \frac{r_i}{e_i}$. We refer to this period as an *updating period*, and both r_i and e_i are the average of values observed on monitors within an updating period. y_i essentially measures the cost reduction yield per unit of error allowance. After it obtains y_i from all monitors, the coordinator performs the following assignment $err_i' = err \frac{y_i}{\sum_i y_i}$, where err_i' is the assignment for monitor i in the next iteration. Intuitively, this updating scheme assigns more error allowance to monitors with higher cost reduction yield. In the previous example, $I_1 = 4$ and $I_2 = 1$ after the initial even distribution of error

allowance. As long as e_2 is not very large, the coordinator would continuously assign more error allowance to monitor 2 until $I_2 = 2$, which increases the overall cost reduction yield.

Typically, the mis-detecting rate we derived in Section III increases super-linearly with growing sampling interval. As a result, y_i decreases with increasing e_i, and the assignment eventually converges to a stable assignment when the monitored data distribution across nodes does not significantly change. Furthermore, the tuning scheme also applies throttling to avoid unnecessary updating. It avoids reallocating err to a monitor i if $err_i < \underline{err}$ where constant $\underline{err}$ is the minimum assignment. Furthermore, it does not perform reallocation if $\max\{y_i/y_j, \forall i, j\} < 0.1$. We set the updating period to be every thousand I_d and $\underline{err}$ to be $\frac{err}{100}$.

V. Evaluation

We deploy a prototype of Volley in a datacenter testbed consisting of 800 virtual machines (VMs) and evaluate Volley with real world network, system and application level monitoring scenarios. We highlight some of the key results below:

- The violation likelihood based adaptation technique saves up to 90% sampling cost. The accuracy loss is smaller or close to the user specified error allowances.
- The distributed sampling coordination technique optimizes the error allowance allocation across monitors and outperforms alternative schemes.

A. Experiment Setup

We setup a virtualized datacenter testbed containing 800 VMs in Emulab [16] to evaluate our approach. Figure 4 illustrates the high-level setup of the environment. It consists of 20 physical servers, each equipped with a 2.4 GHz 64bit Quad Core Xeon E5530 processor, 12 GB RAM and runs XEN-3.0.3 hypervisor. Each server has a single privileged VM/domain called Domain 0 (`Dom0`) which is responsible for managing the other unprivileged VMs/user domains [17]. In addition, each server runs 40 VMs (besides `Dom0`) configured with 1 virtual CPU and 256MB memory. All VMs run 64bit CentOS 5.5. We implemented a virtual network to allow packet forwarding among all 800 VMs with XEN route scripts and `iptables`.

We implemented a prototype of Volley which consists of three main components: agents, monitors and coordinators. An agent runs within a VM and provides monitoring data. Agents play an important role in emulating real world monitoring environments. Depending on the type of monitoring, they either generate network traffic according to pre-collected network traces or provide pre-collected monitoring data to monitors when requested (described below). For each VM, a monitor is created in `Dom0`. Monitors collect monitoring data, process the data to check whether local violations exist and report local violations to coordinators. In addition,

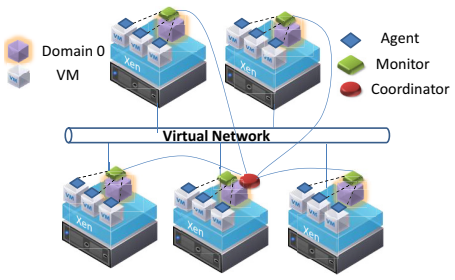
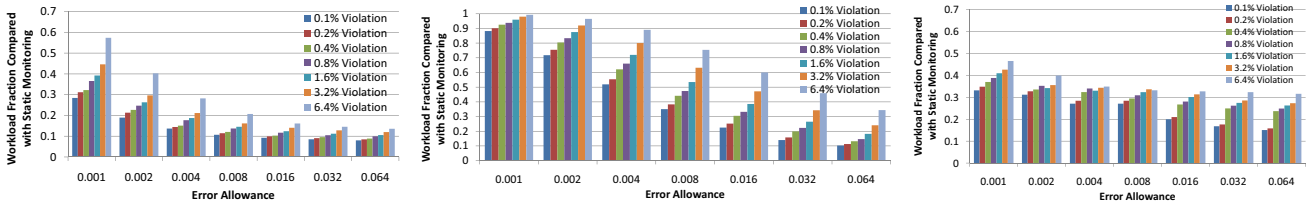


Figure 4. Experiment Setup

they also perform local violation likelihood based sampling adaptation described earlier. A coordinator is created for every 5 physical servers. They process local violation reports and trigger global polls if necessary. We also implemented distributed sampling coordination on coordinators. We next present details on monitoring tasks.

Network level monitoring tasks used in our experiments try to detect distributed denial of service (DDoS) attacks (Section II-A) in a virtualized datacenter. We perform traffic monitoring on each server (`Dom0`), rather than network switches and routers, because only `Dom0` can observe communications between VMs running on the same server. For a task involving a set V of VMs , their corresponding monitors perform sampling by collecting and processing traffic associated with the VM $v \in V$ (within a 15-second interval) to compute the traffic difference $\rho_v = P_i(v) - P_o(v)$ where $P_i(v)$ and $P_o(v)$ are the incoming number of packets with SYN flags set and the outgoing number of packets with both SYN and ACK flags set respectively. The sampling is implemented with `tcpdump` and bash scripts and the default sampling interval is 15 seconds (capture continuously and report every 15 seconds). When ρ_v exceeds the local threshold, the monitor reports a local violation to its coordinator which then communicates with monitors associated with other VMs $V - v$ to check if $\sum_{v \in V} \rho_v > T$.

We port real world traffic observed on Internet2 network [18], a large-scale data-intensive production network, into our testbed. The traces are in netflow v5 format and contain approximately 42,278,745 packet flows collected from the Internet2 backbone network. A flow in the trace records the source and destination IP addresses as well as the traffic information (total bytes, number of packets, protocol, etc.) for a flow of observed packets. We uniformly map addresses observed in netflow logs into VMs in our testbed. If a packet sent from address A to B is recorded in the logs, VM X (where A is mapped to) also sends a packet to VM Y (where B is mapped to). We set SYN and ACK flags with a fixed probability $p = 0.1$ to each packet a VM sends. Note that ρ is not affected by the value of p as p has the same effect to both P_i and P_o. In addition, let F denote the number of packets in a recorded flow. We also scale down the traffic volume to a realistic level by generating only F/n packets for this flow where n is the average number of



(a) Network Level Monitoring (b) System Level Monitoring (c) Application Level Monitoring

Figure 5. Monitoring Overhead Saving under Different Error Allowance and State Alert Rates

addresses mapped to a VM.

System level monitoring tasks track the state of OS level performance metrics on VMs running in our testbed. A system level task triggers a state alert when the average value of a certain metric exceeds a given threshold, e.g. an alert is generated when the average memory utilization on VM-1 to VM-10 exceeds 80%. To create VMs in a production environment, we port a performance dataset[19] collected from hundreds of computing nodes to our VMs. This dataset contains performance values on 66 system metrics including available CPU, free memory, virtual memory statistics(vmstat), disk usage, network usage, etc. To perform sampling, a monitor queries its assigned VM for a certain performance metric value and the agent running inside the VM responds with the value recorded in the dataset. The default sampling interval is 5 seconds.

Application level monitoring tasks watch the throughput state of web applications deployed in datacenters. For example, Amazon EC2 can dynamically add new server instances to a web application when the monitored throughput exceeds a certain level[10]. We port traces of HTTP requests ($>$ 1 billion) collected from a website hosted by a set of 30 distributed web servers[20]. Similar to system level monitoring, agents running on VMs respond with web server access logs (since last sampling point) in the dataset when queried by monitors so that they mimic VMs running a web application. The default sampling interval is 1 second.

Thresholds. Monitoring datasets used in our experiment are not labeled for identifying state violations. Hence, for a state monitoring task on metric m, we assign its monitoring threshold by taking $(100-k)$-th percentile of m's values. For example, when $k = 1$, a network-level task reports DDoS alerts if $\rho > Q(\rho, 99)$ where $Q(\rho, 99)$ is the 99th percentile of ρ observed through the lifetime of the task. Similarly, when $k = 10$, a system-level task report state alerts if memory utilization $\mu > Q(\mu, 90)$. We believe this is a reasonable way to create state monitoring tasks as many state monitoring tasks try to detect a small percentage of violation events. We also vary the value of selectivity parameter k to evaluate the impact of selectivity in tasks.

B. Results

Monitoring Efficiency. Figure 5(a) illustrates the results for our network monitoring experiments where each task checks whether the traffic difference ρ on a single VM exceeds a threshold set by the aforementioned selectivity k (we illustrate results on distributed monitoring tasks (multiple VMs) later in Figure 8). We are interested in the ratio of sampling operations (y-axis) performed by Volley over those performed by periodical sampling (with interval I_d). We vary both the error allowance (x-axis) and the alert selectivity k in monitoring tasks (series) to test their impact on monitoring efficiency. Recall that the error allowance specifies the maximum percentage of state alerts allowed to be missed. An error allowance of 1% means that the user can tolerate at most 1% of state alerts not being detected. We see that dynamic sampling reduces monitoring overhead by 40%-90%. Clearly, the larger the error allowance, the more sampling operations Volley can save by reducing monitoring frequency. The alert state selectivity k also plays an important role, e.g. varying k from 6.4% to 0.1% can lead to 40% cost reduction. Recall that $k = 6.4$ means that 6.4% of monitored values would trigger state alerts. This is because lower k leads to fewer state alerts and higher thresholds in monitoring tasks, which allows Volley to use longer monitoring intervals when previous observed values are far away from the threshold (low violation likelihood). Since real world tasks often have small k, e.g. a task with a 15-second monitoring interval, generating one alert event per hour leads to a $k = 1/240 \approx 0.0042$. We expect Volley to save considerable overhead for many monitoring tasks.

Figure 5(b) shows the results for system level monitoring, where each monitoring task checks if the value of a single metric on a certain VM violates a threshold chosen by the aforementioned selectivity k. The results suggest that Volley also effectively reduces the monitoring overhead, with relatively smaller cost saving ratios compared with the network monitoring case. This is because changes in traffic are often less than changes in system metric values (e.g. CPU utilization). This is especially true for network traffic observed at night.

We show the results of application level monitoring in Figure 5(c) where each task checks whether the access rate of a certain object, e.g. a video or a web page, on a certain VM exceeds the k-determined threshold by analyzing the recent access logs on the VM. We observe similar cost savings in this figure. The high cost reduction achieved in the application level monitoring is due to the bursty

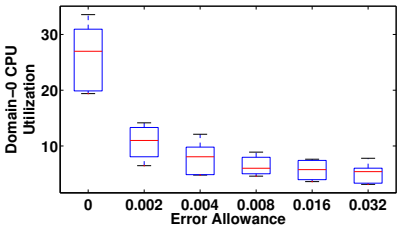


Figure 6. CPU Utilization

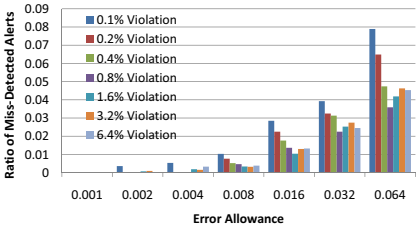


Figure 7. Actual Mis-Detection Rates

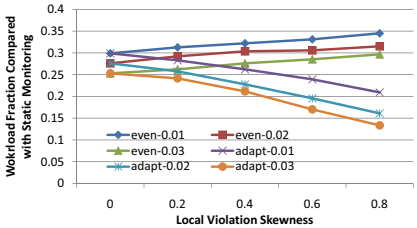


Figure 8. Distributed Coordination

nature of accesses. It allows our adaptation to use large monitoring intervals during off-peak times. We believe that our techniques can provide substantial benefits when this type of change pattern occurs in many other applications (e.g. e-business websites) where diurnal effects and bursty request arrival are common.

Figure 6 uses box plots to illustrate the distribution of `Dom0` CPU resource consumption (percentage) caused by network-level monitoring tasks with increasing error allowance. The upper and lower bound of boxes mark the 0.75 and 0.25 quantile. The line inside the box indicates the median. The whiskers are lines extending from each end of the box to show the extent of the rest of the utilization data. CPU resources are primarily consumed by packet collection and deep packet inspection, and the variation in utilization is caused by network traffic changes. When the error allowance is 0, our violation-likelihood based sampling is essentially periodical sampling and introduces fairly high CPU utilization (20-34%) which is prohibitively high for `Dom0`. This is because `Dom0` needs to access hardware on behalf of all user VMs, and IO intensive user VMs may consume lots of `Dom0` CPU cycles. When `Dom0` is saturated with monitoring and IO overhead, all VMs running on the same server experience seriously degraded IO performance[17]. With increasing error allowance, our approach quickly reduces the CPU utilization by at least a half (up to 80%) and substantially improves the efficiency and scalability of monitoring.

Although system/application level monitoring tasks incur less overhead compared with the network monitoring case, Volley can still save significant monitoring cost when such tasks are performed by monitoring services that charge users based on sampling frequency[10]. Furthermore, the aggregated cost of these tasks is still considerable for data-center monitoring. Reducing sampling cost not only relieves resource contention between application and infrastructure management, but also improves the datacenter management scalability[7].

Monitoring Accuracy. Figure 7 shows the actual mis-detection rate (y-axis) of alerts for the system-level monitoring experiments. We see that the actual mis-detection rate is lower than the specified error allowance in most cases. Among different state monitoring tasks, those with high alert selectivity often have relatively large mis-detection rates. There are two reasons. First, high selectivity leads to few alerts which reduces the denominator in the mis-detection rate. Second, high selectivity also makes Volley prefer low frequency which increases the chance of missing alerts. We do not show the results on network and application level monitoring as the results are similar.

Distributed Sampling Coordination. Figure 8 illustrates the performance of different error allowance distribution schemes in network monitoring tasks. To vary the cost reduction yield on monitors, we change the local violation rates by varying the local thresholds. Initially, we assign a local threshold to each monitor so that all monitors have the same local violation rate. We then gradually change the local violation rate distribution to a Zipf distribution[21] which is commonly used to approximate skewed distributions in many situations. The x-axis shows the skewness of the distribution, and the distribution is uniform when skewness is 0. The y-axis shows the the ratio between the total number of sampling performed by one scheme and that performed by a periodical sampling scheme with the default sampling frequency. We compare the performance of our iterative tuning scheme (adapt) described in Section IV with an alternative scheme (even) which always divides the global error allowance evenly among monitors. The cost reduction of the even scheme gradually degrades with increasing skewness of the local violation rate distribution. This is because when the cost reduction yields on monitors are not the same, the even scheme cannot maximize the cost reduction yield over all monitors. The adaptive scheme reduces cost significantly more as it continuously allocates error allowance to monitors with high yields. Since a few monitors account for most local violations under a skewed Zipf distribution, the adaptive scheme can move error allowance from these monitors to those with higher cost reduction yield.

VI. Related Work

Most existing works in distributed state monitoring [3], [5], [22] study the problem of employing distributed constraints to minimize communication cost of distributed state monitoring. While these works study the communication-efficient detection of state violations in a distributed manner based on the assumption that monitoring data is always available (with no cost), we study a lower level problem on collecting monitoring data. We investigate the fundamental

relation between sampling intervals and accuracy, and propose violation likelihood based dynamic sampling schemes to enable efficient monitoring with controlled accuracy.

A number of existing works in sensor networks use correlation to minimize energy consumption on sensor nodes[23], [24]. Our work differs from these works in several aspects. First, these works[23] often leverage the broadcast feature of sensor networks, while our system architecture is very different from sensor networks and does not have broadcast features. Second, we aim at reducing sampling cost while these works focus on reducing communication cost to preserve energy. Third, while sensor networks usually run a single or a few tasks, we have to consider multi-task correlation in large-scale distributed environments. Finally, some works (e.g. [24]) make assumptions on value distributions, while our approach makes no such assumptions. Some scenarios such as network monitoring employ random sampling to collect a partial snapshot for state monitoring (e.g., a random subset of packets [1]). Volley is complementary to random sampling as it can be used together with random sampling to offer additional cost savings by scheduling sampling operations.

VII. Conclusions and Future Work

We have presented Volley, a violation likelihood based efficient state monitoring approach that operates at both single-node and multi-node levels to explore task-level error allowance and inter-task state correlation to minimize monitoring cost through accuracy-driven dynamic sampling. Volley works best when inter-sampling monitoring value changes have relatively stable distributions or when anomalies tend to cause continuous global violations. While Volley safeguards monitoring error in a best-effort manner, our experiment results show that it introduces few mis-detections in several real world monitoring tasks due to its conservative adaptation (Chebyshev's inequality and additive-increase/multiplicative-decrease-like adaptation scheme). As part of our ongoing work, we are studying techniques to support advanced state monitoring forms (e.g. tasks with aggregation time window).

Acknowledgements

The first author was partially supported by IBM 2011-2012 PhD fellowship and a grant from NSF NetSE program when this work was performed initially. The fourth author is partially supported by grants from NSF Network Science and Engineering (NetSE) and NSF Trustworthy Cyberspace (SaTC), an IBM faculty award and a grant from Intel ISTC on Cloud Computing.

References

[1] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *SIGCOMM*, 2002.

[2] B. Raghavan, K. V. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud control with distributed rate limiting," in *SIGCOMM*, 2007, pp. 337–348.

[3] S. R. Kashyap, J. Ramamirtham, R. Rastogi, and P. Shukla, "Efficient constraint monitoring using adaptive thresholds," in *ICDE*, 2008.

[4] S. Meng, S. R. Kashyap, C. Venkatramani, and L. Liu, "Resource-aware application state monitoring," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 12, pp. 2315–2329, 2012.

[5] S. Meng, L. Liu, and T. Wang, "State monitoring in cloud datacenters," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 9, pp. 1328–1344, 2011.

[6] J. S. Chase, D. C. Anderson, P. N. Thakar, A. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centres," in *SOSP*, 2001.

[7] S. Meng, L. Liu, and V. Soundararajan, "Tide: Achieving self-scaling in virtualized datacenter management middleware," in *Middleware*, 2010.

[8] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *CCS*, 2003.

[9] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks*, vol. 44, no. 5, pp. 643–666, 2004.

[10] EC2 CloudWatch, http://aws.amazon.com/cloudwatch/, 2012.

[11] "The ntp project," http://www.ntp.org/ntpfaq/NTP-s-algo.htm.

[12] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood, "Deep packet inspection using parallel bloom filters," *IEEE Micro*, vol. 24, no. 1, pp. 52–61, 2004.

[13] S. Meng, A. K. Iyengar, I. M. Rouvellou, and L. Liu, "Violation likelihood based state monitoring."

[14] G. Grimmett and D. Stirzaker, *Probability and Random Processes 3rd ed.* Oxford, 2001.

[15] D. E. Knuth, *The Art of Computer Programming, Vol. 2, 3rd Ed.* Addison-Wesley, 1998.

[16] 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.

[17] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Middleware*, 2006.

[18] The Internet2 Data, http://www.internet2.edu/observatory/archive, 2012.

[19] Y. Zhao, Y. Tan, Z. Gong, X. Gu, and M. Wamboldt, "Self-correlating predictive information tracking for large-scale production systems," in *ICAC*, 2009, pp. 33–42.

[20] WorldCup Trace, http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[21] G. K. Zipf, *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.

[22] S. Meng, A. Iyengar, I. Rouvellou, L. Liu, K. Lee, B. Palanisamy, and Y. Tang, "Reliable state monitoring in cloud datacenters," in *IEEE CLOUD*, 2012, pp. 951–958.

[23] A. Silberstein, R. Braynard, and J. Y. 0001, "Constraint chaining: on energy-efficient continuous monitoring in sensor networks," in *SIGMOD Conference*, 2006, pp. 157–168.

[24] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong, "Approximate data collection in sensor networks using probabilistic models," in *ICDE*, 2006.