Resilient Self-Compressive Monitoring for Large-Scale Hosting Infrastructures

Yongmin Tan, *Student Member, IEEE*, Vinay Venkatesh, and Xiaohui Gu, *Senior Member, IEEE*

Abstract—Large-scale hosting infrastructures have become the fundamental platforms for many real world systems such as cloud computing infrastructures, enterprise data centers, and massive data processing systems. However, it is a challenging task to achieve both scalability and high precision while monitoring a large number of intra-node and inter-node attributes (e.g., CPU usage, free memory, free disk, inter-node network delay). In this paper, we present the design and implementation of a *R*esilient self-*C*ompressive *M*onitoring (RCM) system for large-scale hosting infrastructures. RCM achieves scalable distributed monitoring by performing *online data compression* to reduce remote data collection cost. RCM provides failure resilience to achieve robust monitoring for dynamic distributed systems where host and network failures are common. We have conducted extensive experiments using a set of real monitoring data from NCSU's virtual computing lab (VCL), PlanetLab, a Google cluster, and real Internet traffic matrices. The experimental results show that RCM can achieve up to 200% higher compression ratio and several orders of magnitude less overhead than the existing approaches.

Index Terms—Online data compression, distributed system monitoring

1 Introduction

Large-scale distributed hosting infrastructures have become the fundamental platforms for many real world production systems such as enterprise data centers, cloud systems [1], and massive data processing systems [2]–[4]. A production hosting infrastructure typically consists of i) a large number of distributed worker nodes that execute different application tasks; and ii) a set of management nodes that provide various configuration and optimization services. As the complexity and scale of those distributed systems continue to grow, it has become imperative to provide automatic system management support [5]. Among those system management modules, system monitoring serves as one of the fundamental building blocks. A distributed monitoring system typically deploys monitoring agents on distributed worker nodes and configures those agents to continuously collect various metrics and periodically report sampled metric values to management nodes, which is illustrated by Fig. 1.

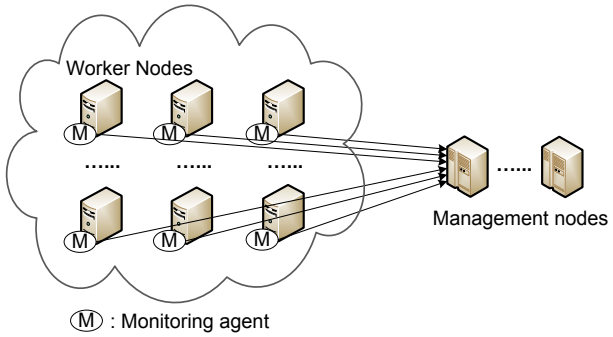


Fig. 1. Distributed monitoring system.

To achieve efficiency and accuracy, the system management node often desires to obtain *complete* and *fine-grained* information about all hosts and network connections within the hosting infrastructure. For example, our previous work [6], [7] has shown that fine-grained monitoring data can help significantly improve the resource utilization of the hosing infrastructure. Fig. 2 shows the service level objective (SLO) violation rate of a dynamic resource scaling system [7] under different monitoring granularities (i.e., 1 second v.s. 1 minute monitoring interval) for the RUBiS online auction benchmark application [8]. We can see that fine-grained monitoring can reduce the SLO violation rate from 17.4% to 4.3%. If we allow a tight approximation error (0.05) in the distributed monitoring system, we observe that the SLO rate is almost unaffected. This implies that fine-grained monitoring with a tight error bound is effective for system management. Similarly, online performance anomaly detection and prediction systems [9]–[11] also depend on fine-grained monitoring data to achieve high accuracy.

However, it is a challenging task to deploy fine-grained monitoring for large-scale hosting infrastructures due to the scalability concern. A production hosting infrastructure often comprises thousands of physical hosts and many more virtual machines (VMs), each of which can be associated with hundreds

- *Yongmin Tan and Xiaohui Gu are with Department of Computer Science, North Carolina State University, Raleigh, NC, 27695. Email: ytan2@ncsu.edu, gu@csc.ncsu.edu*
- *Vinay Venkatesh is with IBM, Research Triangle Park, NC 27709. Email: venkatvi@us.ibm.com*

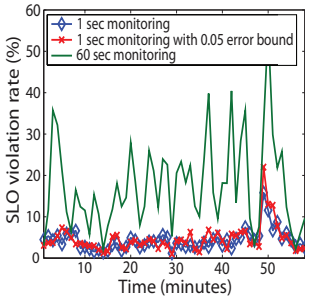


Fig. 2. A case study for fine-grained monitoring.

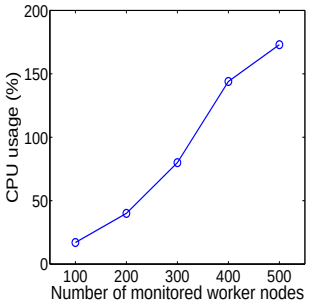


Fig. 3. Scalability bottleneck at the management node.

of dynamic metrics [12], [13]. For example, the IBM Tivoli monitoring system [13] can collect over 600 metrics on a single host. Hence, most production hosting infrastructures [12], [14] typically use a long update interval (e.g., every five minutes) to avoid excessive monitoring overhead. Fig. 3 shows the CPU usage of a dual-core 2.4 GHz server that monitors different numbers of worker nodes. Here, we assume that each worker node reports 90 intra-node attributes and 10 inter-node attributes at one second sampling interval to the management server. The size of each attribute is 8 bytes. We use the web workload generator httperf [15] to emulate different monitoring workloads. Fig. 3 shows that the management node is almost overloaded when the number of worker nodes becomes large (e.g., 500). Thus, without reducing the monitoring traffic to the management node, it is impractical to apply fine-grained monitoring to large-scale hosting infrastructures.

Previous work has identified the scalability challenge of fine-grained monitoring and proposed various solutions: 1) employing decentralized architectures such as hierarchical aggregation [16] or peer-to-peer structure [17], [18] to distribute monitoring workload or 2) trading off information coverage [19] or precision [20] for lower monitoring cost. To address the problem, one promising approach is to perform online data compression to reduce the monitoring traffic from the distributed worker nodes to the management node. Recent work [11], [21] has proposed to exploit temporal and/or spatial correlations for data compression. However, only exploring temporal correlation has limited compression power while discovering spatial correlation is often costly due to the expensive clustering operation [21].

In this paper, we present the design and implementation of a Resilient, self-Compressive Monitoring system (RCM) for large-scale hosting infrastructures. Our goal is to alleviate the bottleneck on the management node by reducing the monitoring traffic from distributed worker nodes to the management node. To achieve this goal, RCM takes a novel image-based approach. We model snapshots of dynamic monitoring attributes as a sequence of system images. Each image is partitioned into a set of blocks. RCM then performs online reference block search to find the optimal reference block for each system image block. During runtime, RCM suppresses the remote updates on those attributes whose values can be inferred by their reference block values within a pre-defined error bound. Furthermore, host failures are common in large-scale distributed systems [22]. For example, in our past study about PlanetLab host failures [23], we observed both transient and persistent host failures that lasted from five minutes to two days. Those host failures can be caused by either network failures or host crashes. Thus, the online compression scheme also needs to be failure-resilient in order to handle those host failures. RCM achieves failure-oblivious compression by maintaining a few backup reference blocks and dynamically switching to one backup reference block if the original reference block fails.

Specifically, this paper makes the following contributions:

- We present a fast and efficient reference block search algorithm that uses a dual-diamond search pattern to find the optimal or near-optimal reference block with low overhead.
- We introduce light-weight failure management mechanisms to achieve resilient online distributed monitoring data compression.
- We conduct an analytical study to quantify the advantage of our image-based compression approach over previous correlation-based compression schemes.
- We have implemented a prototype of RCM and conducted extensive experiments using real system monitoring data collected on PlanetLab [24], NCSU's virtual computing lab (VCL) [14], a Google cluster [25], and real Internet traffic matrices [26].

Our experimental results show that RCM can achieve up to 95% compression ratio under a range of tight error bounds (e.g., 0.01-0.1). RCM can improve the compression ratio by up to 200% compared to the simple temporal correlation based approach but still maintain a low overhead. Under host failures, RCM can achieve similar compression performance to the non-failure case with little extra cost.

The rest of the paper is organized as follows. Section 2 gives a preliminary introduction about the distributed monitoring system model and the problem formulation. Section 3 describes the design details of RCM. Section 4 presents the experimental evaluation. Section 5 compares our work with related work. Finally, the paper concludes in Section 6.

2 Preliminary

In this section, we first introduce the distributed monitoring system model. We then describe the problem

TABLE 1
Notations.

Notation	Description
N	number of worker nodes
v_i	the i'th worker node
A	set of all intra-node attributes
D	set of all inter-node attributes
$a^t_{i,k}$	intra-node attribute a_k on node i at time t
$d^t_{i,j,k}$	inter-node attribute d_k between node i and j at time t
b	block size
e_i	error bound for attribute i
T	monitoring interval
CR	compression ratio
U_{orig}	the number of uncompressed attribute updates in bytes per second
U_s	the number of suppressed attribute updates in bytes per second
N_b	number of blocks in one system image
I	number of reference images
R	number of training rounds
L	number of images in one compression phase
q	number of backup reference blocks
S	size of each monitoring data update in bytes

$$\begin{bmatrix} 0 & \cdots & d^t_{1,j,k} & \cdots & d^t_{1,N,k} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ d^t_{i,1,k} & \cdots & 0 & \cdots & d^t_{i,N,k} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ d^t_{N,1,k} & \cdots & d^t_{N,j,k} & \cdots & 0 \end{bmatrix}$$

($d^t_{i,j,k}$: inter-node attribute d_k between node i and j at time t)

Fig. 4. System image of an inter-node attribute for a distributed system of N nodes.

$$\begin{bmatrix} a^t_{1,k} & \cdots & a^t_{M,k} \\ a^t_{M+1,k} & \cdots & a^t_{2M,k} \\ \vdots & \vdots & \vdots \\ a^t_{N-M,k} & \cdots & a^t_{N,k} \end{bmatrix}$$

($a^t_{i,k}$ = intra-node attribute a_k of the node i at time t)

Fig. 5. System image of an intra-node attribute for a distributed system of N nodes.

formulation. Table 1 summarizes all the notations used in this paper.

2.1 Monitoring System Model

We consider a large-scale distributed hosting infrastructure that consists of N worker nodes, denoted by $\{v_1, \ldots v_N\}$. RCM installs monitoring agents on all worker nodes and configures those monitoring agents to report their local attribute values to the management node using a certain sampling rate (e.g., every 10 seconds). We classify distributed system attributes into two categories: 1) *intra-node* attributes that contain information relating to each node (e.g. CPU load, memory usage, disk I/O statistics), and 2) *inter-node* attributes that denote measurements between different nodes (e.g. network delay and bandwidth).

On each worker node, the monitoring agent periodically samples each intra-node attribute to form a time series $\{a^1_{i,k}, \ldots, a^t_{i,k}, \ldots, a^{t+m}_{i,k}\}$, where $a^t_{i,k}$ denotes the sampled value for the intra-node attribute a_k collected on node v_i at time t. Similarly, the monitoring agent periodically samples each inter-node attribute to form a time series $\{d^1_{i,j,k}, \ldots, d^t_{i,j,k}, \ldots, d^{t+m}_{i,j,k}\}$ where $d^t_{i,j,k}$ denotes the inter-node attribute d_k between node i and j at time t.

2.2 Problem Formulation

The goal of the RCM system is to achieve fine-grained monitoring with low cost. The basic idea is to suppress the update of the attribute value from a worker node to the management node at time t if the management node can infer the worker node's attribute value using a known reference value. Let a_i denote one real attribute value and a'_i denote its reference value. RCM allows the user to specify an error bound e_i ($e_i \geq 0$). If $|a_i - a'_i|/a_i \leq e_i$, RCM can suppress the update of this attribute value from the worker node and restore the value on the management node using its reference value within the error bound e_i. In general, more attribute value updates can be suppressed under a larger error bound e_i. The error bound is typically application-dependent, which can be configured by the application or the system management modules. If different applications and system management modules have different error bound requirements, RCM will use the lowest error bound to meet the precision requirements of all applications and system management modules.

To quantify the effectiveness of the online compression algorithm, we define the compression ratio (CR) as follows:

$$CR = \frac{U_s}{U_{orig}} \qquad (0 \leq CR \leq 1) \tag{1}$$

where U_s is the number of attribute values whose updates are suppressed by RCM in bytes per second and U_{orig} is the number of original attribute updates without using any compression in bytes per second. The larger the compression ratio, the more monitoring cost reduction can be achieved. To maximize the compression ratio, RCM needs to find the optimal reference value for each attribute. However, the challenge here is how to discover the optimal reference values for all the monitored attributes using a light-weight online compression algorithm.

We propose a novel image-based approach to scalable online monitoring data compression. We model

snapshots of dynamic monitoring attributes collected on distributed worker nodes in the hosting infrastructure as a sequence of *system images*.

For monitoring an inter-node attribute d_k (e.g., network delay) of a distributed system consisting of N nodes, the system image at time t is an $N \times N$ square matrix: the element at i'th row and j'th column denotes the attribute value between node i and node j, illustrated by Fig. 4. Note that we only need to build half image to reduce the monitoring traffic if $d_{i,j,k}$ and $d_{j,i,k}$ have very similar values. We choose to build the complete image to make our model generic, which can support those asymmetric inter-node attributes as well.

For monitoring an intra-node attribute a_k (e.g., CPU load) of a distributed system consisting of N nodes, the system image at time t is an $M \times K$ matrix ($M \cdot K = N$) where each element denotes the attribute value of one particular node, illustrated by Fig. 5. We conducted experiments to study how to pick M and K. Our results show that it is optimal to make M and K similar, as we will show in Section 4.

We further partition the system image into a set of small blocks, each of which contains $b \times b$ attribute values (e.g., $b = 2$). We perform reference value search at the block level for efficiency. We strive to find the optimal reference block for each block using the fast reference block search algorithm that will be described in the next section.

The compression scheme provided by RCM differs from the traditional compression techniques in the following major aspects. First, RCM needs to handle decentralized monitoring data that are located on different geographically dispersed hosts. This requires all the monitoring agents and the management node to perform compression together in a coordinated way. Second, RCM deals with dynamic live monitoring data, which requires an adaptive online training algorithm to maintain good performance. Third, distributed hosts may experience transient or persistent failures, which requires the compression algorithm to be failure resilient.

3 System Design

In this section, we first provide an overview of RCM. We then describe the optimal reference block search algorithm. Next, we present the failure-resilient online compression scheme. Finally, we present the analytical study to compare RCM with other alternative schemes.

3.1 Overview

RCM performs online compression over distributed monitoring data, which is illustrated by Fig. 6. The runtime operation of RCM alternates between two phases: the training phase and the compression phase. During the training phase, all worker nodes send

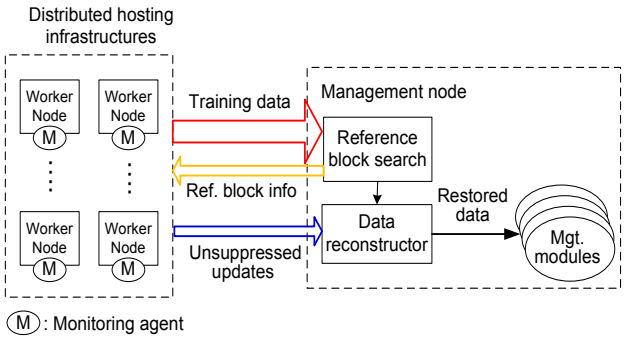


Fig. 6. RCM distributed monitoring architecture.

complete monitoring data to the management node. RCM then builds a sequence of system images using the uncompressed monitoring data. For each system image block, RCM finds the optimal reference based on the compression ratio metric. The management node stores the optimal reference block information (i.e., image index and block position) locally and also sends corresponding reference block information to different worker nodes.

After the training phase is done, RCM starts to perform compression using the reference block information derived during the training phase. Each worker node compares the current attribute value with its reference value. If the difference is within the pre-defined error bound, the worker node suppresses the attribute update for reducing the monitoring data traffic. If the management node does not receive the attribute update from a worker node, it will use the attribute's reference value to restore the suppressed attribute update.

RCM on the management node reconstructs all the monitoring data and delivers the complete monitoring information to different system management modules. If the error bound is set to be zero, RCM achieves lossless compression; If the error bound is set to be non-zero, RCM provides lossy compression but assures that the approximation error is within the pre-defined bound. RCM does not require any explicit synchronization between the worker nodes and the management node. When the worker node sends attribute updates to the management node, the message is denoted by a tuple {*timestamp, a set of attribute values*}. The worker node can send values of different attributes through different communication ports. Thus, on the management node side, we can find out which attributes are omitted due to compression and infer which sampled values of those attributes are missing by checking the gap between different timestamps.

There are several design trade-offs in RCM. First, we need to decide a proper training interval. Intuitively, frequent training helps to maintain a high compression ratio, especially for highly dynamic distributed systems where the optimal reference block

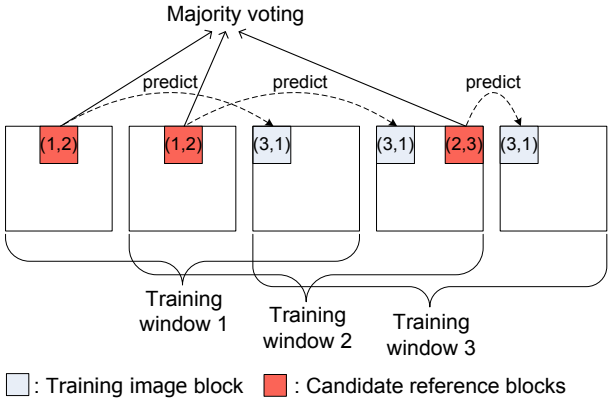


Fig. 7. Online training phase.

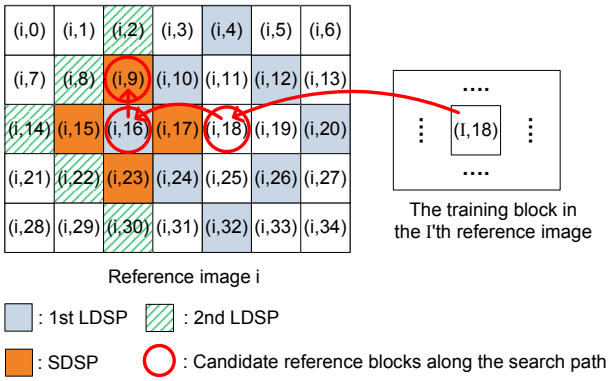


Fig. 8. RCM's reference block search algorithm.

might change from time to time. However, too frequent training also significantly lowers the overall compression ratio because we cannot perform any compression during the training phase (i.e., the "zero compression" effect). Hence, we typically use a moderate training interval for a balanced trade-off. Second, RCM can also support the threshold-driven training, which triggers the training phase whenever the compression ratio drops below a pre-defined threshold. However, we find it difficult to decide an appropriate compression ratio threshold for unseen live monitoring data streams. Furthermore, this scheme might frequently trigger the training if the compression ratio keeps fluctuating around the threshold.

3.2 Online Reference Block Search

One key step in our online compression scheme is to find a good reference block for each system image block. If an attribute value in the current block can be inferred from the corresponding attribute value of its reference block within the error bound, the monitoring agent on the worker node does not need to report the current attribute value to the management node.

As mentioned in the previous subsection, RCM employs a training phase to search the optimal reference block. The training phase consists of R (e.g., $R = 3$) training windows, illustrated by Fig. 7. In each training window, RCM on the management node examines I (e.g., $I = 3$) consecutive system images. We denote those I images as the *reference images*. The last reference image in the training window is called the *training image*. Each block is represented by a coordinate (*image index*, *block position*) where the image index denotes which reference image in the training window and the block position denotes the block sequence number inside one system image. The training phase determines a mapping from the coordinate of each training image block to the coordinate of the reference block. For each block in the training image, RCM checks a number of candidate blocks in all the reference images to select the best reference block. The best reference block is defined as the block that can give the highest compression ratio. RCM repeats this process for R consecutive training windows and use the majority voting to decide the best reference block for each training image block in the training image. Fig. 7 shows an example of the online training phase. The best reference blocks for the current training image block (3,1) during three training windows are block (1,2), (1,2), and (2,3). Using the majority voting, we choose the block (1,2) as the best reference block for the block (3,1). If multiple reference blocks have the same frequency of being the best reference block, we break the tie by selecting the reference block that achieves the highest average compression ratio. Note that RCM only uses the coordinate mappings found during the training phase to perform compression.

Ideally, we want to find the optimal reference block that can achieve the highest compression ratio. However, exhaustively searching all reference blocks will inevitably incur long training time. Thus, for practicality, we want to adopt some fast reference block search algorithms to find near-optimal reference blocks with low overhead. RCM provides a fast reference block search algorithm inspired by a video coding technique [27]. The basic idea is to gradually increase the search range and terminate the search process immediately when little compression performance improvement can be achieved. This search strategy can achieve good tradeoff between the search coverage and the search overhead.

Specifically, our reference block search algorithm uses a dual-diamond search pattern, illustrated by Fig. 8. The first pattern, called *large diamond search pattern* (LDSP), comprises eight blocks surrounding the center block to compose a diamond shape. In Fig. 8, blocks (i,4), (i,10), (i,12), (i,16), (i,20), (i,24), (i,26) and (i,32) form one LDSP with block (i,18) as the center. Blocks (i,2), (i,8), (i,10), (i,14), (i,18), (i,22), (i,24) and (i,30) form another LDSP with block (i,16) as the center. Note that some blocks are overlapped between different LDSPs. The second pattern, called *small diamond search pattern* (SDSP), consists of four

blocks surrounding the center block. In Fig. 8, blocks (i,9), (i,15), (i,17) and (i,23) compose one SDSP with block (i,16) as the center. LDSP is repeatedly used until the best reference block occurs at the center block. We then switch the search pattern from LDSP to SDSP. Among the four blocks in the SDSP and the center block, the block yielding the highest compression ratio is selected as the final reference block. We now describe the major reference block search steps in details as follows. We use the search conducted in the reference image i ($1 \leq i \leq I$) for the training image block (I,18) as an example.

Step 1): The initial LDSP is centered at the co-located block in the reference image i (i.e., block (i,18) in Fig. 8). The center block and the eight neighboring blocks are tested individually by checking the compression ratio that can be achieved when using each of them as the reference block. If the block that achieves the highest compression ratio is located at the center position, we go to Step 3; Otherwise, we go to Step 2. In Fig. 8, the best block of the initial LDSP is block (i,16). Thus, our search proceeds to Step 2.

Step 2): The neighboring block achieving the highest compression ratio in the previous search step (i.e., block (i,16) in Fig. 8) is re-positioned as the center block to form a new LDSP. If the eight neighboring blocks in this new LDSP do not achieve higher compression ratio than the center block, we go to Step 3. This indicates that we want to terminate the broad search since little compression performance improvement can be achieved. Otherwise, we repeat Step 2 to explore more blocks. In Fig. 8, the best block of the second LDSP is center block (i,16) itself. Thus, our search proceeds to Step 3.

Step 3): Switch the search pattern from LDSP to SDSP. The best block found in this step is the final reference block. In Fig. 8, the final reference block is block (i,9).

For each training window, the same search process is conducted on all the I consecutive system images independently. We then compare the best reference blocks of all reference images and choose the one that has the highest compression ratio as the final best reference block. If multiple candidates have the same compression ratio, we break the tie by choosing the best reference block in the image that is closest to the training image in time.

As illustrated by Fig. 8, the search path is formed by the movement of center blocks of successive LDSPs and the movement of the center block of the last LDSP to the final reference block of the SDSP. The blocks along this search path are the candidate reference blocks found in successive diamond search rounds. The compression ratio values obtained by these candidate reference blocks should continuously increase. We also observe that candidate blocks can partially overlap between neighboring search steps. Our algorithm will mark those blocks that have been examined in previous search rounds to avoid repetitive searches.

During the reference block search, RCM excludes the co-located block in the $(I-1)'th$ reference image. The reason is that RCM always uses the last value prediction by default, which checks whether the values of the current block at time t can be predicted by the values of itself at time $t-1$. For better failure resilience, RCM can be configured to exclude those co-located blocks in other reference images as well. The reason is that distributed hosts might experience transient failures. The availability of the affected host might change frequently. Thus, it is unreliable to use any of those recent co-located blocks as the reference block since they might be unavailable too.

At first glance, RCM is similar to a mixture of the temporal and spatial correlation schemes. However, RCM is more effective than these traditional correlation-based approaches because RCM applies a broader search range. We will formally quantify the search range comparison between RCM and correlation-based schemes in Section 3.4. More importantly, RCM has much lower overhead than previous spatial correlation-based scheme, which will be shown in Section 4.

3.3 Failure Resilience

The failed hosts that serve as reference blocks will affect the compression performance since RCM cannot use those reference blocks anymore. However, it is a non-trivial task to distinguish a temporary unavailable node from a healthy node that suppresses the value update for reducing the monitoring cost. Thus, the management node cannot directly rely on the monitoring data to detect the host failures. One simple solution is to require each host to send heartbeat messages periodically, which however will incur extra monitoring overhead. Instead, RCM leverages the inter-node attribute monitoring to detect host failures. Specifically, to monitor inter-node attributes such as network delays and bandwidth, each monitoring agent needs to ping all the other hosts with a pre-defined sampling interval. If the monitoring agent on host A does not get the response from another host B at time t, it will fill in a NULL value in its inter-node attribute report to indicate the host failure B at time t. The management node can thus detect the host failure B when it aggregates the inter-node attribute reports from all the monitoring agents. If there is no inter-node attribute monitoring, RCM can either require each host to send periodical heartbeat messages or rely on existing failure detection schemes (e.g., [28]) to check host liveness.

To achieve *fast* failure recovery, RCM maintains a few additional reference blocks as backups. Specifically, we denote q as the number of backup reference blocks supported by RCM, which is a configurable parameter in our system. We describe how to get q

backup reference blocks under two different conditions.

1) $q \leq (I-1)$. By running the reference block search algorithm on each of the I reference system images, RCM can find I reference blocks for each block in the training image. We rank the I candidate reference blocks in the descending order of their compression ratios. Since $q \leq (I-1)$, we can choose the first one as the primary reference block and the remaining q blocks in the ranked list as the backup reference blocks. If there are R training rounds, we will get R ranked lists. We then perform the majority voting for $(q+1)$ times. Each majority voting chooses one block from R blocks that appear at the same position in R ranked lists. Hence, we can obtain the final $(q+1)$ reference blocks.

2) $q > (I-1)$. We use the same algorithm as above to get I candidate reference blocks ranked using the compression ratio metric. We still choose the first one as the primary reference block and the remaining $I-1$ reference blocks as the backup reference blocks. However, since $q > (I-1)$, we need to find additional $(q-I+1)$ backup reference blocks. Since we have used up all I reference blocks found during the training phase, RCM has to assign a random block in a random reference image as an additional backup reference block. By assigning a random block, we avoid excessive extra overhead of having to search good backup reference blocks again so that we achieve good tradeoff between reliability and cost.

Our failure resilience management scheme introduces little extra overhead since the backup reference block search algorithm is performed as a byproduct of the optimal reference block search algorithm. During the compression phase, the management node can quickly replace the failed reference block with one of its live backups. As mentioned above, we sort all the backups in the decreasing order of their compression ratios and pick the first live backup when the primary reference block fails. If all the backup reference blocks for an attribute fail by chance, RCM stops performing compression on the attribute and will wait until the next training phase to find a set of new reference blocks.

3.4 Analytical Study

We now present our analytical study to formally compare RCM with a set of common alternative schemes: 1) the *temporal* correlation algorithm that suppresses the monitoring updates if the last attribute value can be used to predict the current attribute value within the error bound; 2) the *spatial* correlation algorithm that uses the k-medoids clustering algorithm [29] to group all monitored nodes into different groups. We elect one node in the group (i.e. the cluster head, usually the medoid of each cluster) as the representative node. Other cluster members do not need to send their updates if the difference between their values and the cluster head is within the error bound; 3) the *temporal+spatial* correlation algorithm developed by the InfoTrack system [21] that leverages both temporal and spatial correlations among attribute values to suppress distributed monitoring traffic; and 4) the *neighbor* search algorithm that performs a similar reference block search as RCM but its search range is limited to eight immediate neighbor blocks (i.e., upleft, up, upright, right, downright, down, downleft, and left) in the reference image.

The details about the analytical study, including the compression ratio and overhead comparisons among different approaches, can be found in Section 1 of the online supplemental material. Here, we provide a summary of our analytical results: RCM can achieve higher compression ratio than the other alternative schemes. RCM typically has sub-linear overhead, which is significantly lower than the overhead of the spatial correlation algorithm.

4 System Evaluation

In this section, we first describe our system implementation followed by the data trace descriptions. We then present and analyze our experimental results.

4.1 System Implementation

We have implemented a prototype of the RCM system and conducted extensive experiments using several monitoring traces collected on real-world distributed systems. All the trace-driven compression experiments were conducted on a desktop machine with Intel Core Duo CPU 2.4 GHz and 4GB RAM. For comparison, we also implemented several alternative compression algorithms that have been discussed in Section 3.4.

For RCM and the neighbor search algorithm, we configure the length of each compression phase to be 300 system images for the following two reasons: 1) This value is not too small so that the "zero-compression" impact caused by the training phase can become negligible; and 2) This value is not too large so that it can make the training frequent enough to maintain good compression performance. For consistency, the spatial correlation and the temporal+spatial correlation approaches also use the same interval to perform clustering periodically. The number of reference images I and the number of training rounds R in RCM are both set to be 3. The block size b is set to be 4. We found those parameter settings work well for all the datasets we tested. We also conducted sensitivity studies to show how the choices of different parameter values affect the performance of RCM.

TABLE 2
Statistics of the monitoring traces.

ID	Description	System image dimension	Total data size	Coefficient of variation (CV)
D1	VCL IP statistics (Datagrams/sec)	20x20	18MB	1.53
D2	VCL processor (DPC queued/sec)	20x20	36MB	1.33
D3	PlanetLab CPU (%)	20x20	195MB	0.30
D4	PlanetLab memory (MB)	20x20	261MB	0.39
D5	Google CPU (normalized)	36x36	86MB	0.68
D6	Google memory (normalized)	36x36	86MB	0.59
D7	PlanetLab delay (ms)	464x464	92GB	0.22
D8	PlanetLab delay (ms)	466x466	82GB	0.25
D9	PlanetLab delay (ms)	438x438	106GB	0.18
D10	Traffic matrices (Kbps)	23x23	126MB	0.41

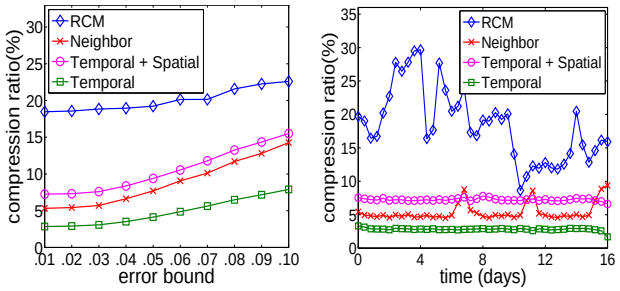
4.2 Trace Data Collection and Statistics

We have collected several real-world distributed system monitoring data traces to evaluate the RCM system. Table 2 summarizes the characteristics of different traces. For intra-node attributes (i.e., D1-D6), a system image of dimension $N \times N$ means that we are monitoring N^2 nodes. For inter-node attributes (i.e., D7-D10), a system image of dimension $N \times N$ means that we are monitoring N nodes. Data size is the total file size of each monitoring data trace. The coefficient of variation (CV) is defined as the ratio of the standard deviation σ to the mean μ (i.e., $CV = \sigma/\mu$). For example, in Table 2, the CV of the VCL IP statistics dataset (D1) is 1.53. It means that the standard deviation of D1 is 1.53 times of the mean of D1, which indicates that this dataset is highly fluctuating. We can use CV as a normalized measure of data variability across different datasets. Generally speaking, the dataset with a smaller CV will have lower variations than other datasets. In Table 2, the dataset with the lowest variation is the third PlanetLab delay trace (D9) that has a CV value of 0.18.

More details on our trace data collection can be found in Section 2 of the online supplemental material.

4.3 Results and Analysis

We first compare RCM with the other alternative compression schemes under no host failures. We then present the compression results of RCM under host failures. We also compare RCM with an offline compression scheme `gzip`. Next, we conduct sensitivity studies to show how the choices of various parameter values affect the performance of RCM. Finally, we measure the overhead of RCM.



(a) overall (b) error bound = 0.01

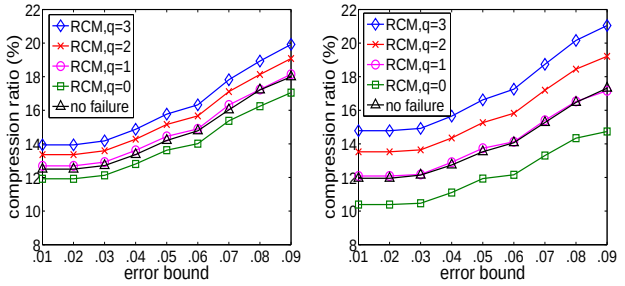
Fig. 9. Compression ratio comparison for the VCL IP Statistics trace.

4.3.1 Compression Comparison Without Host Failures

Summary of results. We conducted experiments on eight datasets described in Table 2 (i.e., D1-D7 and D10). Our experimental results show that RCM can achieve average compression ratio of 28.2%-48.8% under a range of tight error bounds (0.01-0.1) for these datasets. RCM consistently outperforms the other alternative compression schemes. RCM can improve the compression ratio by 24%, 18% and 46% on average compared to the neighbor search algorithm, the temproal+spatial correlation algorithm, and the temporal correlation algorithm, respectively.

Here, we present the detailed compression comparison results of the VCL IP statistics dataset. The results of the other datasets can be found in Section 3.1 of the online supplemental material.

Fig. 9(a) shows the average compression ratio achieved by different schemes under various error bounds for the VCL IP statistics attribute (datagrams/sec). Fig. 9(b) shows the continuously sampled compression ratio values under a fixed error bound (0.01). The attribute is sampled every five minutes and the whole trace lasts about 16 days. According to the statistics, this dataset is highly dynamic. We observe that RCM significantly outperforms all the other schemes under different error bounds. RCM can achieve more than 200% higher compression ratio over the temporal correlation scheme under tight error bounds (e.g., 0.01). We also observe that the neighbor search scheme is slightly worse than the temporal+spatial correlation scheme. The reason is that the neighbor search algorithm has a smaller reference block search range than the spatial+temporal correlation scheme. However, the neighbor search scheme still has the advantage over the temporal+spatial scheme since its computational overhead is much smaller. The compression ratio fluctuation of RCM shown in Fig. 9(b) indicates the dynamic properties of this dataset. It shows that RCM can explore the



(a) 10% failure rate (b) 30% failure rate

Fig. 10. Compression ratio comparison for the Google cluster CPU trace under host failures.

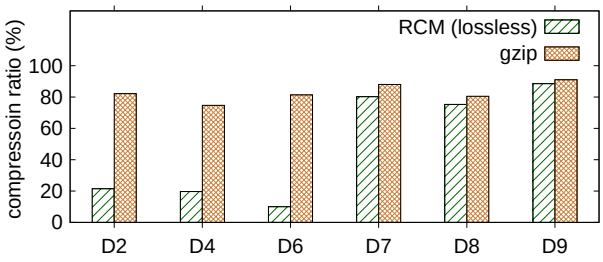


Fig. 11. Compression ratio comparison with `gzip`.

changing compressibility of the monitoring data to achieve highest possible compression ratio throughout the monitoring time.

RCM achieves varied compression ratios across different datasets. This indicates that the optimal compression performance depends on the dynamic input data behavior. The monitoring data with low variations are usually easier to be compressed than other data with high variations. Despite this, RCM consistently outperforms the other alternative schemes for all the datasets we tested. The fundamental reason is that RCM strives to explore a broader search range than temporal and spatial correlation schemes to search the best reference blocks so that the chance of successful compression is greatly increased.

4.3.2 Compression Comparison With Host Failures

Summary of results. We conducted experiments on three datasets described in Table 2 (i.e., D5, D6 and D7) under certain percentage of host failures. Our experimental results show that RCM with additional backup reference blocks can tolerate host failures and achieve similar compression ratios to the case where there is no host failure. Furthermore, RCM can achieve better compression performance as the number of backup reference blocks increases.

Here, we present the experimental results of the Google cluster CPU usage trace in details. The results of the other datasets can be found in Section 3.2 of the online supplemental material.

Fig. 10 shows the compression results for the Google cluster CPU usage trace under 10% and 30% host failures. We compare the overall compression ratio achieved by RCM with different numbers of backup reference blocks (i.e., $q = 0$ to 3). We also show the compression result of the original non-failure trace under the same experiment settings. We have several interesting observations: 1) RCM without any backup reference block only has 10%-20% compression ratio loss compared to the non-failure case. It reflects that RCM inherently has some failure resilience capability; 2) RCM with one backup reference block can achieve similar compression ratio to the non-failure case. It indicates that one backup reference block is already very effective in most cases; 3) RCM can achieve better compression performance when the number of backup reference blocks increases. The reason is that RCM will continue to check next available reference block for both host failures and compression failures (i.e., one reference block fails to provide reference values within a pre-defined error bound). Thus, examining more backup reference blocks will increase the probability of successful compression; and 4) RCM with more than one backup reference block can achieve even higher compression ratio than the non-failure case. The reason is that our backup reference blocks not only covers host failures but also compression failures. If a host failure coincides with a compression failure, more backup reference blocks will increase the chance of successful compression.

The performance of RCM with failure resilience support depends on the quality of the backup blocks. Recall that we rank multiple candidate blocks in the descending order of the compression ratio that can be achieved. We choose the top one as the primary reference block and the remaining ones as the backups. If the differences between the primary reference block and those backup reference blocks are small, the chance of successful compression using the backup reference block is similar to that of using the primary reference block. Thus, RCM with backup reference blocks can achieve similar performance as the no failure case on those data traces that have low attribute value variations.

4.3.3 Compression Comparison With `gzip`

We compare RCM with `gzip`, a widely-used file compression program that is based on a combination of Lempel-Ziv [30] and Huffman coding [31] algorithms. Fig. 11 shows the comparison results for a subset of the datasets described in Table 2. We also test with two more PlanetLab inter-node delay datasets (D8 and D9) that were collected in a similar way with dataset D7 but at different time. For each dataset, we run `gzip` offline and compare the compressed file size with its original size to get the compression ratio. We set the error bound to be zero for RCM since `gzip` is a lossless compression scheme. We observe that RCM can achieve similar compression ratios as

gzip for the PlanetLab inter-node delay datasets that have low attribute value variations. We want to emphasize that RCM performs online compression over live monitoring data streams while gzip can only perform offline compression after the monitoring data have been sent to the management node. Thus, RCM can alleviate the processing bottleneck on the management node, which cannot be achieved by any offline data compression scheme.

4.3.4 Sensitivity Studies

We conduct sensitivity experiments to study the impact of different parameters (e.g., block size, training interval, number of reference images, number of training rounds, system image size) on the compression performance of RCM and the neighbor search algorithm.

Summary of results. We have the following major observations: 1) The block size has noticeable impact on the compression performance. RCM achieves the highest compression ratio when the block size is equal to four for all the datasets in our experiments; 2) The training interval has significant influence on the compression performance. We find that a moderate length of training interval (e.g., 300 system images) can achieve the highest compression ratio; 3) The number of reference images and the number of training rounds have little impact to the performance of RCM; and 4) The system image size affects the neighbor search algorithm only. The neighbor search algorithm achieves the highest compression ratio for square-shape images. In contrast, RCM is not sensitive to different image sizes.

More detailed results can be found in Section 3.3 of the online supplemental material.

4.3.5 Overhead Measurements

Our overhead measurements were collected by running several trace-driven compression experiments on a desktop machine with Intel Core Duo CPU 2.4 GHz and 4GB RAM. We report the training time, the compression time, and the memory usage.

Summary of results. Our results show that RCM has much lower training overhead than the spatial correlation scheme. Specifically, RCM incurs approximately 30 milliseconds of training time for the Google cluster CPU usage dataset containing 1296 hosts, which saves more than 80% training time compared to the spatial correlation scheme. The compression time for RCM is less than 90 microseconds, which is similar to the other approaches. For more challenging and larger datasets (e.g., the PlanetLab inter-node delay dataset consisting of 464 hosts), RCM can still keep the training time within 1 second, which is several orders of magnitude smaller than the training time of the spatial correlation scheme. The compression time for RCM is less than 10 milliseconds. Furthermore, we also find that the backup reference blocks only introduce little extra training and compression overhead. Finally, the memory usage of RCM is only tens of megabytes.

More detailed overhead statistics and their comparisons among different compression schemes can be found in Section 3.4 of the online supplemental material.

5 Related Work

Correlations among distributed data sources can often be exploited for data compression. Correlation-based approaches have been studied under different contexts such as sensor network monitoring [32]–[34], distributed event tracking [35], and resource discovery [36]. Several previous work [11], [20] has proposed to leverage correlation patterns to reduce the monitoring cost. InfoTrack [21] explores both spatial and temporal correlations to compress the live monitoring streams in a large-scale distributed system. However, we find that the temporal correlation scheme has limited compression ratio while the spatial correlation scheme is often costly due to the expensive clustering operations. In contrast, RCM uses light-weight, image-based reference block search algorithms to enable a broader search range so that the compression ratio can be significantly improved without imposing too much overhead. Zhang et al. proposes to leverage spatial and temporal correlations to infer missing values from other received values in Internet traffic monitoring systems [37]. In comparison, RCM addresses an orthogonal problem of reducing the monitoring cost of known values.

In our previous work OLIC [38], we presented an initial design and implementation of an image-based online compressive monitoring system to reducing the distributed monitoring cost. RCM extends OLIC by adding failure resilience support to achieve robust monitoring under host failures. Furthermore, we theoretically prove that RCM outperforms previous correlation-based schemes in terms of higher compression ratio and lower compression overhead.

More discussions on the related work can be found in Section 4 of the online supplemental material.

6 Conclusions

In this paper, we have presented RCM, a novel image-based resilient self-compressive monitoring system for large-scale hosting infrastructures. RCM models snapshots of the monitored distributed system using a sequence of system images and applies light-weight online reference block search algorithms to compress distributed monitoring data. RCM is failure resilient, which can tolerate host and network failures that are common in real-world hosting infrastructures. To the best of our knowledge, we make the first attempt to adopt an image-based approach to achieving efficient and robust distributed monitoring traffic reduction.

We have implemented the RCM system and conducted extensive experiments using a wide range of real-system monitoring data collected on PlanetLab, VCL, a Google cluster, and real Internet traffic matrices. Our prototype implementation shows that RCM is practical and efficient, which can achieve higher compression ratio with lower overhead than previous compression schemes.

ACKNOWLEDGMENT

This work was sponsored in part by NSF CNS0915567 grant, NSF CNS0915861 grant, NSF CAREER Award CNS1149445, U.S. Army Research Office (ARO) under grant W911NF-10-1-0273, IBM Faculty Awards and Google Research Awards. Any opinions expressed in this paper are those of the authors and do not necessarily reflect the views of NSF, ARO, or U.S. Government. The authors would like to thank the anonymous reviewers for their insightful comments.

REFERENCES

[1] "Amazon Elastic Compute Cloud," http://aws.amazon.com/ec2/.
[2] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proc. of OSDI*, 2004.
[3] "Apache Hadoop System," http://hadoop.apache.org/core/.
[4] L. Amini, N. Jain, A. Sehgal, J. Silber, and O. Verscheure, "Adaptive Control of Extreme-scale Stream Processing Systems," in *Proc. of ICDCS*, 2006.
[5] M. Parashar and S. Hariri, "Autonomic Computing: An Overview," in *Proc. of UPP*, 2004.
[6] Z. Gong and X. Gu, "PAC: Pattern-driven Application Consolidation for Efficient Cloud Computing," in *Proc. of MASCOTS*, 2010.
[7] Z. Gong, X. Gu, and J. Wilkes, "PRESS: PRedictive Elastic ReSource Scaling for Cloud Systems," in *Proc. of CNSM*, 2010.
[8] "RUBiS Online Auction System," http://rubis.ow2.org/.
[9] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase, "Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control," in *Proc. of OSDI*, 2004.
[10] X. Gu and H. Wang, "Online Anomaly Prediction for Robust Cluster Systems," in *Proc. of ICDE*, 2009.
[11] L. Huang and et al., "Communication-Efficient Online Detection of Network-Wide Anomalies," in *Proc. of IEEE INFOCOM*, 2007.
[12] "CoMon," http://comon.cs.princeton.edu/.
[13] "IBM Tivoli Monitoring software," http://www-01.ibm.com/software/tivoli/.
[14] "NCSU Virtual Computing Lab," http://vcl.ncsu.edu/.
[15] "httperf," http://code.google.com/p/httperf/.
[16] R. Van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 164–206, 2003.
[17] P. Yalagandula and M. Dahlin, "A Scalable Distributed Information Management System," in *Proc. of SIGCOMM 2004*, Aug. 2004.
[18] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Design and Implementation Tradeoffs for Wide-Area Resource Discovery," in *Proc. of HPDC-14*, 2005.
[19] J. Liang, X. Gu, and K. Nahrstedt, "Self-Configuring Information Management for Large-Scale Service Overlays," in *Proc. of INFOCOM*, 2007.
[20] N. Jain, D. Kit, P. Mahajan, P. Yalagandula, M. Dahlin, and Y. Zhang, "STAR: Self-Tuning Aggregation for Scalable Monitoring," in *Proc. of VLDB*, 2007.
[21] Y. Zhao, Y. Tan, Z. Gong, X. Gu, and M. Wamboldt, "Self-Correlating Predictive Information Tracking for Large-Scale Production Systems," in *Proc. of ICAC*, 2009.
[22] J. W. Mickens and B. D. Noble, "Exploiting Availability Prediction in Distributed Systems," in *Proc. of NSDI*, 2006.
[23] Y. Tan and X. Gu, "On Predictability of System Anomalies in Real World," in *Proc. of MASCOTS*, 2010.
[24] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology Into the Internet," in *Proc. of HotNets-I*, 2002.
[25] "Google Cluster Data," http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.
[26] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing Public Intradomain Traffic Matrices to the Research Community"," *Computer Communication Review*, vol. 36, no. 1, pp. 83–86, 2006.
[27] S. Zhu and K. K. Ma, "A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation," *IEEE Trans. Image Processing*, vol. 9, no. 2, pp. 287–290, Feb. 2000.
[28] J. B. Leners, H. Wu, W.-L. Hung, M. K. Aguilera, and M. Walfish, "Detecting failures in distributed systems with the falcon spy network," in *Proc. of SOSP*, 2011.
[29] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.
[30] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *Information Theory, IEEE Transactions on*, vol. 24, no. 5, pp. 530 – 536, sep 1978.
[31] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.
[32] M. C. Vuran and I. F. Akyildiz, "Spatial Correlation-Based Collaborative Medium Access Control in Wireless Sensor Networks," *IEEE/ACM Transactions on Networking (TON)*, 2006.
[33] S. Krishnamurthy, T. He, G. Zhou, J. A. Stankovic, and S. Son, "RESTORE: A Real-time Event Correlation and Storage Service for Sensor Networks," in *Proc. of ICNSS*, 2006.
[34] A. Deshpand, E. C. Guestrin, and S. R. Madden, "Model-driven data acquisition in sensor networks," in *Proc. of VLDB*, 2002.
[35] A. Jain, E. Y. Chang, and Y.-F. Wang, "Adaptive Stream Resource Management Using Kalman Filters," in *Proc. of SIGMOD*, 2004.
[36] M. Cardosa and A. Chandra, "Resource Bundles: Using Aggregation for Statistical Wide-Area Resource Discovery and Allocation," in *Proc. of ICDCS*, 2008.
[37] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices," in *Proc. of SIGCOMM*, 2009.
[38] Y. Tan, X. Gu, and V. Venkatesh, "OLIC: Online information compression for scalable hosting infrastructure monitoring," in *Proc. of IWQoS*, 2011.

Yongmin Tan is a PhD candidate in the Department of Computer Science at the North Carolina State University. He received the BE degree and ME degree, both in electrical engineering from Shanghai Jiaotong University, Shanghai, China in 2005 and 2008 respectively, and has interned with NEC Lab America. He is a student member of IEEE.

Vinay Venkatesh is a software engineer in IBM, Research Triangle Park, NC, USA. He received his BE degree in computer science from M.S. Ramaiah Institute of Technology, Bangalore, India in June 2005. Later, he worked at Infosys Technologies Limited, Bangalore, India for 2 years. He received the MS degree in computer science from North Carolina State University in December 2010.

Xiaohui Gu is an assistant professor in the Department of Computer Science at the North Carolina State University. She received her PhD degree in 2004 and MS degree in 2001 from the Department of Computer Science, University of Illinois at Urbana-Champaign. She received her BS degree in computer science from Peking University, Beijing, China in 1999. She was a research staff member at IBM T. J. Watson Research Center, Hawthorne, New York, between 2004 and 2007. She received ILLIAC fellowship, David J. Kuck Best Master Thesis Award, and Saburo Muroga Fellowship from University of Illinois at Urbana-Champaign. She also received the IBM Invention Achievement Awards in 2004, 2006, and 2007. She has filed eight patents, and has published more than 50 research papers in international journals and major peer-reviewed conference proceedings. She is a recipient of NSF Career Award, four IBM Faculty Awards 2008, 2009, 2010, 2011, and two Google Research Awards 2009, 2011, a best paper award from IEEE CNSM 2010, and NCSU Faculty Research and Professional Development Award. She is a Senior Member of IEEE.