OLIC: OnLine Information Compression for Scalable Hosting Infrastructure Monitoring

Yongmin Tan
Department of Computer Science
North Carolina State University
ytan2@ncsu.edu

Vinay Venkatesh
IBM RTP
venkatvi@us.ibm.com

Xiaohui Gu
Department of Computer Science
North Carolina State University
gu@csc.ncsu.edu

***Abstract*—Quality-of-service (QoS) management often requires a continuous monitoring service to provide updated information about different hosts and network links in the managed system. However, it is a challenging task to achieve both scalability and precision for monitoring various intra-node and inter-node metrics (e.g., CPU, memory, disk, network delay) in a large-scale hosting infrastructure. In this paper, we present a novel OnLine Information Compression (OLIC) system to achieve scalable fine-grained hosting infrastructure monitoring. OLIC models continuous snapshots of a hosting infrastructure as a sequence of images and performs online monitoring data compression to significantly reduce the monitoring cost. We have implemented a prototype of the OLIC system and deployed it on the PlanetLab and NCSU's virtual computing lab (VCL). We have conducted extensive experiments using a set of real monitoring data from VCL, Planetlab, and a Google cluster as well as a real Internet traffic matrix trace. The experimental results show that OLIC can achieve much higher compression ratios with several orders of magnitude less overhead than previous approaches.**

I. Introduction

Large-scale distributed hosting infrastructures have become 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. Particularly, quality-of-service (QoS) management plays a key role to convince users to migrate their applications into those hosting infrastructures. However, to achieve automatic QoS management, the first step is to provide a continuous monitoring service that can collect various resource and performance metrics (e.g., host resource availability, application resource usages, inter-node network delays/bandwidth) in the hosting infrastructure.

To achieve efficiency, the QoS management module often desires to obtain *complete* and *fine-grained* information about all hosts and network connections within the hosting infrastructure. Our previous work [5], [6] has shown that fine-grained monitoring information can greatly improve QoS provisioning performance. Similarly, online performance anomaly detection system [7]–[9] also depends on fine-grained monitoring data to achieve high accuracy.

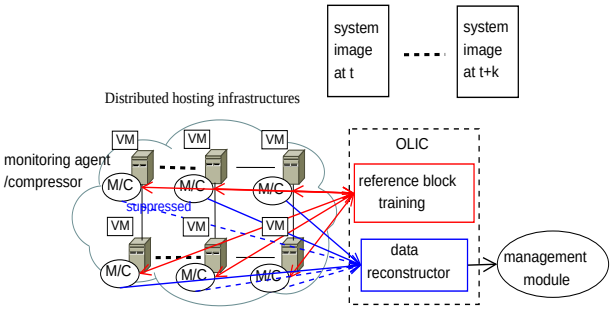


Fig. 1. OLIC distributed monitoring architecture.

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 hosts and many more virtual machines (VMs), each of which can be associated with tens of or hundreds of dynamic metrics [10], [11]. For example, the IBM Tivoli monitoring system [11] can collect over 600 metrics on a host running Windows OS. Hence, most production hosting infrastructures [10], [13] typically use a long update interval (e.g., every five minutes) to tradeoff information precision for scalability, which unfortunately will significantly affect the performance of the QoS management functions [5]–[8].

Previous research work has identified the challenge of scalable distributed monitoring and proposed various solutions to address the problem: i) employing decentralized architectures such as hierarchical aggregation [14] or peer-to-peer structure [15], [16] to distribute monitoring workload; or ii) trading off information coverage [17] or precision [18] for lower monitoring cost. In contrast, the goal of our research is to provide a fine-grained and full coverage monitoring system that can be applied to any distributed hosting infrastructure. Recent work [9], [19] has proposed to leverage temporal and/or spatial correlations to reduce monitoring overhead. However, exploring temporal correlation alone has limited cost reduction [19] while spatial correlation discovery can be costly when the size of the hosting infrastructure is large.

In this paper, we present a novel light-weight OnLine Information Compression (OLIC) framework that explores a new image-based approach to scalable distributed monitoring.

978-1-4577-0103-0/11/$26.00

Figure 1 shows the overall architecture of the OLIC system. OLIC models snapshots of dynamic monitoring metrics of the whole hosting infrastructure as a sequence of system images. We partition each system image into a set of blocks and dynamically search the optimal reference block from a window of recent snapshots for each block. OLIC reduces monitoring cost by *suppressing* the remote update of those attributes whose values can be inferred by their reference values within a user defined error bound. Compared to previous correlation-based approaches, OLIC employs a larger search range to find better reference blocks. Moreover, OLIC does not enforce a group of data blocks to share a common reference block as in the spatial correlation based scheme.

OLIC can reconstruct all monitoring data and deliver full-coverage and fine-grained information to other system management modules. If the error bound is set to be 0, OLIC delivers the exact original attribute value to the management module; If the error bound is non-zero, OLIC provides approximate values for some attributes. However, all approximation errors are guaranteed to be within the specified error bound. Our experiments will show that a small approximation error in monitoring data does not significantly affect the performance of the management function.

This paper makes the following contributions:

- We propose OLIC, a light-weight online information compression framework to enable scalable, full-coverage, fine-grained monitoring for large-scale hosting infrastructures.
- We have implemented a prototype of the OLIC system and deployed it on the PlanetLab [20] and NCSU virtual computing lab (VCL) [13]. Our prototype implementation shows that our approach is feasible and practical for real world hosting infrastructures.
- We conducted extensive experiments using real system monitoring data from PlanetLab, VCL, a Google cluster [21], and real Internet traffic matrices [22]. Our experimental results show that the OLIC can achieve up to 95% monitoring cost reduction under a range of tight error bounds (0.01-0.1) and 70% reduction for lossless compression (i.e. 0 error bound). OLIC can improve the compression ratio by up to 200% with more than several orders of magnitudes less overhead than other alternative schemes.

The rest of the paper is organized as follows. Section II gives an overview about our system model and problem formulation. Section III describes the design details of the OLIC system. Section IV presents the prototype implementation and experimental evaluation. Section V compares our work with related work. Finally, the paper concludes in Section VI.

II. PRELIMINARY

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

A. Monitoring System Model

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

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 also periodically sample each inter-node attribute to form a time series $\{d^1_{i,j}, \ldots, d^t_{i,j}, \ldots, d^{t+m}_{i,j}\}$ where $d^t_{i,j}$ denotes the value of the inter-node attribute $d_{i,j}$ at time t.

OLIC performs compressed information collection from all monitoring agents for reducing the distributed monitoring cost. On the management node, OLIC decompresses the monitoring data and delivers complete, fine-grained monitoring data time series to other QoS management modules.

B. Problem Formulation

The goal of the OLIC monitoring system is to provide full-coverage and fine-grained monitoring information to the management node with low information collection 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 attribute value using other reference values that are already known to the management node. OLIC allows the user to define an error bound $e (e \geq 0)$ to indicate the maximum approximation error that can be tolerated by the management function. If $|a_i - a'_i|/a_i \leq e$, OLIC can suppress the update of the attribute value a_i from the worker node and restore the value of a_i on the management node using its reference value a'_i within the error bound e.

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

$$CR = \frac{N_{compressed}}{N_{orig}} \quad (1)$$

$N_{compressed}$ is the number of attribute values whose updates are suppressed by OLIC and N_{orig} is the number of original attribute updates without any compression. The larger the compression ratio, the more monitoring cost reduction can be achieved by the OLIC system. To maximize the compression ratio, OLIC needs to find the optimal reference value for each attribute whose value can be used as reference with the highest probability. However, the key question is how to discover such reference values online and maintain the efficiency of reference values while monitoring a dynamic hosting infrastructure. The goal of the online compression

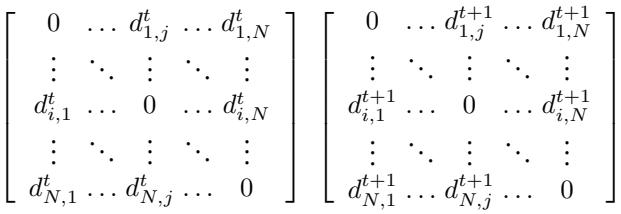


Image at time t Image at time $t+1$

($d^t_{i,j}$: inter-node attribute between node *i* to node *j*)

Fig. 2. System image sequence for an inter-node attribute for a distributed system of N nodes.

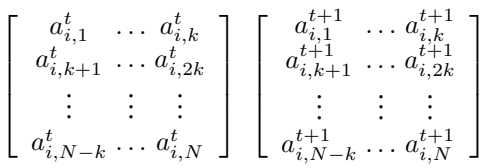


Image at time t Image at time $t+1$

($a^t_{i,k}$ = attribute a_i of the node k)

Fig. 3. System image sequence for an intra-node attribute for a distributed system of N nodes.

algorithm is to efficiently address this question. Furthermore, the online compression algorithm needs to be light-weight by itself. Otherwise, the cost of the compression will defeat the original purpose of the compression.

We propose a novel image based approach to achieving light-weight online compression for the distributed monitoring system. We model a snapshot of a distributed system using a *system image*, which is described in detail as follows:

For monitoring an inter-node attribute (e.g., network delay) of a distributed system consisting of N nodes, the system image at time t comprises $N \times N$ pixels, illustrated by Figure 2. The pixel at $i'th$ row and $j'th$ column denotes the attribute value between the node i and the node j at time t. Note that the main diagonal of this image should have all zero values since the inter-node attribute value does not exist for one node itself.

For monitoring an intra-node attribute (e.g., CPU load) of a distributed system consisting of N nodes, the system image at time t comprises N pixels where each pixel denotes the attribute value at one particular node. We can organize the N pixels into l rows and k columns where $l \cdot k = N$, illustrated by Figure 3. To fully utilize the power of our search algorithm described in Section III-A, we choose the values of l and k so that the system image is close to a square matrix.

We partition each system image into a set of small blocks, each of which contains $n \times n$ attribute samples where n is a tunable small number. We can then perform reference value search at block level instead of the pixel level. For each block, we strive to find the optimal reference block for each block using a fast reference block search algorithm that will be described in the next section.

There are several design issues we need to consider. First, our information compression system needs to handle decentralized monitoring data where different image blocks are disseminated on different distributed hosts. This requires all monitoring agents and the management node to perform the online information compression together in a coordinated way. Second, our system deals with dynamic live monitoring data, which requires an adaptive online training algorithm to maintain the efficiency of the compression.

Different from the static, offline compression scheme (e.g., gzip) that can only be applied after the data have been reported to the management node, our approach performs online compression over live monitoring data streams during monitoring runtime. Thus, our approach can reduce end-system resource and network bandwidth consumption on both monitored worker nodes and management node, which cannot be achieved by previous offline compression techniques.

III. System Design

In this section, we present the design and algorithm details of the OLIC system. We first describe our reference block search algorithm. Next, we present the online information compression algorithm.

A. Online Reference Block Training

One key step in our online compression scheme is to select a good reference block for each block. As long as an attribute value within one block can be inferred from the attribute value of its corresponding block within the user defined error bound, the monitoring agent does not need to report the current value to the management node. Ideally, we wish to find the optimal reference block for each block that can achieve the highest compression ratio defined by Equation 1. However, finding the optimal reference block would require us to search all the blocks in all historical system images. This will inevitably impose high overhead to the monitoring system. Thus, in practice, we have to adopt fast reference block search algorithms to find near-optimal reference blocks with low overhead.

OLIC employs a training phase to perform online reference block search, illustrated by Figure 4. During the training phase, the management node examines a window of consecutive system images preceding the current system image. We call those images *reference images* and the current image as *training image*. For each block in the training image, the management node examines a number of blocks in all reference images as well as the current image to find its best reference block. We then slide our training window to an earlier time by one step and designate the image immediately preceding the current image as the training image. We then apply the reference block search algorithm to find another set of reference blocks for each block in the current training image. We repeat the above training process a few times and use majority voting to decide the best reference block for each block.

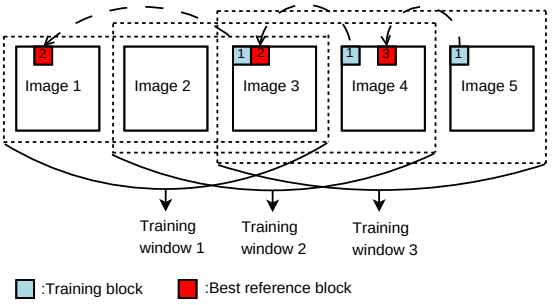


Fig. 4. Online training phase.

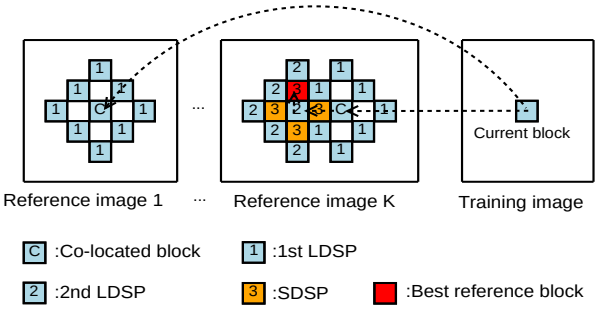


Fig. 5. Reference block search using dual diamond pattern.

For example, in Figure 4, the best reference blocks for block 1 during the three rounds of training are $\{2, 2, 3\}$. Thus, we will select block 2 as the reference block for block 1. If multiple reference blocks have the same frequency, we break tie by selecting the reference block with the highest average compression ratio. Note that we will not achieve any compression during the reference block training phase since all monitoring agents need to send all their recent sample values to the management node during this process. Thus, we cannot perform training over a large number of reference images.

We provide a fast and efficient reference block search algorithm inspired by similar techniques in the video coding area [23]. The idea is to greedily increase the search range to explore more candidate reference blocks and terminate the search immediately when little compression improvement can be achieved. This search strategy can achieve good tradeoff between search coverage and search overhead.

Specifically, the search algorithm follows dual-diamond search patterns illustrated by Figure 5. The first pattern is a *large diamond search pattern* (LDSP) that searches eight blocks surround the center block following a diamond shape and the center block. The second pattern is a *small diamond search pattern* (SDSP) that searches four blocks surrounding the center block which form a small diamond with the center block. Our search algorithm repeatedly conduct LDSP search until the best reference block occurs at the center block. Next, the search pattern is switched from LDSP to SDSP to find the best reference block among the five blocks included in SDSP. For example, in Figure 5, the search path includes two LDSPs and one SDSP. The arrows indicate the movement of the diamond center. Finally, the red block in the reference image K is chosen as the best reference block.

In the worst case, LDSP needs to search all blocks in the current image with a computation complexity of $O(M^2)$, where M denotes the number of blocks. However, in practice, we found that the search algorithm has sub-linear overhead in most cases since the search process often terminates after a few search rounds.

B. OLIC Compression Algorithm

We now describe the detailed online compression algorithms of the OLIC system. The runtime operation of the OLIC system involves both the management node and all distributed monitoring agents. The online compression process alternates between two phases: the training phase and the compression phase.

During the training phase, all monitoring agents send all monitoring data to the management node. Thus, OLIC achieves zero compression at this stage. When the management node accumulates a number of raw system images, it executes the online reference block training algorithm described in Section III-A to derive the reference blocks that give the best compression ratio. The management node then records the reference block information (e.g., block location, the index of the reference image within the training window) for each image block. Next, the management node sends the reference block information to the corresponding monitoring agents.

During the compression phase, each monitoring agent continuously compares the current sample value of each attribute with the corresponding reference value that has been reported to the management node in the reference block. If the difference is within the pre-defined error bound, the monitoring agent will omit the report of its current value to the management node. Otherwise, the monitoring agent will send the current attribute value to the management node. On the management node side, if the management node does not receive the update from a monitoring agent during the current sampling period, it assumes that it can infer the attribute value using the attribute's reference value. Otherwise, the management node will use the most recent value reported by the monitoring agent as the sample value for the current sampling period. One tricky situation is that the management node may not be able to distinguish between a failed node or a healthy node that performs information compression when it does not receive updates from that node. We will further discuss this in the next section.

The compression power of the OLIC system depends on the effectiveness of the reference blocks. Since distributed monitoring data streams are often dynamic, the effectiveness of the reference blocks may become weaker after a period of time. The management node can calculate recent compression ratios achieved by the current reference block assignment using Equation 1. The management node can trigger the reference block training periodically or when compression ratios fall below a certain threshold. Figure 6 shows the pseudo-code of the training phase and compression phase on

K: number of reference images
r: number of training rounds

Training()
1. build $(K + r)$ system images
2. for i =1 to r
3. load images i to $(i + K - 1)$ as reference images
4. load the image $(i + K)$ as the current training image
5. run GS or LS to find best reference blocks
6. decide best reference blocks using majority voting
7. send the reference block information back to corresponding monitoring agents

Compression()
1. for each worker node at time t
2. if (no update from this node)
3. if (this node has failed)
4. reassign reference hosts for affected blocks
5. else
6. use the attribute's reference value
7. else
8. use the most recent reported value

Fig. 6. OLIC algorithm on the management node.

the management node.

IV. System Evaluation

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

A. Prototype Implementation

We have implemented a prototype of the OLIC system and deployed it on: 1) the PlanetLab [20] that is a wide-area network testbed, and 2) the NCSU virtual computing lab (VCL) [13] that is a production virtualized hosting infrastructure similar to Amazon EC2 [1]. We deploy a monitoring agent on each host and run the management node program on a dedicated server in our lab as shown in Figure 1. The server machine has configuration of Intel Core Duo CPU 2.4 GHz with 4GB RAM. Each monitoring agent on the PlanetLab collects about 66 attributes that are supported by the PlanetLab CoMon monitoring tool [10] (e.g., CPU load, free memory, available CPU). Each monitoring sensor also periodically pings other nodes in the system to collect inter-node attributes such as network delay and bandwidth. The monitoring agent on VCL host is connected to the IBM Tivoli Monitoring agent [11] that can collect hundreds of attributes.

For comparison, we also implemented several alternative online compression algorithms: 1) the *temporal* algorithm that suppresses the monitoring updates if the last value can be used to predict the current value within the pre-defined error bound[1]; 2) the *spatial* algorithm uses the k-medoids clustering algorithm [26] to group all monitored nodes into different groups. We elect one node in the group (i.e. cluster head, usually the medoid of each cluster) as the representative. Other cluster members do not need to send their updates if the difference between their values and the cluster head is within a predefined error bound; 3) the *temporal+spatial* algorithm developped by our previous work (the InfoTrack system) [19] that leverages both temporal and spatial correlations among attribute values from different nodes to suppress distributed monitoring traffic; and 4) the *neighbor* search algorithm that performs a similar online compression process as OLIC but its reference block search algorithm only examines eight neighboring blocks and the co-located blocks in the reference images.

In most of our experiments, we trigger the online training algorithm with a fixed training interval ranging within [200,300] system images. The number of reference images K and the number of training rounds r are both set to be 3. The spatial and the temporal+spatial approaches also use the same training interval to perform clustering periodically. We will conduct sensitivity studies to discuss the impact of these parameters.

B. Trace Collection

To compare the performance of different compression algorithms, we use real world distributed system monitoring data to drive our experiments. Both the monitoring agents and the management node are fully implemented with the monitoring traces replayed at different monitoring agents.

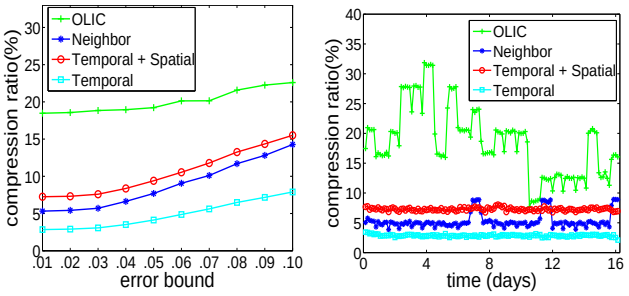
The VCL monitoring traces are collected by the production VCL system using the IBM Tivoli monitoring software [11]. The trace dataset contains various performance attributes for 400 VCL nodes from Oct.18th, 2010 to Nov.3rd, 2010. The sampling interval is five minutes. In our experiments, we test our algorithms on the IP statistics attribute (datagrams/sec) and windows NT processor attribute (DPC queued/sec).

The PlanetLab data were collected by our distributed monitoring system [17], [19] deployed on 500 PlanetLab nodes. The monitoring agent on each node collected various system-level attributes (i.e., intra-node attributes) that are supported by the PlanetLab CoMon monitoring tool [10] (e.g., CPU load, free memory, available CPU etc.) at a sampling interval of ten seconds. Each monitoring agent also periodically pings other nodes in the system to collect inter-node attributes such as network delay and bandwidth. We collected a dataset of intra-node attributes containing 400 nodes from Jan.29th, 2009 to Feb.3rd, 2009. We collected another dataset of inter-node attributes containing 464 nodes from Oct.4th, 2009 to Oct.6th, 2009.

To test inter-node attributes, we also used a real Internet traffic matrices collected by previous research work from a transit network [22]. This dataset contains traffic matrices sampled every 15 minutes for a period of about four months.

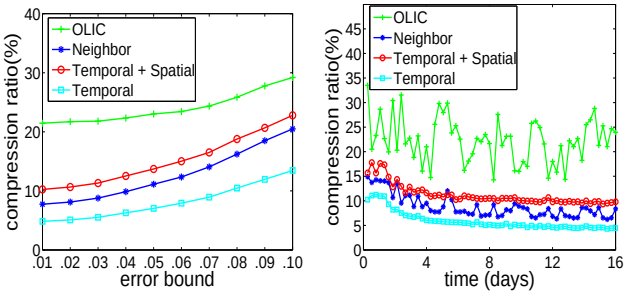
We also got a small sample of real application workload trace data from a Google cluster [21]. The dataset contains normalized CPU and memory usage attributes for more than 30,000 different jobs, with a sampling interval of five minutes. We selected a subset of these jobs (i.e., 1296 jobs) which have larger variations in the raw data. Since different jobs

[1]We have implemented other temporal prediction algorithms such as Kalman filter, which however have very similar performance with the last value based approach [19].



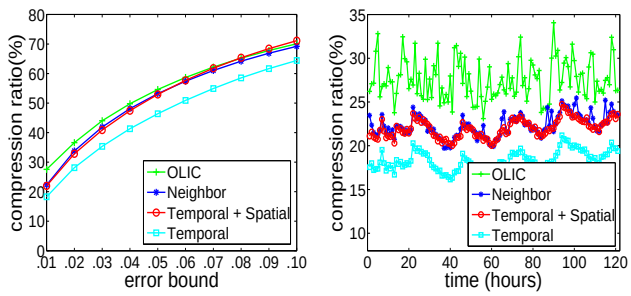
(a) overall (b) error bound = 0.01

Fig. 7. Compression ratio comparison for the VCL IP Statistics trace. (Intra-node attribute, 400 nodes, Mean: 38.69, Standard deviation: 82.58, Sampling interval: 5 minutes, Total data size: 18MB)



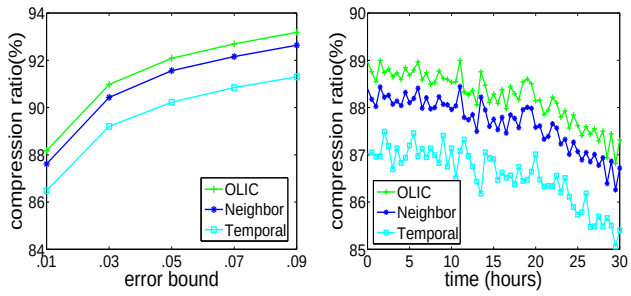
(a) overall (b) error bound = 0.01

Fig. 8. Compression ratio comparison for the VCL NT Processor trace. (Intra-node attribute, 400 nodes, Mean: 45.4, Standard deviation: 58.23, Sampling interval: 5 minutes, Total data size: 36MB)



(a) overall (b) error bound = 0.01

Fig. 9. Compression ratio comparison for the PlanetLab free memory trace. (Intra-node attribute, 400 nodes, Mean: 107, Standard deviation: 42, Sampling interval: 10 seconds, Total data size: 744MB)



(a) overall (b) error bound = 0.01

Fig. 10. Compression ratio comparison for the PlanetLab inter-node delay trace. (Inter-node attribute, 464 nodes, Mean: 241.8, Standard deviation: 58.9, Sampling interval: 10 seconds, Total data size: 92GB)

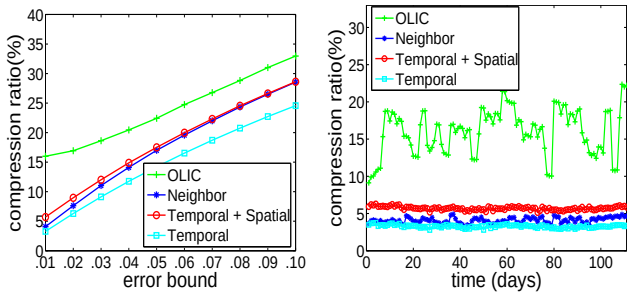
have varied execution length in the original trace, we repeated short jobs from randomized start points within the trace. We also assume that those jobs are executed on different hosts.

C. Results and Analysis

We first present the compression comparison results using the VCL monitoring datasets. Figure 7(a) shows the average compression ratio achieved by different schemes under various error bounds for the IP statistics attribute (datagrams/sec). Figure 7(b) shows the compression ratio changes in time-series for different approaches under 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 fluctuating with a very large standard deviation. We observe that OLIC significantly outperforms all the other schemes with more than 200% higher compression ratios under tight error bound requirements. The temporal algorithm can only achieve very low compression ratios since the monitoring data are highly dynamic. The compression ratio fluctuation of OLIC scheme in Figure 7(b) is caused by the variation degree change in the monitoring data. It indicates that that OLIC can fully explore the changing compressibility of the monitoring data to maintain the highest compression ratio. Figure 8(a) and Figure 8(b) show the compression result of NT processor attribute (DPC Queued/sec) in the VCL processor statistics trace. The results are consistent with those of the IP statistics trace.

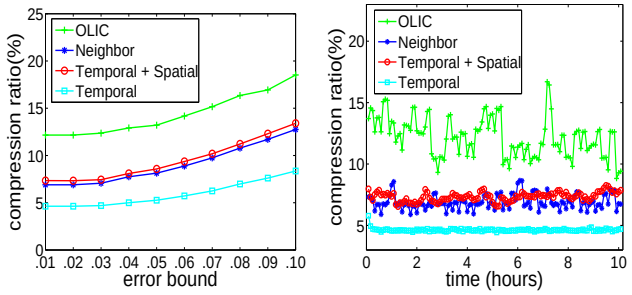
We then present the results of monitoring intra-node attributes on the PlanetLab. Each monitoring attribute is sampled every 10 seconds and the whole trace lasts about six days. Figure 9(a) and Figure 9(b) shows the average compression results and time-series compression ratio changes for PlanetLab free memory attribute trace. Different from the VCL datasets, the Planetlab monitoring data are more stable with smaller standard deviations. Thus, we can see that the PlanetLab monitoring data are much easier to be compressed with up to 70-80% compression ratios. This also is the reason why the performance of all approaches are very close. However, OLIC and temporal+spatial algorithms still perform better than the temporal algorithm. We will show later that OLIC has much lower overhead than the temporal+spatial algorithm. Furthermore, OLIC still performs the best under rigid error bounds (e.g., 0.01).

Next we present the compression results for PlanetLab inter-node delay dataset in Figure 10(a) and Figure 10(b). The size of this dataset (92GB) is much bigger than the other datasets. We observe that this dataset is also very stable and easy to be compressed. OLIC can achieve 88% compression ratio under a very tight 0.01 error bound and 70% compression ratio for lossless compression (i.e. 0 error bound). The reason is that the measured delay between two PlanetLab nodes are stable within certain period of time. We note that the inter-node metric shows better compression potential. However, it is impractical to apply previous spatial-temporal correlation to the inter-node metrics due to its high overhead. In contrast,



(a) overall (b) error bound = 0.01

Fig. 11. Compression ratio comparison for the traffic matrices trace. (Inter-node attribute, 23 nodes, Mean: 17040, Standard deviation: 52578, Sampling interval: 15 minutes, Total data size: 126MB)



(a) overall (b) error bound = 0.01

Fig. 12. Compression ratio comparison for the Google cluster cpu trace. (Intra-node attribute, 1296 nodes, Mean: 0.039, Standard deviation: 0.026, Sampling interval: 5 minutes, Total data size: 86MB)

OLIC can easily achieve a very high compression ratio with little overhead. In our experiment, the spatial algorithm took around 6 minutes while OLIC only needs 50 milliseconds for one training period.

Now we present the compression results for another inter-node attribute trace - Internet traffic matrices, shown by Figure 11(a) and Figure 11(b). Again, we observe that the OLIC algorithm consistently outperforms the other algorithms over different error bound settings. In Figure 11(b), the fluctuating compression ratio curve for OLIC indicates that OLIC can dynamically discover the best reference blocks to maintain the highest compression ratio.

Next, we present the compression results for the CPU monitoring datasets from a Google cluster. Google normalizes original data using some secrete linear function for privacy protection. However, the normalized data preserve the changing pattern of the original data. Figure 12(a) and Figure 12(b) show the compression result for the cpu attribute. Again, we observe that the OLIC algorithm consistently outperforms all the other algorithms.

As we have seen, OLIC achieves varying compression ratios because of the data variability among different datasets. However, OLIC consistently outperforms other alternative schemes in all test cases. The fundamental reason is that OLIC dynamically and intelligently explores much broader search range than temporal and spatial approaches so that the chance of finding best reference blocks for higher compression ratio is greatly increased.

Error bound	**0.01**	**0.05**
Block 2x2	17.71	33.60
Block 4x4	**19.98**	**36.32**
Block 6x6	18.72	34.94
150 mins. interval	14.74	27.50
1000 mins. interval	18.72	34.94
3000 mins. interval	**19.18**	**36.68**
40000 mins. interval	14.41	31.93
K=5, r=5	**20.04**	**35.31**
K=3, r=3	19.45	35.29
K=1, r=1	18.72	34.94
Image 36x36	**18.72**	**34.94**
Image 24x54	18.65	34.17
Image 18x72	18.58	34.08

TABLE I
SENSITIVITY EXPERIMENT RESULTS FOR THE GOOGLE CLUSTER CPU TRACE.

We conduct sensitivity experiments to investigate the impact of different parameters (i.e., block size, training interval, reference image number, training rounds). Table I shows a subset of results for the Google cluster CPU trace with the optimal parameter settings highlighted.

First, block size decides the granularity of the search algorithms. A larger block size can reduce the number of search steps and thus decrease the computational overhead. However, under coarse granularity, it will lead to less accurate block matching. On the other hand, a smaller block size performs fine-grained search but has limited search range. It is also less robust to noises in data (e.g., some transient similarity). In our experiments, we observe that block size 4x4 achieves the best performance.

Second, we evaluate the impact of training interval. The reference block training is triggered more frequently under a smaller training interval. The consequence is that we can always have an accurate model. However, the overall compression ratio will be affected since those images during the training phase have zero compression ratio. In contrast, a larger training interval can mitigate the negative effect of zero compression ratio but runs the risk of losing reference block freshness. The normalized Google cluster CPU trace contains 7500 data samples and the sampling interval is five minutes. We observe that training interval of 3000 minutes (i.e. 600 data samples) achieves the best performance. It is also interesting to observe that too small training interval (e.g., 150 minutes) brings even worse performance than using one training model from beginning to the end (i.e. 40000 minutes).

Third, we alter the number of reference images K and training rounds r in one training phase. The results show that having more reference images and training rounds only brings marginal performance gains.

Fourth, we look into the effect of different image organizations for the intra-node attribute. We organize one image using different combinations of row and column sizes. We observe that our diamond search algorithm is robust to different image

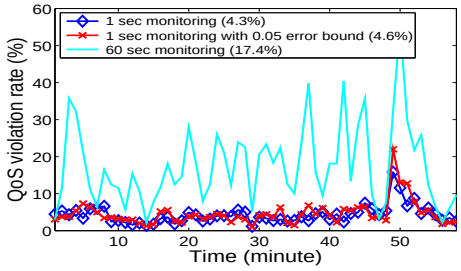


Fig. 13. A case for fine-grained monitoring.

Algorithm	Memory	Training	Compression
OLIC	43 MB	0.051±0.021 s	17.57±1.19 ms
Neighbor	38 MB	0.047±0.022 s	13.51±1.25 ms
Spatial	40 MB	400±190 s	12.89±1.43 ms

TABLE II
SYSTEM OVERHEAD COMPARISON FOR COMPRESSING AN INTER-NODE ATTRIBUTE ON 464 PLANETLAB HOSTS.

sizes since it has a flexible search pattern.

We conducted a case study to qualify the benefit of fine-grained monitoring and the impact of approximation error on the performance of QoS management . Figure 13 shows the QoS provisioning performance of a dynamic resource scaling system [6] under different monitoring granularities (i.e., 1 second v.s. 1 minute monitoring interval) for the RUBiS online auction benchmark application. The resource scaling system dynamically adjusts the resource allocation based on the predicted application resource demand. The resource demand predictor is trained using the sampled monitoring data. We can see that fine-grained monitoring (one second sampling interval) can reduce QoS violation rate to 4.3% from 17.4% achieved under a 1-minute sampling interval. We then introduce a 0.05 approximation error and re-run the scaling experiments. We can see that the QoS violation rate is almost unaffected under a small approximation error.

Finally, we evaluate the overhead of different compression algorithms for compressing the inter-node network delay attribute on 464 PlanetLab hosts in Table II. For OLIC and neighbor search algorithms, the training overhead includes the time of searching best reference blocks during one training phase. For the spatial approach, the training overhead includes the time of performing k-medoids clustering during one training phase. Since the training overhead of the temporal algorithm is negligible, we did not show its overhead. The compression time includes the time of performing compression for one system image by using the reference block information obtained during the training phase. We observe that both OLIC and neighbor search algorithm have significantly smaller training overhead than the spatial approach. The neighbor search algorithm is a bit faster than OLIC since its search range is even smaller. However, we have already shown that the compression performance of OLIC consistently outperforms the neighbor search algorithm for all datasets due to its broader search coverage. We observe that the memory consumption for all three approaches are around 40 MB. We also observe that OLIC can finish one round training within tens of milliseconds, which well meet our online compression goal.

V. RELATED WORK

Distributed systems and network monitoring have been extensively studied before. Previous work (e.g. Astrolabe [14], SDIMS [15], Mercury [27], SWORD [16]) has proposed to leverage hierarchical or decentralized architectures to achieve scalable distributed monitoring. Chen et al. proposed an algebraic approach that selectively monitors a subset of paths to fully predict the loss rate and latency of all paths in an overlay network [28] with the assumption that the underlying network topology is known. In contrast, our work focuses on monitoring an arbitrary distributed hosting infrastructure without any prior assumption about the monitored distributed system.

Exploring correlation patterns among distributed data sources have been studied under different context such as sensor network monitoring [29]–[31], distributed event tracking [32], and resource discovery [33]. Several previous work [9], [18], including some of our own [19], has proposed to leverage correlation patterns to reduce monitoring cost. In contrast, our work explores a new image-based online compression approach to reducing distributed monitoring overhead. Zhang et al. proposed to leverage spatial and temporal correlations to infer missing values from other received values in Internet traffic monitoring systems [34]. In comparison, our work addresses an orthogonal problem of reducing the collection cost of known values.

Compression techniques have been extensively studied in video streaming applications [35], [36]. Our work is inspired by the video compression technique that encodes large video data at the source, transmits the compressed video data for lower communication cost, and then decodes the compressed data at the receiver to restore the original data. However, our work needs to address a set of new challenges since our source data are distributed on different hosts that can experience transient or persistent failures from time to time. Offline data compression has been well studied. For example, VPC3 [37] is an offline trace compression algorithm for large log files, which utilizes value predictors to identify and amplify patterns in the log files so that compression/decompression can be achieved more effectively and faster. Flight data recorder [38] is an online system call tracing tool with system call compression support. In contrast, our work focuses on online compression of dynamic metric values.

VI. CONCLUSIONS

In this paper, we have presented OLIC, a novel image-based online information compression framework for monitoring large-scale hosting infrastructures. OLIC models snapshots of the monitored distributed system using a sequence of system images and apply light-weight online reference block search schemes to compress distributed monitoring data streams. OLIC performs online reference block training using a dual diamond reference block search algorithms inspired

by the video compression techniques. To the best of our knowledge, OLIC makes the first attempt to adopt an image-based approach to achieving efficient distributed monitoring traffic reduction. We have implemented the OLIC system and deployed it on the PlanetLab and NCSU virtual computing lab (VCL). We conducted extensive experiments using a range of real system monitoring data from PlanetLab, VCL, a Google cluster, and a real Internet traffic matrix dataset. Our prototype implementation indicates that OLIC is practical and efficient, which can achieve the best compression performance with much lower overhead compared to previous schemes.

VII. Acknowledgements

This work was sponsored in part by NSF CNS0915567 grant, NSF CNS0915861 grant, U.S. Army Research Office (ARO) under grant W911NF-10-1-0273, and IBM Faculty Award. Any opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the 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] Z. Gong and X. Gu, "PAC: Pattern-driven Application Consolidation for Efficient Cloud Computing," in *Proc. of MASCOTS*, 2010.
[6] Z. Gong, X. Gu, and J. Wilkes, "PRESS: PRedictive Elastic ReSource Scaling for Cloud Systems," in *Proc. of CNSM*, 2010.
[7] 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.
[8] X. Gu and H. Wang, "Online Anomaly Prediction for Robust Cluster Systems," in *Proc. of ICDE*, 2009.
[9] L. Huang and et al., "Communication-Efficient Online Detection of Network-Wide Anomalies," in *Proc. of IEEE INFOCOM*, 2007.
[10] "CoMon," http://comon.cs.princeton.edu/.
[11] "IBM Tivoli Monitoring software," http://www-01.ibm.com/software/tivoli/.
[12] "World Community Grid," *http://www.worldcommunitygrid.org*.
[13] "NCSU Virtual Computing Lab," http://vcl.ncsu.edu/.
[14] 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.
[15] P. Yalagandula and M. Dahlin, "A Scalable Distributed Information Management System," in *Proc. of SIGCOMM 2004*, Aug. 2004.
[16] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Design and Implementation Tradeoffs for Wide-Area Resource Discovery," in *Proc. of HPDC-14*, 2005.
[17] J. Liang, X. Gu, and K. Nahrstedt, "Self-Configuring Information Management for Large-Scale Service Overlays," in *Proc. of INFOCOM*, 2007.
[18] 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.
[19] 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.
[20] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology Into the Internet," in *Proc. of HotNets-I*, 2002.
[21] "Google Cluster Data," http://googleresearch.blogspot.com /2010/01/google-cluster-data.html.
[22] 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.
[23] 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.
[24] J. W. Mickens and B. D. Noble, "Exploiting Availability Prediction in Distributed Systems," in *Proc. of NSDI*, 2006.
[25] Y. Tan and X. Gu, "On Predictability of System Anomalies in Real World," in *Proc. of MASCOTS*, 2010.
[26] S. Theodoridis and K. Koutroumbas, *Pattern Recognition 3rd edition*, 2006.
[27] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," in *Proc. of SIGCOMM*, 2004.
[28] Y. Chen, D. Bindel, H. Song, and R. H. Katz, "An Algebraic Approach to Practical and Scalable Overlay Network Monitoring," in *Proc. of ACM SIGCOMM*, 2004.
[29] 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.
[30] 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.
[31] A. Deshpand, E. C. Guestrin, and S. R. Madden, "Model-driven data acquisition in sensor networks," in *Proc. of VLDB*, 2002.
[32] A. Jain, E. Y. Chang, and Y.-F. Wang, "Adaptive Stream Resource Management Using Kalman Filters," in *Proc. of SIGMOD*, 2004.
[33] M. Cardosa and A. Chandra, "Resource Bundles: Using Aggregation for Statistical Wide-Area Resource Discovery and Allocation," in *Proc. of ICDCS*, 2008.
[34] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, "Spatio-temporal compressive sensing and internet traffic matrices," in *Proc. of SIGCOMM*, 2009.
[35] T. Sikora, "Trends and Perspectives in Image and Video Coding," *Proceedings of IEEE*, vol. 93, no. 1, pp. 6–17, Jan. 2005.
[36] G. J. Sullivan and T. Wiegand, "Video Compression: From Concepts to the H.264/AVC Standard," *Proceedings of IEEE*, vol. 93, no. 1, pp. 18–31, Jan. 2005.
[37] M. Burtscher, "VPC3: A Fast and Effective Trace-Compression Algorithm," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 167–176, 2004.
[38] C. Verbowski, E. Kiciman, A. Kumar, B. Daniels, S. Lu, J. Lee, and Y. Wang, "Flight Data Recorder: Monitoring Persistent-State Interactions to Improve Systems Management," in *Proc. of OSDI*, Nov. 2006.