Supplemental Material: Resilient Self-Compressive Monitoring for Large-Scale Hosting Infrastructures

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

1 Detailed Results of Analytical Study

Without compression, a distributed monitoring system needs to configure all the monitoring agents to periodically report their collected attribute values to the management node. Let N denote the number of worker nodes in the distributed system, T denote the monitoring interval in seconds, and S denote the message size for reporting the attribute value in bytes. We can calculate the total size of these attribute updates without compression in bytes per second, U_{orig}, as follows:

$$U_{orig} = \frac{1}{T} \cdot N \cdot S \tag{2}$$

With compression, part of the attribute updates can be suppressed if their values can be inferred by their reference values within the error bound. We use U_s to denote the total size of these suppressed attribute update cost in bytes per second. We first analyze the temporal correlation scheme that uses the last value prediction to suppress the attribute updates.

Lemma 1. *Let p_1 denote the percentage of nodes whose attribute values can be inferred from its last value. The compression ratio achieved by the temporal correlation algorithm (CR_t) is p_1.*

Proof: Since p_1 denotes the percentage of nodes whose attribute values can be inferred from its last value, $(p_1 \cdot N)$ nodes do not need to send the attribute updates. Hence, we derive U_s, the total size of the suppressed attribute updates in bytes per second for the temporal correlation scheme as follows:

$$U_s = \frac{1}{T} \cdot p_1 \cdot N \cdot S$$

Thus, we can calculate CR_t as follows:

$$CR_t = \frac{U_s}{U_{orig}} = p_1$$

□

We then study the spatial correlation scheme that uses the clustering algorithm to group all the monitored nodes in the current system image into different clusters.

Lemma 2. *Assume all the N monitored nodes can be clustered into m groups. Let p_2 denote the percentage of nodes whose attribute values can be predicted by its cluster head. The compression ratio achieved by the spatial correlation algorithm (CR_s) is $p_2 \cdot \frac{N-m}{N}$.*

Proof: For m nodes (i.e., cluster heads), we need to send their unsuppressed attribute updates. For the other $(N-m)$ nodes (i.e., cluster members), since p_2 denotes the percentage of nodes whose attribute values can be predicted by their cluster heads, $p_2 \cdot (N-m)$ nodes do not need to send the attribute updates. Thus, we derive U_s, the total size of the suppressed attribute updates in bytes per second for the spatial correlation scheme as follows:

$$U_s = \frac{1}{T} \cdot p_2 \cdot (N-m) \cdot S$$

Hence, we can calculate CR_s as follows:

$$CR_s = \frac{U_s}{U_{orig}} = p_2 \cdot \frac{N-m}{N}$$

□

Now we consider the integrated approach that combines the temporal and spatial correlation schemes.

Lemma 3. *Let p_1 denote the percentage of nodes whose attribute values can be inferred based on the temporal correlation scheme. Let p_2' denote the percentage of cluster members whose attribute values cannot be inferred by the temporal correlation scheme but can be inferred by the cluster heads. The compression ratio achieved by the integrated approach (CR_{t+s}) is $p_1 + p_2' \frac{N-m}{N}$.*

Proof: For m nodes (i.e., cluster heads), $(p_1 \cdot m)$ of them do not need to send the attribute updates since their values can be inferred based on the temporal correlation scheme. For the other $(N-m)$ nodes (i.e., cluster members), p_1 of their values can be inferred by the temporal scheme and p_2' of their values can be

inferred by the cluster heads. Hence, there is totally $(p_1+p_2')\cdot(N-m)$ nodes do not need to send their attribute updates. We can derive U_s, the total size of the suppressed attribute updates in bytes per second for the integrated "temporal+spatial" correlation scheme as follows:

$$U_s = \frac{1}{T}(p_1 \cdot m + (p_1+p_2')\cdot(N-m))\cdot S$$

Thus, we can calculate CR_{t+s} as follows:

$$CR_{t+s} = \frac{U_s}{U_{orig}} = p_1 + p_2'\frac{N-m}{N}$$

□

Next, we analyze the neighbor search algorithm and RCM. We put their analysis together since they only differ in the reference block search patterns. For these two schemes, we always use the temporal correlation scheme first to check whether the current attribute value can be inferred by the last value. We also need to take the training overhead into account since no compression can be achieved during the training phase.

Lemma 4. *Let p_1 denote the percentage of nodes that can be inferred by the temporal correlation scheme, p_4 denote the percentage of nodes whose attribute values cannot be inferred by the temporal correlation scheme but can be inferred by its reference block value. Let I denote the number of reference images in one training round, R denote the number of training rounds, and L denote the number of images in one compression phase. The average compression ratio achieved by RCM (CR_{RCM}) is $\frac{L\cdot(p_1+p_4)}{I+R-1+L}$. When $I+R \ll L$, $CR'_{RCM} = p_1+p_4$.*

Proof: Let us consider a period that contains one training phase and one compression phase of RCM. The training phase has $(I+R-1)$ system images. During the training phase, we need to send complete and unsuppressed attribute updates for the optimal reference block search algorithm. The compression phase has L system images. For the N nodes in one image, p_1 of their values can be inferred by the temporal scheme and p_4 of their values can be inferred by the reference block values. The total number of nodes that do not need to send their attribute updates during the compression phase are $L\cdot(p_1+p_4)\cdot N$. Hence, we can derive U_s, the total size of the suppressed attribute updates in bytes per second for RCM as follows:

$$U_s = \frac{L\cdot(p_1+p_4)\cdot N\cdot S}{(I+R-1+L)\cdot T}$$

Thus, we can calculate CR_{RCM} as follows:

$$CR_{RCM} = \frac{U_s}{U_{orig}} = \frac{L\cdot(p_1+p_4)}{I+R-1+L}$$

When $I+R \ll L$, CR_{RCM} can be approximated as: $CR'_{RCM} = p_1+p_4$.

□

After we have analyzed different compression schemes individually, now we compare the compression capability of RCM with the most powerful correlation-based scheme: the integrated "temporal+spatial" correlation scheme.

Proposition 1. *The compression ratio of RCM is larger than the compression ratio of the integrated temporal+spatial correlation scheme (i.e., $CR_{RCM} > CR_{t+s}$) if each search path in the RCM's reference block search algorithm always covers the corresponding cluster head.*

Proof: Based on Lemma 3, we have $CR_{t+s} = p_1 + p_2'\frac{N-m}{N}$. Now we further analyze p_2'.

The spatial correlation scheme uses k-medoid clustering to form m clusters among N nodes in the current image. p_2' is defined as the percentage of cluster members whose attribute values cannot be inferred by the temporal correlation scheme but can be inferred by m cluster heads.

Let P_{b_I} denote the percentage of nodes whose attribute values can be inferred by the best reference block value obtained by RCM in the current system image. For each node, if the search path in the RCM's reference block search algorithm covers the cluster head for this node, RCM achieves no lower compression ratio than the spatial correlation scheme, that is, P_{b_I} is no smaller than p_2':

$$P_{b_I} \geq p_2' \tag{3}$$

In RCM, the compression phase is much larger than the training phase. According to Lemma 4, if $I+R \ll L$, the "zero compression" impact caused by the training phase will be negligible. Thus, CR_{RCM} can be approximated by p_1+p_4. Now we further analyze p_4.

We define P_{b_i} as the percentage of nodes whose attribute values can be inferred by the best reference block found by RCM in the i'th reference image. Since RCM picks the reference block that achieves the highest compression ratio among I best reference blocks as the final reference block, we can derive the following relation between p_4 and P_{b_i} $(i=1,2,...,I)$:

$$p_4 \geq max(P_{b_1}, P_{b_2}, ..., P_{b_I}) \tag{4}$$

By concatenating inequality (4) and (3), we can get $p_4 \geq p_2'$. Based on Lemma 3 and Lemma 4, we can derive the following relation:

$$CR'_{RCM} = p_1+p_4 \geq p_1+p_2' > p_1+p_2'\frac{N-m}{N} = CR_{t+s} \tag{5}$$

Hence, the compression ratio achieved by RCM is larger than the integrated temporal+spatial correlation scheme if every search path in the RCM's reference block search algorithm always covers the corresponding cluster head. The intuition behind this analytical result is that RCM always employs a broader search range (i.e., I recent system images)

than the temporal+spatial correlation scheme that only searches the current image.

□

Furthermore, we compare the compression capability of RCM with the neighbor search algorithm.

Proposition 2. *The compression ratio of RCM is larger than the compression ratio of the neighbor search scheme (i.e., $CR_{RCM} > CR_{neighbor}$).*

Proof: Let p_3 denote the percentage of nodes whose attribute values cannot be inferred by the temporal correlation scheme but can be inferred by its reference block value found by the neighbor search algorithm. Since the neighbor search algorithm uses the same online training process as RCM and only differs in the reference block search algorithm, we can leverage Lemma 4 to calculate $CR_{neighbor}$. Since I, R and L are all constant, $CR_{RCM}/CR_{neighbor} = (p_1+p_4)/(p_1+p_3)$. Now we only need to prove $p_4 > p_3$.

Let N_3 denote the number of nodes whose attribute values cannot be inferred by the temporal correlation scheme but can be inferred by its reference block value found by the neighbor search algorithm. Let N_4 denote the number of nodes whose attribute values cannot be inferred by the temporal correlation scheme but can be inferred by its reference block value found by RCM. Thus, we have $N_3 = N \cdot p_3$ and $N_4 = N \cdot p_4$. Intuitively, the probability of finding the best reference block for the current block can be increased by applying a broader search range to the reference images. Therefore, N_3 and N_4 are proportional to the number of candidate blocks examined by the neighbor search algorithm and RCM.

The neighbor search algorithm examines eight surrounding blocks in I reference images in each training window. Hence, N_3 follows the following proportional relation:

$$N_3 \propto I \cdot 8, \quad (I \geq 1) \tag{6}$$

RCM's reference block search algorithm checks eight neighboring blocks in each large diamond search and four neighboring blocks in the final small diamond search. Let l_i denote the number of large diamond searches for the i'th reference image ($l_i \geq 1$). Thus, N_4 follows the following proportional relation:

$$N_4 \propto \sum_{i=1}^{I}(8 \cdot l_i + 4), \quad (I \geq 1, l_i \geq 1) \tag{7}$$

Comparing Equation (6) and (7), N_4 is larger than N_3 even if there is only one large diamond search round (i.e., $l_i = 1$). Hence, RCM can achieve higher compression ratio than the neighbor search algorithm.

□

We now analyze the training cost of different compression algorithms. The training overhead of the neighbor search algorithm and RCM is mainly caused by the optimal reference block search algorithm. The neighbor search algorithm checks eight immediate surrounding blocks in each of the I reference images for each block in the training image. Thus, the training cost is $O(8N_bI)$ where N_b denotes the number of blocks in one system image. I is typically a small constant (e.g., $I = 3$ in our experiments). Hence, $O(8N_bI)$ can be simplified as $O(N_b)$.

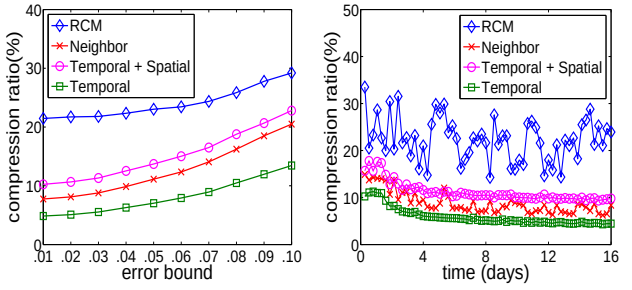
The RCM's reference block search algorithm checks eight neighboring blocks in each LDSP and four neighboring blocks in the final SDSP in each of the I reference images. In the worst case, this search path might cover all blocks in one system image. Thus, the training cost of RCM's reference block search algorithm in the worst case is $O(N_b^2)$. However, RCM's reference block search algorithm can have sub-linear training cost under the condition that the large diamond search terminates after only a few rounds. Such condition can be satisfied when the current block has strong temporal and spatial correlations with its neighboring blocks. Thus, the search algorithm can stop at one neighboring block that is not far away from the current block and thus avoid exploring more farther blocks. Furthermore, several parameters can also affect this condition. For example, by using a large block size b we can reduce the number of search steps. The search process may also terminate early under a loose error bound e_i.

The spatial correlation algorithm uses the k-medoids clustering to group N monitored nodes into different groups. Its training cost is $O(N^3)$ in the worst case [1], which is significantly higher than RCM's reference block search algorithm.

2 Details of the Trace Data Collection

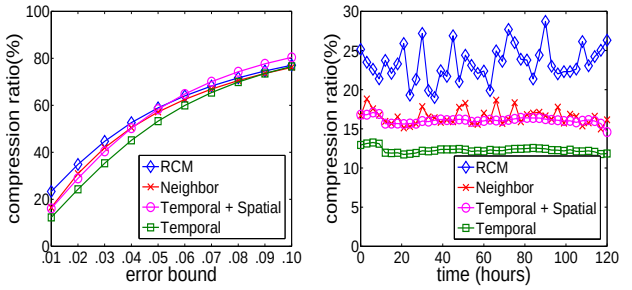
The VCL monitoring traces were collected by the production VCL system using the IBM Tivoli monitoring software [2]. The traces contain various performance attributes collected on 400 VCL nodes from Oct. 18th, 2010 to Nov. 3rd, 2010. The sampling interval is 5 minutes. In our experiments, we test our algorithms on the IP statistics attribute (datagrams/sec) and the windows NT processor attribute (DPC queued/sec).

The PlanetLab data traces were collected by our distributed monitoring system [3], [4] 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 [5] (e.g., CPU load, free memory, available CPU etc.) at a sampling interval of 10 seconds. Each monitoring agent also periodically pinged other nodes in the system to collect inter-node attributes such as network delay and bandwidth. We collected a set of intra-node attributes containing 400 nodes from Jan. 29th, 2009 to Feb. 3rd, 2009. We collected three sets of inter-node attributes containing more than 400 nodes from Sep. 21st, 2009 to Sep. 23rd,



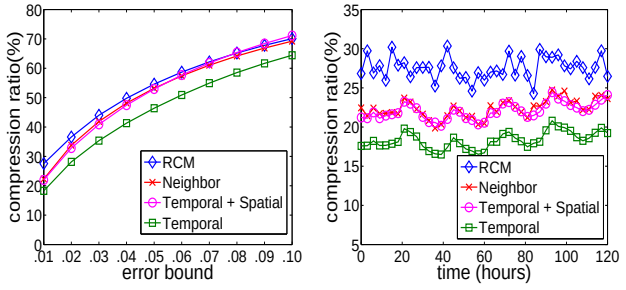
(a) overall (b) error bound = 0.01

Fig. 12. Compression ratio comparison for the VCL NT Processor trace.



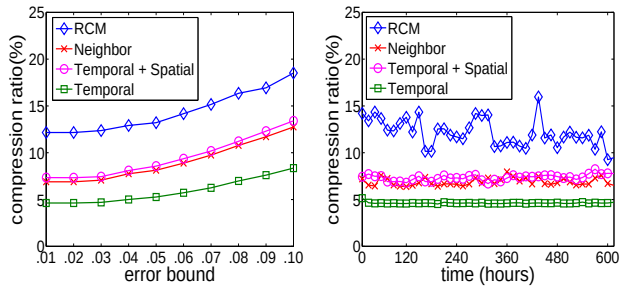
(a) overall (b) error bound = 0.01

Fig. 13. Compression ratio comparison for the PlanetLab CPU load trace.



(a) overall (b) error bound = 0.01

Fig. 14. Compression ratio comparison for the PlanetLab free memory trace.



(a) overall (b) error bound = 0.01

Fig. 15. Compression ratio comparison for the Google cluster CPU trace.

2009, from Oct. 4th, 2009 to Oct. 6th, 2009 and from Nov. 5th, 2009 to Nov. 9th, 2009, respectively.

We got a small sample of real application workload trace data collected on a Google cluster [6]. The dataset contains normalized CPU and memory usage attributes for more than 30,000 different jobs, with a sampling interval of 5 minutes. We selected a subset of these jobs (i.e., 1296 jobs) that have larger variations in the raw data. We also assume that those jobs are executed on different hosts.

To test with inter-node attributes, we also used real Internet traffic matrices collected by previous research work from a transit network [7]. In this dataset, one traffic matrix was computed per 15 minutes for a period of four months.

3 Additional Experimental Results

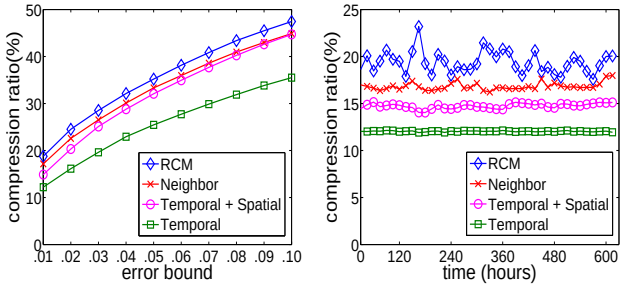
3.1 Compression Comparison Without Host Failures

Fig. 12 shows the compression results of the NT processor attribute (DPC Queued/sec). The results are consistent with the results of the IP statistics trace described in Fig. 9 of the main paper.

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. Fig. 13 shows the average compression results and the continuously sampled compression ratio values under the tight error bound (0.01) for the PlanetLab CPU load attribute trace. Fig. 14 shows the average compression results and the continuously sampled compression ratio results for the PlanetLab free memory attribute trace. Different from the VCL datasets, the PlanetLab monitoring data are more stable with smaller CV. Thus, we can see that the PlanetLab monitoring data are much easier to compress with up to 70%-80% compression ratios. This is also the reason why the performance of all compression approaches are close. However, RCM and the temporal+spatial correlation algorithms still perform slightly better than the temporal correlation algorithm. We will show later that RCM has much lower overhead than the temporal+spatial correlation algorithm. Furthermore, RCM remains the best under tight error bounds (e.g., 0.01).

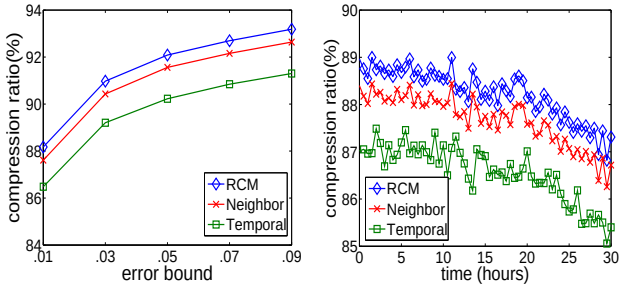
Next, we present the compression results for the memory and CPU usage datasets collected on a Google cluster. Google normalizes the original data using some secrete linear function for privacy protection. However, the normalized data preserve the statistical changing pattern of the original data. Fig. 15 shows the compression results for the CPU trace. Fig. 16 shows the compression results for the memory trace. Again, we observe that RCM consistently outperforms all the other algorithms.

Now we present the compression results for a PlanetLab inter-node delay dataset in Fig. 17. The size of this dataset (92GB) is much bigger than the



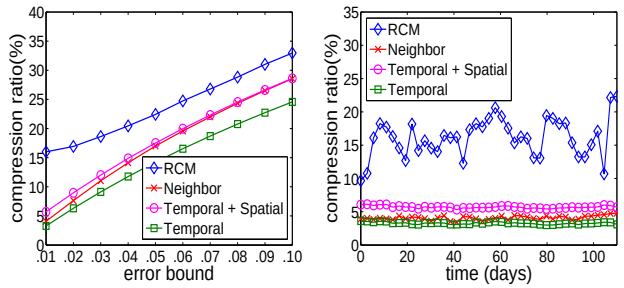
(a) overall (b) error bound = 0.01

Fig. 16. Compression ratio comparison for the Google cluster memory trace.



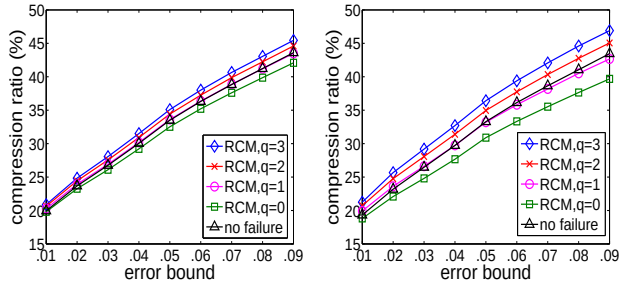
(a) overall (b) error bound = 0.01

Fig. 17. Compression ratio comparison for the PlanetLab inter-node delay trace.



(a) overall (b) error bound = 0.01

Fig. 18. Compression ratio comparison for the traffic matrices trace.



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

Fig. 19. Compression ratio comparison for the Google cluster memory trace under host failures.

other datasets. We observe that this dataset has the lowest CV and thus is easy to compress. RCM can achieve 88% compression ratio under a tight 0.01 error bound. We do not use the temporal+spatial correlation approach for this dataset since it is impractical to apply this approach due to its prohibitive overhead. In our experiment, the spatial correlation algorithm took six minutes while RCM only spent around 500 milliseconds for one training round, which is more than two orders of magnitude faster.

Finally, we present the compression results for another inter-node attribute trace, the Internet traffic matrices. The results are shown in Fig. 18. Again, we observe that the RCM consistently outperforms the other algorithms over different error bound settings. In Fig. 18(b), the fluctuating compression ratio curve of RCM indicates that RCM can dynamically discover the best reference blocks to maintain high compression ratios.

3.2 Compression Comparison With Host Failures

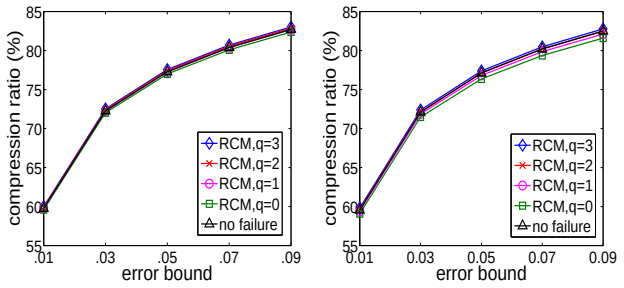
Fig. 19 shows the compression comparison results for the Google cluster memory usage trace with 10% and 30% host failures. We have similar observations to the Google cluster CPU usage trace described in Fig. 10 of the main paper.

We show the compression results for the first PlanetLab inter-node delay dataset in Fig. 20. Compared to the results of two intra-node Google cluster traces, we observe that the compression ratio improvement made by the backup reference blocks are less significant. The reason is that this data trace has less variations. As a result, attribute values are compressed by the last value prediction approach most of time. Our experiment statistics show that approximately 90% of the compression ratio is contributed by the last value prediction approach on average. Hence, the host failures have little impact to the compression ratio and the backup reference blocks are rarely used to replace any failed reference block.

3.3 Sensitivity Studies

We conduct sensitivity experiments to study the impact of different parameters (e.g., block size, training interval, the number of reference images, the number of training rounds) on the compression performance of RCM and the neighbor search algorithm. Table 3 shows the summarized results of all the intra-node attributes datasets. Table 4 shows the summarized results of all the inter-node attributes datasets. We report the average compression ratio results for all the datasets we tested. The optimal setting of each parameter setting is highlighted in bold.

First, block size b 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, with coarser granularity, it will lead to less accurate block matching. In contrast,



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

Fig. 20. Compression ratio comparison for the PlanetLab inter-node delay trace under host failures.

TABLE 3
Sensitivity study summarization of all the intra-node attribute datasets.

Error bound	0.01		0.05	
Algorithms	Neighbor	RCM	Neighbor	RCM
b=2	11.14	17.74	24.18	31.33
b=4	**12.97**	**20.28**	**28.30**	**34.02**
b=6	12.46	18.84	27.12	32.67
30 images	9.66	15.50	21.25	25.56
100 images	12.27	19.74	26.89	32.39
300 images	**13.03**	**20.15**	**28.36**	**34.08**
3000 images	9.38	15.11	24.53	29.65
I=5, R=5	**12.87**	**20.29**	**28.06**	**33.85**
I=3, R=3	12.61	19.60	27.75	33.57
I=1, R=1	12.35	18.88	27.41	33.23
$M = K$	**12.87**	**20.07**	**28.06**	**33.79**
$M < K$	12.35	19.74	27.17	33.10
$M \ll K$	11.34	19.73	26.42	33.08

a smaller block size performs fine-grained search but has limited search range. It is also less robust to noises in the data (e.g., some transient similarities). We observe that RCM achieves the best compression performance when b is equal to 4.

Second, we evaluate the impact of training interval. We find that the length of training interval cannot be either too small or too large. On the one hand, a too small training interval exacerbates the "zero-compression"impact caused by the training phase. On the other hand, a too large training interval degrades the reference block effectiveness. We observe that a moderate length of training interval (i.e., 300 images) achieves the best compression performance.

Third, we test different combinations of the number of reference images I and the number of training rounds R in one training phase. Our results show that having more reference images and training rounds only brings marginal compression performance gains.

Fourth, we study the effect of different system image organizations for all the intra-node attributes. For a system image consisting of N nodes, we organize the image using different combinations of M and K ($M \cdot K = N$). We observe that the neighbor search algorithm achieves the highest compression ratio for square-shape images. In contrast, RCM is not sensitive to different image shapes since it has a more flexible search pattern than the neighbor search algorithm.

TABLE 4
Sensitivity study summarization of all the inter-node attribute datasets.

Error bound	0.01		0.05	
Algorithms	Neighbor	RCM	Neighbor	RCM
b=2	39.42	45.49	46.08	52.98
b=4	**45.99**	**52.12**	**53.95**	**57.60**
b=6	44.11	48.33	51.70	55.29
30 images	34.03	40.09	40.46	43.20
100 images	43.23	51.12	51.14	54.72
300 images	**45.99**	**52.12**	**53.95**	**57.60**
3000 images	33.11	39.09	46.64	50.11
I=5, R=5	**45.99**	**52.12**	**53.95**	**57.60**
I=3, R=3	45.05	50.29	53.38	57.02
I=1, R=1	44.11	48.46	52.80	56.44

TABLE 5
System overhead comparison for compressing an intra-node Google cluster CPU usage trace containing 1296 hosts.

Algorithm	Memory	Training	Compression
RCM (q=3)	8 MB	32±2.7 ms	86±12.5 μs
RCM (q=2)	8 MB	32±2.0 ms	77±4.1 μs
RCM (q=1)	8 MB	31±2.6 ms	72±16.2 μs
RCM (q=0)	8 MB	26±3.3 ms	63±7.0 μs
Neighbor	8 MB	14±1.0 ms	57±0.5 μs
Spatial	12 MB	198±17.0 ms	64±2.0 μs

3.4 Overhead Measurements

Our overhead measurements were collected by running the trace-driven compression experiments on a desktop machine with Intel Core Duo CPU 2.4 GHz and 4GB RAM. Table 5 shows the results for compressing an intra-node Google cluster CPU usage trace that contains 1296 hosts. Table 6 shows the results for compressing a PlanetLab inter-node delay trace that contains 464 hosts. These two traces are the largest intra-node and inter-node attribute data traces that we have collected. For RCM and the neighbor search schemes, the training overhead includes the time of searching the best reference blocks during one training phase. For the spatial scheme, the training overhead includes the time of performing k-medoids clustering during one training phase. For all the schemes, the compression overhead includes the time of performing compression for one system image. We have several observations: 1) Both RCM and the neighbor search scheme have much lower training overhead than the spatial scheme. Specifically, RCM is several orders of magnitude faster than the spatial scheme for the PlanetLab inter-node monitoring

TABLE 6
System overhead comparison for compressing a PlanetLab inter-node delay trace containing 464 hosts.

Algorithm	Memory	Training	Compression
RCM (q=3)	34 MB	0.530±0.027 s	8.8±0.2 ms
RCM (q=2)	34 MB	0.527±0.013 s	8.3±0.4 ms
RCM (q=1)	34 MB	0.525±0.012 s	7.7±0.3 ms
RCM (q=0)	34 MB	0.510±0.019 s	7.2±0.1 ms
Neighbor	26 MB	0.474±0.019 s	6.7±0.5 ms
Spatial	40 MB	300±17.0 s	6.8±0.6 ms

data that collects about 212K samples per ten seconds. Thus, RCM is more scalable than the previous correlation-based schemes; 2) RCM incurs slightly more overhead than the neighbor search algorithm since it searches more candidate reference blocks; and 3) The backup reference blocks (i.e., when $q > 0$) only introduce little extra training and compression overhead. We also observe that the memory consumption for all the schemes are small. We can see that RCM is light-weight, which makes it suitable for performing online monitoring data compression for large-scale hosting infrastructures.

The overhead of RCM is linear to the number of the attributes being tracked. For example, if we monitor 600 *intra-node* attributes, the training phase will take about 18 seconds while the compression will take about 42 milliseconds. If we monitor 10 *inter-node* attributes, the training phase will cost about 5 seconds while the compression will spend about 70 milliseconds. Note that those computation time results are measured using our currently unoptimized implementation of RCM and a low-end desktop PC. We can further accelerate the training time and compression time by optimizing the implementation of RCM and using more powerful server machines.

4 Additional Related Work

Distributed system monitoring have been extensively studied before. A major challenge of building large-scale distributed monitoring systems is the scalability concern. Some approaches employ specific monitoring structure to achieve scalable distributed monitoring, such as the peer-to-peer structure used by Astrolabe [8] and the hierarchical structure used by SDIMS [9], Mercury [10] and Ganglia [11]. Other approaches trade information coverage or precision for lower monitoring cost. InfoEye [3] achieves minimum monitoring overhead at the cost of losing information coverage since only a subset of monitored attributes that satisfy the latest statistical conditions will be pushed to the management node. STAR [12] proposes a self-tuning algorithm that adaptively sets precision constraints to bound the numerical error in query results. In contrast, we solve the scalability problem by applying online data compression techniques on live monitoring streams to reduce the monitoring cost. RCM does not enforce any specific structure to the monitored infrastructure, which makes it easier to be deployed in different hosting infrastructures. RCM can achieve fine-grained and complete monitoring with only negligible information precision loss within a pre-defined error bound.

Compression techniques have been extensively studied in video coding applications [13]. RCM 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 reconstruct the original data. However, our work addresses a set of new challenges since our source data are distributed on different hosts that can experience transient or persistent failures. Offline data compression has also been well studied. For example, LZW and Deflate algorithm are widely used by data compression tools such as gzip. VPC3 [14] is an offline trace compression algorithm for large log files. Different from those offline compression schemes that can only be applied after the data have been reported to the management node, RCM performs online compression over live monitoring data streams during monitoring runtime. Thus, RCM can reduce the end-system resource and network bandwidth consumption on both worker nodes and management nodes. Flight data recorder [15] is an online system call tracing tool with system call compression support. In contrast, our work focuses on the online compression of dynamic system attributes.

References

[1] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[2] "IBM Tivoli Monitoring software," http://www-01.ibm.com/software/tivoli/.

[3] J. Liang, X. Gu, and K. Nahrstedt, "Self-Configuring Information Management for Large-Scale Service Overlays," in *Proc. of INFOCOM*, 2007.

[4] 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.

[5] "CoMon," http://comon.cs.princeton.edu/.

[6] "Google Cluster Data," http://googleresearch.blogspot.com/2010/01/google-cluster-data.html.

[7] 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.

[8] 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.

[9] P. Yalagandula and M. Dahlin, "A Scalable Distributed Information Management System," in *Proc. of SIGCOMM 2004*, Aug. 2004.

[10] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," in *Proc. of SIGCOMM*, 2004.

[11] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817 – 840, 2004.

[12] 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, pp. 962–973.
[13] T. Sikora, "Trends and Perspectives in Image and Video Coding," *Proceedings of IEEE*, vol. 93, no. 1, pp. 6–17, Jan. 2005.
[14] M. Burtscher, "VPC3: A Fast and Effective Trace-Compression Algorithm," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 167–176, 2004.
[15] 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.