ABSTRACT

VENKATESH, VINAY. A Video Coding Approach to Compressive Distributed System Monitoring. (Under the direction of Dr. Xiaohui (Helen) Gu.)

Large scale distributed systems have become important platforms for many real world production systems. Automatic and continuous management of such distributed computing infrastructures is challenging since the management solution has to be both scalable and efficient. This research presents a new distributed system monitoring framework based on video coding techniques. Our approach models the distributed system metrics as an image and uses both intra-image and inter-image encoding to compress the monitoring traffic. This research explores the design, implementation, and evaluation of two such video coding algorithms: local neighbor search and diamond search. We have implemented the prototype of the monitoring system and tested it on the Planetlab and and data from real Internet traffic matrices. The experimental results show that our approach can achieve more than 80% compression ratio. Compared to previous compression schemes, our scheme can improve the compression ratio by up to 50% and reduce the overhead by 80%.

A Video Coding Approach to Compressive Distributed System Monitoring

by
Vinay Venkatesh

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2010

APPROVED BY:

Dr. Xiaosong Ma

Dr. Rudra Dutta

Dr. Xiaohui (Helen) Gu
Chair of Advisory Committee

DEDICATION

To my Family.

BIOGRAPHY

Vinay was born in Bangalore, India. He received his Bachelor of Engineering 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. With the defense of this thesis, he will receive the Master of Science in Computer Science from North Carolina State University in December 2010.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Xiaohui (Helen) Gu for her valuable guidance and support to conduct productive research . I thank Dr. Xiaosong Ma and Dr. Rudra Dutta for their consent to serve in my thesis committee. I would like to extend my thanks to the DANCE research group members, especially to Yongmin Tan and Zhenhuan Gong for their advice and help during my research.

I also thank the Department of Computer Science and NCSU libraries for providing the necessary facilities. My family has been an inspiration throughout my life and I extend my deepest gratitude to them for their unconditional support. Finally, I have made friends along the way. They have helped me, one way or the other, in my effort to complete this thesis. Many thanks to Karthik Srinivas Murthy, Ashwin Bindingnavile Nanjaraju and Marilyn Chris Fernandes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The last few decades have seen computing becomes a part of our day to day life. Most of the applications today follow a distributed computing paradigm in which the user accesses resources spread across different locations. Low cost computers and high bandwidth networks are enabling this paradigm to become a common practice. The distributed resources may be located within a few feet or may be present half way across the world. Distributed computing infrastructures such as PlanetLab [4] and world community Grid [5] are serving as large-scale resource pools for scientific research. PlanetLab has dedicated nodes spread across the world under different autonomous administrations which co-operate together to merge their computing resources. The world community grid depends upon individuals collectively contributing their unused computer cycles. Efficiently managing such large-scale distributed systems has become a challenging problem. In particular, monitoring and querying the state of the distributed system form the key components of the distributed information management service which is one of the fundamental building blocks for automatic distributed system management [19].

Distributed information management service must be capable of tracking system information such as CPU, memory, disk usage and bandwidth dynamically. Applications or system processes need to know not only the various attribute values on a per node basis but also the link information between different nodes. For example, distributed stream processing applications

such as Spade [8] need to be aware of the network link information between these distributed hosts for optimal performance. Two important factors, namely scalability and accuracy, need to be addressed while designing the large-scale distributed information management service. On one hand, quality-of-service (QoS) sensitive applications require accurate up-to-date distributed system information. On the other hand, a large-scale distributed system can include tens of thousands of geographically distributed nodes. The actual monitoring traffic for such a system would be in the range of 6.1Mb/sec at a sampling rate of ten seconds and 400 metrics collected per node. The challenge is to provide scalability in the framework and at the same time maximize the accuracy of the collected information.

Previous distributed monitoring systems [11, 18, 3] have proposed different solutions for large scale information management. Most are centralized where all the collected data are aggregated and analyzed in one node whereas few others propose decentralized hierarchical systems [14] to distribute the monitoring workload. Some approaches[10, 9] also provide configurable schemes that can tradeoff information coverage and precision for lower monitoring cost. Infotrack [21] explores spatial and temporal correlation using clustering (k-means) and metric value prediction techniques (Kalman filter) to achieve compression in the monitored traffic. However, clustering algorithm has a large computational overhead and is not suitable for monitoring massive scale metrics such as inter-node attributes.

In this thesis, we present the design and implementation of a video coding based compressive monitoring system for large-scale distributed computing infrastructures. The video coding technique utilized both temporal and spatial correlations to find the best reference block within the recent snapshots preceding the currently processed snapshot. A block in the video coding domain is a group of adjacent pixels. To apply such a coding technique for compressive distributed system monitoring, we need to map the system monitoring domain and the video coding domain. This is done by modeling the continuous snapshots of the monitored distributed system as a sequence of system image frames.

Video coding techniques are then applied to find the best reference block for each data

block in the system image to achieve compressive monitoring. The monitored traffic can be compressed by sending the difference between the current block and the reference block. We introduce two reference block search algorithms: i) *the neighbor search algorithm* that examines up to eight neighboring blocks and ii) *the diamond search algorithm* that has a extended search range beyond just the neighbor blocks. The node layouts in the system snapshots are organized in such a way that nodes from the same domain are adjacent to each other. This approach helps in maintaining the neighboring pixels similarity feature in a video frame.

The optimal reference block tends to change over time. In order to maintain a better compression ratio, we periodically perform re-training to update the reference blocks for all blocks. This best reference block is used until the next re-training phase which can be triggered either periodically or when the overall system monitoring data compression ratio falls below a certain threshold. The neighbor and diamond search algorithms have a lower complexity. This is reflected by the considerable reduction in training time when compared with clustering algorithm.

We have implemented the prototype of the compressive distributed monitoring system and tested on the planet-lab [4]. We have utilized around 300 planetlab nodes and have collected data for 66 per-node metrics and inter-node attributes such as network delay and bandwidth. Experimental results show that the coding based approach is efficient and light weight. The video coding approach used in combination with last value based approach increases the overall compression ratio by upto 50% in comparison with bare last value approach. We also show the coding based approach has much less over head in terms of online training time and are two orders of magnitude lower complexity in comparison to clustering based algorithms.

The rest of the thesis is organized as follows. Chapter 2 gives an overview of the system model and the approach. Chapter 3 describes the design details of the coding based compressive distributed monitoring system. Chapter 4 presents the prototype implementation and experimental results. Chapter 5 compares our work with other related work.

Chapter 2

System Overview

In this Chapter we shall provide the background on the information management model used in our approach. Subsequently we shall also provide the problem formulation followed by the overview of our approach. The various notations used in this thesis are summarized in the Table 2.1.

Table 2.1: Notations.

Notation	**Description**
N	total number of monitored nodes
v_i	monitored node
A	set of all intra-node attributes
$a_{i,k}^t$	intra-node attribute a_k collected on node v_i at time t
$d_{i,j}^t$	inter-node attribute between v_i and v_j at time t
S_i	size of attribute a_i
T	sampling interval
CR	compression ratio
e_i	error bound for attribute i
m	total number of blocks in one frame
n	the dimension of one block

2.1 System Model

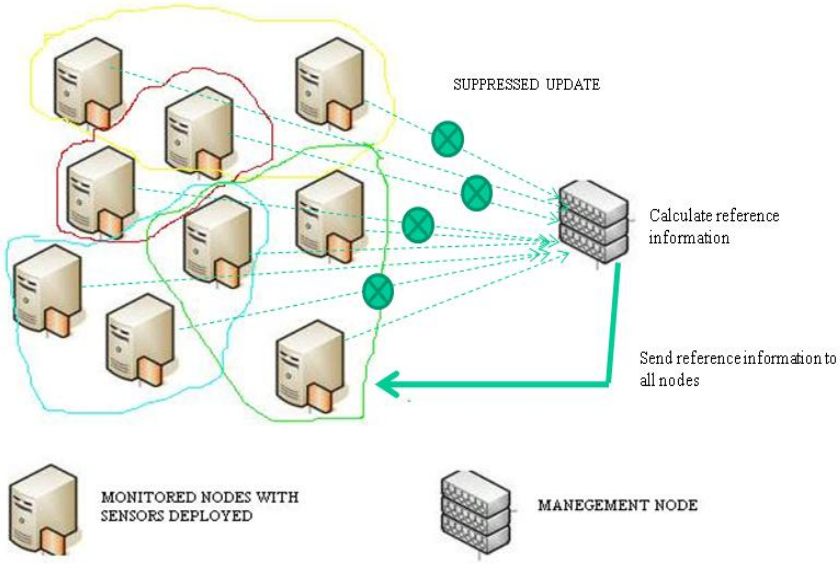

Figure 2.1: System Architecture.

The system model which we have considered and depicted in figure 2.1 is a collection of nodes $\{v_1, \ldots v_N\}$ which are geographically distributed. On each of these nodes, we install a monitoring sensor. This sensor is capable of monitoring two types of system metrics. The first group is the intra-node attributes which includes the metrics related the monitored system (e.g. Free Memory, Available CPU etc). For a particular node vi, intra-node attributes are denoted by $A_i = \{a_{i,1}, \ldots, a_{i,|A_i|}\}$. The second group is the inter-node attributes which consists of metrics dealing with a combination of nodes (e.g. Latency, Bandwidth etc). An inter-node attribute for a pair of nodes v_i and v_j, is defined as $d_{i,j}$.

The deployed sensor on each node periodically samples an attribute $a_{i,k}$ 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 instance t. The sensor also periodically samples the inter-node attributes 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. The sensor continuously streams the metric values to the management node. This information aggregated on the management node can be utilized for various services which includes getting an insight of the monitored system and also to identify long term metric patterns. The management node is also capable of controlling the behavior (changing the update interval, start/stop monitoring) of the deployed sensors.

2.2 Problem formulation

Any information management system must be able to provide scalable and continuous monitoring of nodes. The system must have low overhead and acceptable information precision. We employ two such correlation techniques to reduce the monitoring costs. Spatial correlation is exhibited by an attribute if the value of that attribute can be inferred by the same attribute of a different node. If the inferred value is within a certain error bound, the sensor needs not send the value of that attribute and the management node can infer the value from the other node. Temporal correlation is exhibited by an attribute a_k of a node v_i if its value at time t can be inferred using the same attribute value of the same node at previous time. If the attribute value $a^t_{i,k}$ can be inferred within the defined error bound, the monitoring sensor can compress the traffic by discarding $a^t_{i,k}$ since the management node can infer the attribute value by itself. Spatial and temporal correlation for an attribute can be combined together to further reduce monitoring costs. An attribute value of a node at time t can be inferred using the sampled value of itself at some previous time or the sampled value on other nodes at the same time or some previous time instance. Large scale distributed systems performing similar jobs tend to exhibit temporal and spatial correlation. However, it is challenging to discover real time temporal spatial correlations but yet still have low overhead.

To infer such temporal and spatial correlations, we can use certain techniques of video coding [15, 16]. Any video is basically a sequence of video frames and each video frame is made up of image blocks. A video codec is made up of encoder and decoder. The encoder encodes each block in the intra-mode or in the inter-mode. Intra-mode exploits spatial correlations in the same

$$
\begin{bmatrix} 0 & \dots & \dots & d^t_{1,N} & \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ d^t_{i,1} & \dots & \dots & d^t_{i,N} & \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ d^t_{N,1} & \dots & \dots & 0 & \end{bmatrix}
\begin{bmatrix} 0 & \dots & \dots & d^{t+1}_{1,N} \\ \vdots & \ddots & \ddots & \vdots \\ d^{t+1}_{i,1} & \dots & \dots & d^{t+1}_{i,N} \\ \vdots & \ddots & \ddots & \vdots \\ d^{t+1}_{N,1} & \dots & \dots & 0 \end{bmatrix}
$$

time t time $t+1$ ...

Figure 2.2: Inter-node attributes organization for continuous system snapshots.

video frame and the prediction signal P is estimated using position specific linear combinations of previously encoded samples from adjacent blocks. Inter-mode exploits temporal correlations and the prediction signal P in this case is formed my motion compensated prediction. The encoder applies motion estimation to search for the best spatial displacement motion vectors from one or more reference frames which are previously encoded. The residual of the intra or inter prediction, which is the difference between the original block and its prediction P is transformed by a frequency transform.The transform coefficients are then scaled, quantized, entropy coded, and transmitted together with the prediction side information. The overall compression ratio is a joint contribution of several techniques. At decoder side, the compressed bit stream is reconstructed by reversing the operations conducted at encoder side.

2.3 Approach Overview

In-order to apply video coding techniques to the compressive system monitoring problem, we need to map both the domains and this necessitates the need to formulate a pattern for the data layout. The inter-node attribute are laid out in the form of a matrix as shown in figure 2.2. The rows and the columns represent different nodes. The element at ith row and jth column represents the value of the inter-node attribute (e.g. Latency) between node i and j. The principle diagonal of the matrix has all zero values since inter-node attributes does not

$$
\begin{array}{cc}
\begin{bmatrix} a^t_{1,1} & \cdots & a^t_{1,k} & \cdots & a^t_{1,A} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a^t_{i,1} & \cdots & a^t_{i,k} & \cdots & a^t_{i,A} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a^t_{N,1} & \cdots & a^t_{N,k} & \cdots & a^t_{N,A} \end{bmatrix} & \begin{bmatrix} a^{t+1}_{1,1} & \cdots & \cdots & a^{t+1}_{1,A} \\ \vdots & \ddots & \ddots & \vdots \\ a^{t+1}_{i,1} & \cdots & \cdots & a^{t+1}_{i,A} \\ \vdots & \ddots & \ddots & \vdots \\ a^{t+1}_{N,1} & \cdots & \cdots & a^{t+1}_{N,A} \end{bmatrix} \\
\text{time } t & \text{time } t+1 \quad \ldots
\end{array}
$$

Figure 2.3: Intra-node attributes organization for continuous system snapshots.

exist from a node to itself. To mimic the grouping of pixels to form macro blocks in the video coding techniques, we group $n \times n$ elements into a block as the basic processing unit where the value of n can varied. Nodes from the same domain are grouped together and are laid out next to each other since this kind of arrangement results in stronger correlation between neighboring blocks and helps to boost the performance.

For the intra-node attributes, the data layout is shown in figure 2.3. Each element in the matrix represents the attribute of an individual node. The node from the same domain again are placed adjacent to each other as this results in correlated attributes coming close to each other and helps in boosting performance.

Motion search techniques as employed in video coding are used to exploit the temporal and spatial correlation in the data frame by searching the best reference block for each block. If the difference between the current value and the reference value is within the user defined error bound, the sensor need not send the value to the management node and the management node can infer value from the reference value. The best reference block is found during the training phase and used to compress the following frames.

Video coding and compressive system monitoring are entirely different subjects and there arises a few problems when trying to map both the domains. First each block in our data frame denotes a numerical value of an inter-node or an intra-node attribute. It is equivalent to an individual pixel of an image. Since the video coding techniques are applied to a group of

pixels, we group $n \times n$ attributes into blocks which act as the basic processing units. Secondly, neighboring blocks in a video frame have more or less similar video contents and exhibit strong correlations. Most of the search range of motion estimation techniques is reasonably small since the object movement among the consecutive video frames is unlikely to be very quick. As stated before, in order to mimic this adjacent block correlation, we group nodes from the same domain into our data frame. Thirdly the video coding deals with centralized image data and all the compression algorithms are done at the encoder in an offline manner. In contrast the compressive system monitoring needs to handle decentralized monitoring data and this requires the monitoring sensors and the management node to perform online compression together. The performance of the compressive monitoring system is evaluated by a compression ratio (CR) metric that is calculated as follows,

$$CR = \frac{N_{matched}}{N_{total}} \tag{2.1}$$

Where $N_{matched}$ is the number of attribute values in current block which can be inferred from the reference values within a predefined error bound. N_{total} is the total number of attributes in one block. The compression ratio is averaged over all the blocks in the current frame to get the total compression ratio for a frame. Since node failures are common in most of the distributed systems [12], the CR calculation must be able to handle such scenarios. A NULL value is filled in to indicate failure if a node is not able to get response from another node. To calculate average compression ratio, we should not count these NULL values, so Eq. (2.1) becomes Eq. (2.2).

$$CR = \frac{N_{valid-matched}}{N_{valid-total}} \tag{2.2}$$

where $N_{valid-matched}$ is the number of matched valid attribute values. $N_{valid-total}$ is the number of total valid attribute values in one block.

Chapter 3

System Design

In this section we present the design details of the compressive monitoring system. We will first present the distributed compressive monitoring scheme followed by two reference block search algorithms used by the compressive monitoring system.

3.1 Compressive Distributed System Monitoring

Our current compressive system monitoring is a hybrid of Last value based approach and the coding approach. The sensor decides for each attribute value which approach is used and informs the management node regarding the same. Compressive system monitoring on the management node runs in two phases. One is the training phase and the other is the testing phase. During the training phase, the management node loads initial K frames as the reference frames. This is done by asking the sensor nodes to report all the attribute values without compression. All the frames are partitioned into $n \times n$ blocks where n is a tunable parameter. For each block, one of the specified block search algorithms is run to find the best reference block. Once the best reference block is chosen for particular block, the reference block location and the index of the reference frame are stored. This is similar to the motion estimation techniques of video coding where the system searches for the best reference block in the previous system snapshots for each block in the current system snapshot. The management

Algorithm 1: Management node Algorithm.

```
Input: ∑ A sequence of data frames to be compressed
Input: M_i: data frame at time interval i
Input: n: the size of each block in a frame
Input: K: the number of reference frames
Input: e: error bound
Input: T_p: Training period size
Input: timeout: Time at which to retrigger training
partition each M_i in ∑ into blocks of n × n;
load K frames in ∑ as reference frames into a sliding window;
load frames from (K + 1)...(K + T_p);
for each frame in set (K + 1)...(K + T_p) do
    for each block b_i do
        for each reference frame do
            do either neighbor search or diamond search;
        end
    end
    Update best block information for each block;
    Shift current frame into sliding window of reference frames discarding the first frame;
end
For each block choose the best block which repeatedly gives best compression ratio;
Send reference information to the sensor nodes;
for each successive frame do
    for each attribute value do
        Compare with reference last value or coding reference value;
        Save values which cannot be compressed;
    end
    if(timeout) retrigger training;
end
```

node sends the reference block information to the appropriate sensor nodes. The pseudo-code of the management node operations are shown in Algorithm 1. During the testing phase, the management node compares each attribute value with the coding based reference value utilizing the reference block information found during the training phase or with the reference last value depending on which approach the sensor node is using and calculates the residual value. The management node only needs to store the reference block indices and residual values. As time progresses, the correlations found tend to become weaker and the overall compression ratio

Algorithm 2: Sensor node Algorithm.

Input: $a_{i,t}$: the observed value of attribute value a_i at time t
Input: a_i^l: the reference last value of attribute value a_i
Input: a_i^r: the reference coding value of attribute value a_i
Input: e: error bound

```
Receive coding reference information from the management node;
if |a_{i,t} − a_i^l|/a_{i,t} < e then
    Don't send update ;
else
    Switch to coding method. Inform management node;
    if |a_{i,t} − a_i^r|/a_{i,t} < e then
        Don't send update ;
    else
        Push the value of a_{i,t} to the management node;
```

of the system might reduce. To overcome this scenario, we periodically retrigger training to update the reference information.

On the sensor node side of the operations, each sensor receives the reference block information from the management node and uses this information to compress monitoring traffic. By default, the sensor utilizes last value based approach. If this approach fails, then the sensor immediately switches to the coding based approach utilizing the reference block information sent by the management node. This switch between the approaches is informed to the management node by a bitset. The sensor will push the values to the management node if the current value cannot be inferred either by the last value based approach or coding approach. The pseudo-code of the sensor side operations is depicted in Algorithm 2.

3.2 Neighbour Search Algorithm

The neighbor search algorithm considers up to eight neighboring blocks (upleft, upright, left, right, bottom left, bottom right, up ,down) and the co-located block for the current block in all the reference frames. The search pattern is illustrated in figure 3.1. The search algorithm is

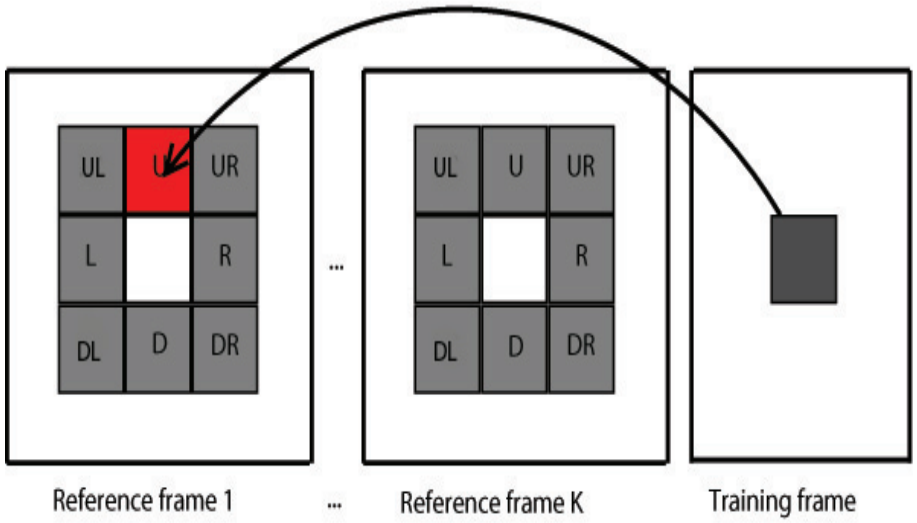

Figure 3.1: Search pattern for neighbor search Algorithm.

applied to all the blocks in the current frame. If K denotes the number of reference frames and m denotes the number of blocks in the given frame, the computational complexity of neighbor search algorithm would be $O(Km)$.

3.3 Diamond Search Algorithm

The neighbor search algorithm though simple and fast, has a limited search range since it only examines the neighboring block and itself. This might work out for video frames because of the strong adjacent block correlations but might not always for data frames. One of the straightforward ways of increasing the search range and finding better reference blocks is to exhaustively check for all blocks in the reference frame. This results in increased computational complexity. The idea here is to find an optimal balance between search range and the quality of reference blocks. The diamond search algorithm [22] provides such a feasible solution.

The diamond search algorithm utilizes two search patterns. One is the large diamond search pattern (LDSP) and other is the small diamond search pattern (SDSP) as illustrated in figure

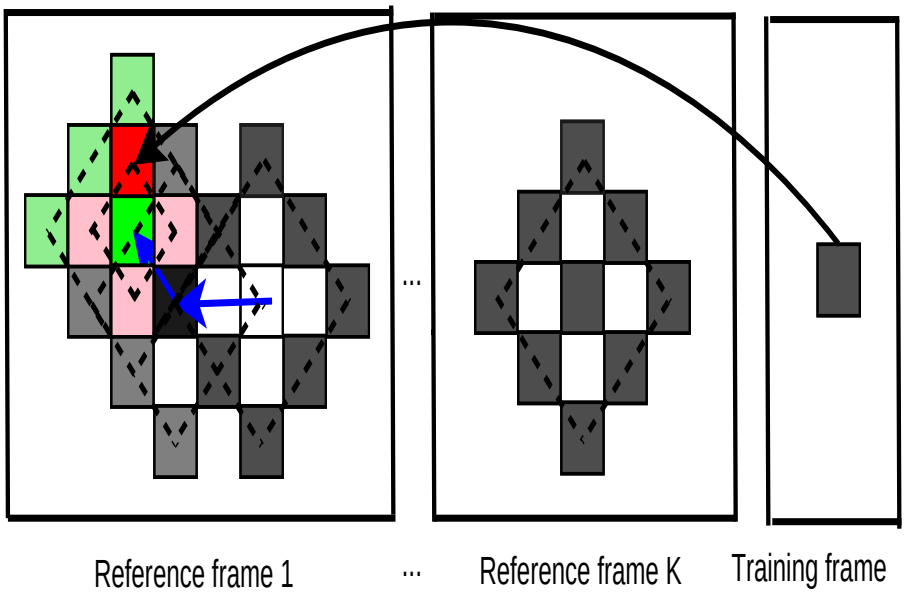

Figure 3.2: Search pattern for diamond search Algorithm.

3.2. The LDSP searches nine blocks out of which eight surround the center block to form a diamond. SDSP searches five blocks out of which four form a smaller diamond around the center block. The diamond search algorithm repeatedly searches using LDSP until the best matching block occurs at the center. Then, the search pattern is switched to SDSP and the best block found during this stage is considered the best reference block.

The diamond search algorithm doesn't restrict the number of search steps. In the best case the LSDP is run only once and if K denotes the number of reference frames and m denotes the number of blocks in the give frame, the computational complexity of diamond search algorithm would be $O(Km)$. In the worst case, LDSP will run over all the blocks and the complexity of the diamond search algorithm would be $O(Km^2)$. In most cases the average complexity is found to be $O(Km)$.

Chapter 4

Experimental Evaluation

The proposed compressive system monitoring has been implemented and deployed on PlanetLab [4]. We have a web interface showing the real-time monitoring of the system. The experimental evaluation of our compressive monitoring system has been done using real system monitoring data. In this section we first describe the system implementation details followed by the trace description. We then present our experimental results.

4.1 Prototype Implementation

We have implemented the prototype of the proposed system using C/C++. A monitoring sensor is deployed on each of the PlanetLab host and the management node program is run on a dedicated server. The server machine has configuration of Intel Core Duo CPU 2.4 GHz with 4GB of RAM. Each of the monitoring sensor also collects various intra-node metrics like CPU load, free memory etc. For the inter-node attributes, each monitoring sensor pings other nodes in the system and collects information such as latency and bandwidth. Based on pre-defined periodic reporting interval, the monitoring sensor reports the intra-node and inter-node attributes to the management node. The collected attributes are associated with the IP address to identify the information source and the management node aggregates all the information reported by the various sensors.

We also implement several alternative algorithms. The first is basic temporal correlation based compression algorithm denoted by *Temporal*. For an attribute a_i, the Temporal correlation uses the value at time $t-1$ as the predicted value at time t if they are within the predefined error bound. If they are not, the reference value gets updated with the current value which will be used for future prediction. The Temporal correlation will have good performance if the attribute values do not fluctuate frequently with large variation. We also have implemented the spatial correlation based compression algorithm denoted by *Spatial*. Using the k-medoids algorithm, the Spatial method clusters all the attributes into different groups based on the value of a_i. The medoid of each group is elected as the cluster head. The cluster members do not need to send their updates if the difference between their value and the cluster head is within the predefined error bound. The integrated approach that considers both the temporal and spatial correlation is also implemented.

4.2 Trace Collection and Experimental Results

The proposed system has been deployed on around 300 PlanetLab nodes. We have collected 66 different metrics shown in table 4.1 with a sampling rate of 10 seconds. We have mainly concentrated on evaluating our compression algorithms on some of the challenging attributes that are highly dynamic such as network delay, CPU load and free memory. The data were collected for a period of one month starting in December 2009. We also have evaluated our approach using real Internet traffic matrices collected by previous research work from a transit network [20]. This dataset contains traffic matrices taken per 15 minutes for a period of about four months. Statistics of the various traces are listed in table 4.2.

First we present the result for the intra-node attributes. In figure 4.1, we compare the average compression ratio (Equation 2.2) of different methods under various error bounds for the CPU load attribute. We observe that the coding based algorithms continuously performs better than the other alternatives at tighter error bounds. Figures 4.2 and 4.3 show the continuously sampled compression ratios for different approaches with error bound 1% and 5% respectively.

Table 4.1: Monitored metrics on Planetlab.

Monitored Attributes		
LOAD1	LOAD5	AVAILCPU
UPTIME	FREEMEM	FREEDISK
DISKUSAGE	DISKSIZE	MYFREEDISK
NUMSLICE	LIVESLICE	VMSTAT1-17
CPUUSE	RWFS	LOAD11
LOAD12	LOAD13	SERVTEST1
SERVTEST2	MEMINFO1	MEMINFO2
MEMINFO3	BURP	CPUHOG
MEMHOG	TXHOG	RXHOG
PROCHOG	TXRATE	RXRATE
PURKS1-10	PUKPUKS1-10	

Table 4.2: Statistics of the Traces.

Description	**Min**	**Max**	**Mean**	**Std.**
network delay trace	0.017	10449	241.8	585.9
CPU load trace	0.1	100	13.26	11.18
Free memory trace	26	2820400	173505	191687.8
Internet Traffic matrices trace	0.24	592867	17040	52578

The percentage value in the legend indicates the mean compression ratio achieved by different algorithms. We can observe that our coding based approach performs better or atleast on par with the combined Spatial and Temporal method.

We ran a similar set of experiments using the free memory trace. In figure 4.4, we compare the average compression ratio (Equation 2.2) of different methods under various error bounds for the free memory attribute. We still observe that the coding based algorithms perform better than the other alternatives and the gain is much higher at tighter error bounds. The coding based algorithms are light-weight and provide a better choice. Figures 4.5 and 4.6 show the continuously sampled compression ratios for different approaches with error bound 1% and 5% respectively. The percentage value in the legend indicates the mean compression ratio achieved by different algorithms. We can observe that the diamond search performs better than the neighbor search and the combined Spatial and Temporal methods.

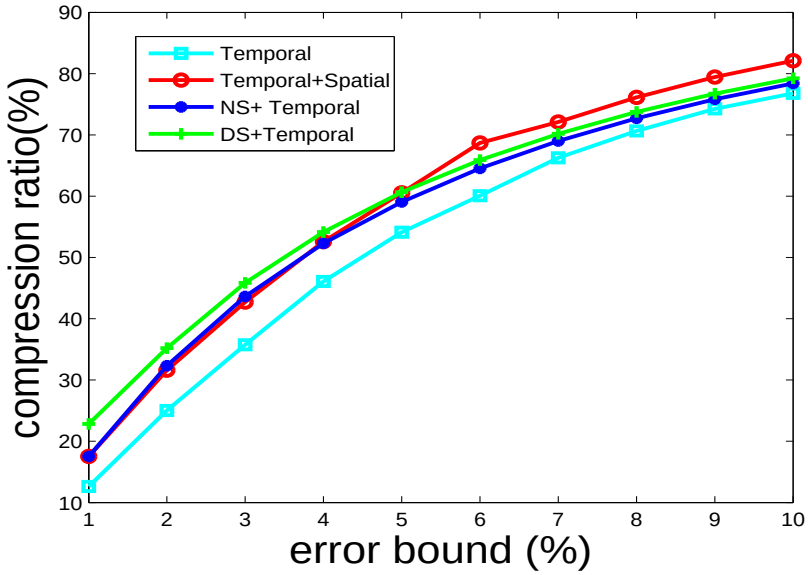

Figure 4.1: Compression ratio comparison under different error bounds for the CPU load trace.

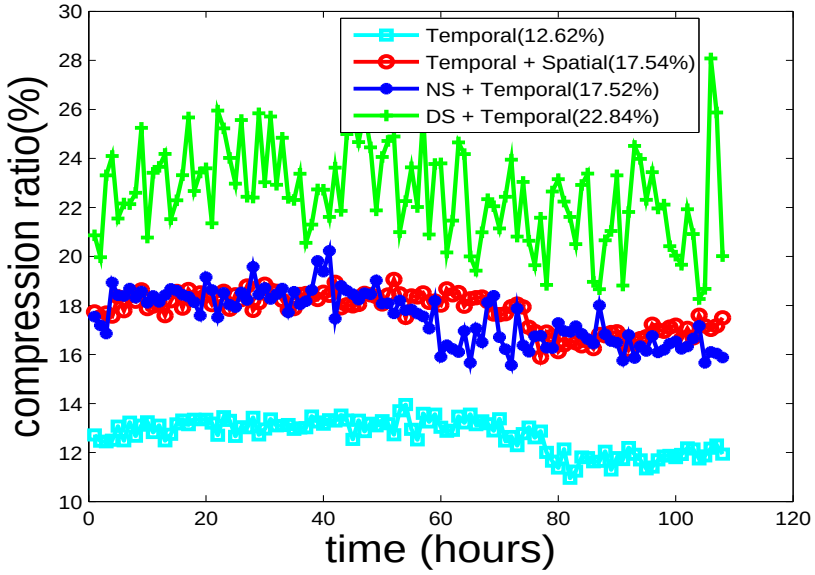

Figure 4.2: Continuous compression ratio comparison for the CPU load trace. Error bound = 1%.

Next we present the results for the inter-node attributes. Figure 4.7 shows the average compression ratio achieved by different compression approaches under various error bounds.

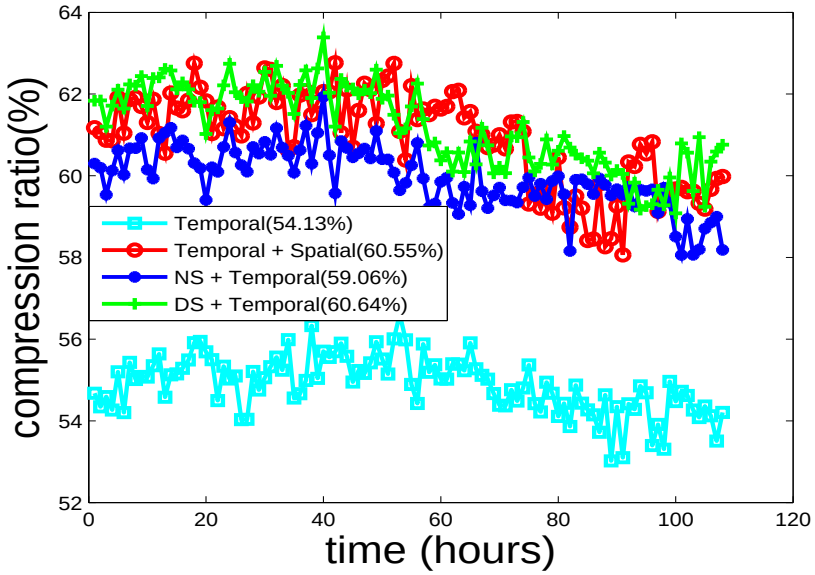

Figure 4.3: Continuous compression ratio comparison for the CPU load trace. Error bound = 5%.

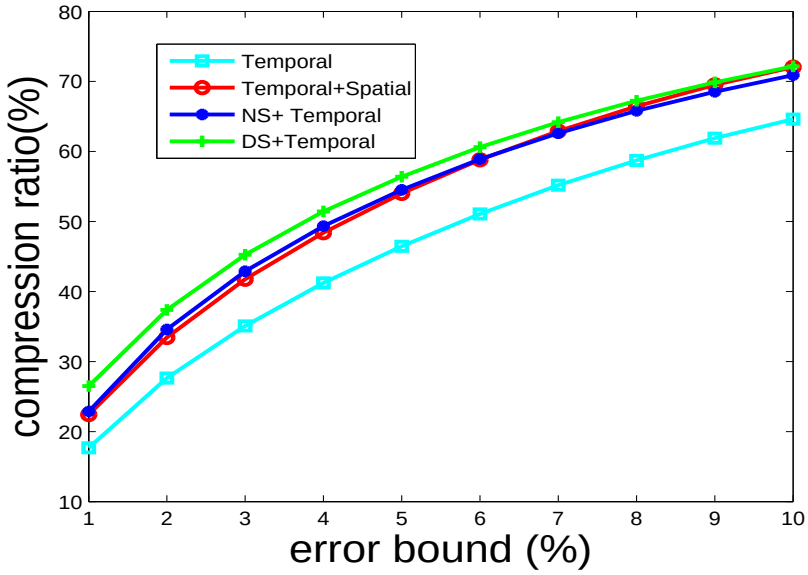

Figure 4.4: Compression ratio comparison under different error bounds for the free memory load trace.

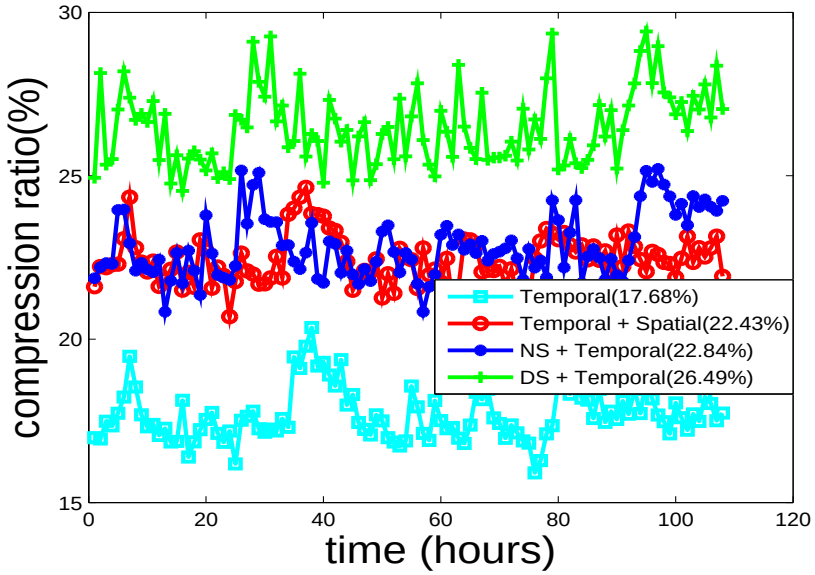

Figure 4.5: Continuous compression ratio comparison for the free memory load trace. Error bound = 1%.

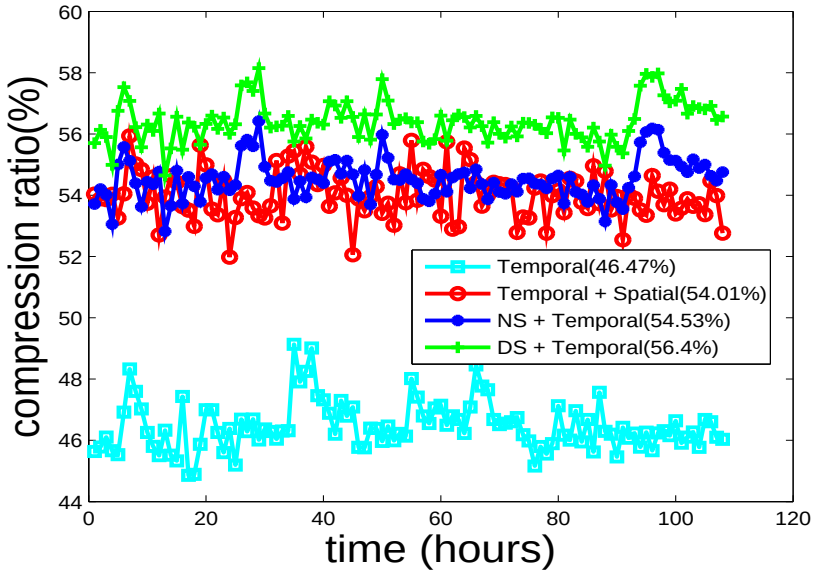

Figure 4.6: Continuous compression ratio comparison for the free memory load trace. Error bound = 5%.

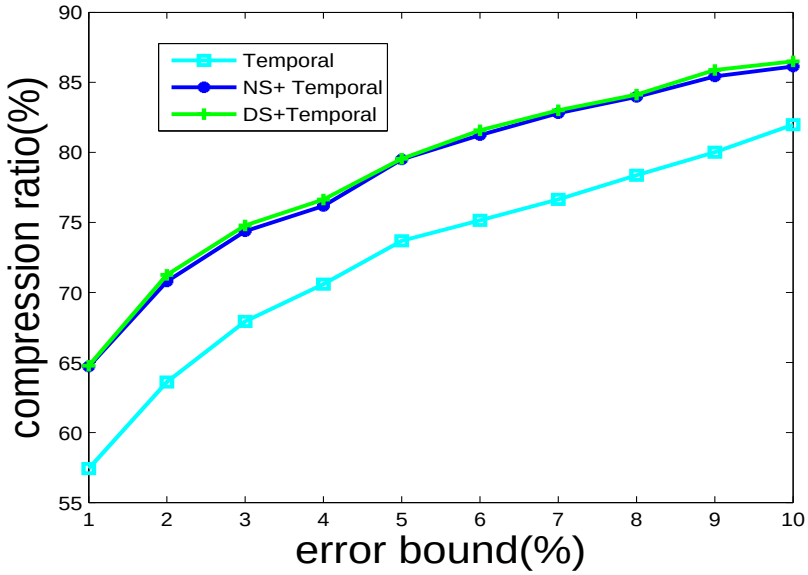

Figure 4.7: Compression ratio comparison under different error bounds for the inter-node delay trace.

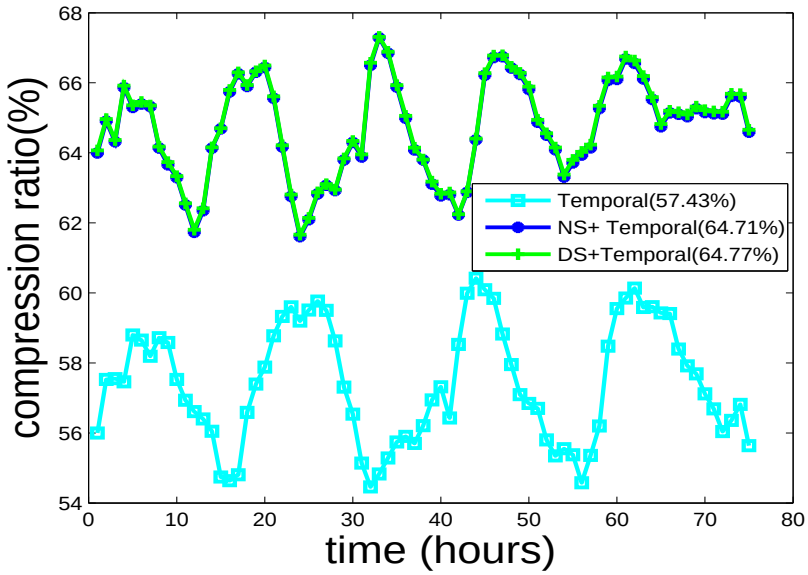

Figure 4.8: Continuous compression ratio comparison for the inter-node delay trace. Error bound = 1%.

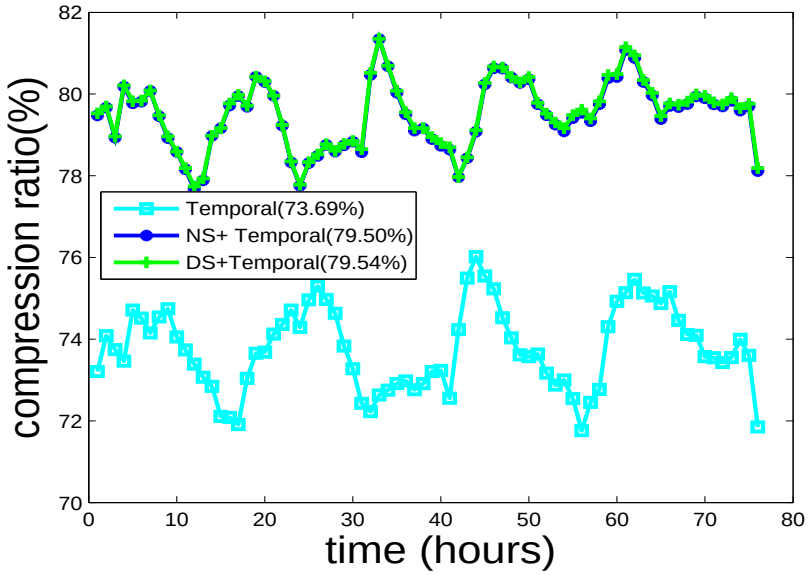

Figure 4.9: Continuous compression ratio comparison for the inter-node delay trace. Error bound = 5%.

For the inter-node attribute, we do not compare with the Spatial and the combined Spatial and Temporal method because the clustering algorithm is too expensive to perform for such large data sets and it takes more than 15 minutes to complete one round of clustering. In contrast, our coding based algorithm only takes tens of seconds. In figures 4.8 and 4.9 we show the continuously sampled compression ratios for different approaches with error bound 1% and 5% respectively.

We also have evaluated our approach using the Internet traffic matrices trace. Figure 4.10, shows the average compression ratio achieved by different compression approaches under various error bounds. In figures 4.11 and 4.12 we show the continuously sampled compression ratios for different approaches with error bound 1% and 5% respectively. The percentage value in the legend indicates the mean compression ratio achieved by different algorithms. Once again we observe that the diamond search performs better than the neighbor search and the combined Spatial and Temporal methods because of its superior search range.

We next consider the effects of varying the algorithm parameters like block size and training

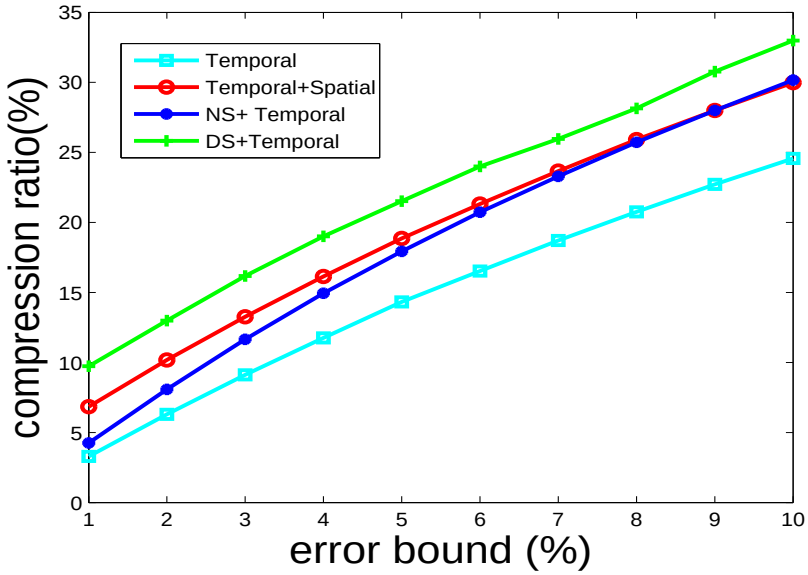

Figure 4.10: Compression ratio comparison under different error bounds for the traffic matrices trace.

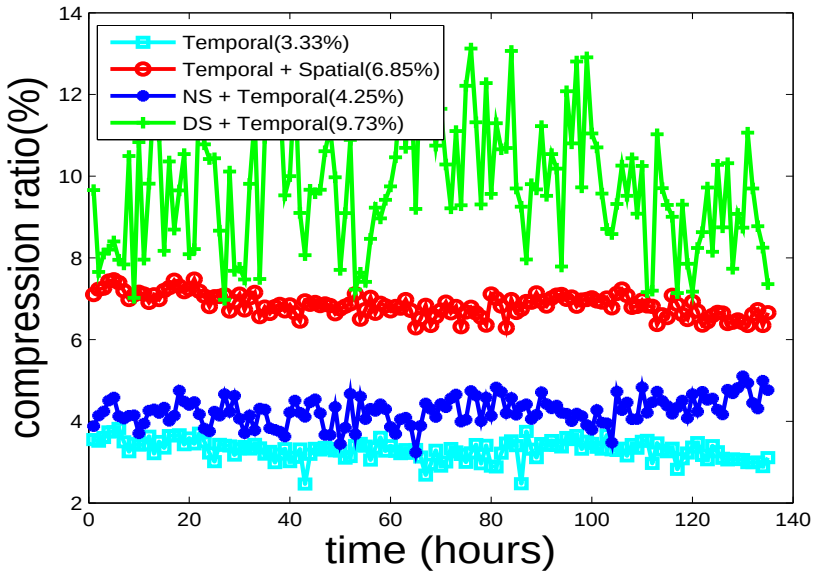

Figure 4.11: Continuous compression ratio comparison for the traffic matrices trace. Error bound = 1%.

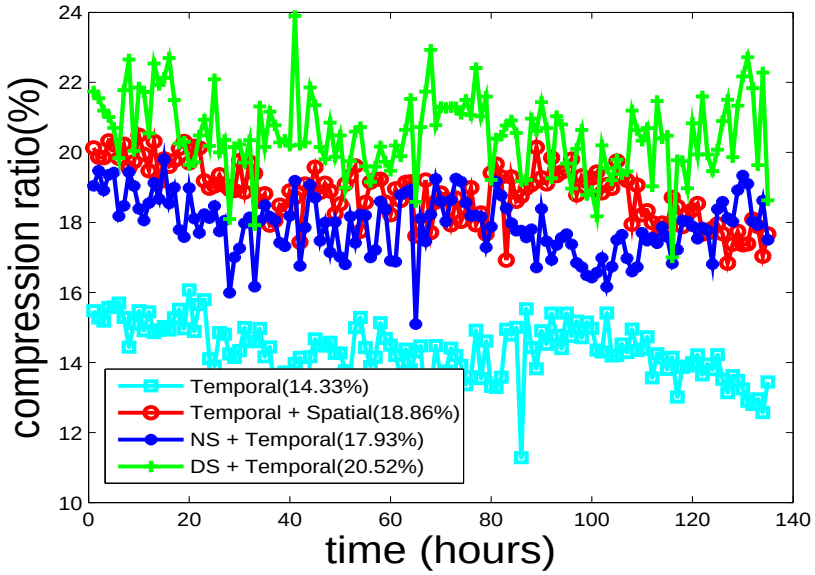

Figure 4.12: Continuous compression ratio comparison for the traffic matrices trace. Error bound = 5%.

period. Figures 4.13 and 4.14 show the results achieved when the block size is changed to 4×4, 2×2 and 6×6 for the free memory metric. We can observe that the block size of 4×4 gives better results for the free memory trace. Figures 4.15 and 4.16 show the results achieved when the re-training timeout is varied to 5, 10 and 15 minutes. We can observe that frequent re-training can achieve better compression ratio.

Finally we compare the overhead of different approaches. Table 4.3 summarizes the overhead measurements. We can clearly see that our coding based approach is much more light-weight than the clustering based Spatial algorithm.

Table 4.3: Compression processing overhead for different approaches.

Trace	**Num of Samples**	**DS**	**NS**	**Spatial**
inter-node delay	120K	157s	120s	1195s
CPU load	400	0.95s	0.78s	1.7s

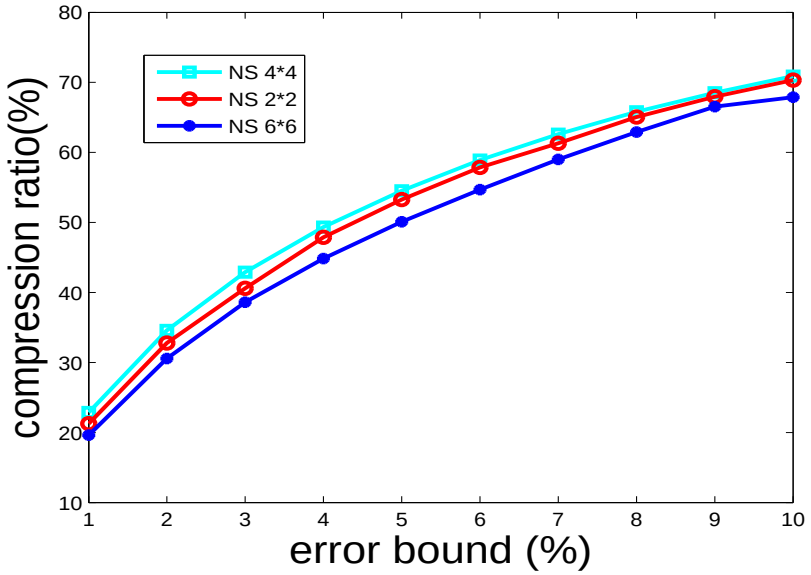

Figure 4.13: Compression ratio comparison for the free memory load trace with different block sizes for the neighbor search.

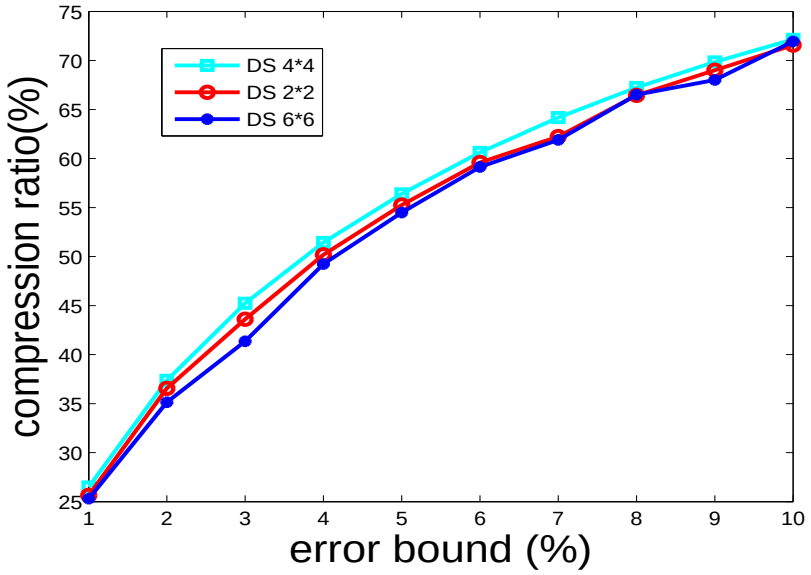

Figure 4.14: Compression ratio comparison for the free memory load trace with different block sizes for the diamond search.

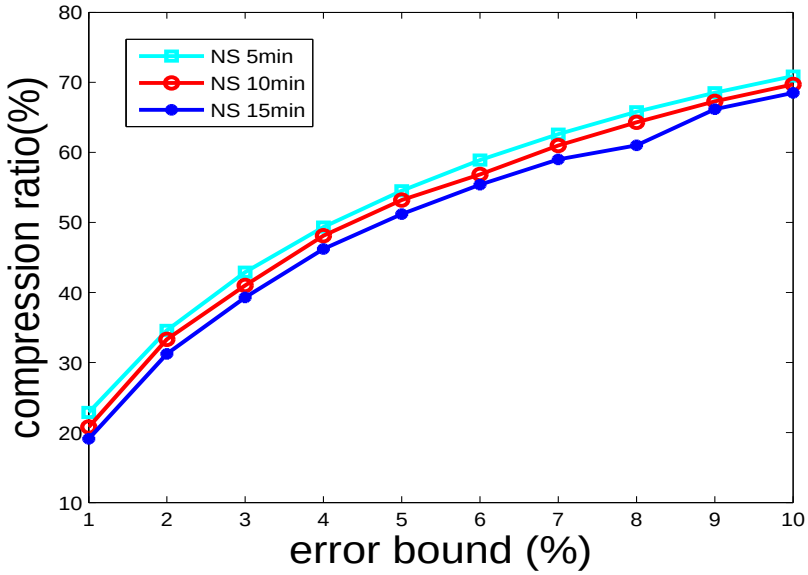

Figure 4.15: Compression ratio comparison for the free memory load trace with different retraining timeout for the neighbor search.

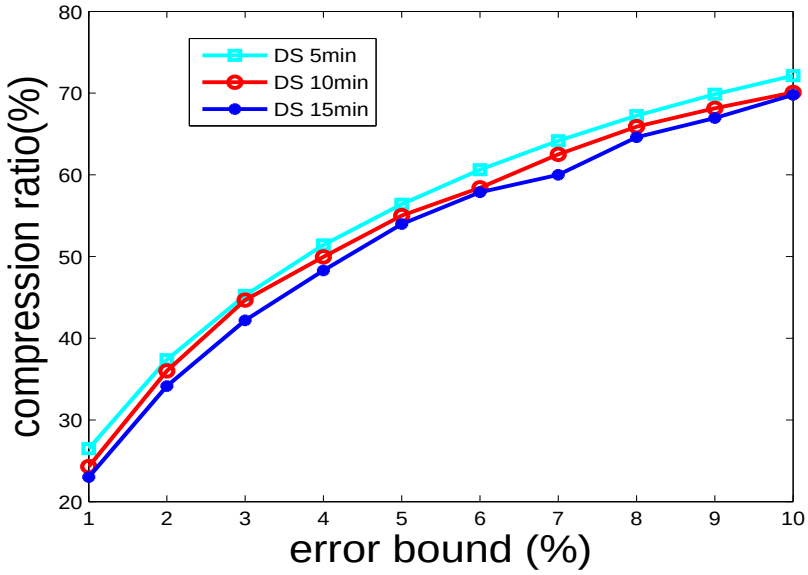

Figure 4.16: Compression ratio comparison for the free memory load trace with different retraining timeout for the diamond search.

Chapter 5

Related Work

Information monitoring services help in managing nodes in a large scale distributed system. There has been a lot of previous research work in this area. CoMon [2] is a monitoring system that collects data on 400-450 PlanetLab nodes and reports them on a Web Interface. It has been providing services like monitoring, community-aided problem identification, troubleshooting login failures and Node selections. However, the drawback of this system is that the frequency of data collected is low in order to maintain the storage and communication costs. Also, the system monitors data over a fixed time interval (5 minutes).

Many systems like SWORD [13], SDIMS [19] and Mercury [6] have proposed decentralized architectures to provide scalable information management. Astrolabe [14] is a similar large scale monitoring system that collects data, allows rapid updates and provides aggregated results that are generated dynamically. Astrolabe uses gossip protocol to collect data and creates a hierarchical structure to run SQL aggregation queries. However, due to the data replication on the Astrolabe agents, the system may not be scalable for many attributes. Also, the system does not provide security against faulty aggregation queries.

VPC3 [7] is an off line trace compression algorithms intended for trace files which tend to be large. It utilizes value predictors to identify and amplify patterns in the log files so that compression/decompression can be achieved more effectively and faster. This approach is

suitable for trace databases and cannot be used compress/decompress on line streams. FDR [17] is an online system call tracing tool used for troubleshooting and it mainly deals with system call compression only unlike ours which deals with collecting different metrics values and transporting them to the management node with low over head.

Commercial monitoring solutions such as Amazon Cloud watch [1] provides monitoring utilities for the Amazon EC2 cloud. Metrics provided include CPU utilization m disk reads and writes, network traffic. Monitored data is stored for 2 weeks. Amazon Cloud watch also provides Auto scaling which allows addition /removal of instances based on the Cloud watch metrics.

Spatio-temporal compressive sensing [20] is being used in Internet traffic matrices to deal with missing values. The solution developed exploits the presence of both global and local structures in the real world traffic matrices to reliably perform TM estimation, TM prediction and anomaly detection. However it does not provide a common interface to measure intra and inter nodal attributes in large scale distributed systems and much of the work deals with recovering missing values.

InfoTrack [21] explores both spatial and temporal correlation to collect data and report statistics on a large scale distributed system. It tracks 66 dynamic attributes like CPU Usage, Memory Statistics on more than 300 distributed nodes. Infotrack utilizes data clustering algorithms and metric value predictors to reduce the monitoring traffic. Our work on the other hand employs a completely new approach of using video coding techniques to exploit similar correlations and reduce the monitoring overhead.

Chapter 6

Conclusion

In this thesis, we have presented a coding based compressive distributed monitoring system. We model the continuous monitoring data as sequence of images and apply video coding techniques to compress the monitoring streams. We have applied two simple reference block search algorithms to efficiently compress the monitoring streams with low overhead. To the best of our knowledge , our work marks the first attempt to apply video coding techniques for scalable distributed system monitoring. We have run experiments using the inter-node and intra-node attribute data collected from more than 300 Planetlab hosts. Our prototype implementation and experimental results show that it is practical and efficient to apply coding techniques for compressive distributed system monitoring.

REFERENCES

[1] Amazon CloudWatch. http://aws.amazon.com/cloudwatch/.

[2] CoMon. http://comon.cs.princeton.edu/.

[3] IBM Tivoli Monitoring. www.ibm.com/software/tivoli/products/monitor.

[4] PlanetLab. http://www.planet-lab.org/.

[5] World Community Grid. http://www.worldcommunitygrid.org/.

[6] A.R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *SIGCOMM 2004*, August 2004.

[7] M. Burtscher. Vpc3: a fast and effective trace-compression algorithm. In *SIGMETRICS*, pages 167–176, 2004.

[8] B. Gedik, H. Andrade, K. Wu, P.S.Yu, and M. Doo. Spade: the system s declarative stream processing engine. In *ACM SIGMOD*, pages 1123–1134, 2008.

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

[10] J. Liang, X. Gu, and K. Nahrstedt. Self-Configuring Information Management for Large-Scale Service Overlays. *Proc. of INFOCOM , 472-480*, 2007.

[11] M.L. Massie, B.N. Chun, and D.E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30, July 2004.

[12] J.W. Mickens and B.D. Noble. Exploiting availability prediction in distributed systems. In *NSDI*, 2006.

[13] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In *HPDC-14*, July 2005.

[14] R. Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalabel technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

[15] T. Sikora. Trends and perspectives in image and videocoding. *Proc. of IEEE*, pages 6–17, January 2005.

[16] G. J. Sullivan and T. Wiegand. Video compression: From concepts to the h.264/avc standard. *Proc. of IEEE*, pages 6–17, January 2005.

[17] C. Verbowski. Flight Data Recorder: Always-on Tracing and Scalable Analysis of Persistent State Interactions to Improve Systems and Security Management. In *Proceedings of OSDI*, 2006.

[18] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An Information Plane for Networked Systems. *HotNets-II*, 2003.

[19] P. Yalagandula and M. Dahlin. A Scalable Distributed Information Management System. *Proc. of SIGCOMM 2004*, August 2004.

[20] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu. Spatio-temporal compressive sensing and internet traffic matrices. In *SIGCOMM*, pages 267–278, 2009.

[21] Y. Zhao, Y. Tan, Z. Gong, X. Gu, and M. Wamboldt. Self-correlating predictive information tracking for large-scale production system. *ICAC 09 , 33-42*, 2009.

[22] S. Zhu and K.K. Ma. A new diamond search algorithm for fast block matching motion estimation. *IEEE Trans.Image Processing*, pages 287–290, February 2000.