Self-Correlating Predictive Information Tracking for Large-Scale Production Systems

Ying Zhao
Tsinghua University
Beijing, China
yingz@cs.tsinghua.edu

Yongmin Tan, Zhenhuan Gong, Xiaohui Gu
North Carolina State University
Raleigh, NC
{ytan2,zgong, gu}@ csc.ncsu.edu

Mike Wamboldt
IBM RTP
Durham, NC
wamboldt@us.ibm.com

ABSTRACT

Automatic management of large-scale production systems requires a continuous monitoring service to keep track of the states of the managed system. However, it is challenging to achieve both scalability and high information precision while continuously monitoring a large amount of *distributed* and *time-varying* metrics in large-scale production systems. In this paper, we present a new self-correlating, predictive information tracking system called *InfoTrack*, which employs lightweight temporal and spatial correlation discovery methods to minimize continuous monitoring cost. InfoTrack combines both metric value prediction within individual nodes and adaptive clustering among distributed nodes to suppress remote information update in distributed system monitoring. We have implemented a prototype of the InfoTrack system and deployed the system on the PlanetLab. We evaluated the performance of the InfoTrack system using both real system traces and micro-benchmark prototype experiments. The experimental results show that InfoTrack can reduce the continuous monitoring cost by 50-90% while maintaining high information precision (i.e., within 0.01-0.05 error bound).

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Measurement techniques]

General Terms

Design, Experimentation

1. INTRODUCTION

Large-scale distributed computing infrastructures have become important platforms for many important real-world production systems such as enterprise data centers, virtualized computing infrastructures, web service hosting centers, and online data stream processing systems. As these distributed computing infrastructures continue to grow, how to efficiently manage those complex infrastructures has become a challenging problem. Inspired by how human nervous system reacts to external changes, the autonomic computing paradigm [24] has recently been proposed as a viable approach to building self-managed systems.

To achieve automatic management of a large-scale production system, the first step is to gain insightful understanding about the managed system. Information tracking is one of the fundamental building blocks of autonomic systems, which can capture complete, time-varying system information (e.g., resource availability, service response time, virtual machine (VM) resource consumptions, application component states) and make it available via some query interfaces to other system controllers. For example, a job scheduler may issue a multi-attribute range query such as “find ten hosts that have at least 10% free CPU time and 20MB memory and 10GB disk space” or a top-k query such as “return the three hosts that have the highest CPU load in the past one hour”. The anomaly predictor [31, 13] needs to acquire continuous runtime system measurements to build system state classification models.

A large-scale distributed computing infrastructure typically consists of i) hundreds of or thousands of distributed *worker nodes* that execute different application tasks; and ii) a set of *management nodes* that monitor the conditions of all worker nodes and perform various system management tasks. To perform automatic system management, the management node first needs to gain understanding about the managed systems. In many cases (e.g., global job scheduling, resource optimizer, anomaly prediction and diagnosis), the management node needs to acquire complete, fine-grained, and continuous monitoring about the whole system. For this purpose, we need to deploy monitoring sensors on all worker nodes that periodically sample various metric values (e.g., resource metrics, performance metrics) about the local worker nodes and continuously report the metric values to the management nodes.

However, it is a challenging task to provide *scalable* and *precise* continuous system monitoring for large-scale production systems. On one hand, system controllers reside within the management nodes desire to get up-to-date, precise, and global information about the whole distributed infrastructure in order to better accomplish their management tasks. On the other hand, the system can include a large number of geographically dispersed nodes, and each node can be associated with tens of or hundreds of dynamic attributes [1, 2]. For example, the World Community Grid [5] consists of many thousands of nodes and IBM Tivoli Monitoring [2] can collect over 600 metrics on a host running Windows OS. Obtaining accurate information about all nodes with their complete information continuously would inevitably involve high monitoring cost.

Existing production system monitoring solutions [1, 10] typically configure long information update interval (e.g., several minutes) to tradeoff information precision for low monitoring cost. However, many automatic system management tasks desire more

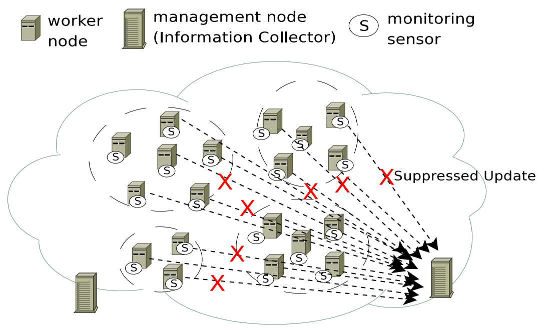


Figure 1: Suppressed Information tracking for large-scale distributed systems.

notation	meaning
N	total number of monitored nodes
A	set of all attributes
a_i	system state attribute
S_i	size of attribute a_i
l_i	total number clusters for a_i
T	tracking interval
CR	total compression ratio
e_i	error bound requirement for a_i
$p_{i,1}$	% of nodes with temporal correlation for a_i
$p_{i,2}$	% of nodes with spatial correlation for a_i
$p_{i,3}$	% of nodes with temporal&spatial correlation for a_i

Table 1: Notations.

fine-grained, up-to-date, monitoring data. Previous research work on scalable distributed monitoring can be broadly classified into two categories: i) employing decentralized architectures such as hierarchical aggregation [27] or peer-to-peer structure [30, 23] to distribute monitoring workload; and ii) trading off information coverage [21] or information precision [17] for lower monitoring cost. In contrast, our research focuses on an orthogonal problem, that is how to *explore temporal and spatial correlation patterns among distributed monitoring metrics to perform online suppression over continuous monitoring data so as to minimize system monitoring cost*. Our solution can be generally applied to any centralized or decentralized monitoring architecture.

In this paper, we present the design and implementation of the *InfoTrack* system, a new *self-correlating, predictive* information tracking system to reduce monitoring cost without losing information coverage or precision. By suppressing remote information update, our approach not only reduces monitoring network traffic but also lower the end-system resource consumption (e.g., CPU, memory, disk storage, power) for processing monitoring data. InfoTrack employs light-weight schemes to discover various metric correlation patterns on-the-fly within the monitored system. We explore both temporal correlation within one node (e.g., self-similarity) and spatial correlation among distributed nodes (e.g., group-similarity) to suppress unnecessary remote information update, illustrated by Figure 1.

- To leverage the temporal correlations, we install a metric value predictor P_i at both the monitoring node and the management node. If the attribute value at time t, denoted by $a_{i,t}$ can be predicted by P_i within a pre-defined error bound, we can suppress the remote update about $a_{i,t}$ from the monitoring node to the management node since the management node can infer the attribute value using the same predictor.
- To leverage spatial correlation for a monitored attribute a_i, we cluster all monitored nodes into different groups based on the values of a_i. We elect one node in the group (e.g., cluster head) to report its measurement values for a_i. The management node can infer the attribute values of the other nodes in the group based on the spatial correlation function.
- Our approach is *integrated*, which comprehensively considers both temporal and spatial correlations for reducing tracking cost. Our approach is also *adaptive*, which dynamically updates metric value predictors and clusters to adapt to changing correlation patterns.

We have implemented a prototype of the InfoTrack system and tested it on both PlanetLab [25], a wide-area network testbed consisting of several hundreds of hosts dispersed over wide-area networks, and Virtual Computing Lab [4], a virtualization-based computing infrastructure consisting of several hundreds of blade servers. The InfoTrack system is continuously running on the Planetlab and VCL and the information tracking results can be accessed live via a web interface[3]. Our experiments used about 300 PlanetLab nodes and collected more than three months real system attribute data (66 attributes per node) on the PlanetLab. Our experimental results show that InfoTrack can achieve 80-90% compression ratio (i.e., percentage of suppressed remote information updates) for tracking highly dynamic attributes (e.g., CPU load, memory usage) within 0.05 error bound, and more than 95% compression ratio for tracking relatively stable attribute values within 0.01 error bound. We also measured the overhead of our algorithms, which shows that our approach is light-weight and scalable.

The rest of the paper is organized as follows. Section 2 gives an overview about our system model, approaches, and problem formulation. Section 3 describes the design details of the InfoTrack system. Section 4 presents the prototype implementation and experimental evaluation. Section 5 compares our work with related work. Finally, the paper concludes in Section 7.

2. SYSTEM OVERVIEW

In this section, we give an overview of the InfoTrack system. We first introduce the information management system model. We then describe our self-correlating predictive information tracking approach. Third, we derive the system cost model and present the problem formulation. We summarize the notations used in this paper in Table 1.

2.1 System Model

We consider a networked system that has N nodes $\{v_1, ...v_N\}$to be monitored and a set of management nodes, illustrated by Figure 1. Each node is associated with a set of attributes (e.g., resource consumptions, performance metrics, system component states) that are denoted by $A = \{a_1, ..., a_{|A|}\}$. Each attribute a_i is denoted by a name (e.g., CPU load) and a value (e.g., 10%, 20KB). Unless specified otherwise, we use a_i to represent both name and value of the attribute. On each node, there is a monitoring sensor that periodically samples the attribute values to produce a time series $\{a_{i,1}, ..., a_{i,t}, ..., a_{i,k}\}$ where $a_{i,t}$ denote the sampled value for the attribute a_i at time t. The management nodes receives continuous information update from distributed monitoring sensors to provide the distributed information tracking service. The goal of the information tracking service is to gain insightful knowledge about the managed system. With continuous monitoring, the information tracking service can capture not only snapshots of the networked system but also its evolving behavior.

2.2 Approach Overview

To achieve scalable distributed information tracking, we propose to explore temporal and spatial correlations among distributed monitoring streams to compress tracking traffic. We say an attribute exhibits temporal correlation if we can infer its value at time t denoted by $a_{i,t}$ using previous m values $\{a_{i,t-m}, ..., a_{i,t-1}\}$. We can install a predictor at both monitoring site and the management node. If the attribute time series $\{a_{i,1}, ..., a_{i,t}\}$ can be inferred by the predictor within the user-defined error bound, the monitoring sensor does not need to report $\{a_{i,1}, ..., a_{i,t}\}$ to the management node since the managed node can infer the attribute values using the same predictor.

We say a group of worker nodes exhibit spatial correlation for an attribute a_i if the nodes within the group possess correlated values for a_i. Two nodes are said to have correlated attribute values if one node has an attribute value $a_{i,t}$ at time t then the other node has an attribute value $f(a_{i,t})$ where f denotes some correlation function. The correlation function can take different forms such as $f(a_{i,t}) = a_{i,t}$, $f(a_{i,t}) = a_{i,t} + C$ or $f(a_{i,t}) = a_{i,t} \cdot K$, where C and K are constants. In this paper, we assume the correlation function takes the form of $f(a_{i,t}) = a_{i,t}$ to simplify explanations. To reduce tracking cost, we elect one node in the group (e.g., cluster head) as the representative to report its measurement values for a_i. All the other nodes in the group do not need to report the values for $a_{i,t}$ if the management node can infer their attribute values based on the correlation function $f(a_{i,t})$ within the user-defined error bound.

Our approach is based on the observation that real-world distributed production systems often exhibit temporal and spatial correlations. For example, a host can remain at a certain resource level during night when no new jobs are allocated to the node. In distributed systems, a group of hosts (e.g., nearby neighbors within one administration domain) may exhibit similar behavior when they are assigned to execute similar computing tasks. As we will show in Section 4, we collected several months of measurement traces on the PlanetLab and discovered significant temporal and spatial correlation patterns. Thus, we can explore those correlation properties to reduce distributed information tracking cost without losing information coverage or precision. Note that our approach can be applied to both centralized or decentralized monitoring system where the system can consist of one management node or multiple collaborative management nodes. For simplicity, we will use the case of single management node to explain our algorithm. However, our approach can be extended to the case of multiple management nodes straightforwardly.

2.3 Information Tracking Cost Analysis

Without exploring temporal or spatial correlations, a distributed information tracking system will configure all monitoring sensors to periodically report all attribute values to the management node. Let us assume the networked system consists of N nodes, each of which is associated with $|A|$ attributes $A = \{a_1, ..., a_{|A|}\}$, the update interval is T, and the message size for reporting the attribute a_i is S_i. We define the distributed information tracking cost as the amount of total measurement data delivered from all monitoring sensors to the management node every second. The original tracking cost without suppression is

$$C_{Orig} = \frac{1}{T} \cdot \sum_{a_i \in A} N \cdot S_i \tag{1}$$

Our approach first reduces the tracking cost by exploring the temporal correlation. If the attribute value of a_i can be inferred at time t within a certain error bound e_i (e.g., $|a_{i,t} - P_i(t)|/a_{i,t} \le e_i$) at one node, the monitoring sensor on that node does not need to report $a_{i,t}$ to the management node. Let us assume on average the management node can infer attribute values for $p_{i,1}$ percent of nodes for a_i, the total tracking cost is reduced to

$$C_T = \frac{1}{T} \cdot \sum_{a_i \in A} (1 - p_{i,1}) N \cdot S_i \tag{2}$$

We now derive the cost reduction brought by exploring spatial correlations to suppress remote information updates. Let us assume all monitored nodes can be clustered into l_i groups based on the values of a_i. The nodes within one group possess similar values for a_i. To reduce tracking cost, we elect one node in the group (e.g., cluster head) to report its measurement values for a_i. Thus, the management node can infer the attribute values of the other nodes in the group based on the spatial correlation function. If we assume on average we can infer attribute values for $p_{i,2}$ percent of nodes for a_i, the total tracking cost consists of two parts: the cost for cluster heads to report attribute values to the management node, and the cost for cluster members whose attribute values can not be inferred accurately to report their values, which can be defined as follows,

$$C_S = \frac{1}{T} \cdot \sum_{a_i \in A} l_i \cdot S_i + (1 - p_{i,2}) \cdot (N - l_i) \cdot S_i \tag{3}$$

In order to minimize the monitoring cost, we need to form good clusters, in which more attribute values of cluster members can be inferred from the value reported by the cluster head. The number of clusters is also an important factor in the monitoring cost function. With fewer clusters, the first cost compoment is smaller. However, larger clusters tend to include heterogeneous cluster members and increases the second cost component. Hence, to minimize the monitoring cost, we need to balance the two cost compoments by forming proper number of clusters.

We now derive the tracking cost of an integrated approach considering both temporal and spatial correlations. Assume on average the management node can infer attribute values for $p_{i,1}$ percent of cluster heads or cluster members based on the temporal correlation. Assume also the management node can infer $p'_{i,2}$ percent of cluster members whose attribute values cannot be inferred by the temporal correlations but can be inferred based on the reported or predicted values of the cluster heads. The total tracking cost becomes

$$C_I = \frac{1}{T} \cdot \sum_{a_i \in A} l_i (1 - p_{i,1}) S_i + (1 - p_{i,1} - p'_{i,2}) \cdot (N - l_i) \cdot S_i \tag{4}$$

Compared to the original information tracking service, our approach reduce the information tracking cost by suppressing remote updates of those attribute values that can inferred from temporal or spatial correlations. We use C_{to} to define the overhead of updating temporal predictors on the management node; We use C_{so} to denote the dynamic cluster update cost. We will quantify C_{to} and C_{so} when we describe specific temporal and spatial correlation discovery schemes in Section 3. We define the total tracking cost of the InfoTrack system as $C_{InfoTrack} = C_I + C_{to} + C_{so}$. We define the information tracking compression ratio (CR) as follows:

$$CR = \frac{C_{Orig} - C_{InfoTrack}}{C_{Orig}} = \frac{C_{Orig} - (C_I + C_{to} + C_{so})}{C_{Orig}} \tag{5}$$

The various cost functions derived in this section do not include the cost of initializing the tracking system, which includes the cost of building the initial predictors and clusters for each attribute value.

Such initial cost depends on the actual algorithms employed in the system, and we give a detailed analysis in Section 4.5.

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 *dynamic, 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.

3. SYSTEM DESIGN AND ALGORITHMS

In this section, we present the design and algorithm details of the InfoTrack system. We first describe the approach of exploring temporal correlations to reduce information tracking cost. Next, we present how to suppress information tracking cost by exploring spatial correlations. Finally, we present the integrated approach exploring both temporal and spatial correlations.

3.1 Exploring Temporal Correlation

To explore temporal correlation of an attribute a_i for reducing tracking cost, we install a predictor P_i at both the monitored site and the management node. If the attribute value at time t denoted by $a_{i,t}$ can be predicted by P_i within a certain error bound (e.g., $|a_{i,t} - P_i(t)|/a_{i,t} \leq e_i$), the monitoring sensor does not need to report $a_{i,t}$ to the management node since the management node can infer the attribute value using P_i. If the monitoring sensor detects that prediction error exceeds the pre-defined threshold by comparing the inferred value with the real measurement value (e.g., $|a_{i,t} - P_i(t)|/a_{i,t} > e_i$), the monitoring sensor performs normal information update by sending the measurement value of a_i to the management node. If the monitoring sensor detects that the predictor makes frequent errors, it constructs a new prediction function P'_i and transfers P'_i to the management node to replace the old prediction function.

Our InfoTrack system is a generic framework, in which any prediction approach can be used to explore temporal correlation. However, to ensure low tracking cost, we need to keep the prediction overhead low. In this paper, we consider two such light-weight predictors, a last-value based simple method and Kalman filter [18]. The last value based method uses the value at time $t-1$ as the predicted value for time t. Thus, if the attribute value does not fluctuate frequently, the last value predictor can accurately predict the metric value most of time. The advantage of this simple approach is that the new predictor P'_i is the measurement value $a_{i,t}$ itself. Hence, no additional traffic is generated for updating the new predictor P'_i at the management node since the monitoring sensor already reports $a_{i,t}$ to the management node based on our predictive monitoring protocol.

We also apply the Kalman filter [8, 16] to achieve predictive information tracking. The Kalman filter assumes that the process has n internal states and m observable measurements. The internal states and measurements at time $t-1$ and t are governed by the following equations:

$$x_t = \mathbf{A}x_{t-1} + w_{t-1} \tag{6}$$

$$z_t = \mathbf{H}x_t + v_t \tag{7}$$

where x and z are the state and measurement vectors of the process, respectively. The random variables w_t and v_t represent the process and measurement noise, which follows normal probability distributions with process noise covariance Q and measurement noise covariance R, respectively. The internal states propagate from time $t-1$ to time t through the state transition matrix $\mathbf{A}$, and the measurements are determined by a linear combination of internal states through a $(m \times n)$ matrix H. For simplicity, A, H, Q and R are assumed to be constant and we assume the internal states are observable measurements. The Kalman filter estimates the state vector in two steps: *prediction* and *correction*. The prediction of x_t is made by following Equation 6 and then corrected by the weighted difference of the true measurement and the prediction if a true measurement is available. The correction weight is obtained by applying the least squares method to minimize error covariance [8]. The process noise covariance Q controls the smoothing power of the Kalman filter.

The Kalman filter works in the following way in our information tracking system. At the beginning, for each monitored attribute a_i, the Kalman filter is initialized using the same A,H,Q,and R on both the monitoring site and the management node, and the true attribute value is pushed from the monitoring sensor to the management node to start the Kalman filter on both sides. Then, when the Kalman filter makes a prediction $\bar{a}_{i,t}$ at time t, the monitoring sensor checks whether $\bar{a}_{i,t}$ is within a certain error bound. If $\bar{a}_{i,t}$ is close enough to the true value $a_{i,t}$, the sensor does not report $a_{i,t}$ to the management site. The Kalman filter installed on the management node predicts the same $\bar{a}_{i,t}$, and the management node uses this predicted value as the observation value at time t. Both Kalman filters make steps of predictions without correction from true observations until $\bar{a}_{i,t}$ exceeds the predefined error bound, in which case the sensor pushes the observation of a_i to the management node, and both Kalman filters correct their predictions accordingly.

From Equation 5 and 2, we can see that we need to have a tradeoff between the predictor update cost (i.e., C_{to} in Equation 5) and the prediction accuracy, which determines the percentage of nodes that need to send attribute values to the management node (i.e., $p_{i,1}$ in Equation 2). In InfoTrack, both the last value based method and Kalman filter method only require the observed value of a_i for updating the predictors on the management node. Hence, there is no extra predictor update cost (i.e., $C_{to} = 0$ in Equation 5) for both approaches, and Equation 5 can be simplified as follows:

$$CR = 1 - \frac{\sum_{a_i \in A} (1 - p_{i,1}) \cdot S_i}{\sum_{a_i \in A} S_i}, \tag{8}$$

where S_i is the message size for reporting the attribute a_i.

3.2 Exploring Spatial Correlation

To explore spatial correlations for a monitored attribute a_i, we cluster all monitored nodes into different groups based on the values of a_i. The nodes within one group possess similar values for a_i. To reduce information tracking overhead, we elect one node in the group (e.g., cluster head) to report its measurement values for a_i. The management node can infer the attribute values of the other cluster members based on the spatial correlation function. Thus, we can reduce the tracking cost when the management node can infer the attribute values of the cluster members within a certain error bound using the spatial correlation function and the attribute value of the cluster head.

To form closely corelated clusters, we use the *Pearson correlation coefficient* as the similarity measure for clustering algorithms. Given a window size w, we can form a vector $[a_{i,t}, .., a_{i,t+w}]$ for each monitored node. The similarity between two such vectors $V = [v_1, ..., v_n]$ and $U = [u_1, ..., u_n]$ is defined as follows:

$$\mathrm{sim}(V,U) = \frac{\sum_{i=1}^{n}(v_i - \bar{v})(u_i - \bar{u})}{\sqrt{\sum_{i=1}^{n}(v_i - \bar{v})^2}\sqrt{\sum_{i=1}^{n}(u_i - \bar{u})^2}}, \qquad (9)$$

where $\bar{v}$ and $\bar{u}$ are the mean of the values of the V and U vectors, respectively. Note that we need to push all these values to the management node to initiate clustering process, we prefer a smaller value of w that can also lead to reasonable clustering results.

We employ two widely used clustering algorithms for our purpose: a typical partitional clustering algorithm k-means [22], and a typical agglomerative algorithm $UPGMA$ [15].

The k-means [22] algorithm computes a k-way clustering of a set of objects as follows. Initially, a set of k objects is selected from the datasets to act as the *seeds* of the k clusters and each object is assigned to the cluster corresponding to its most similar seed. Then, the centroid of each cluster is computed and objects are moved corresponding to their most similar centroids. This process is repeated until it converges to produce the final k clusters. The UPGMA (*i.e*, Unweighted Pair Group Method with Arithmetic mean) algorithm [15] finds the clusters by initially assigning each object to its own cluster and then repeatedly merging pairs of clusters until a certain stopping criteria is met. The UPGMA scheme measures the similarity of two clusters as the average of the pairwise similarity of the objects from each cluster. The agglomerative clustering algorithms produce a hierarchical tree at the end, and a k-way clustering soluction can be obtained by cutting the tree using various criteria.

Comparing these two algorithms, k-means has lower computational complexity, but may suffer from bad initial seed choices. In addition, k-means requires the number of clusters as an explicite input parameter, whereas UPGMA produces a hierarchical tree and we can cut the tree to form natural clusters.

After clustering different monitored nodes into groups, we select the one with the median value of attribute a_i as the cluster head. There are several ways for the management node to infer the attribute values of cluster members from the values reported by their cluster heads. For simplicity, we record the difference between the last reported values of the monitored node and its cluster head, and add this difference to the newly reported value as the inferred value.

During runtime, we need to dynamically update the cluster to maintain efficiency. We consider two types of changes in clusters. First, a monitored node may not exhibit the similar measurement value for attribute a_i as its cluster head after a certain period of time. In this case, we want to regroup the monitored site to its most similar cluster. We maintain a credit value for each monitored node on the management node. If the monitored node can be represented by the cluster head, the credit value is incremented. Otherwise, the credit value is decremented. The monitored node is re-assigned to another cluster when its credit value is below a certain threshold (denoted as θ_2). Second, a cluster head may not be the best representative of the cluster after a period of time, in which case we need to select a new cluster head. A new cluster head is selected when the fraction of nodes that need to updates their attribute values is above a certain threshold (denoted as θ_1) for a cluster. The management node needs to send control messages to monitoring sensors when changing cluster membership of a node or changing a cluster head, which forms the spatial correlation discovery overhead (i.e., C_{so} in Equation 5). This cluster adaptation cost is closely related to the number of re-assignments per tracking interval. As we will show in Section 4, the cluster adaptation cost is very small.

Note that since the re-assignment of cluster head requires sending cluster information to the newly selected cluster head, we need to avoid frequent cluster head re-assignment by setting a high θ_1 value. Similarly, re-assigning a monitored site back and forth can be eliminated by setting a low θ_2 value.

```
Input:
$a_{i,t}$: the observation value of $a_i$ at time $t$
$\bar{a}_{i,t}$: the predicted value of $a_i$ by Kalman filter at time $t$
$\hat{a}_{i,t}$: the $a_i$ value recieved from the cluster head at time $t$
$e_i$: error bound for $a_i$

SensorReport($a_{i,t}, \bar{a}_{i,t}, \hat{a}_{i,t}, e_i$)
1.  if the node is a cluster head
2.      if ($|a_{i,t} - \bar{a}_{i,t}|/a_{i,t}) > e_i$ )
3.          report $a_{i,t}$ to the management node
4.          push $a_{i,t}$ to its cluster members
5.      else
6.          push $\bar{a}_{i,t}$ to its cluster members
7.  else
8.      if ($|a_{i,t} - \bar{a}_{i,t}|/a_{i,t}) > e_i$ )
9.          if ($|a_{i,t} - \hat{a}_{i,t}|/a_{i,t}) > e_i$ )
10.             report $a_{i,t}$ to the management node
16.return
```

Figure 2: Integrated information update algorithm on the worker node.

3.3 Integrated Approach

We now present the integrated approach exploring both temporal and spatial correlations, which is shown in Figure 2. To achieve compressed information tracking, each monitoring sensor performs selective information report. The sensor reports the true observation value for an attribute a_i only when the management node cannot infer the value of a_i based on either temporal or spatial correlation functions within a certain error bound. If the monitored node is a cluster head, its monitoring sensor decides whether to report its observation value based on the accuracy of the predicted value given by the metric value predictor. Note that both the monitored node and the management node run the same predictor to predict the value of a_i. The monitoring sensor only reports the observation value of a_i to the management node if the prediction error exceeds a certain error bound.

For non-cluster-head monitored site, its monitoring sensor performs selective information report based on both temporal and spatial correlations. First, the monitoring sensor checks the predicted value of an attribute a_i given by its own metric value predictor. If the predicted value is within the error bound, the monitoring sensor will not report the observation value of a_i to the management node. Otherwise, the monitoring sensor checks the a_i value received from the cluster head, denoted by $\hat{a}_i$, which is either the observation value of a_i reported to the management node by the cluster head or the predicted value. If the cluster head value $\hat{a}_i$ can be used by the management node to infer the attribute value of the monitored site within the error bound, the monitoring sensor will not report its observation value to the management node. The monitoring sensor needs to report the observation value of a_i to the management node only when both the predicted value given by the metric value predictor and the attribute value of the cluster head cannot achieve desired accuracy. One complication is that the management node needs to know which value (e.g., metric value predictor or cluster head) to use if only one of the two values is usable. Under those cases, either the cluster head or the monitoring sensor needs to send a flag to the management node to indicate which correlation (i.e., temporal or spatial) should be used to infer the remote attribute value.

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 2: Monitored metrics on Planetlab.

When a new node arrives in the system, the management node assigns the node to a cluster based upon the attribute similarity between the node and cluster heads, and predictors are installed on both the worker node and management node. Worker nodes send signals to the management node periodically if they do not need to report their metric values for a long period. In this way the management node knows that a node leaves or becomes unreachable if it does not receive any metric report or life signal over a certain period. When a non-cluster-head node leaves, the management node simply removes the node from its cluster. Whereas, the management node needs to select a new cluster head if a cluster-head node leaves.

The analysis of the cost model for the integrated approach is similar to the separated ones. The information tracking compression ratio (CR) can be determined by Equation 5 and 4. As discussed above, our light-weight temporal and spatial approaches ensure a zero C_{to} and very small C_{so}. Hence, Equation 5 can be simplified as follows:

$$CR = 1 - \frac{\sum_{a_i \in A} l_i(1-p_{i,1})S_i + (1-p_{i,1}-p'_{i,2}) \cdot (N-l_i) \cdot S_i}{N \cdot \sum\limits_{a_i \in A} S_i}, \tag{10}$$

where S_i is the message size for reporting the attribute a_i, l_i is the number of clusters for attribute a_i, $p_{i,1}$ is the percent of nodes whose values can be inferred by temporal metric value predictors, and $p'_{i,2}$ is the percent of cluster members whose attribute values cannot be inferred by the temporal correlations but can be inferred based on the reported or predicted values of the cluster heads.

4. SYSTEM EVALUATION

4.1 Prototype Implementation

We have implemented a prototype of the InfoTrack system and deployed the system on the Planetlab [25] and VCL [4]. The tracking results of InfoTrack can be accessed live via the web interface mentioned in the Introduction. The monitoring sensor collects about 66 attributes (e.g., available CPU, free memory, disk usage, load1, load5, load10, number of live slices, uptime, etc.) on the PlanetLab, shown by Table 2. Each complete information report has about 2000 bytes.

We perform temporal correlation inference by using the simple last value approach or by running Kalman filters on both monitoring sites and the management node. The management node discovers spatial correlations using k-means or $UPGMA$ algorithms. The management node initializes the tracking process, pushes the error bound to each sensor, and invokes temporal correlation predictors on both sides. Each sensor executes the attribute report algorithm shown by Figure 2, which periodically collects resource attribute values and only reports the collected attribute values to the management when the correlation-based inference error is out of bound.

Trace Data	Mean	Avg. Std.	Avg. SV
CPU-10	80.2	5.65	1.34
CPU-30	80.2	5.65	2.58
MEM-10	78.2	11.29	2.29
MEM-30	78.2	11.29	3.11
Load5	8.85	2.71	-

Table 3: Statistics of Data Sets

4.2 Traces and Their Characteristics

Our experiments used about 300 PlanetLab nodes and collected several months real system attribute data on the PlanetLab without interrupting normal workload on each node. We track about 66 system attributes and set the report interval to be 10 or 30 seconds. Our system can easily achieve high compression ratio for tracking relatively stable attributes such as uptime, disk usage, load5, load10 within 0.01 error bound. Thus, our experiments focus on evaluating our algorithms on tracking most challenging attributes such as CPU load and free memory. To this end, we extracted four sets of trace data starting from March 20, 2008 for more than a week: CPU load observed every 10 seconds (*CPU-10*), CPU load observed every 30 seconds (*CPU-30*), memory usage observed every 10 seconds (*MEM-10*), and memory usage observed every 30 seconds (*MEM-30*).

We use these trace data sets to evaluate our various temporal correlation, spatial correlation, and integrated models on different system state attributes, as well as different tracking intervals. Some statistics of the data sets are shown in Table 3. We include the average and average standard deviation of Load5 (i.e., the load average for the past 5 minutes in terms of how many active processes are competing for the CPU) in Table 3 as well to demonstrate the workload and conditions when the traces were acquired. In addition to the mean CPU load (in percentage) and mean memory usage (in percentage), we also calculated average standard deviation (labelled as "Avg. Std") along time intervals averaged over all nodes and average step variance (labelled as "Avg. SV"). The average step variance is the absolute difference between two consecutive measurements averaged over all time intervals and all nodes. The average standard deviation indicates the range of the measurements varying along time intervals, whereas the average step variance indicates how rapidly each measurement changes. As shown in Table 3, memory usage exhibits larger variance than CPU load. Hence, we expect that the two memory usage data sets are harder for compression. Varying report intervals from 10 seconds to 30 seconds increases the average step variance. Since our models compress data based on past measurements, we expect that our models perform better on trace data with smaller average step variances.

4.3 Evaluation Methodology

We evaluate our correlation-aware information tracking models using randomized test on the four trace data sets. For a given trace, we randomly select a starting point in the trace and start to evalute our models for the next 9000 samples (for CPU-10 and MEM-10) and 3000 samples (for CPU-30 and MEM-30), respectively. The number of tested samples is chosen to be 9000 and 300 so that the evaluation period covers one day.

For our temporal correlation aware tracking model with the Kalman filter, we set the process noise covariance Q equal to 10 and

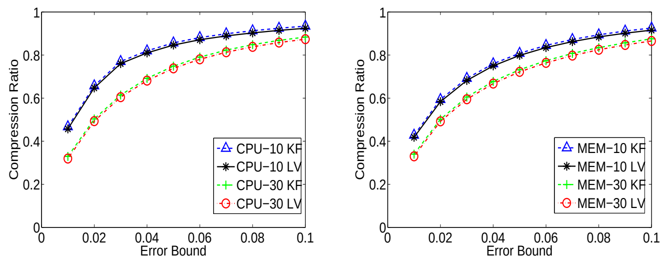
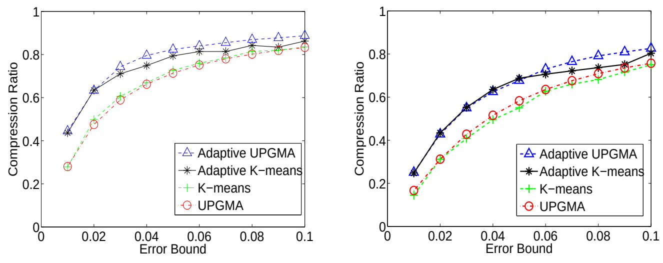
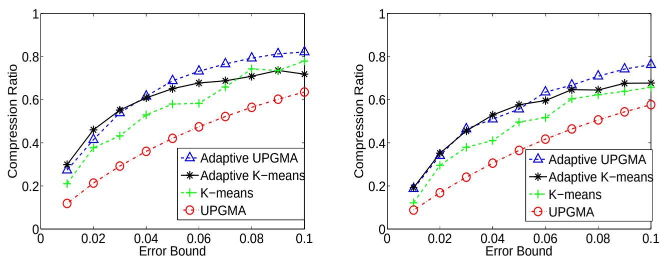


Figure 3: Mean tracking cost reduction based on the temporal correlation.



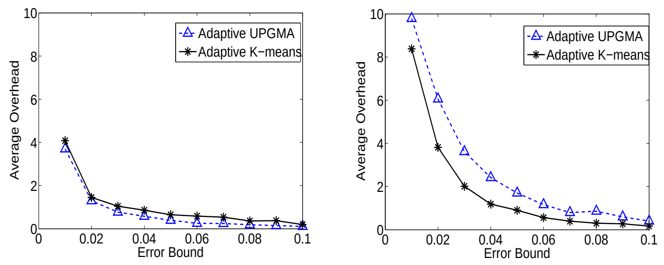
(a) CPU-10 (b) CPU-30

Figure 4: Mean CPU metric tracking cost reduction based on the spatial correlation.



(a) MEM-10 (b) MEM-30

Figure 5: Mean memory metric tracking cost reduction based on the spatial correlation.



(a) CPU-10 (b) MEM-10

Figure 6: Average overhead of adaptive clustering algorithms.

measurement noise covariance R equal to 1, as discussed in Section 3.1. For our spatial correlation-aware tracking model, we test both k-means and $UPGMA$ clustering algorithms. We empirically chose to use a window size of 12 for calculating correlation coefficient (Equation 9). The number of clusters produced by $UPGMA$ is controlled to be between 10 to 20, as the values in this range tend to perform well. This is done by varying the inconsistency coefficient cutoff value. We use the same number of clusters as the input to the k-means clustering algorithm.

The temporal correlation-aware, spatial correlation-aware, and integrated information tracking models are evaluted with an error bound value ranging from 0.01 to 0.1. The performance of various models are assessed using *Compression Ratio* (Equation 5). Each experiment is repeated 200 times, and the average compression ratio is reported.

4.4 Results and Analysis

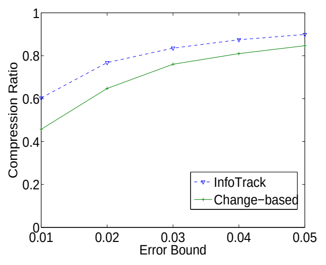
We first present the results for our temporal correlation-aware model, which uses the last value based approach or Kalman filter to infer dynamic attribute values. Figure 3 shows the compression ratio achieved by the last value model and Kalman filter model with various error bound values for CPU-10, CPU-30, MEM-10, and MEM-30. From the figure, we can clearly see that the benefit of employing the temporal correlation-aware model to achieve compressed information tracking. The larger the error bound is allowed, the higher the compression ratio can be achieved by our system. The two temporal correlation-aware models perform similarly and can achieve compression ratio over 90%, with error bound around 0.1. As we expected, our model performs better on CPU load data than on memory usage data since the former data set shows bigger step variances than the latter data set. The compression ratios achieved on CPU-10 and MEM-10 is better than those on CPU-30 and MEM-30.

We conduct the second set of experiments to study the effectiveness of our spatial correlation aware information tracking model. Figure 4 and Figure 5 show the compression ratio achieved by different clustering algorithms on CPU-10, CPU-30, MEM-10, and MEM-30. We run our experiments using four clustering algorithms in total, namely $UPGMA$, k-means, and their adaptive versions with dynamic cluster adjustments.

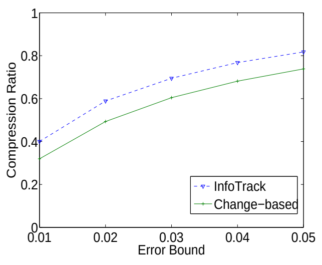
There are a number of observations we can make from those figures. First, we observe that our spatial correlation aware tracking model can reduce the tracking cost significantly. The two adaptive clustering algorithms can achieve a compression ratio of more than 50% with tight error bounds of less than 0.05 for all traces. Second, similar to the results of our temporal correlation aware tracking model, the results here also show the trace data with longer report intervals are harder to compress than the trace data with shorter report intervals. Third, the adaptive clustering techniques improve the compression ratio in most cases. On average the adaptive $UPGMA$ clustering algorithm outperforms the $UPGMA$ algorithm by more than 10% on CPU-10 and CPU-30, and around 20% on MEM-10 and MEM-30. The adaptive k-means clustering algorithm also outperforms the k-means clustering algorithm by 5% to 7% on average on four trace data sets. Finally, the two adaptive clustering algorithms perform similarly for all data sets.

Figure 6 shows the cluster adaptation cost introduced by the adaptive techniques on CPU-10 and MEM-10. The average overhead shown in Figure 6 is defined as the number of re-assignments of either cluster heads or cluster members per report interval. We observe that the number of re-assignments quickly reaches to a small number as the error bound increases. Hence, the adaptive techniques improve our tracking compression ratios with little additional cost.

We now present the results of the integrated self-suppressing information tracking approaches. The two temporal approaches and two spatial approaches can have four combinations. We show the results of combining Kalman filter with adaptive $UPGMA$, and other combinations show similiar trends as well. Figure 7 and Figure 8 show the compression ratio achieved by the integrated

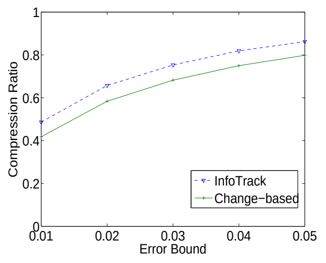


(a) CPU-10

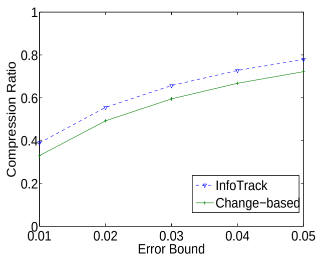


(b) CPU-30

Figure 7: Mean CPU metric tracking cost reduction of the integrated approach.



(a) MEM-10



(b) MEM-30

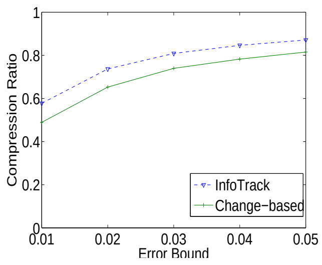
Figure 8: Mean memory metric tracking cost reduction of the integrated approach.

model (labelled as InfoTrack in all figures) on CPU-10, CPU-30, MEM-10, and MEM-30, and the results of a simple change based model as a baseline. The change based model performs information update when the attribute value change exceeds the error bound. We observe that the tracking models exploring both temporal and spatial correlations can consistently improve the compression ratio further than the tracking models exploring only temporal or spatial correlations. The integrated approach can achieve more than 60% compression ratio on CPU-10 with a tight error bound of 0.01, and 90% compression ratio with an error bound of 0.05. On MEM-10, the integrated approach can achieve 48% to 85% of compression ratio. The improvement achieved by InfoTrack over the change-based approach can be as high as 40% for CPU-10 and 25% for MEM-10, especially under a tight error bound. Note that the performance of our intergrated approach can be further improved by plugging in more advanced or application tailored temporal and spatial correlation models. However, the focus of this paper is not to find the best single temporal or spatial correlation models, but to present a generic framework which allows the integration of exploring both spatial and temporal correlations.

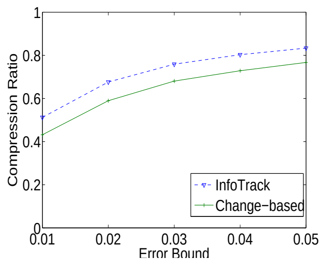
To evaluate InfoTrack with more dynamic systems, i.e., when system metric values have bigger average standard deviations and and bigger average step variances, we selected the PlanetLab traces of two days (denoted as *Trace1* and *Trace2*) with the highest average standard deviations of CPU load during the entire period

Trace Data	CPULoad Mean	CPULoad Avg. Std.	Load5 Mean	Load5 Avg. Std.
Trace1	77.1	6.92	6.72	2.23
Trace2	72.5	7.03	6.49	1.96

Table 4: Statistics of Trace1 and Trace2



(a) Trace1



(b) Trace2

Figure 9: Mean CPU metric tracking cost reduction of the integrated approach.

that we collected real system attribute data on the PlanetLab. Both traces contain CPU load values with the report interval of 10 seconds for 24 hours. Some statistics of the two traces are shown in Table 4. We show the mean CPU metric tracking cost reduction of the integrated approach in Figure 9. We can observe that with higher variations on metric values, InfoTrack can still achieve simiiliar improvements over the change-based approach as the ones shown in Figure 7, and achieve more than 50% compression ratio with an error bound of 0.01, and more than 80% with an error bound of 0.05. Other system metrics show similar trends on *Trace1* and *Trace2*, and the results are omitted due to space limitation.

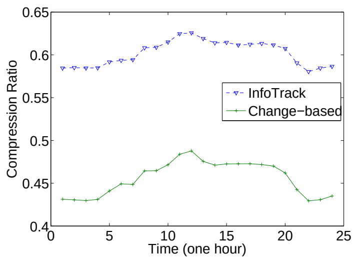
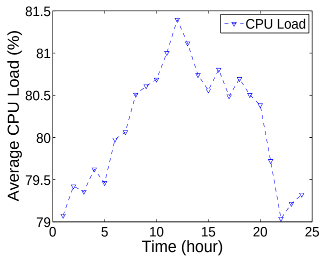


Figure 10: Continuous CPU metric tracking cost reduction with error bound = 0.01.



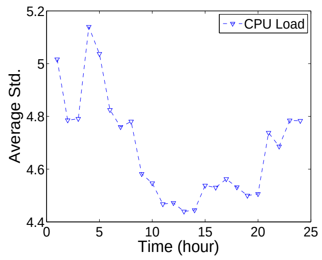


Figure 11: Average CPU load and standard deviation of CPU-10.

In the experiment results we have shown so far, the compression ratio is calculated by averaging over the entire tracking period (which is about 24 hours). We also want to see how the compression ratio changes over time. To do so, we evalute the performance of a tracking model by calculating the compression ratio value for each hour. Figure 10 shows such compression ratios achieved by Info-Track and a simple change-based model on CPU-10 with an error bound of 0.01. We observe that there is a clear “day and night” pattern in the results. To explain this performance pattern,

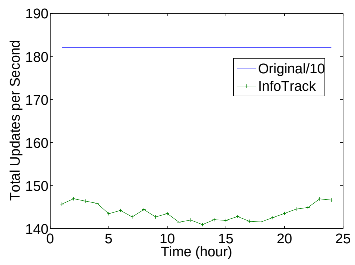


Figure 12: Continuous total tracking cost reduction on all 66 metrics with error bound = 0.01.

Node type	memory	computation
Management node (cluster creation)	100 MB	5 min
Management node (cluster update)	10 MB	2 ms
Management node (prediction)	50 MB	50 ms
Worker node	10 KB	0.01ms

Table 5: Total InfoTrack overhead for tracking 100 metrics on 5000 nodes.

we also plot the average CPU load and standard deviation of CPU load for each hour averaged over days in Figure 11. The performance pattern can be well explained by the same pattern in the average CPU load and average standard deviation. The correlation-aware tracking models can achieve higher compression ratios when average CPU load is high and average standard deviation is low.

4.5 Micro-benchmark Experiments

Now we show the results of micro-benchmark experiments of our InfoTrack system on PlanetLab. This set of experiments involves 276 PlanetLab nodes and we use InfoTrack to track the whole set of 66 features in every 10 seconds over 24 hours starting from August 26, 2008, and the average CPU load and average Load5 are 79.5% and 6.87, respectively, during that period. In Figure 12, we show the total tracking cost in terms of total updates per second of InfoTrack with an error bound value of 0.01 and the original tracking cost without any compression divided by 10 over 24 hours[1]. Note that in this set of experiments the Kalman Filter and adaptive UPGMA are used as the temporal and spatial correlation approach, respectively. As shown in Figure 12, our system significantly reduces the tracking cost by more than one order of magnitude with a tight error bound of 0.01. By suppressing remote information update, our approach can reduce both monitoring network traffic and end-system resource consumption. In particular, InfoTrack reduced monitoring network traffic to be under 5 Kbps. More importantly, fewer updates also reduce database write on the management node and save system resource consumption (e.g., CPU, memory, database capacity, disk storage, power) for processing or storing monitoring data.

In many real-world large-scale networked systems such as enterprise data centers, the number of nodes can easily grow over thousands or even tens of thousands, in which case the original monitoring network traffic can exceed tens of Mbps. InfoTrack can significantly reduce the tracking cost and is well suited for tracking dynamic information in large-scale networked systems.

We now evaluate the resource overhead of the InfoTrack system to verify that our approach is light-weight. Table 5 summarizes the overhead measurement results. We run the management node software on a desktop machine with 1.2GHz CPU and 1G memory to track M attributes on N nodes, where M varies from 20 to 100 and N varies from 500 to 5000. For 5000 nodes, the k-means algorithm creates clusters for one feature under one minute, and the UPGAM algorithm under five minutes. The memory consumption for creating clusters is under 2MB for k-means, and 100MB for UPGMA. Note that the cluster creation is rarely invoked. Once the clusters are created, maintaining and updating clusters is light-weight. With 5000 nodes and 100 attributes, the cluster update time is under 2ms, and the memory consumption of the management node is under 10MB. For metric value prediction, the last value approach consumes 2MB memory to store last values. With 5000 nodes and 100 attributes, the prediction time using Kalman filter is under 50ms, and the memory consumption is under 50MB.

The InfoTrack software running on each monitored worker node is very light-weight. We run the software on the same desktop machine. With 100 attributes, the prediction time using Kalman filter is under 0.01ms, and the memory consumption is under 10KB.

9[1]Showing 1/10 of original tracking cost allows us to see the varying compression performance achieved by InfoTrack.

5. RELATED WORK

Distributed information management is critical for managing large-scale networked systems. For example, both the CoMon PlanetLab monitoring service [1] and the Grid Monitoring/Discovery Service (MDS [10]) have proven extremely useful for their user communities. However, for practical purposes, both systems are statically configured with long update interval (e.g., five minutes for the CoMon infrastructure). Previous work (e.g., Astrolabe [27], SDIMS [30], Mercury [7], SWORD [23],NodeWiz [6] and PIER [14]) has proposed to leverage decentralized structures to achieve scalable information management. Other research work (e.g., InfoEye [21], STAR [17]) has proposed to trade off information coverage or information precision for lower monitoring cost. Different from the above work, the focus of our research is on exploring correlation patterns to achieve self-compressing continuous information tracking, which can be used to not only answer various information queries but also extract important system patterns to guide system management decisions.

Distributed information monitoring has been recognized by recent work as an important component for system management. Singh et al. developed a declarative query system for distributed system monitoring and problem diagnosis [26]. AjaxScope [19] is a distributed Web application monitoring platform using online instrumentation of JavaScript code. FDR [28] is an online system call tracing tool used for misconfiguration troubleshooting. FDR focuses on system call compression while our work focuses on compressing numerical value metric tracking traffic by exploring spatial and temporal correlations.

Exploring correlation patterns among distributed data sources have been studied under different context such as sensor network monitoring [29, 20, 11], distributed event tracking [16], and resource discovery [9]. Although the general idea of exploring temporal and spatial correlations is not new, we shall emphasize applying the idea to distributed information tracking over large-scale networked systems requires non-trivial system analysis and design. In our case, it means discovering dynamic spatial and temporal correlation patterns among distributed information sources using light-weight methods instead of assuming a specific probabilistic model (e.g., Gaussians) as in wireless sensor networks. To the best of our knowledge, our work makes the first step to combine temporal and spatial correlation discovery for reducing distributed information tracking cost in large-scale distributed production systems.

6. ACKNOWLEDGEMENT

This work was sponsored in part by NSFC grant 60703058, IBM Faculty Award, CACC Award, and NCSU startup funding.

7. CONCLUSION

In this paper, we have presented InfoTrack, a new self-correlating predictive information tracking system for managing large-scale distributed production systems. InfoTrack explores both temporal and spatial correlations in the distributed system to suppress remote information update while preserving information coverage and precision. InfoTrack integrates both metric value predictions and adaptive clustering algorithms to reduce information tracking cost. We have implemented the InfoTrack system and tested it on the PlanetLab tracking 66 dynamic attributes on more than 300 distributed hosts. We learned the following lessons from our prototype implementation and experiments: 1) spatial and temporal correlation patterns exist in real-world production systems and can be efficiently discovered using light-weight schemes; 2) correlation-aware information tracking can easily achieve more than 95% compression ratio (e.g., percentage of reduced remote information updates) for tracking relatively stable attribute values within 0.01 error bound; 3) for highly dynamic metrics, our system can achieve more than 50% compression ratio within a tight error bound of 0.01, and more than 90% compression ratio within an error bound of 0.05. As part of our on-going work, we are deploying and testing the InfoTrack system on more complicated commercial monitoring system such as IBM Tivoli running on large-scale distributed production systems such as VCL [4] at NCSU and enterprise data centers, and explore more advanced data modeling techniques, for example, the Minimum Description Length data modeling technique [12], to achieve better prediction of data variations.

8. REFERENCES

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

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

[3] NCSU Information Monitoring System. http://dance.csc.ncsu.edu/projects/infoscope.

[4] NCSU Virtual Computing Lab. http://vcl.ncsu.edu/.

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

[6] Sujoy Basu, Sujata Banerjee, Puneet Sharma, and Sung-Ju Lee. NodeWiz: Peer-to-peer Resource Discovery for Grids. In *Proceedings of IEEE/ACM GP2PC*, 2005.

[7] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting scalable multi-attribute range queries. In *SIGCOMM 2004*, August 2004.

[8] R. G. Brown. *Introduction to Random Signal Analysis and Kalman Filtering*. Wiley, 1983.

[9] M. Cardosa and A. Chandra. Resource Bundles: Using Aggregation for Statistical Wide-Area Resource Discovery and Allocation. *Proc. of ICDCS*, 2008.

[10] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *HPDC-10*, 2001.

[11] A. Deshpand, E. C. Guestrin, and S. R. Madden. Model-driven data acquisition in sensor networks. In *Proceedings of VLDB*, 2002.

[12] P.D. Grunwald. *The Minimum Description Length Principle*. MIT Press, 2007.

[13] X. Gu, S. Papadimitriou, P. S. Yu, and S. P Chang. Toward Predictive Failure Management for Distributed Stream Processing Systems. *Proc. of ICDCS*, 2008.

[14] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the internet with PIER. In *Proceedings of 29th VLDB Conference*, 2003.

[15] A.K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[16] Ankur Jain, Edward Y. Chang, and Yuan-Fang Wang. Adaptive stream resource management using Kalman filters. In *Proceedings of SIGMOD*, 2004.

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

[18] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering*, 82:35–45, 1960.

[19] E. Kiciman and B. Livshits. AjaxScope: A Platform for Remotely Monitoring the Client-side Behavior of Web 2.0 Applications. In *Proceedings of SOSP*, 2007.

[20] S. Krishnamurthy, T. He, G. Zhou, J. A. Stankovic, and S. Son. RESTORE: A Real-time Event Correlation and Storage Service for Sensor Networks. *Proc. of International Conference on Networked Sensing Systems*, 2006.

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

[22] J. MacQueen. Some methods for classification and analysis of multivariate observations, 1967.

[23] David Oppenheimer, Jeannie Albrecht, David Patterson, and Amin Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In *HPDC-14*, July 2005.

[24] Manish Parashar and Salim Hariri. Autonomic Computing: An Overview. In *UPP 2004*, Mont Saint-Michel, France, 2004.

[25] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. In *HotNets-I*, Princeton, New Jersey, October 2002.

[26] A. Singh, P. Maniatis, T. Roscoe, and P. Druschel. Using Queries for Distributed Monitoring and Forensics. *Proc. of Eurosys*, 2006.

[27] Robbert van Renesse, Kenneth Birman, and Werner 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.

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

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

[31] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox. Ensemble of models for automated diagnosis of system performance problems. *Proc. of DSN*, 2005.