Adaptive Sampling and Statistical Inference for Anomaly Detection

A Thesis

Submitted to the Faculty

of

Drexel University

by

Tingshan Huang

in partial fulfillment of the

requirements for the degree

of

Doctor of Philosophy

December 2015

© Copyright 2015
Tingshan Huang. All Rights Reserved.

Dedications

To my fiancé Shuyang and my parents who are always there for me.

Acknowledgments

I would like to express my deepest gratitude to my advisors, Drs. Nagarajan Kandasamy and Harish Sethu, for their continuous inspiration, support and guidance in the last 65 months. I am grateful to Drs. Kandasamy and Sethu for introducing me into the field of anomaly detection at the first place and for their constant encouragement in my work and career. I appreciate their time and effort in delivering knowledge, insight, guidance and suggestions that made this dissertation possible. I am fortunate to have two co-advisors who are outstandingly knowledgeable and inspiring. What I have learned from them has not only reshaped my way of thinking, but also my attitude toward life.

Many thanks are due to the members of my committee, Drs. Nagarajan Kandasamy, Spiros Mancoridis, James Shackleford, Steven Weber and Harish Sethu. I would like to thank them all for agreeing to spend their precious time to serve on my committee and for providing valuable feedback that allows me to improve my work. I would also like to express my gratitude to Drs. Matthew Stamm and John Walsh for their helpful suggestions on my work.

I am also thankful to my friends and colleagues at Drexel University, especially Xiaoyu Chu, Rui Wang, Yao Yu, Bradford Boyle, Raymond Canzanese, Guyue Han, Salvador DeCelles and Ni An, for their invaluable help and friendship. I am grateful for the assistance provided by ECE staff over the years, especially those from Chad Morris, Kathy Bryant, Tanita Chapelle, Phyllis D. Watson and Sean Clark.

Finally, I owe a debt of gratitude to my family, especially my fiancé Shuyang Chen, for their continuous understanding, encouragement, and love.

Table of Contents

List of Tables

List of Figures

Abstract

Adaptive Sampling and Statistical Inference for Anomaly Detection

Tingshan Huang

Advisors: Drs. Nagarajan Kandasamy and Harish Sethu

Given the rising threat of malware and the increasing inadequacy of signature-based solutions, online performance monitoring has emerged as a critical component of the security infrastructure of data centers and networked systems. Most of the systems that require monitoring are usually large-scale, highly dynamic and time-evolving. These facts add to the complexity of both monitoring and the underlying techniques for anomaly detection. Furthermore, one cannot ignore the costs associated with monitoring and detection which can interfere with the normal operation of a system and deplete the supply of resources available for the system. Therefore, securing modern systems calls for efficient monitoring strategies and anomaly detection techniques that can deal with massive data with high efficiency and report unusual events effectively.

This dissertation contributes new algorithms and implementation strategies toward a significant improvement in the effectiveness and efficiency of two components of security infrastructure: (1) system monitoring and (2) anomaly detection. For system monitoring purposes, we develop two techniques which reduce the cost associated with information collection: i) a non-sampling technique and ii) a sampling technique. The non-sampling technique is based on compression and employs the best basis algorithm to automatically select the basis for compressing the data according to the structure of the data. The sampling technique improves upon compressive sampling, a recent signal processing technique for acquiring data at low cost. This enhances the technique of compressive sampling by employing it in an adaptive-rate model wherein the sampling rate for compressive sampling is adaptively tuned to the data being sampled. Our simulation results on measurements collected from a data center show that these two data collection techniques achieve small information loss with reduced monitoring cost. The best basis algorithm can select the basis in which the data is most concisely represented, allowing a reduced sample size for monitoring. The adaptive-rate model for compressive sampling allows us to save 70% in sample size, compared with the constant-rate model.

For anomaly detection, this dissertation develops three techniques to allow efficient detection of anomalies. In the first technique, we exploit the properties maintained in the samples of compressive sampling and apply state-of-the-art anomaly detection techniques directly to compressed measurements. Simulation results show that the detection rate of abrupt changes using the compressed measurements is greater than 95% when the size of the measurements is only 18%. In our second approach, we characterize performance-related measurements as a stream of covariance matrices, one for each designated window of time, and then propose a new metric to quantify changes in the covariance matrices. The observed changes are then employed to infer anomalies in the system. In our third approach, anomalies in a system are detected using a low-complexity distributed algorithm when only steams of raw measurement vectors, one for each time window, are available and distributed among multiple locations. We apply our techniques on real network traffic data and show that these two techniques furnish existing methods with more details about the anomalous changes.

1. Introduction

This dissertation develops new algorithmic strategies for system monitoring and anomaly detection for improved security and reliability. It borrows from and contributes to three fields: signal processing, data science and system security. Our contribution to signal processing is through enhanced techniques based on recent ideas in compressive sampling to accomplish resource-efficient system monitoring. Our contribution to data science stems from improved efficiency in the real-time detection of changes in the features of a data stream. Our contribution to system security arises from the application of the techniques developed in the aforementioned fields for intrusion detection and system reliability.

Given the rising threat of malware and the increasing inadequacy of signature-based solutions, online performance monitoring has emerged as a critical component of the security infrastructure of data centers and networked systems. In fact, the last few years have seen a significant shift in the security industry—from solutions based on end-system anti-virus products to network-wide monitoring of anomalies [1, 2]. Recent well-known intrusions have further accelerated this trend and led to wide acknowledgment of the need for anomaly detection for effective system security [3, 4]. FireEye Inc., a company that provides anomaly detection solutions, has increased its revenue by 50% in just 2014 [5]. Detection of anomalous patterns and mitigation actions in response to anomalies are now considered integral to effective system monitoring. Such monitoring helps establish a foundation of security and has become imperative in almost all variety of systems vulnerable to security threats—a data center, a cloud computing infrastructure, computers on a LAN, or routers and systems in the Internet.

System monitoring provides system administrators the information necessary to assess and improve their current security state. With monitored information, system administrators can proactively report and repair system vulnerabilities. The monitored information can also be stored to serve as a detailed log, allowing system managers to browse historical data to identify and uncover the details of an attack or a system failure. Besides, the monitoring drives long-term decisions such

as capacity planning to improve the overall utilization of system resources.

The growing Internet of Things makes system monitoring challenging. As more devices are connected to the Internet, a system becomes more vulnerable to designed attacks. Many everyday appliances such as printers and air conditioners have become possible gateways for malware, leading to a strikingly increased number of data breach incidents over the last few years [6]. Furthermore, most of these systems that require monitoring are usually large-scale, highly dynamic and time-evolving. This fact adds to the complexity of both monitoring and the underlying techniques for anomaly detection. Therefore, securing modern systems calls for efficient monitoring strategies and anomaly detection techniques that can deal with massive data with high efficiency and report unusual events effectively.

This dissertation contributes new algorithms and implementation strategies toward a significant improvement in the effectiveness and efficiency of two components of security infrastructure: system monitoring and anomaly detection. For system monitoring purposes, we propose two techniques which reduce the cost associated with information collection. One of our proposed techniques is based on compression, a non-sampling technique. The new idea of this technique is to automatically select the basis for compressing the data according to the structure of the data. Our second technique for resource-efficient system monitoring is a sampling technique which improves upon compressive sampling, a recent signal processing technique for acquiring data at low cost. This technique enhances the technique of compressive sampling by employing it in an adaptive-rate model wherein the sampling rate for compressive sampling is adaptively tuned to the data being sampled. For anomaly detection, we propose to apply state-of-the-art anomaly detection techniques directly on the compressed form of data. Furthermore, we develop a new metric based on principal component analysis (PCA) to quantify the anomalous state of a system and use this metric to detect anomalies.

In the rest of this chapter, we introduce the background that forms the foundation of this work, related research and an extended summary of the contributions of this dissertation. Section 1.1 introduces the challenges of system monitoring and describes the model assumed in this dissertation. Section 1.2 presents the techniques of applying compression and compressive sampling for system monitoring in details. Section 1.3 covers the development of anomaly detection techniques and

related research using PCA. Section 1.4 presents two proposed strategies for system monitoring that are based on compression and compression sampling respectively. Section 1.5 summarizes the contributions of this dissertation for anomaly detection.

1.1 Resource-efficient system monitoring

Consider a data center with hardware/software sensors recording performance-related data of the system as shown in Figure 1.1. The monitoring information may include hardware factors such as the runtime state of devices, software factors such as network activity within the data center and memory usage of virtual machines, and environmental factors such as the temperature at different locations. The collected information is then transmitted over a network to a monitoring station where data analysis and visualization are performed before the information is stored.

These measurements are important for making decisions on thermal systems and workload redistribution of the data center. Therefore it is important that the collected information should reflect the real-time status of the data center. However, collecting these measurements has its costs. Sensors that add sensing-related code within the application code to monitor the performance-related data can easily interfere with the online application. Transmitting the collected information over the network consumes network bandwidth. Furthermore, if the sensors communicate with the monitoring station over a wireless network, both the act of collecting information at the sensors and transmitting the collected information over the wireless network consume power. Logging the data for future use, such as analysis aimed at capacity planning, consumes disk space. It is therefore desirable for the monitoring system to minimize the above-described costs.

As shown in Figure 1.2, there are two major categories of techniques for monitoring: monitoring with known correlation and monitoring with unknown correlation. In monitoring with known correlation, the relationship between performance-related measurements is assumed to be known and exploited to reduce the cost of monitoring. In monitoring with unknown correlation, the relationship between performance-related measurements to be collected is unknown, and each one of the performance-related factors is monitored independently.

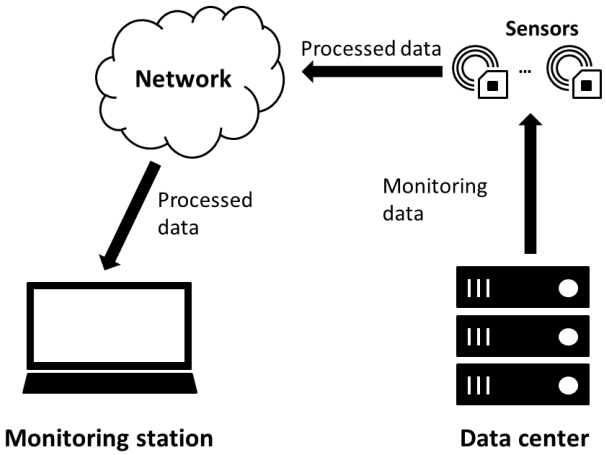


Figure 1.1: The system monitoring model in the case of monitoring a data center.

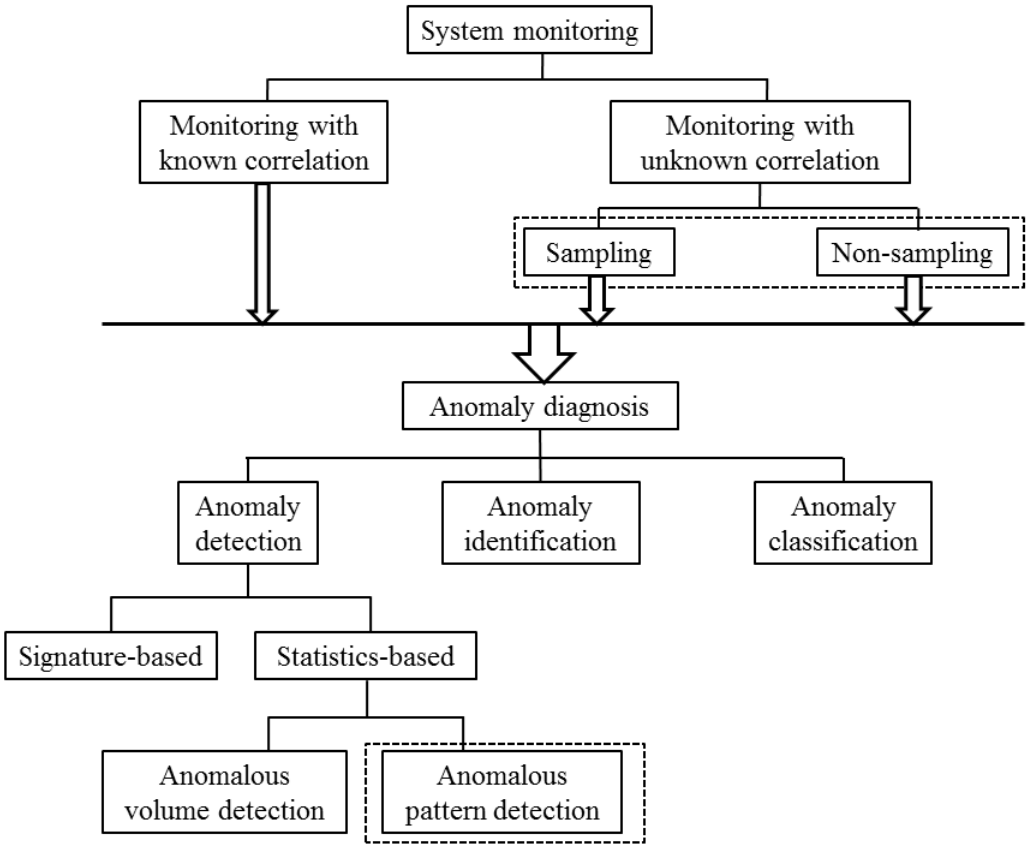


Figure 1.2: Categorization of methods for system monitoring and anomaly detection. The categories to which the proposed methods of this dissertation belong are enclosed in the dotted lines.

1.1.1 Monitoring with known correlation

In the first category of monitoring techniques, performance-related measurements are considered as multiple variables that are correlated, and their correlation is assumed to be known prior to the monitoring. This correlated relationship is exploited during the data collection to reduce monitoring cost.

Consider a data center where environmental measurements (e.g., the fan temperature) and software/hardware factors (e.g., the CPU utilization and power usage of each machine) are collected at physically distributed remote sensors and then transmitted to a monitoring station for analysis. The monitoring factors are correlated. For example, temperature readings are similar at nearby locations. When the CPU utilization of a machine is high, so is its fan temperature and power usage. The correlation between measurements can be exploited to reduce the cost associated with the data collection at each sensor as well as the cost associated with transmitting the data to the monitoring station.

Clustering is one of the methods that exploit the correlation between measurements to reduce the size of data transmitted over the network [7, 8, 9, 10]. This method divides sensors into subsets, and sensors whose measurements are correlated are put in the same subset. These subsets are also referred to as clusters. Within each cluster, one sensor is selected as the cluster head, and each sensor only needs to communicate to its cluster head. The cluster head is responsible for collecting the raw data, i.e., measurements from all the other sensors in its cluster, and sending the aggregated information to the monitoring station. Instead of using fixed cluster heads, techniques discussed in [11, 12, 13, 14] apply self-organization to dynamically assign a cluster head to each cluster.

The aggregation of the raw data performed by the cluster head exploits the correlation within the data to reduce information redundancy [15, 16]. Therefore, the size of the aggregated data is smaller than the overall size of the raw data collected by all the sensors. As a result, the size of the data transmitted over the network and then logged at the monitoring station can be reduced significantly by employing the clustering method.

The monitoring cost can be further reduced by reducing the cost associated with the data collection at each sensor. For example, methods such as distributed compression [17, 18] and distributed

source coding [19, 20] can be employed at sensors within the same cluster to gather correlated data. It has also been proposed that some sensor nodes can be turned off, or go to sleep, to reduce sampling cost [21, 22].

1.1.2 Monitoring with unknown correlation

In the second category wherein the correlation between measurements is unknown, each of the performance-related measurements has to be collected independently. Without any assumption on the correlation between measurements, techniques in this category can be applied on a broader range of data. One straightforward method is to log the mean value of data over each fixed period of time. This method uses a sliding time window with a predetermined length, and records the mean of the monitored data within each time window. The series of mean values are then used for data analysis [23]. While it is easy to implement and requires low memory, this method has a few drawbacks. First, the output of averaging is sensitive to the window length. A larger window size leads to a smoother data series, while interesting data patterns within each time window get erased by the act of averaging. Besides, to calculate the mean, it is necessary to aggregate all the observed data within a time window. However, such aggregation is impractical for some applications. For example, inspecting every packet passing through each router of a backbone network for network traffic monitoring causes unacceptable delay [24].

One traditional strategy for monitoring is random sampling [25, 26], where each one of the data points is chosen to be collected with a probability that is predetermined by the sampling rate. Random sampling is widely used because it requires very little memory consumption and CPU power on the monitoring nodes. In the case of traffic monitoring, random sampling can be applied on the packets to examine the ongoing traffic without impairing the network traffic. However, the technique of random sampling can be biased for some applications [25]. When a monitoring system is used to estimate flow statistics, a router keeps the statistics of active flows that are passing through. The obtained result is useful for network monitoring and accounting, and helps identify anomalies in the network. Using random sampling, packets that belong to a larger traffic flow are more likely to be sampled than those which belong to a small flow. As a result, the estimated flow distribution using the randomly selected samples is biased toward larger flows. Data with such bias can lead to

incorrect decisions by network operators [27].

Sampling with priority is another technique used to remove the bias mentioned above [28]. Given an ultimate goal for monitoring, the sampling rate for each data series is adjusted based on the contribution of the data to the goal. For example, Meng *et al.* propose a method in [29] to detect violations that are defined as extreme values exceeding some threshold in the performance-related measurements of data centers. The sampling rate is updated dynamically: it is increased when the likelihood of detecting a violation is high, and decreased when the likelihood of detecting a violation is low. Similarly, Wang *et al.* propose the Residual-Geometric sampling for monitoring the distribution of network traffic flow at a router [30]. In this sampling strategy, a packet belonging to a new traffic flow is sampled with a predetermined probability, while packets belonging to the existing traffic flows are always sampled. The collected samples are then used to estimate traffic flow distribution, and the estimation result is more accurate than is possible with random sampling. The techniques of sampling with priority are, however, application specific, and it is difficult to extend them for other applications.

1.2 Resource-efficient system monitoring using compression and compressive sampling

In this dissertation, we consider the following two methods for monitoring: compression and compressive sampling. These two methods are monitoring techniques with unknown correlations and therefore can be applied universally. In particular, compression is a non-sampling method, while compressive sampling is a sampling method. They have been widely applied for they allow a good reconstruction of the original data with small sample sizes.

1.2.1 Resource-efficient system monitoring using compression

The system monitoring using compression involves five steps, as shown in Figure 1.3. The first four steps are implemented by the sensors, and the last step is carried out at the monitoring station. In the first two steps, the data is collected and then transformed into another basis. Next, the coefficients in the new basis that are the largest in magnitude are preserved as the compressed form of the original data. The compressed data is then sent to the monitoring station. In the final step,

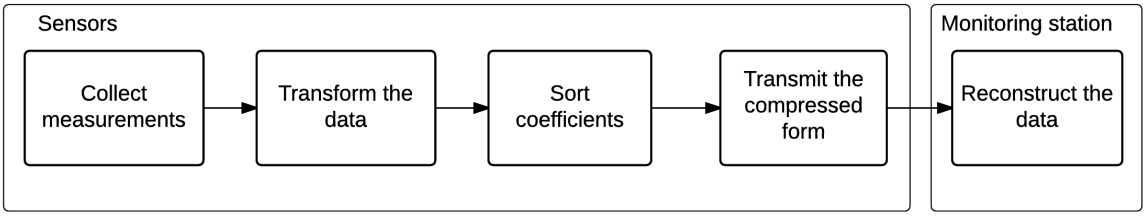
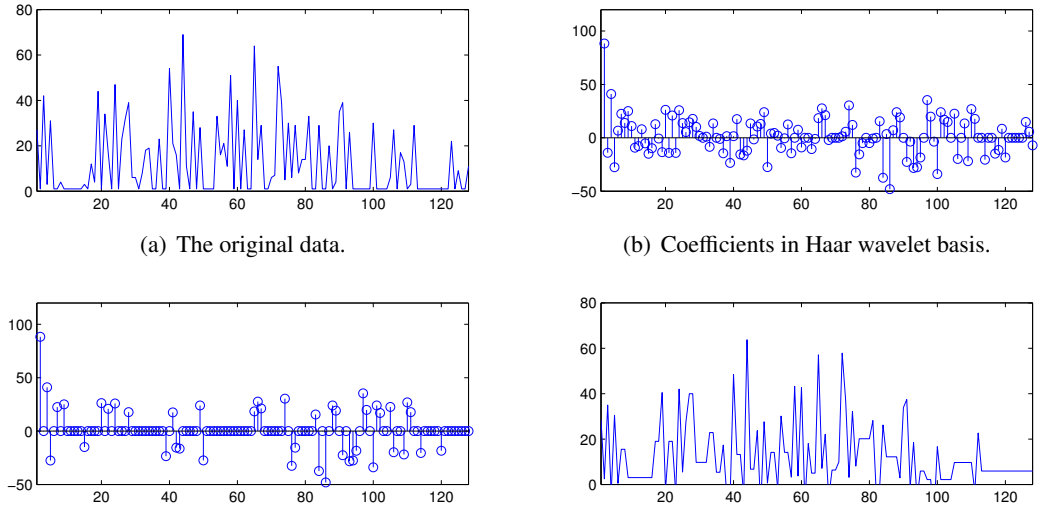


Figure 1.3: The system monitoring model using compression.



(a) The original data.

(b) Coefficients in Haar wavelet basis.

(c) Coefficients with large magnitudes.

(d) Reconstructed data.

Figure 1.4: The form of data at different stages of the monitoring model using compression. (a) the original data to be collected. (b) the representation of the data in the Haar wavelet basis. (c) the compressed form of the data after applying thresholding on the coefficients. (d) the reconstructed data after applying inverse wavelet transform on the compressed form of data.

inverse transform is applied on these coefficients to approximate the original data. An example of the data at different stages of this monitoring model is shown in Figure 1.4.

The technique of compression exploits the property of sparsity, which shows how concisely the data can be expressed in one basis [31]. Under the basis in which the data is more sparsely represented, fewer coefficients need to be preserved in the compressed form. Therefore, it is desirable to find a basis in which the data to be collected is sparse.

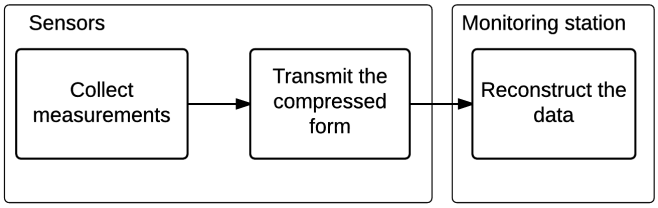
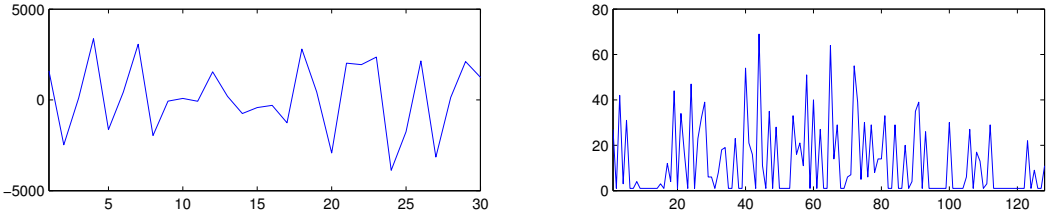


Figure 1.5: The system monitoring model using compressive sampling

1.2.2 Resource-efficient system monitoring using compressive sampling

Compressive sampling, also known as compressive sensing or compressed sensing, is a sampling technique that guarantees accurate reconstruction of sparse data from only a small amount of samples [32]. Similar to compression, a compressed form of the original data is obtained in the sampling process. However, the sampling process of compressive sampling is simply multiplying the original data in the form of a vector with a sensing matrix. Besides, the number of the resulting samples is far fewer than the length of the original data. Reconstruction algorithms are applied on the obtained samples to approximate the original data. A class of reconstruction algorithms called basis pursuit or iterative hard thresholding pursuit (HTP) [33] is used to reconstruct the original data. The model of monitoring using compressive sampling consists of three steps, as shown in Figure 1.5. In the first two steps, the sensors sample the data in a compressed form and send the compressed data to the monitoring station. In the final step, the monitoring station applies the reconstruction algorithm to recover the original data from the samples. Using the same original data as in Figure 1.4, we show in Figure 1.6 the compressed form of data and the reconstructed data in the monitoring model using compressive sampling.

Two sets of bases, the sensing basis and the representation basis, are used in compressive sampling. The sensing matrix for collecting the data is composed of the sensing basis, and the representation basis is for representing the data. The compressive sampling theory is based on two properties associated with these two bases, the sparsity property of the data in the representation basis and the property of incoherence between the sensing basis and the representation basis. The sparsity property allows the data to be sparsely represented in the representation basis. When the sensing and representation bases are uncorrelated or incoherent, a spike in one basis will be represented as a



(a) Compressed samples. (b) Reconstructed data.

Figure 1.6: The form of data at different stages of the monitoring model using compressive sampling. The original data is the same as in Figure 1.4. (a) the sampled data that is also in the compressed form. (b) the reconstructed data after applying reconstruction algorithm on the compressed form of data.

spread-out waveform in the other. For example, the Dirac delta basis and the Fourier basis are incoherent. In this case, data shown as a spike in the Dirac basis is spread out when represented in the Fourier basis. When the sensing and representation bases are incoherent, using the sensing basis to sample the data that is sparsely represented in the representation basis is also referred to as incoherent sampling. The property of incoherence allows us to capture the useful information content embedded in the data and condense it into a small amount of data by incoherent sampling.

Reconstruction of the original data using HTP is considered to be exact with high probability if the sample size is greater than a lower bound defined by the sparsity and the coherence [34]. Typically, when the coherence as well as the sparsity are small, we need only a few samples to recover the original data exactly with high probability. It is therefore desirable to find a representation basis in which the data to be collected is sparse and a sensing basis that has small coherence with the representation basis.

1.2.3 Comparing compression and compressive sampling

Both compressive sampling and compression exploit the sparsity property. Since the sample size as a result of compression and compressive sampling is small, the network bandwidth required to transmit the samples to the monitoring station is reduced and so is the hard-disk space required to store them. When operators wish to analyze the original data, there is a way to use numerical optimization to reconstruct the full-length data from the sample set. However, these two methods are fundamentally different.

In compression, the original data is transformed into the representation basis, and the obtained coefficients are sorted to get the compressed form of the original data. The sampling process of compressive sampling, which is multiplying the data with a sensing matrix to obtain the compressed form, is comparatively simpler with a smaller sampling overhead. As a result, the simple sampling process of compressive sampling can significantly reduce the intrusion of monitoring on the system. The sample size as a result of compressive sampling, nonetheless, can be larger than that of compression. The reconstruction of compressive sampling is more complex compared with the inverse transformation employed by compression.

1.3 Anomaly detection

Anomaly detection is a technique that can detect attacks and unusual states of a system. It has been widely applied in system monitoring, such as intrusion detection in a network [35] and in a cloud computing environment [36]. Anomalies are data patterns that do not conform to a well defined notion of normal behavior. These anomalous patterns are usually caused by malicious attacks on a network (e.g., denial-of-service (DoS) attacks, port scans), system breakdown, or measurement faults. For example, a DoS attack can lead to a sudden increased number of requests to one server. The occurrence of a hardware fault in one single machine can lead to run-time computing failures in the cloud computing system. Many of these anomalies can be a threat to the health of a system. It is critical for system operators to apply anomaly diagnosis to detect these unusual patterns promptly and accurately, and identify the irregularities automatically.

Anomaly detection along with anomaly identification and classification constitute anomaly diagnosis, as shown in Figure 1.2. Anomaly identification and classification are also known as root cause analysis. Anomaly detection systems monitor performance-related features and send alarm messages whenever an abnormal change of any kind is observed. Root cause analysis tries to classify the anomaly based on characteristics of the anomalous pattern and identify the underlying cause.

Intuitively, anomaly detection seems no more difficult than performing a comparison: use existing statistical-analysis techniques to compare the performance-related measurements with a statisti-

cal model of normal behavior, and generate an output if there exist any statistical outliers. However, there are many pitfalls in this intuition. First, it is difficult to get statistical definitions of normal behaviors. Traffic of a backbone network, for example, varies significantly all the time, and change of settings at the routers can lead to different traffic patterns. Besides, the normal behavior of a system keeps evolving all the time, thus it may take a long time to find a normal pattern. Even if we have a model for the normal pattern, the detection result is sensitive to detection settings such as the boundary between normal and abnormal behaviors. Furthermore, limitations on available measurements will degrade the performance of an anomaly detection system.

With these challenges, two basic categories of techniques have been developed for anomaly detection as shown in Figure 1.2: signature-based and statistic-based.

1.3.1 Signature-based techniques for anomaly detection

Signature-based techniques detect anomalies by looking for a pattern that matches signatures of known anomalies [36]. In [37], Feather *et al*. use a signature matching mechanism to detect failures of an Ethernet network. Moore *et al*. use the property of address uniformity in several popular DoS toolkits to diagnose DoS in [38]. Violation detection in data centers detects anomalies that are defined as extreme values exceeding a threshold [29]. Many software systems and toolkits for network security, such as Bro and Snort proposed in [39] and [40], have been developed based on signature-based techniques. However, techniques in this category have the limitation that they can only detect the known anomalies and thus compromise system security when some unknown anomalies occur in the system.

1.3.2 Statistic-based techniques for anomaly detection

The statistic-based techniques do not require any prior knowledge about the anomalies. Instead, techniques in this category look for significant changes in short-term statistics of performance-related data. Therefore, these techniques are also referred to as change detection and can be effective for both known anomalies and unknown anomalies. Based on the type of anomalies they target, the methods in this category can be divided into two subtypes: volume anomaly detection and pattern anomaly detection.

1. The techniques for volume anomaly detection are the methods that detect unusual volume changes in the univariate data. Many methods have been proposed to apply statistical techniques to detection volume anomalies. For example, Barford *et al*. treat anomalies as deviations in the overall traffic volume, and use the traffic variances at different time-frequency scales to distinguish predictable and anomalous traffic volume changes [41]. It has been proposed to apply a variety of time series forecast models (e.g., ARIMA and Holt-Winters) on network traffic, and look for traffic flows with large forecast errors to detect traffic anomalies [42, 43, 44, 45]. For example, the AnomalyDetection R package tool recently released by Twitter on GitHub [45] aims to detect anomalies in time series data. This tool assumes a smooth transition in normal data series such as activities of uploading photos on Twitter, and uses seasonal hybrid extreme studentized deviate (S-H-ESD) test to quantify deviation of a datapoint from its prediction. Datapoints that deviate from their predictions are flagged as anomalies in seasonal univariate time series. This tool is able to detect both global and local anomalies that do not necessarily appear to be extreme values.

2. The techniques for pattern anomaly detection are the methods that detect unusual patterns in the multivariate data. In particular, these techniques explore correlations between different measurements and detect anomalies by checking changes of correlation. For example, Lakhina *et al.* propose to apply PCA in [46] to explore anomalous deviations in the distribution of network-wide traffic, and use the deviations to signal, identify and classify anomalies. Lan *et al.* [47] employ PCA and independent component analysis (ICA) to extract features from the data of large-scale systems, apply unsupervised learning on the extracted features, and identify anomalous nodes of the system as the nodes acting differently from the others. Guan *et al.* propose in [48] to exploit the most relevant principal components (MRPCs) to describe system failures in cloud computing infrastructures, and adaptively identify the detected anomalies.

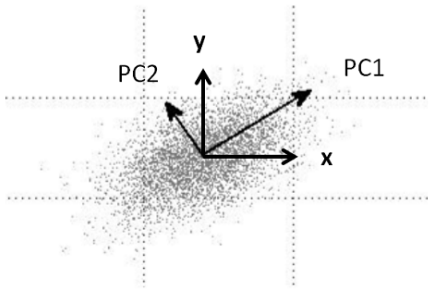


Figure 1.7: Principal components of two-dimensional data.

1.3.3 Principal component analysis for anomaly detection

PCA is a technique for dimension reduction which is widely used for anomaly detection in network [46, 49, 50, 51], cloud computing infrastructure [48, 52] and other large-scale systems [47]. Given a set of observations that are high-dimensional, applying PCA on the data gives us a set of orthogonal axes, which are called principal components and along which the observations are correlated. Furthermore, these principal components are ordered by the strength of correlation the data exhibits along their directions. The correlation captured by each principal component is quantified by the variance of the data along its direction. As a result, the first principal component captures the strongest correlation between the features, the second principal component captures the second strongest, and so on [53]. Figure 1.7 shows two principal components, PC1 and PC2, as the result of applying PCA on a two-dimensional dataset. The first principal component, PC1, gives us the direction along which the data is most correlated.

The performance-related data of a system are inevitably correlated. For example, a correlation exists between the traffic passing through two neighboring routers. The thermal readings at different nodes in a sensor network are correlated, and the readings at two nearby nodes are similar. The response time of an application server is correlated with the CPU utilization of the machine that hosts the application. Because of such correlations, the number of principal components that capture the major patterns is usually smaller than the dimension of the original dataset. As a result, we are able to reduce the dimension of highly correlated datasets and thus reduce the complexity of data analysis by employing PCA.

Feature extraction using PCA usually involves applying PCA on the datasets to generate a com-

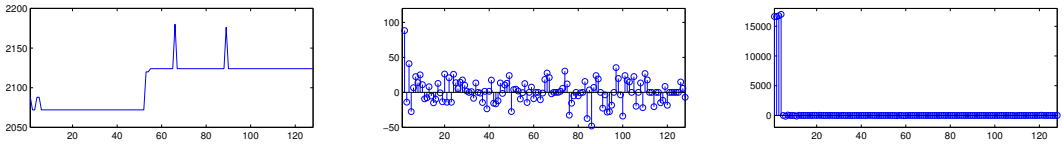
plete set of principal components. Then, a set of principal components are selected such that they account for a certain amount of variance exceeding a predetermined threshold [47, 49, 50, 51, 52]. In [49], the principal components that are chosen as the extracted features form the *normal subspace*, and the *anomalous subspace* is composed of the rest of the principal components. An underlying assumption in [49] is that anomalies exhibit correlations that are different from those captured by the normal subspace. Based on this assumption, the projection of data without anomalies onto the normal subspace should be large, while its projection onto the anomalous subspace should be small. Similarly, the projection of anomalous data onto the anomalous subspace should not be small. The anomaly detection is implemented by projecting the test data onto the anomalous subspace, and sending an alert if the projection is above a threshold. This method of applying PCA to construct normal and anomalous subspaces for anomaly detection is referred to as the *subspace method*.

1.4 Contribution to resource-efficient monitoring

As shown in Figure 1.2, one part of this dissertation is to develop two simple yet effective monitoring tools. One of the monitoring tools is based on compression, a non-sampling technique. We term this monitoring tool as C-MON. The other monitoring tool called CS-MON employs compressive sampling, a sampling technique. The technique of compression and compressive sampling are for monitoring with unknown correlation, and thus our monitoring tools can be applied to monitor each one of the performance-related measurements. With no assumption on the correlation between the monitored data, these tools can be applied for general purposes.

1.4.1 C-MON: monitoring using compression in the best basis

In C-MON, compression is applied on performance-related measurements, and the compressed form of the data is sent to the monitoring station for data analysis. Note that the data can be represented at different sparsity levels under different bases. In Figure 1.8, we show that the sparsity of a dataset is different in two wavelet bases. For the basis in which the data is more sparsely represented, fewer coefficients need to be preserved in the compressed form. Using the same sample size, both compressive sampling and compression can reconstruct the data with greater fidelity if



(a) The original data (b) In Haar wavelet basis. (c) In db2 wavelet basis.

Figure 1.8: The sparsity of data in (a) in two wavelet bases: Haar and db2. (a) An example of data series reflecting the memory allocated to a virtual machine. (b) Representation of the data in Haar wavelet basis. (c) Representation of the data in db2 wavelet basis. The data is more concisely represented in the db2 basis.

we know beforehand the underlying basis in which the data is sparse.

Selecting the appropriate basis under which the measured data is most sparse requires some experimentation using representative training data from the system. We propose to employ a basis selection strategy, *best basis algorithm*, that reduces this burden on the system operator by automatically adapting the representation basis to the structure of the sampled data.

In our design, the best basis should satisfy two conditions. First, the representation of the data in this basis should be sparse. Second, the approximation error as a result of using the compressed form of the data in this basis should be small. Considering these two conditions, we formulate a cost function associated with each candidate basis. There are two terms in this cost function, one for the sparsity and the other for the approximation error [54]. We then choose the best basis as the one that minimizes the cost function.

The system monitoring model of C-MON is shown in Figure 1.9. Comparing with the compression-based monitoring model in Figure 1.3, C-MON has an additional component which implements the best basis strategy. Given a set of candidate bases, C-MON employs the best basis strategy to find the basis in which the data can be most sparsely represented. The data is then compressed in this best basis so that the size of data in its compressed form is the smallest. As a result, the network traffic for sending the monitored information to the monitoring station is reduced by employing the best basis strategy.

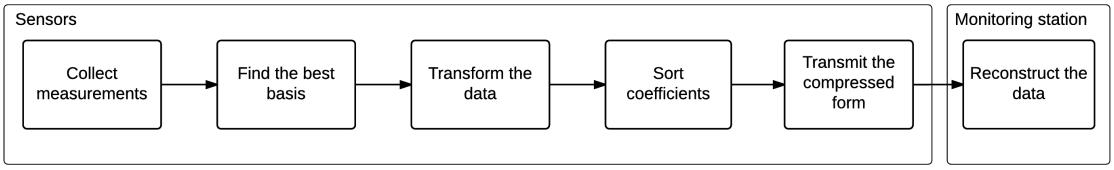


Figure 1.9: The system monitoring model of C-MON.

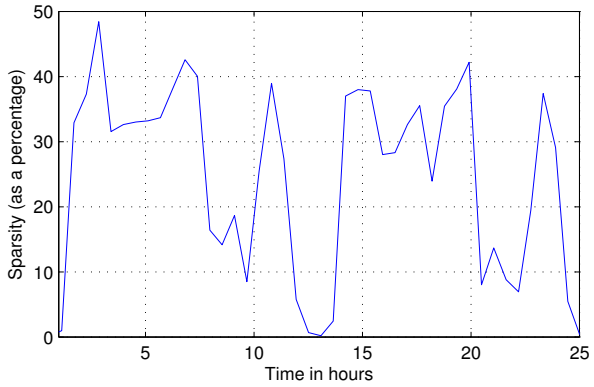


Figure 1.10: Change in the sparsity level over time for the measurement of memory usage of a virtual machine in a data center.

1.4.2 CS-MON: monitoring using adaptive-rate compressive sampling

The other monitoring tool we develop in this dissertation, termed CS-MON, applies the technique of compressive sampling on the data.

The technique of compressive sampling has been widely used for cognitive radio [55, 56, 57, 58], MRI [59, 60, 61, 62], radar imaging [63, 64, 65, 66] and video coding [67, 68, 69] for the underlying sparsity property of the data. It has been rarely used for system monitoring except in [70] for the monitoring of wireless sensor networks, in [71] for the monitoring of power systems, and in [72] for the monitoring of data centers.

In works that apply compressive sampling for system monitoring (e.g., [72]) the sparsity of the data when represented in the underlying basis is assumed to be constant or bounded over time. Under such an assumption, one can use a fixed rate for sampling the data. This assumption of a constant sparsity, however, rarely holds in practice. For example, we measure the memory usage of a virtual machine in our case study of monitoring a data center. In particular, the data records the

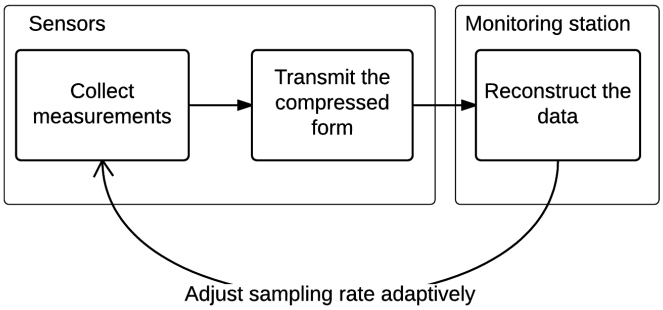


Figure 1.11: The system monitoring model of CS-MON.

amount of memory allocated by processes in the system using malloc() calls. We find this data to be sparse in Haar wavelet basis, and plot in Figure 1.10 the sparsity of the data during a 25-hour period. The sparsity is defined as the percentage of coefficients needed within the Haar wavelet basis to capture 99% of the information contained within the data. The sparsity of the data varies significantly over time. So a simple sampling strategy, say one that chooses a fixed rate based on an upper bound on the sparsity level—between 45-50% in this particular case—incurs a large sample size. We can perform much better in terms of reducing the number of samples needed by dynamically increasing or decreasing the sampling rate as the sparsity changes.

In this dissertation, we improve the compressive sampling strategy in Figure 1.5 by adding a rate adjustment component inspired by the work in [73]. The flow chart for CS-MON, the system monitoring tool using adaptive-rate compressive sampling, is shown in Figure 1.11. Our idea is that by collecting some relevant side information from the data—which could have been under or over sampled during any particular time window—we can estimate the underlying sparsity and predict it for some succeeding time windows. The number of samples collected over these succeeding windows can then be adjusted based on the predicted sparsity values. Our goal is to achieve an overall reduction in sample size without compromising the reconstruction quality.

We apply CS-MON strategy for online monitoring of a data center as a case study. In particular, we show that to achieve the same reconstruction quality, this monitoring strategy is able to reduce the sample size by 70% compared with constant-rate compressive sampling. We also use the reconstructed data of compressive sampling for two cases of anomaly detection: violation detection and trend detection.

1. *Violation detection*. Violations are defined as values that exceed some threshold. These violations are usually performance-related bottlenecks or anomalies. We use the metric of hit rate to evaluate the performance of CS-MON. Here, a hit is defined as a spike in the reconstructed data that matches a similar spike in the original data. Our result shows that this strategy is able to achieve 90% hit rate using about 30% of the original data.

2. *Trend detection*. A decreasing or increasing trend in the data can reflect performance deterioration associated with software aging or resource exhaustion. We use a linear model to fit the reconstructed data, and detect the trend by checking the estimated slope of the linear model. Our result shows that using around 30% of the original data, we are able to detect the increasing trend in the measurements collected from our testbed with high confidence.

1.5 Contribution to anomaly detection

1.5.1 Anomaly detection with compressed measurements

Compressive sampling used in the aforementioned CS-MON allows for a very simple sampling strategy on the local machine. When operators wish to analyze the original signal, there is a way to use numerical optimization to reconstruct the full-length signal from the sample set. The reconstructed signal allows for real-time anomaly detection and diagnosis, and also helps drive decisions of a longer-term nature such as intelligent capacity planning. The recovery process is typically posed as a linear programming (LP) problem and solved under some sparsity assumptions using a class of reconstruction algorithms called basis pursuit or iterative hard thresholding pursuit [33]. Though modern LP solvers are quite efficient, each monitoring station may be responsible for recovering and analyzing hundreds of signals belonging to many servers, making it important to reduce the corresponding overhead.

We propose to apply anomaly detection directly on compressed measurements to reduce the computational cost incurred by the monitoring station, associated with recovering the original signal from the sample set and analyzing it for anomalies. Our work makes the following contributions towards this goal:

- We prove from a theoretical viewpoint that the compressed samples preserve statistical prop-

erties of the original data such as variance and mean. This result allows for the detection of abrupt changes and trends by *directly analyzing* just the compressed samples without having to reconstruct the full-length signal.

- Since the sampling process is just a linear projection of the original data, we also prove that the compressed samples approximately *preserve spectral properties* such as correlation between data points, the length of the data vectors, as well as the distance between two vectors, under such a projection. This result allows for well-known anomaly detection methods such as PCA to be used directly on the compressed samples; performance is almost equivalent to the case in which the raw data is completely available.

We illustrate the usefulness of the approach via case studies using IBM's Trade Performance Benchmark (also known as Trade6). We measure signals from the disk and memory subsystems using a CS-based sampling strategy, and analyze the compressed samples for possible anomalies. The first scenario involves detecting abrupt changes in the signal during which the magnitude exceeds some nominal threshold value. In the second scenario, we wish to detect the gradual deterioration of system performance, say over hours or days, associated with software aging by statistically analyzing the appropriate signals for the existence of trends [74, 75]. We use a long-running Trade6 application having a small memory leak and evaluate the ability of the approach to estimate a positive slope in the compressed data even in the presence of seasonal variations and periodicity in the signal. Finally, we evaluate the efficacy of applying PCA to the compressed samples to detect abrupt changes in the signal.

Abrupt changes can be detected using a sample size of 25% with a hit rate of 95% for a fixed false alarm rate of 5%; trends can be detected with a confidence interval of 95% using a sample size of 6%. Finally, the hit rate achieved by the PCA-based analysis when using the compressed samples to detect abrupt changes is higher than 95% when the false alarm is fixed at 0.5%. The corresponding sample size is about 18%. These results point to the feasibility of adopting a two-step anomaly detection process at the monitoring station: the received compressed data is examined for possible anomalies; if one is suspected, the relevant portion of the signal is fully reconstructed to localize and further analyze the anomaly.

1.5.2 Anomaly detection with the maximum subspace distance

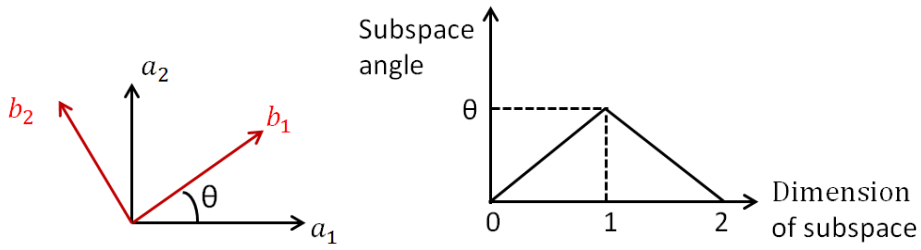
The subspace method proposed in [49] has been widely applied and is shown to be effective in detecting anomalies for traffic monitoring [49] and for cloud computing security [52]. However, this method has the following drawbacks [51, 76]: the detection rate of the method is sensitive to the chosen dimension for the normal subspace, and anomalies that reside within the normal subspace will be missed by the detection tool.

As mentioned earlier, PCA is applied on a set of training data and then the most dominant principal components are selected to form the normal subspace. The dimension of the normal subspace, or the number of principal components, is selected such that it accounts for a predetermined amount of variance in the training dataset. In this way, the detection tool proposed in [49] is able to detect anomalies that do not comply with the correlations captured in the normal subspace. As a result, the range of anomalies that can be detected relies on the training data and user input of threshold for the variance that should be captured by the normal subspace. In the following, we refer to this method as the *variance-based subspace method*.

Considering these drawbacks associated with the variance-based subspace method, we propose a new metric named *subspace distance* to differentiate between two behavior datasets of a system, and use this metric to detect changes in correlation patterns. More importantly, we show that the value of our proposed metric explains the significant correlation changes, if any, between the measured data and the training data. Consequently, this metric provides more information about the detected anomalies.

We choose the second-order statistics, the covariance matrix, to summarize a system behavior. By using the covariance matrix, we place no prior assumption on the distribution of measured data. Besides, the result in [77] shows that anomalies lead to changes in the covariance matrix and that the deviations in a covariance matrix corresponding to different anomalies are different. Inspired by work in [77], we aim to develop a general metric to quantify the difference between two covariance matrices, and use the measured difference to detect and identify anomalies.

The subspace distance is calculated as follows. After applying PCA on the two covariance matrices, one for each of the system behaviors, we obtain two sets of principal components. We



(a) Two sets of subspaces (b) The subspace angle.

Figure 1.12: The example of subspace distance between two sets of principal components. (a) Two sets of principal components: $\{a_1, a_2\}$ and $\{b_1, b_2\}$. (b) The subspace angle between a pair of subsets of principal components when the number of principal components included in each subset increases from 0 to 2. The subspace angle is maximum when the subspace dimension is 1. The maximum subspace angle is θ, i.e., the angle between a_1 and b_1.

then choose a subset from each set of principal components, and calculate the angle between these two subsets. The subspace distance is the maximum angle between any pair of subsets of principal components. The two subsets of principal components corresponding to the maximum angle are the signature patterns of these two system behaviors. In the example shown in Figure 1.12, $\{a_1, a_2\}$ and $\{b_1, b_2\}$ are the principal components of two system behaviors. The maximum subspace angle is θ, the angle between $\{a_1\}$ and $\{b_1\}$. As a result, the subspace distance between these two behaviors is θ and is achieved when $\{a_1\}$ and $\{b_1\}$ are chosen as the signature pattern of each behavior.

Particularly, the metric of subspace distance quantifies the difference between correlation patterns of two system behaviors. These two behaviors can be the normal and anomalous behaviors of a system, in which scenario the metric quantifies the anomalous change in the system behavior. They can also be the behaviors of a system at different stages, in which scenario the metric quantifies the change of system status.

Given two system behaviors, one during the training phase and the other one during the test phase, we derive the maximum subspace distance and obtain two subsets of principal components corresponding to the maximum subspace distance. We then examine these two subsets of principal components for the significant correlation change caused by anomalies occurred during the test phase. We refer to this method as the *distance-based subspace method*. In the following, we present two efficient algorithms that allow us to obtain information about the correlation changes in centralized and decentralized scenarios.

The centralized algorithm for subspace distance estimation

The definition of our proposed subspace distance shows that two complete sets of principal components need to be calculated first. However, it can be computationally expensive to calculate principal components for datasets with high dimensions (e.g., network flow data from each source IP address can have up to 2^{32} dimensions). We propose an efficient algorithm that estimates the subspace distance with much lower computational complexity.

The key ideas of our proposed algorithm are listed below.

1. *Reduction of search range*. The subspace distance is found by comparing pairs of subsets of principal components. We prove that the searching range can be reduced from $O(N^2)$ to $O(N)$, where N is the dimension of the measurements.

2. *Using only a subset of principal components*. Instead of calculating two complete sets of principal components, we choose to calculate one principal component for each dataset at a time using the power iteration method [78]. The order for calculating the principal components is the same as the significance of these principal components.

3. *Simplified computation of subspace angle*. We develop a simplified procedure of computing the subspace angle between any pair of subsets of principal components.

4. *Setting a stop condition for searching*. To avoid calculating two complete sets of principal components, we set a stopping condition for the search of subspace distance. We prove that the stop condition of our algorithm can get us an approximation of the subspace distance with bounded error.

We validate each of these ideas with mathematical proofs and experimental results. We then develop a centralized algorithm based on these ideas. We show that the estimated subspace distance as a result of our algorithm is close to the actual subspace distance with significantly lower complexity.

The decentralized algorithm for subspace distance estimation

The data required for computing covariance matrices is usually collected at physically distributed nodes, such as network traffic at different routers and temperature readings at different

sensor nodes. In order to perform the centralized algorithm, a data sink is required to collect all the data from the collection nodes, compute covariance matrices, and run the algorithm.

In a distributed mode where there is no such data sink, a collection node will need to collaborate with its neighbors to estimate principal components in a distributed fashion. In this setting, the covariance matrix is unknown to each node, and the power iteration method for calculating the principal components has to be implemented in a distributed manner. We propose that this is feasible with Gaussian gossiping [79, 80]. In the decentralized algorithm, each node will only need to communicate information with its neighbors and estimate its corresponding entry in the principal components instead of computing the complete vectors of principal components. With partial knowledge about the principal components, each node can compute the subspace angle in a distributed fashion and collaborate with its neighbors to estimate the subspace distance. We derive the tradeoff between the accuracy of the estimation and the volume of communications among the sensing nodes as a result of our decentralized algorithm.

2. An Efficient Strategy for Online Performance Monitoring of Datacenters via Adaptive Sampling

Performance monitoring of datacenters provides vital information for dynamic resource provisioning, anomaly detection, capacity planning and metering decisions. Online monitoring, however, incurs a variety of costs: the very act of monitoring a system interferes with its performance, consuming network bandwidth and disk space. With the goal of reducing these costs of online monitoring, this chapter develops and validates a strategy based on adaptive-rate compressive sampling. It exploits the fact that the signals of interest often can be sparsified under an appropriate representation basis and that the sampling rate can be tuned as a function of sparsity. We use the Trade6 application as our experimental platform and measure the signals of interest—in our case, signals pertaining to memory and disk I/O activity—using adaptive sampling. We then evaluate whether the reconstructed signals can be used for trend detection to track the gradual deterioration of system performance associated with software aging. Our experiments show that the signals recovered by the our methods can be used to detect, with high confidence, the existence of trends within the original signal. We also evaluate the reconstructed signals for threshold-violation detection wherein the magnitude of the signal exceeds a preset value. Our experiments show that performance bottlenecks and anomalies that manifest themselves in portions of the signal where its magnitude exceeds a threshold value can also be detected using the reconstructed signals. Most importantly, detection of these anomalies is achieved using a substantially reduced sample size—a reduction of more than 70% when compared to the standard fixed-rate sampling method. The material presented in this chapter was previously published in [81, 82, 83].

2.1 Introduction

Online performance monitoring of both the IT infrastructure and the physical facility is vital to ensuring the effective and efficient operation of datacenters [84, 85]. Examples of monitoring solutions include the Tivoli Monitoring software from IBM for the IT infrastructure [86] and the

Datacenter Environmental Edge from HP that monitors temperature, humidity, and state of the power network within the datacenter. The monitored information has a variety of uses. It supplies the metering data used by pay as you use pricing models for cloud resources. It also drives real-time performance management decisions such as dynamic provisioning of IT resources to match the incoming workload, detection and mitigation of performance-related hotspots and bottlenecks, and fault diagnosis. In the case of intermittent problems that are hard to isolate, browsing back through historical data can help identify and localize recurring problems affecting the same portion of the IT infrastructure at different times. The monitored information also drives decisions of a longer-term nature; for example, intelligent capacity planning that identifies resources that are over-utilized or under-utilized and aims to improve utilization by adding or removing appropriate resources.

We consider a server cluster wherein software-based sensors embedded within the IT infrastructure measure various performance-related parameters associated with the cluster. These measurements include high-level metrics such as response time and throughput as well as low-level metrics such as processor utilization, I/O, memory, and network activity. The information collected by the sensors is transmitted over a network to a monitoring station for data analysis and visualization. Online monitoring, however, incurs a variety of costs. First, the very act of monitoring an application interferes with its performance. If sensing-related code is merged with the application code, this change may interfere with the timing characteristics of the application or if sensors execute as separate processes, they contend for CPU resources along with the original application. Transmitting the monitored data over a network consumes bandwidth. Finally, logging the data for future use such as analysis aimed at capacity planning consumes disk space. So, when monitoring a large-scale computing system it is desirable to minimize the above-described costs, which is the focus of this chapter.

Traditional methods of sampling signals use Shannon's theorem: the sampling rate must be at least twice the signal bandwidth to capture all the information content present in the signal. However, the theory of *compressive sampling* (*CS*), a recent development in signal processing, states that we can recover a certain class of signals from the original measurements using far fewer samples than that used by techniques that rely on Shannon's theorem [32, 34, 87, 88]. Compressive sampling contends that many natural signals are *sparse* in that they have concise representations

when expressed in the proper basis. This property is used to capture the useful information content embedded in the signal and condense it into a small amount of data. In other words, one can acquire these signals from the underlying system directly in a compressed form. From a viewpoint of reducing the costs associated with monitoring, compressive sampling allows for a very simple sensing strategy. Rather than tailoring the sensing scheme to the specific signal being measured, a signal-independent strategy such as randomized sampling can be used, significantly reducing the intrusion of monitoring on application performance. Also, since signals are acquired directly in compressed form, the network bandwidth required to transmit these few samples to the monitoring station is reduced and so is the hard-disk space required to store them. When operators wish to analyze the original signal, there is a way to use numerical optimization to reconstruct the full-length signal from the sample set.

We develop and experimentally validate an *adaptive sampling strategy within the CS framework* for online monitoring, exploiting the fact that the signals of interest often can be sparsified under an appropriate representation basis and that the sampling rate itself can be tuned as a function of sparsity. We use IBM's Trade6 benchmark, a stock-trading service which allows users to browse, buy, and sell stocks, as our testbed and subject it to a dynamic workload while measuring signals pertaining to the memory and disk I/O subsystems. The following two data sampling, encoding, and signal-recovery strategies are developed and evaluated:

- The sparsity of the data when encoded in a basis function determines the quality of the final reconstruction, especially under low sampling rates. Using the data from the testbed, we show how to find basis functions in which this data can be most concisely represented. More specifically, we describe a basis selection algorithm called *Best Basis* that automatically adapts the representation basis to the structure of the underlying signal being sampled. Using the best-basis algorithm, we develop a strategy called C-MON that takes advantage of signal compressibility to reduce the data transfer and storage costs associated with online monitoring.

- When it can be encoded sparsely using an appropriate basis, we show how the signal can be acquired from the testbed directly in a compressed form. We use *adaptive compressive*

sampling where the key idea is to dynamically tune the sampling rate as the signal sparsity changes: in time windows where the signal is sparse we reduce the sampling rate and in windows where the signal is less concise the rate is increased, all the while ensuring that the chosen sampling rate guarantees a user-defined signal recovery quality. At the monitoring station, the process of recovering the original signal from these samples is posed as a linear programming problem and solved using the hard thresholding pursuit (HTP) approach. We term this strategy for online monitoring as CS-MON.

A major benefit offered by adaptive sampling in the context of datacenter operations is in reducing the overhead of generating, storing, and using performance log files for analysis tasks such as capacity planning, bottleneck detection, and trend forecasting. Here we evaluate the efficacy of detecting threshold violations and trends using the signals reconstructed via C-MON and CS-MON. Under the first scenario, the datacenter operator wishes to detect performance-related bottlenecks or anomalies that manifest themselves as the magnitude of the signal exceeding some nominal threshold value. The performance of the C-MON and CS-MON strategies in this regard is quantified by a hit-rate metric; we show that by selecting the threshold value appropriately, both strategies achieve a hit rate of 90% with a false alarm rate of only 0.1%. In the second scenario, the operator wishes to detect the gradual deterioration of system performance, say over hours or days, associated with software aging [89]. Common causes involve resource exhaustion due to memory leaks and bloat, unreleased network sockets and file locks, and unterminated threads. One way of determining if software aging exists is to statistically analyze the appropriate signals for the existence of trends [74, 75]. We use a long-running Trade6 application having a small memory leak of about 100 KB/minute, and evaluate the performance of C-MON and CS-MON in terms of their ability to estimate a positive slope in the reconstructed data (even in the presence of seasonal variations and periodicity in the signal).

In terms of the sample collection cost, both strategies achieve a notable reduction compared to fixed-rate sampling. The anomalies described above are detected with high confidence using 12–25% of the samples from the original signals in the case of CS-MON and less than 5% in the case of C-MON. We also quantify the computation and storage costs incurred by CS-MON and C-MON to achieve their respective compression gains, finding that CS-MON imposes significantly less burden

on the local server resources than the C-MON strategy.

The chapter is organized as follows. Section 2.2 describes our experimental testbed. Section 2.3 demonstrates C-MON, the compression of signals generated by the testbed using the Best Basis algorithm, and Sections 2.4 and 2.5 discuss CS-MON, the compressive sampling of signals using an adaptive rate strategy. Section 2.6 presents experimental results evaluating the performance of these strategies in recovering the original signals, and in detecting threshold violation and trends. Section 2.7 discusses related work and Section 2.8 provides some concluding remarks.

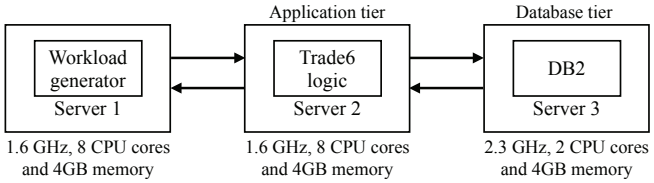
2.2 Experimental Setting

Fig. 2.1(a) shows the system used in our experiments, comprising three servers networked via a gigabit switch. Virtualization of this system is enabled by VMWare's ESX Server running a Linux RedHat kernel. The operating system on the virtual machine (VM) is the SUSE Enterprise Linux Server Edition. The system hosts IBM's Trade6 benchmark, a stock-trading application which allows users to browse, buy, and sell stocks. Users can perform dynamic content retrieval as well as transaction commitments, requiring database reads and writes, respectively. The application logic for Trade6 resides within the IBM WebSphere Application Server, which in turn is hosted by the VM on the server within the application tier. The database component is DB2, hosted on the server running SUSE Enterprise Linux. The database maintains 500 user accounts and information for 3500 stocks.

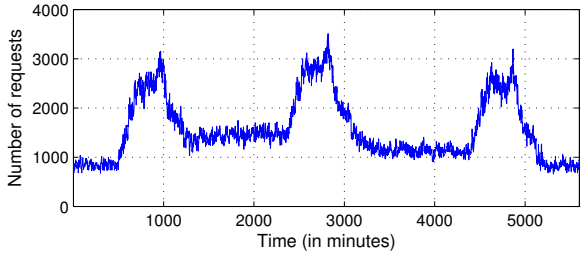
We use httperf [90], an open-loop workload generator, to send a mix of buy/browse transactions to the Trade6 application over a period of 48 hours. The workload traces are synthesized to reflect realistic operating scenarios such as time-of-day variations as well as bursty traffic where request rates vary significantly within short time periods. A sample workload is shown in Fig. 2.1(b), having an average arrival rate of 50 requests per second with a 50/50 mix of buy/browse transactions. Each data point in the figure represents the aggregated workload in a 30-second interval.

The experiments reported in the chapter use the following metrics contained within the `/proc` pseudo file system, specifically the contents of `/proc/meminfo` that reports real-time information about memory usage in Linux systems:

- *Free memory*. This quantity, termed MemFree, reflects the amount of physical memory left



(a) The Trade6 application.



(b) Dynamic workload trace provided to the Trade6 application.

Figure 2.1: (a) The overall system architecture hosting the Trade6 application which implements a stock-trading service that allows users to browse, buy, and sell stocks. (b) Example of workload provided to the testbed in our experiments, plotted in granularity of 30 seconds.

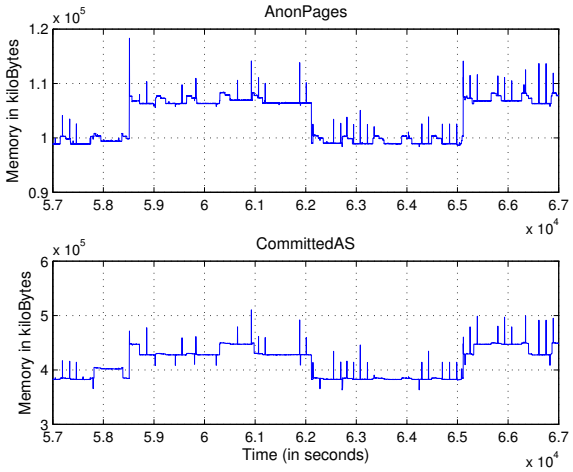


Figure 2.2: Measurements corresponding to the AnonPages and CommittedAS signals collected over a 48-hour operating period.

unused in the system.

- *Committed memory*. This quantity, termed CommittedAS, reflects the total amount of memory allocated by processes in the system using `malloc()` calls, even if the memory has not been used by them as of yet. For example, a process may allocate 1 GB of memory but only

touch 100 MB of it. Although the current memory usage is only 100 MB, the 1 GB allocation is memory that has been committed by the memory subsystem to the process and can be used at any time by the process.

- *Page tables*. This quantity, termed PageTables, reflects the amount of memory dedicated to the lowest level of page tables.

- *Anonymous pages*. This quantity, termed AnonPages, tracks the amount of non-file backed pages mapped to page tables responsible for the user space.

In addition to these features, we also use disk I/O activity measurements (sectors read/written) as part of our evaluation experiments. Our goal is to show that the reconstructed signals can be used to detect various system-level faults. The above-listed features were chosen to support one of the case studies discussed in this chapter: detection of memory leaks. Here we have chosen low-level metrics that are most likely impacted by this fault. Figure 2.2 plots a subset of the features collected during an experimental run of the system lasting 48 hours. The data points are sampled once every two seconds.

2.3 Compressibility of Signals

The fundamental premise behind signal compression is that many natural signals are sparse in that they have concise representations when expressed in the proper basis; this sparsity determines the quality of the subsequent reconstruction. Using the data collected from our testbed, we show how to find basis functions in which this data can be most concisely represented. We also discuss a basis selection algorithm called Best Basis which automatically adapts the representation basis to the structure of the sampled signal.

2.3.1 Sparse Representation of Signals

Let us denote the data to be sampled as $\mathbf{d}$, a vector of length N, and its representation in basis $\mathbf{B}$ as $\mathbf{x}$. In other words, $d = \sum_{i=1}^{N} x_i \mathbf{b}_i = \mathbf{B}\mathbf{x}$, where $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_N]$. For example, if $\mathbf{B}$ is selected to be the Haar wavelet basis, the elements of the vector $\mathbf{x}$ are coefficients of wavelet decomposition

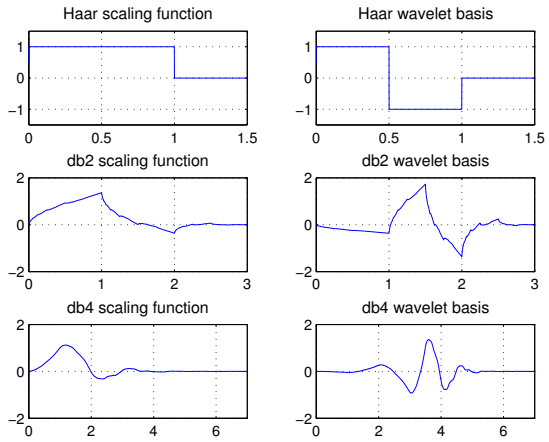


Figure 2.3: Waveforms corresponding to three members of the Daubechies wavelet family. The Haar is the simplest wavelet that captures discontinuities in the data; db2 and db4 also show similarity with our data but have longer waveforms, leading to better frequency resolution.

for signal $\mathbf{d}$. Also, if at most S entries in $\mathbf{x}$ are nonzero, then $\mathbf{x}$ is called an S-sparse vector; if S is small, $\mathbf{d}$ is said to be sparsely represented in the basis $\mathbf{B}$.

As possible basis functions, we consider the following members of the Daubechies wavelet family that can capture signal characteristics in both time and frequency domains: db1 (also known as Haar), db2, and db4 wavelet basis.[1] These wavelets are able to capture sharp or abrupt changes in the signal. Figure 2.3 shows the waveforms for these wavelets. We refer the reader to Walker for a primer on wavelets and their scientific applications [91]. We analyze the basis functions in terms of how concisely they encode the data collected from our system. We first perform a signal transform on each data set to find the corresponding coefficients within the chosen basis and sort them in decreasing order of their magnitude. Then, we use the n largest coefficients where $1 \leq n \leq N$, set all the other coefficients to 0, and reconstruct a new signal $\tilde{\mathbf{d}}(n)$. A relative error metric captures the difference between the original and reconstructed signals as $e(n) = \|\tilde{\mathbf{d}}(n) - \mathbf{d}\|/\|\mathbf{d}\|$, where $\| * \|$ denotes the l_2 norm. Finally, we examine how this relative error changes as the value of n is increased. Table 2.1 summarizes the percentage of coefficients needed to maintain the relative error within 1% for each of the bases. In this respect, the Haar wavelet represents our data most concisely.

The foregoing discussion makes it clear that a signal can have different sparsity levels when

[1]Here 'db' is short for Daubechies and the number after it represents the number of vanishing moments for the corresponding wavelet basis.

Table 2.1: Percentage of coefficients needed to keep the relative error within 1% under each representation basis.

Signal	Haar	db2	db4
AnonPages	0.20%	0.51%	0.66%
Mapped	0.16%	0.37%	0.46%
CommittedAS	0.85%	1.27%	1.49%
PageTables	0.57%	0.64%	0.83%

expressed under different bases. Using the same number of samples, both compressive sampling and compression can reconstruct the signal with greater fidelity if we know beforehand the underlying basis in which the data is sparse. Selecting the appropriate basis under which the measured signal is most sparse requires some experimentation using representative training data from the system. We now discuss a basis selection strategy that reduces this burden on the datacenter operator by automatically adapting the representation basis to the structure of the sampled signal.

2.3.2 The Best Basis Algorithm

Given a set of possible basis functions, we use a thresholding technique to find the basis in which the signal is most sparse [54]. Assume we have a choice between K possible bases and let the representation of the signal in the k^{th} basis $\mathbf{B}_k$ be $\mathbf{x}^{(k)} = [x_1^{(k)}, \cdots, x_N^{(k)}]$. Also let the magnitude of the $(M+1)^{\text{th}}$ largest coefficient in $\mathbf{B}_k$ be t_k, and denote the recovered signal using coefficients with magnitude larger than t_k in basis $\mathbf{B}_k$ as $\mathbf{d}_k$. Let us define a soft-thresholding function as

$$\gamma_{t_k}(x) = \begin{cases} x^2/2, & \text{if } x < t_k \\ t_k x - {t_k}^2/2, & \text{otherwise.} \end{cases} \tag{2.1}$$

Then for the basis $\mathbf{B}_k$, the function

$$R(\mathbf{B}_k) = \sum_{i=1}^{N} \gamma_{t_k}(|x_i^{(k)}|) \tag{2.2}$$

returns the sum of two terms: the quadratic distance between $\mathbf{d}_k$ and the original signal, and the scaled sparsity of $\mathbf{d}_k$ within the selected basis $\mathbf{B}_k$. More formally, the cost function is

$$\frac{1}{2} \parallel \mathbf{d} - \mathbf{d}_k \parallel^2 + t_k \parallel \mathbf{B}_k^T \mathbf{d}_k \parallel_0 \tag{2.3}$$

where $\parallel \mathbf{B}_k^T \mathbf{d}_k \parallel_0$ is the number of nonzero entries in $\mathbf{B}_k^T \mathbf{d}_k$, that is, the sparsity of $\mathbf{d}_k$ in basis $\mathbf{B}_k$. By minimizing this cost function, we obtain the best possible approximation of $\mathbf{d}$ when represented in the most concise basis. In other words, the function $R(\mathbf{B}_k)$ quantifies the risk, in terms of loss in fidelity, of applying the threshold t_k on the coefficients in basis $\mathbf{B}_k$; a smaller risk means that the corresponding coefficients will provide a better approximation of the original signal. Therefore,

$$\mathbf{B}^* = arg \min_{k=1,\cdots,K} R(\mathbf{B}_k) \tag{2.4}$$

provides us with the best basis in which to represent the signal.

2.3.3 Compression-based Online Monitoring Method

The foregoing discussion suggests a straightforward way of using signal compressibility to reduce the data transfer and storage costs associated with online monitoring. Locally at each server or virtual machine, we collect N samples for each signal of interest at some user-specified sampling rate. After the best-basis transformation, say using wavelet decomposition, on these measurements, we obtain a set of N coefficients for the signal and choose the M largest coefficients in terms of magnitude, where $M \ll N$. These M coefficients along with their positions within the larger set of N coefficients are sent to the monitoring station where an approximation of the original measurements is recovered. The chief drawback with this method is the CPU overhead imposed on each local machine due to the sampling and compression process: once the N measurements are obtained, the best-basis algorithm must be invoked followed by a sorting routine to arrange the coefficients in order of decreasing magnitude. This sampling overhead (quantified in Section 2.6) can interfere with the execution of other applications and VMs running on the server.

Table 2.2: Average coherence between an $M \times N$ random Gaussian sensing basis and the different representation bases when $N = 2048$.

Representation basis	Haar	db2	db4
$M = 0.1N$	3.67	4.81	4.77
$M = 0.3N$	3.81	5.05	5.04
$M = 0.5N$	3.90	5.11	5.12

2.4 Compressive Sampling of System Measurements

When the underlying signal can be represented sparsely in an appropriate basis, the signal can be acquired from the system *directly* in a compressed form using the concept of compressive sampling rather than first collecting a number of samples and then compressing them, as was done in the previous section. This allows for a very simple sensing scheme locally at each server: rather than tailoring the sensing to the specific signal being measured, a signal-independent strategy such as randomized sampling can be used. This section familiarizes the reader with the concept of *incoherence*, a key condition underlying compressive sampling that affects the way we sample signals. We then discuss how the original signal is recovered from a small set of samples.

2.4.1 Incoherent Sampling of the Signal

Given two N-dimensional bases Ψ and Φ, the coherence between these bases is defined as the largest coherence between any two basis vectors in Ψ and Φ given by

$$\mu(\Psi, \Phi) = \sqrt{N} \max_{1 \le k,j \le N} \left| \langle \phi_k, \psi_j \rangle \right|, \tag{2.5}$$

where $\langle \phi_k, \psi_j \rangle$ is the dot product of the vectors ϕ_k and ψ_j. Typically the coherence between the two bases lies between 1 and $\sqrt{N}$, and when the value of coherence is small we consider the two bases to be uncorrelated or incoherent. When the sensing and representation bases are uncorrelated, a spike in one basis will be represented as a spread-out waveform in the other. This property allows us to capture the complete information present in the original data using a small number of samples obtained by incoherent sampling.

As a sampling strategy to collect measurements from our testbed, we choose Gaussian random matrices that have a low coherence of $\sqrt{2 \log N}$ relative to any representation matrix with high

probability. Table 2.2 shows the average coherence between an $M \times N$ Gaussian sensing basis and the various representation bases of interest where N is the length of the input data and M is the desired number of samples. As expected, the coherence values are quite low.

Before sample collection, we generate an $M \times N$ Gaussian random matrix G as the underlying sampling matrix. Elements in the matrix are independently chosen from a standard Gaussian distribution and the rows are orthonormalized such that the rows have unit norm and are orthogonal to each other. To obtain the samples from the input data, we simply multiply this matrix G by the vector of data $\mathbf{d}$. For example, assume the data to be sampled is an 8×1 vector

$$\mathbf{d} = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,8} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,8} \\ \vdots & \vdots & \ddots & \vdots \\ B_{8,1} & B_{8,2} & \cdots & B_{8,8} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_8 \end{pmatrix} = B\mathbf{x},$$

where B is an 8×8 matrix corresponding to the Haar wavelet basis and $\mathbf{x}$ is the representation of $\mathbf{d}$ in the Haar basis. Suppose we wish to obtain a 4×1 vector of samples $\mathbf{y}$. The data is multiplied with a 4×8 Gaussian matrix G such that

$$\mathbf{y} = \begin{pmatrix} G_{1,1} & G_{1,2} & \cdots & G_{1,8} \\ G_{2,1} & G_{2,2} & \cdots & G_{2,8} \\ G_{3,1} & G_{3,2} & \cdots & G_{3,8} \\ G_{4,1} & G_{4,2} & \cdots & G_{4,8} \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_8 \end{pmatrix}$$

$$= G\mathbf{d} = GB\mathbf{x} = A\mathbf{x},$$

where $A = GB$ is a 4×8 matrix.

2.4.2 Recovering the Original Signal

The process of incoherent sampling gives us a set of values $\mathbf{y} = G\mathbf{d} = GB\mathbf{x} = A\mathbf{x}$, where $A = GB$. To reconstruct the original data $\mathbf{d}$, we must solve this inverse problem: given a vector $\mathbf{y}$ of length M and matrix A of size $M \times N$ where $M \ll N$, find a sparse vector $\tilde{\mathbf{x}}$ of length N such that

$\mathbf{y} = A\tilde{\mathbf{x}}$. In other words, we are looking for $\tilde{\mathbf{x}}$ as a solution to

$$\min_{\mathbf{b} \in \mathcal{R}^N} \| \mathbf{b} \|_0 \quad \text{subject to: } \mathbf{y} = A\mathbf{b}, \tag{2.6}$$

where $\|\mathbf{b}\|_0$ is the l_0 norm of $\mathbf{b}$, i.e. the number of nonzero entries in $\mathbf{b}$. This problem is underconstrained since the matrix A has more columns than rows; there are infinitely many candidate signals $\mathbf{b}$ for which $A\mathbf{b} = \mathbf{y}$. Since minimizing the l_0 norm is a computationally expensive nonlinear optimization problem, a class of reconstruction algorithms called basis pursuit or iterative hard thresholding pursuit (HTP) is used. Here the problem is recast as one of minimizing the l_1 norm, which is the linear programming problem:

$$\min_{\mathbf{b} \in \mathcal{R}^N} \|\mathbf{b}\|_1 \quad \text{subject to: } \mathbf{y} = A\mathbf{b}. \tag{2.7}$$

We use the iterative hard thresholding pursuit technique previously proposed by Foucart [33] to solve (2.7). If $\mathbf{b}^{(k)}$ denotes the S-sparse vector obtained after the k^{th} iteration, this method performs the following two steps to obtain the next approximation of $\mathbf{b}$:

1. Perform the operation $\mathbf{b}^{(k)} + A^T(\mathbf{y} - A\mathbf{b}^{(k)})$ and identify the indices of the S largest coefficients in the resulting $N \times 1$ vector; store these indices in a set $I^{(k+1)}$.

2. Solve $\mathbf{b}^{(k+1)} = \arg\min_{\mathbf{b^N} \in \mathcal{R}} \|\mathbf{y} - A\mathbf{b}^{(k)}\|_2,\ \text{supp}(\mathbf{b}) \subseteq I^{(k+1)}$. Here, only the coefficients residing in the indices previously stored in $I^{(k+1)}$ are tuned so as to minimize the cost function. The other coefficients are zeros.

The above steps are repeated until $I^{(k+1)} = I^{(k)}$. Reconstruction of the original signal using HTP is considered to be exact with probability exceeding $1 - \delta$ when the number of samples M satisfies the following condition:

$$M \geq C\mu\,(\Psi, \Phi)^2\, S \log \frac{N}{\delta}, \tag{2.8}$$

where δ is a small constant and C is some positive constant [34]. The implication of (2.8) is that when the coherence between the representation and sensing bases, μ, as well as the sparsity metric, S, are small, we need only a few samples to recover the original signal exactly with high probability.

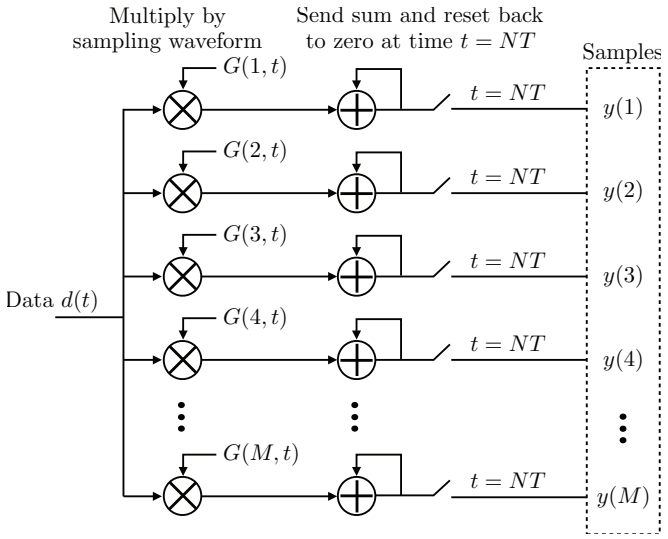


Figure 2.4: Implementation of compressive sampling in our system that takes N data items over a time period as input and returns M samples, where $M \ll N$.

2.4.3 A CS-based Online Monitoring Strategy

Fig. 2.4 shows the implementation of the CS-based method within each local server in which the incoming signal $\mathbf{d}$ is acquired directly in a compressed form $\mathbf{y}$. When a new data item $d(t)$ arrives at time t it is multiplied by the entries in the sampling matrix $G(j,t)$, $j = 1,\ldots,M$ and the partial products are accumulated into $y(j)$. After a period of length $N \times T$, where T is the sampling period, the current values of $y(j)$ are sent out as the M samples to the monitoring station, and then reset back to zero. Effectively, $y(j) = \sum_{t=1}^{N} G(j,t)d(t)$, where $j = 1,\ldots,M$, and thus $\mathbf{y} = G\mathbf{d}$.

At the monitoring station, the original signal is recovered using the HTP algorithm. The key advantage here is that simple, randomized sampling of the data is performed locally on each server, significantly reducing the intrusion of monitoring on application performance. The computational cost associated with signal recovery is offloaded to the monitoring station.

2.5 Adaptive-rate compressive sampling

The previous section assumes that the signal sparsity when represented in the underlying basis is constant or bounded across time windows. Under such an assumption, we can use a fixed rate for

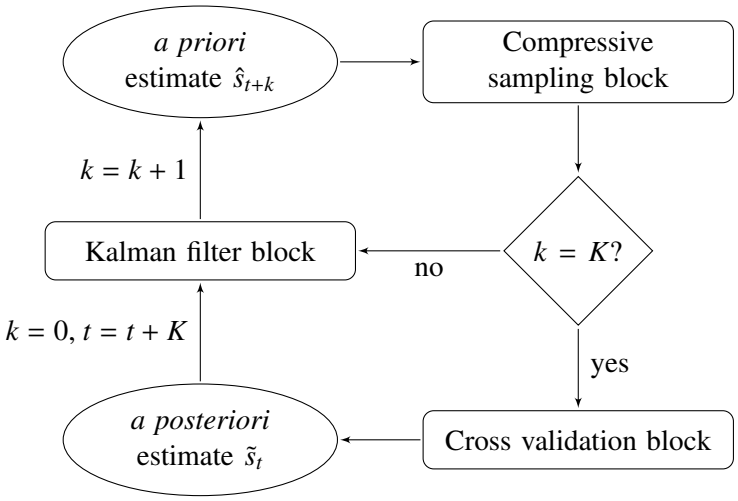


Figure 2.5: Workflow for the adaptive-rate compressive sampling strategy. The workflow comprises three major steps: compressive sampling, cross validation, and prediction of signal sparsity.

sampling the signal. This assumption of a constant signal sparsity within each time window, however, rarely holds in practice. For example, Fig. 1.10 plots the sparsity level in terms of the number of coefficients needed in the Haar wavelet basis to capture 99% of the information contained within the CommittedAS and AnonPages signals (previously shown in Fig. 2.2) within different time windows. The sparsities change significantly over time. So a fixed-rate sampling strategy based on an upper bound on the sparsity level—between 45-50% in this particular case—will collect 100% of the samples at all times. We can perform much better in terms of reducing the number of samples needed by dynamically increasing or decreasing the sampling rate as the signal sparsity changes. This section adds a rate adjustment component to the aforementioned strategy of compression sampling. We show that by collecting some relevant side information from the signal—which could have been under or over sampled during any particular time window—we can estimate the underlying sparsity and predict it for some succeeding time windows. The number of samples collected over these windows can then be adjusted based on the predicted sparsity values. The goal is to achieve an overall reduction in sample size without compromising the reconstruction quality (specified in terms of relative error).

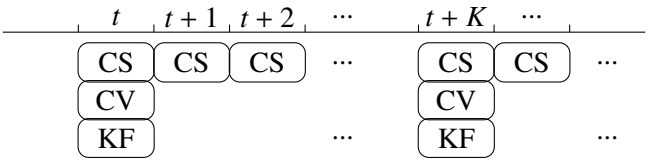


Figure 2.6: The timeline for the execution of each block in the adaptive-rate sampling strategy. Here CS denotes the compressive sampling block, CV, the cross validation block, and KF, the Kalman filter.

2.5.1 Overview of the CS-MON Strategy

Figure 2.5 outlines CS-MON, our strategy for adaptive-rate compressive sampling, comprising three major algorithmic components: a compressive sampling block; a cross validation block to estimate the signal sparsity; and a Kalman filter block to predict the sparsity over future time windows. The side information is collected by the cross validation block once every K time windows and the sparsity is predicted for every time window.

As per Fig. 2.5, an *a priori* estimate of the signal sparsity within the t^{th} time window, $\hat{s}_t$, is obtained (or initialized at start up) and the sampling rate is appropriately adjusted for the subsequent time window based on this estimate. Knowing $\tilde{s}_t$, we use a Kalman filter to predict the sparsity for the next K time windows, $\hat{s}_{t+1}, \hat{s}_{t+2}, \ldots, \hat{s}_{t+K}$. These serve as the *a priori* estimates for time windows $(t + 1)$ through $(t + K)$ and determine the compressive sampling rates within each of the windows. Once every K time widows, cross-validation measurements are collected from the signal and used to generate *a posteriori* estimate of the sparsity, $\tilde{s}_t$, via maximum-likelihood estimation. This estimate forms the basis for the next set of predictions from the Kalman filter. Figure 2.6 shows the execution timeline for each block in our system. The cross-validation and the Kalman filter block are executed once every K time windows whereas the compressive sampling block is executed every time window. Since the Kalman state is only updated once every K windows by the cross-validation measurements, the filter "coasts" in the absence of measurements in the interim.

2.5.2 The Compressive Sampling Block

This block performs compressive sampling within each time window, adjusting the sampling rate for the t^{th} window as per $\hat{s}_t$, the assumed sparsity of the data in this window. Following the

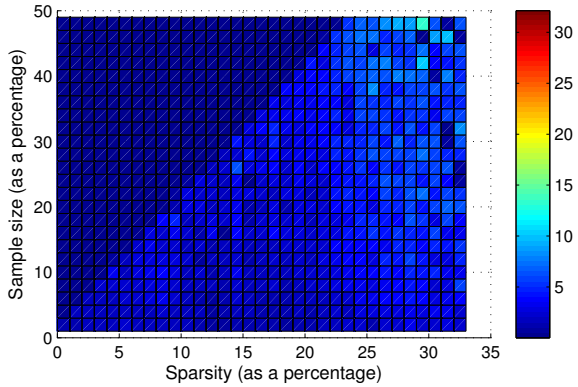


Figure 2.7: The relative error achieved by various sample sizes, given different sparsity levels within the AnonPages signal. Best viewed in color.

notation used in Section 2.3, we denote the data within time window t as an $N \times 1$ vector $\mathbf{d}_t$ and its representation in basis B as $\mathbf{x}_t$ such that $\mathbf{d}_t = B\mathbf{x}_t$. The sample size is set to $M_t(\hat{s}_t)$ which is a function of the assumed sparsity. The $M_t(\hat{s}_t) \times 1$ measurement vector $\mathbf{y}_t(\hat{s}_t)$ is then obtained via incoherent sampling as $\mathbf{y}_t(\hat{s}_t) = G(\hat{s}_t)\mathbf{d}_t = G(\hat{s}_t)B\mathbf{x}_t$, where $G(\hat{s}_t)$ is an $M_t(\hat{s}_t) \times N$ matrix whose construction was previously discussed in Section 2.4. The original data $\mathbf{d}_t$ is then reconstructed as $\hat{\mathbf{d}}_t$, also as described in Section 2.4.

Given that signal sparsity can be estimated at each time window, we can use this information to adjust the sample size such that the desired reconstruction quality is achieved. To achieve this objective, a quantitative relationship between sparsity and sample size needs to be established—empirically, in our case. Consider the AnonPages signal whose sparsity varies over time (see Fig. 1.10). Here, sparsity is defined as the percentage of coefficients that capture 99.5% of the original data. For data belonging to each sparsity level, we try different sample sizes, reconstruct the data and evaluate the result using the relative error metric. Fig. 2.7 shows the relative error achieved using various sample sizes, given different sparsity levels within the AnonPages signal. As expected, portions of the signal which are less sparse require a larger sample size to achieve the same relative error. The figure also implies a linear relationship between sparsity and sample size: for data of length N with sparsity s, a sample size of $M(s) = 5s$ guarantees a reconstruction error below 5% with confidence exceeding 93%. Therefore, if $\hat{s}_t$ is the sparsity for time window t, the corresponding sample size $M_t(\hat{s}_t)$ is chosen as $5\hat{s}_t$.

2.5.3 The Cross-Validation Block

Recall from Fig. 2.5 that the actual sparsity of the underlying data being sampled must be measured periodically, as best as possible, so that an *a posteriori* estimate can be obtained to update the Kalman filter. The cross validation block, executed once every K windows, serves this purpose by performing the following operations:

(1) Measurements $\mathbf{z}_t = H_t\mathbf{d}_t$ are collected from the original signal using a cross-validation matrix H of size $R\times N$ where R is the number of measurements. The entries of H are random numbers independently selected from the Bernoulli distribution with zero mean and variance of $1/R$. We will explain the choice of R later in this section.

(2) Matrix H is then applied on the reconstructed data $\hat{\mathbf{d}}_t$ returned by the compressive sampling block, resulting in $\hat{\mathbf{z}}_t = H\hat{\mathbf{d}}_t$. Note that $\hat{\mathbf{d}}$ is reconstructed using samples obtained under the sparsity assumption of $\hat{s}_t$.

(3) The cross-validation error under the current sparsity assumption $\hat{s}_t$ is then calculated as the error norm

$$e_t^{\mathrm{cv}} = \| \mathbf{z}_t - \hat{\mathbf{z}}_t \| .$$

(4) Using maximum-likelihood estimation, the *a posteriori* estimate for the sparsity is chosen as the value $\tilde{s}_t$ that maximizes the posterior probability $P(e^{\mathrm{cv}} = e_t^{\mathrm{cv}}|s = i, \hat{s} = \hat{s}_t)$:

$$\tilde{s}_t = \arg \max_{i=0,1,\ldots,N} P(e^{\mathrm{cv}} = e_t^{\mathrm{cv}}|s = i, \hat{s} = \hat{s}_t).$$

$P(e^{\mathrm{cv}}|s = i, \hat{s} = \hat{s}_t)$ is the probability density function of the cross-validation error on the condition that $\hat{s}$ is assumed as the sparsity when sampling the data whereas the measured cross-validation data has sparsity s.

The statistical models pertaining to $P(e^{\mathrm{cv}}|s = i, \hat{s} = \hat{s}_t)$ are built empirically using a training data set. We use a sliding window of size N with a step size of L to go through this data set, grouping the

time windows based on their actual sparsity. Once these windows have been collected, the following process is repeated for each window. First the actual sparsity s of the data within the window is calculated. Then we assume different values for the *a priori estimate* $\hat{s}$ (from $i = 0$ through N) and perform the previously discussed steps (1), (2), and (3) on the data. The empirical distribution of e^{cv} when $\hat{s}$ is assumed, but when the actual sparsity is s is then calculated to approximate $P(e^{\text{cv}}|s = i, \hat{s} = \hat{s}_t)$. It is important to note that though the empirical distributions are calculated using a particular set of training data, the effect of the actual and assumed sparsity values on the cross-validation error derived from the training data is representative and is not limited to the data within this set. Therefore, there is no need to update $P(e^{\text{cv}}|s = i, \hat{s} = \hat{s}_t^-)$.

In the foregoing discussion, we argued that a small number of cross-validation measurements can be used to obtain a good approximation of the sparsity of the much larger data stream. In the following we prove this assertion.

The relative error between the original and the reconstructed signals is determined by the exact sparsity and the value assumed during the signal recovery process. First we show that the relative error obtained using the cross-validation measurements is a good approximation of the optimal relative error between the original and reconstructed signals. Given the representation of the signal in the t-th time window, $\mathbf{x}_t$, and an arbitrary $\hat{s}$ as the assumed sparsity, we can define the optimal $\hat{s}$-sparse approximation error as

$$e_t^{\text{opt}}(\hat{s}) = \min_{\|\mathbf{x}\|_0 \leq \hat{s}} \| \mathbf{x}_t - \mathbf{x} \|,$$

where the search space includes all vectors with sparsity $\leq \hat{s}$. Let s be the exact sparsity that captures $1 - \epsilon$ of the signal. If $\hat{s} = s$, then the optimal $\hat{s}$-sparse approximation error $e_t^{\text{opt}}(\hat{s}) = \epsilon \| \mathbf{x}_t \|$, which is the upper bound on the absolute error. If $\hat{s} < s$, then $e_t^{\text{opt}}(\hat{s}) > \epsilon \| \mathbf{x}_t \|$ since the signal will be under-sampled, losing information in the process; similarly if $\hat{s} > s$, then $e_t^{\text{opt}}(\hat{s}) < \epsilon \| \mathbf{x}_t \|$ due to over sampling of the signal. The value of $e_t^{\text{opt}}(\hat{s})$ is indicative of the difference between the actual and assumed sparsities, and therefore we should be able to estimate s by observing $e_t^{\text{opt}}(\hat{s})$. Unfortunately, the $e_t^{\text{opt}}(\hat{s})$ values are unobservable due to the lack of ground truth for $\mathbf{x}_t$, since we do not sample at full rate when CS-MON is operating. However, we can show that the cross-validation

error can be used to obtain a tight upper and lower bounds on $e_t^{\text{opt}}(\hat{s})$, thereby replacing $e_t^{\text{opt}}(\hat{s})$ during sparsity estimation. The cross-validation error e_t^{cv} is observable since both $\mathbf{z}_t$ and $\hat{\mathbf{z}}_t$ are known, and its relationship with $e_t^{\text{opt}}(\hat{s}_t)$,

$$\frac{1}{h(1+\epsilon)} e_t^{\text{cv}}(\hat{s}_t) \leq e_t^{\text{opt}}(\hat{s}_t) \leq \frac{1}{1-\epsilon} e_t^{\text{cv}}(\hat{s}_t) \tag{2.9}$$

holds with high probability. Here h is a constant and ϵ is the upper bound on the relative error. A direct consequence of (2.9) is that the observed $e_t^{\text{cv}}(\hat{s}_t)$ can be used in place of $e_t^{\text{opt}}(\hat{s}_t)$ for the sparsity estimation.

We now provide the proof that $e_t^{\text{opt}}(\hat{s}_t)$ can be bounded by the cross-validation error as stated in (2.9) using the Johnson-Lindenstrauss lemma [92]. First we define an approximation error that quantifies the reconstruction quality achieved by the compressive sampling block as

$$\begin{aligned} e_t^{\text{app}}(\hat{s}_t) &= \| \mathbf{d}_t - \hat{\mathbf{d}}_t \| \\ &= \| \mathbf{x}_t - \hat{\mathbf{x}}_t \|, \end{aligned}$$

where $\hat{\mathbf{x}}_t$ is the representation of $\hat{\mathbf{d}}_t$ in basis B. We can provide upper and lower bounds for $e_t^{\text{app}}(\hat{s}_t)$ in terms of $e_t^{\text{opt}}(\hat{s}_t)$ with high probability for some constant h [73]:

$$e_t^{\text{opt}}(\hat{s}_t) \leq e_t^{\text{app}}(\hat{s}_t) \leq h e_t^{\text{opt}}(\hat{s}_t) \tag{2.10}$$

The constant depends on the sampling matrix $G(\hat{s}_t)$ used during compressive sampling. Since the ground truth $\mathbf{d}_t$ is unknown, the exact value for $e_t^{\text{app}}(\hat{s}_t)$ is unknown as well. The Johnson-Lindenstrauss lemma states that $e_t^{\text{app}}(\hat{s}_t)$ can be bounded by the observable cross-validation error. Let W be an $R \times N$ matrix whose entries are realizations of a random variable r with zero mean and variance $1/R$. If r follows Bernoulli or Gaussian distribution, then, for a predetermined $N \times 1$ vector $\mathbf{d}$,

$$1 - \epsilon \leq \frac{\| W\mathbf{d} \|}{\| \mathbf{d} \|} \leq 1 + \epsilon$$

holds true with probability exceeding $1 - \delta$, where $\delta \in (0, 1)$. According to the lemma, if the row dimension R in the cross-validation matrix is chosen such that it satisfies $R \geq h\epsilon^{-2} \log(1/2\delta)$ where h and ϵ are constants (in our case ϵ is the upper bound for the relative error), then

$$1 - \epsilon \leq \frac{\| \mathbf{z}_t - \hat{\mathbf{z}}_t \|}{\| \mathbf{d}_t - \hat{\mathbf{d}}_t \|} \leq 1 + \epsilon$$

also holds true with probability exceeding $1 - \delta$. We choose the constants as $\epsilon \in (0, \frac{1}{2})$ for the accuracy level and $h = 8$ to satisfy the equality for R (R is chosen to satisfy the equality). Algebraic manipulation of the previous equation gives us

$$\frac{1}{1+\epsilon} e_t^{\text{cv}}(\hat{s}_t) \leq e_t^{\text{app}}(\hat{s}_t) \leq \frac{1}{1-\epsilon} e_t^{\text{cv}}(\hat{s}_t). \tag{2.11}$$

Equation (2.9) follows directly from (2.10) and (2.11), proving that $e_t^{\text{cv}}(\hat{s}_t)$ can indeed be used to estimate the actual sparsity with high confidence.

2.5.4 The Kalman Filter block

The *a posteriori* estimate $\tilde{s}_t$ from the cross validation block is used to update the Kalman filter state and the filter is then used to predict the sparsity values over the next few time windows. Use of the predictive filter further reduces the overall sampling cost—by reducing the number of cross-validation measurements in this case. Internally, the Kalman filter uses a state-space representation of an auto-regressive integrated moving average (ARIMA) model for its predictions [93].

If the cross-validation block is executed once every K time windows, the filter forecasts the sparsity values for the next $K - 1$ time windows, $\hat{s}_{t+1}, \ldots, \hat{s}_{t+K-1}$, which form the inputs to the compressive sampling block. Note that since the filter's state is only updated once every $K - 1$ windows by the cross-validation measurements, the filter "coasts" in the interim, that is, the gains are set to zero.

2.5.5 Summary of the Adaptive-Rate Model Operation

Let the initial condition be that $\mathbf{d}_t$, the data to be sampled during time window t, has some unknown sparsity. To compressively sample this data, we assume its sparsity to be some initial

value $\hat{s}_t$ and use an $M(\hat{s}_t) \times N$ matrix $G(\hat{s}_t)$ to collect $M_t(\hat{s}_t)$ samples, resulting in measurements $\mathbf{y}_t(\hat{s}_t) = G(\hat{s}_t)\mathbf{d}_t = G(\hat{s}_t)B\mathbf{x}_t$. Recall that $M(\hat{s}_t) = 5\hat{s}_t$. The data in this time window is then reconstructed as $\hat{\mathbf{d}}_t$ as described in Section 2.4 by the compressive sampling block.

Once every K time windows cross-validation measurements are collected using an $R \times N$ matrix H with $R \geq 8\epsilon^{-2}\log(1/2\delta)$, resulting in measurements $\mathbf{z}_t = H\mathbf{d}_t = HB\mathbf{x}_t$. The cross-validation error is calculated as $e_t^{\text{cv}}(\hat{s}_t) = \| \mathbf{z}_t - H\hat{\mathbf{d}}_t \|$ which is then supplied to the maximum-likelihood model to obtain the *a posteriori* estimate of the sparsity of the current time window, $\tilde{s}_t$. Once its state is updated using $\tilde{s}_t$, the Kalman filter forecasts the sparsity over the next $K - 1$ time windows. The above process repeats itself.

Note that the sampling matrix $G(\hat{s}_t)$ used by the compressive sampling block is obtained by randomly selecting $M(\hat{s}_t)$ rows from an $N \times N$ random Gaussian matrix, built prior to the sampling process (Section 2.4 discusses the construction of this Gaussian matrix.) The sampling matrix H for collecting the side information is fixed as long as the parameters ϵ and δ remain unchanged.

2.6 Performance Evaluation

We now evaluate the performance of both the compression-based (C-MON) and CS-based (CS-MON) monitoring strategies in terms of their efficacy in: (1) reducing the sample size while guaranteeing a specified signal reconstruction quality; and (2) using the recovered signals to detect threshold violations and trends that could be indicative of performance bottlenecks or degradation in the system. The CPU and storage overhead incurred by these methods is also quantified.

Reconstruction quality can be quantified via the relative-error metric that expresses the normalized error between the original and recovered signals. In many situations, however, it is not essential to recover the original signal exactly, especially if the operator is mainly interested in detecting performance problems with the system. It is more important that the signals reconstructed via the C-MON and CS-MON methods preserve properties in the original signal that can help detect these problems. We consider two such scenarios:

- The datacenter operator wishes to detect performance-related bottlenecks or anomalies that manifest themselves as the magnitude of the signal exceeding some nominal threshold value. The performance of the two strategies in this regard is quantified by a hit-rate metric.

- The operator wishes to detect the gradual performance deterioration associated with software aging or resource exhaustion by analyzing the signals for the existence of trends. The performance is quantified by how accurately C-MON and CS-MON estimate a positive slope, if one exists, in the reconstructed signals.

2.6.1 Signal Reconstruction Quality using CS-MON

We use the signals obtained from our testbed to evaluate the performance of the CS-MON strategy in terms of overall signal reconstruction quality, focusing on the performance of the key blocks comprising the workflow shown in Fig. 2.5.

Estimating a posteriori signal sparsity: We show that, given an arbitrary initial assumption for the signal sparsity, the cross validation block can quickly update its estimation about the actual sparsity. In our result, the average estimation error is within 20% of the actual sparsity value, showing the cross validation block can closely track the actual sparsity by collecting cross-validation measurements periodically.

Assuming $\hat{s}_t$ to be the *a priori* estimate of data sparsity within a time window t, the compressive sampling block collects M_t samples and reconstructs the data. Cross-validation measurements are also collected within this window and maximum-likelihood estimation is performed on the resulting cross-validation error to obtain the *a posteriori* sparsity $\tilde{s}_t$. Since this estimate is influenced by the initial assumption of a sparsity value, we study the quality of $\tilde{s}_t$ obtained under different assumptions for $\hat{s}_t$, including arbitrary guesses.

Figure 2.8 shows an overlay of the actual signal sparsity (the green plot) with the *a posteriori* values $\tilde{s}_t$ obtained by the cross validation block for various initial sparsity guesses. Values along the y-axis correspond to the number of coefficients needed to capture 99% of the AnonPages signal—our definition of sparsity here. The length of each time window is set to $N = 64$. Let us focus on the plot in red, illustrating the case in which during the start of each time window, the compressive sampling block reconstructs the underlying signal assuming a sparsity of 4%, that is $\hat{s}_t = 0.04N$. In this case, the *a priori* value $\hat{s}_t$ used for compressive sampling severely underestimates the actual sparsity during time windows 380 through 425. However, by the end of each of these time windows, the cross validation block is able to correct this estimate and obtain a reasonably close approxima-

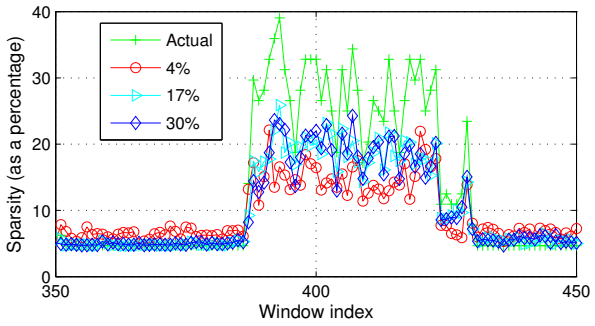
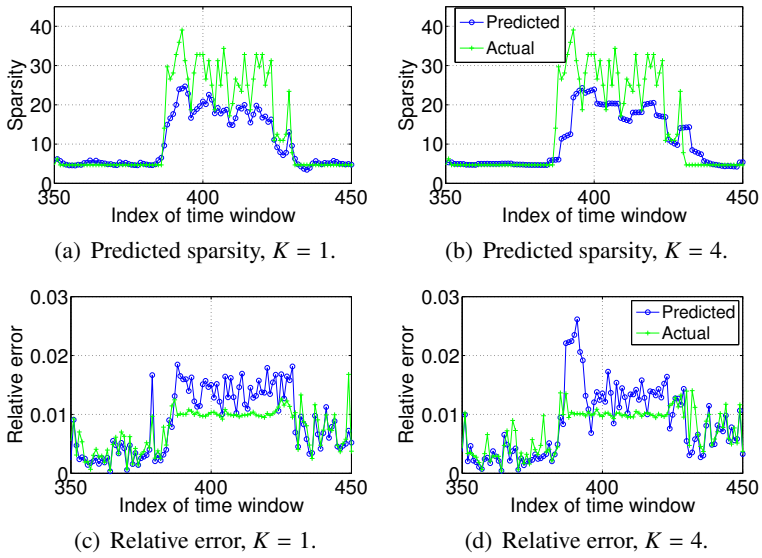


Figure 2.8: Overlay of the actual sparsity of the AnonPages signal with *a posteriori* values $\tilde{s}$ obtained by the cross validation block. Here sparsity is defined as the number of coefficients needed to capture 99% of the original signal. The plots show the $\tilde{s}$ values obtained using the following initial guesses for the sparsity levels: 4%, 17%, and 30%.



(a) Predicted sparsity, $K = 1$.

(b) Predicted sparsity, $K = 4$.

(c) Relative error, $K = 1$.

(d) Relative error, $K = 4$.

Figure 2.9: Predictions for sparsity values provided by the Kalman filter for the AnonPages signal when the filter's state is updated: (a) during each time window, that is $K = 1$, and (b) once every four windows, $K = 4$. The relative error between the actual and reconstructed signals when the filter state is updated each window and once every four windows is shown in (c) and (d), respectively.

tion of the actual sparsity. Also, the *a posteriori* values obtained under different *a priori* sparsity assumptions lie fairly close to each other, and they are not overly sensitive to the *a priori* value $\hat{s}_t$. However, we do observe that guesses closer to the actual value lead to more accurate estimates. The average estimation error is within 20% of the actual sparsity value regardless of the initial guess.

Predicting sparsity for future time windows: Recall from the CS-MON workflow that once the signal

sparsity is estimated by the cross-validation block, a Kalman filter is used to predict the sparsity for the next few time windows to further reduce the sampling overhead involved in obtaining the cross-validation measurements. Our result shows that the prediction error of the Kalman filter block falls within 20% of the actual sparsity.

Figures 2.9(a) and 2.9(b) show the Kalman predictions in relation to the actual sparsity for the AnonPages signal. As before, the window size is set to $N = 64$ and the sparsity is defined as the number of coefficients needed to capture 99% of the data. Figure 2.9(a) shows the case in which the cross validation block is invoked every time window. Figure 2.9(b) shows the case in which the frequency is once every four windows and the Kalman filter is required to predict the values for the windows in between. Since a linear model is used in our filter implementation, the prediction process underestimates the actual sparsity in cases of sudden fluctuations in the sparsity. However, our results show that the predictions track the overall trend exhibited by the actual values, and the average prediction error is within 20%.

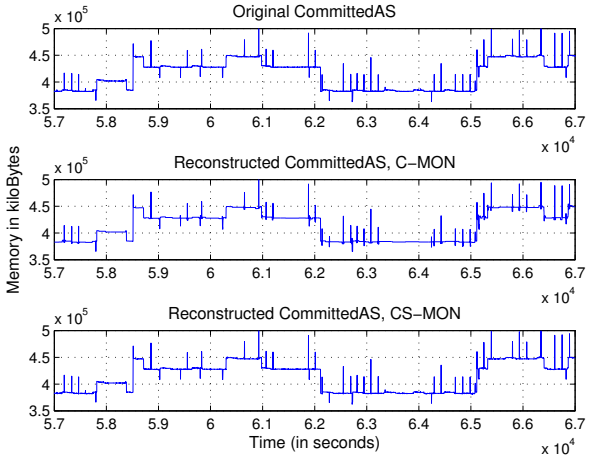
Signal reconstruction quality: We now evaluate the recovery quality achieved by the compressive sampling block using the sparsity predictions provided by the Kalman filter for each time window. Our result shows that although our sparsity prediction is not exact, it nevertheless provides a performance guarantee for monitoring systems in terms of reconstruction quality.

Note that in the ideal case, the relative error achieved as a result of using the actual sparsity is around 1%, which is to be expected since the sparsity accounts for 99% of the data. The relative error between the actual and reconstructed signals is shown in Figs. 2.9(c) and 2.9(d) when using the actual sparsity and the sparsity predictions provided by the filter, respectively. When sparsity is underestimated, the recovered signal incurs a higher relative error, whereas the error is low if more than the necessary number of samples are collected due to an overestimation of the sparsity. We observe that, although we underestimate the sparsity after a sudden increase in the actual value, the resulting reconstruction penalty is small: the relative error as a result of using predicted sparsity is rarely higher than 1.7%.

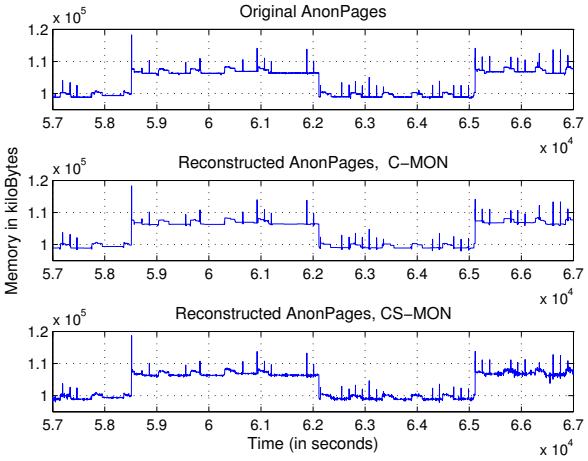
Reduction in sample size: Finally, we demonstrate that the sample size needed to reconstruct the signal while achieving the desired quality is substantially reduced using the adaptive-rate model. Our

Table 2.3: The average sample size used by CS-MON to reconstruct the AnonPages and CommitteAS signals for different sparsity levels.

Error tolerance	AnonPages		CommittedAS	
	$K = 1$	$K = 4$	$K = 1$	$K = 4$
5%	32.94%	32.80%	28.44%	29.13%
10%	27.44%	27.28%	17.19%	17.05%
15%	24.69%	23.94%	11.50%	10.76%



(a) CommittedAS



(b) AnonPages

Figure 2.10: The original CommittedAS and AnonPages signals, and the same signals recovered by C-MON using 5% of the coefficients in the basis found by the best basis algorithm. The signal recovered by CS-MON with 15% error tolerance is also shown for comparison purposes.

result shows that the adaptive-rate model achieves the same reconstruction quality while reducing the sample size by 70% when compared with fixed-rate compressive sampling.

In general, the error tolerance criteria for recovery quality is application-specific, which in turn defines the sparsity level. Table 2.3 lists the average sample size used by the adaptive-rate model per time window to reconstruct the AnonPages and CommittedAS signals under different sparsity criteria. We also show the two cases where the Kalman filter's state is updated during each time window, i.e., $K = 1$, and updated once every four windows, i.e., $K = 4$. The sample sizes are reported as a percentage of the total number of data points N within a time window with $N = 64$. The listed sample sizes include the number of samples obtained for both compressive sampling and for cross validation.

Table 2.3 shows that the adaptive-rate strategy guarantees a reconstruction quality with 5% relative error using substantially small sample sizes for the signals considered in this chapter; for example, about 33% for AnonPages, and 28% for CommittedAS. The sample size needed for a 15% error tolerance is even lower: around 25% for AnonPages and 12% for CommittedAS. Without accounting for time-varying signal sparsity, an upper bound on the sample size, determined by the least sparse portion(s) of the signal, must be used over all time windows to guarantee reconstruction quality. For example, referring back to Fig. 1.10, the upper bound on the sparsity level of the AnonPages signal is around 30%. So, if one always uses this bound for sampling purposes, a sample size corresponding to 30% signal sparsity, i.e., a sample size of 100% is required at all times. This is quite wasteful in terms of sample collection since in many portions of the signal, the sparsity is lower than 5% implying that the sampling rate can be adjusted to be much lower—around 20%—during these portions.

The collection of side information does not add much to the sampling cost. We find that the number of cross-validation measurements needed by CS-MON is quite small; for example, to achieve a relative error of 5% the number of additional measurements due to the cross-validation block accounts for only 0.01% of the overall sample size. The number of cross validation measurements is even smaller when a larger reconstruction error can be tolerated.

Table 2.3 also shows that to achieve a low error tolerance, more samples are needed. We also observe a difference in sample size when K, the frequency at which the cross validation block is

Table 2.4: Relative error between the original AnonPages and CommittedAS signals, and the corresponding reconstructions, achieved by C-MON.

Signal	Basis	% of samples ($M/N \times 100$)		
		2%	5%	10%
AnonPages	Haar	70.56%	0.34%	0.16%
	db2	79.27%	43.09%	0.26%
	db4	84.54%	67.50%	56.42%
	BB	70.56%	0.34%	0.16%
CommittedAS	Haar	70.35%	0.80%	0.37%
	db2	78.68%	42.11%	0.62%
	db4	84.19%	67.33%	56.58%
	BB	70.35%	0.80%	0.37%

invoked, is set to one versus four time windows. This difference is primarily caused by prediction results that either under or overestimate the signal sparsity. For example, when $K = 4$, the Kalman filter provides sparsity predictions without any input from the cross validation block for three time windows leading to prediction errors—mostly underestimating the sudden increase in the sparsity. Consequently, the under-sampling during those time windows results in a smaller sample size.

Moreover, recall that to detect performance-related issues with the system, it is not necessary to recover the original signal exactly but to just preserve properties in it that can help detect these problems. We show later in this section that signals with lower reconstruction quality, specifically those reconstructed to achieve a 15% error tolerance, can be used to detect performance issues with high confidence.

2.6.2 Comparing the CS-MON and C-MON Strategies

We now compare the adaptive sampling strategy to an alternative method also aimed at compressing the data collected at the server—the C-MON strategy discussed in Section 2.3. We find that for varying values of N, C-MON compresses the signal much better than CS-MON, resulting in a lower data transfer overhead to the monitoring station. Table 2.4 summarizes the relative error achieved by C-MON as a function of sample size. Here the original data set comprises $N = 64$ samples and the signal is reconstructed using M coefficients from the wavelet decomposition. The results show that the Best Basis (BB) algorithm successfully identifies the basis under which the signal can be most concisely represented—which happens to be the Haar wavelet for the Anon-

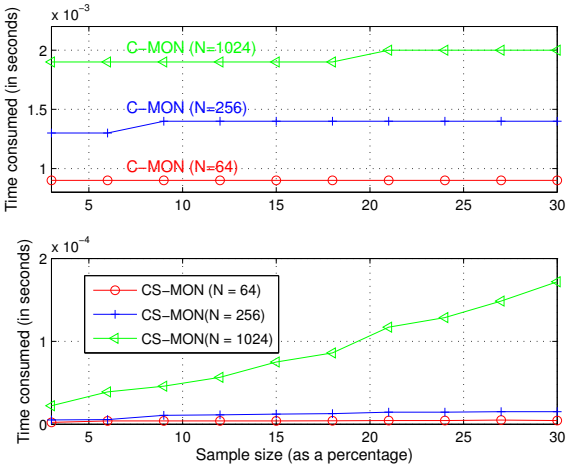


Figure 2.11: The CPU overhead incurred by the sampling and encoding processes associated with the C-MON and CS-MON strategies on the local machine. The signal reconstruction overhead incurred by the monitoring station is not included here. Both strategies were implemented in MATLAB and executed on an AMD Athlon II 3.0 GHz processor.

Pages and CommittedAS signals—thereby achieving good reconstruction quality while using very few coefficients. Fig. 2.10(a) plots the CommittedAS signal recovered by C-MON using 5% of the coefficients. The CS-MON reconstruction with 15% error tolerance is also shown for comparison purposes. Fig. 2.10(b) shows the same set of results for the AnonPages signal. One may conclude from the results summarized in Tables 2.3 and 2.4 that C-MON exploits the compressibility of the underlying signals better than CS-MON.

We quantify the computation and storage costs imposed on the local server by the two strategies to achieve their respective compression levels. Figure 2.11 quantifies the CPU overhead incurred on a per-window basis by the sampling and encoding processes associated with C-MON and CS-MON on the local machine for various values of N. C-MON incurs a higher CPU overhead since the encoding process requires a wavelet transformation on the N values followed by a sorting routine to extract the largest M coefficients. In the CS-MON implementation, the process of incoherent sampling is simply the multiplication of an $M \times N$ matrix with an $N \times 1$ vector, as shown in Fig. 2.4. So, CS-MON incurs considerably less CPU overhead, running about an order of magnitude faster than C-MON. Moreover, since this overhead is incurred on a per-signal basis, CS-MON can be quite CPU-efficient when monitoring large numbers of signals on a single server.

Table 2.5: Storage costs incurred by C-MON and CS-MON under different sparsity levels when sampling the AnonPages signal.

Error tolerance	Storage overhead (in KB)	
	C-MON	CS-MON
5%	1.50	0.16
10%	1.50	0.13
15%	1.50	0.12

Table 2.6: The packetization delay incurred in seconds for various lengths of the measurement window and sampling rate.

Window length, N	Sampling rate			
	0.5 Hz	5 Hz	100 Hz	1000 Hz
32	64	6.4	0.32	0.03
64	128	12.8	0.64	0.06
128	256	25.6	1.28	0.13

C-MON also incurs an increased storage cost over CS-MON, needing a buffer size of $O(N)$ to accommodate N data items whereas a buffer size of $M \leq N$ is required for CS-MON. Table 2.5 shows the storage overhead when using C-MON and CS-MON on the AnonPages signal for a window size of $N = 64$. The overhead due to C-MON includes the buffers needed to store the measurements obtained using the length-N time window and to store the sorted coefficients after the wavelet transformation in the Daubechies basis. However, if the data is transformed in the Haar basis, this can be implemented entirely in place—on the measurement-buffer itself. So, the storage required in this specific case is around $0.5KB$. On the other hand, the CS-MON implementation shown in Fig. 2.4 requires a buffer to simply store the compressed data that is generated as the measurements stream in.

Another equally interesting aspect to consider is how the size of the measurement window—set to $N = 64$ to obtain the results reported in Tables 2.3 and 2.4—affects the monitoring operation. We test C-MON and CS-MON on measurement windows of various lengths and plot, in Fig. 2.12, the relative error achieved by these two strategies as a function of sample size for small measurement windows for the AnonPages signal. The advantage of C-MON and CS-MON in exploiting the compressibility of the signal shrinks with window size.

Recall that N samples must first be collected at the server at some sampling rate to generate

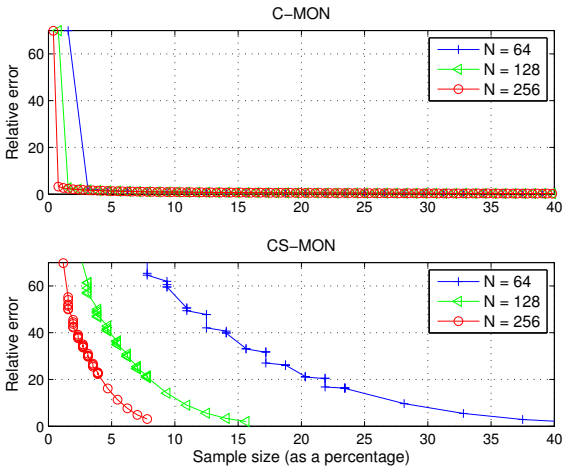


Figure 2.12: The relative error achieved by C-MON and CS-MON when using smaller measurement windows.

a packet of M data items to transmit to the monitoring station, where $M \ll N$. The size of this measurement window dictates how far behind the monitor lags the actual execution of the system, a delay we call the *packetization delay*. Table 2.6 shows this delay, in seconds, as a function of N and the sampling rate. For example, if $N = 128$ with a sampling rate of 0.5 Hz, the delay is 256 seconds. That is, the monitor lags about 4 minutes behind the system. So, as a practical matter, smaller values of N are better suited for real-time monitoring of the system. A key difference between the two strategies is that in the case of CS-MON the compressed samples are ready to be transmitted immediately upon receiving the last data point in the measurement window, whereas the packetization delay for C-MON includes the additional time required for wavelet transformation and sorting to obtain the M largest coefficients.

The foregoing discussion exposes some interesting tradeoffs between the C-MON and CS-MON strategies, particularly those related to the choice of the measurement window size N. In choosing the appropriate N, one must consider the tradeoff between the CPU and storage overhead due to these strategies and the resulting network traffic. If responsive operation is desired then N must be small. However, when monitoring the system for faults that manifest themselves over many minutes or hours, such as memory leaks, or with the intent of simply logging the data for off-line analysis, N can be made larger to better exploit the compressibility of the underlying signals.

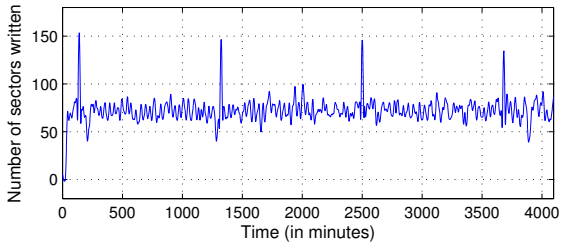


Figure 2.13: The write_activity signal collected from the testbed that measures the number of disk sectors written.

2.6.3 Case Studies

We test the signals reconstructed via the C-MON and CS-MON methods to detect two special cases: abrupt changes in performance-related data and gradual exhaustion of system resources.

Detection of Threshold Violations. As discussed earlier, there are situations in which it is not essential to recover the original signal exactly. If the datacenter operator is mainly interested in detecting performance bottlenecks, hot spots, or anomalies affecting the computing system that manifest themselves as abrupt changes in the signals being monitored, it is more important that the reconstructed signal preserve these characteristics. We evaluate how well threshold violations are detected by C-MON and CS-MON using as an example the *write_activity* signal shown in Fig. 2.13. This signal was collected from our testbed during an experimental run and the signal measures the number of sectors written to the hard disk. Fundamentally, we are interested in how well the reconstructed *write_activity* signal preserves the abrupt changes and spikes present in the original. More formally, if $\mathbf{d}$ is the signal of interest, we consider $\mathbf{d}(i)$, the i^{th} data point within the signal, a $p\%$ violation if it satisfies

$$\mathbf{d}(i) \geq Q(\mathbf{d}, 100 - p), \tag{2.12}$$

where $Q(\mathbf{d}, 100 - p)$ is the $(100 - p)$-th percentile of observations in $\mathbf{d}$. We use a *hit-rate* metric to characterize the number of abrupt changes that can be detected using the reconstructed signal by defining a hit as follows: at time t within the original and recovered signals, a spike occurring in the recovered signal matches a similar spike in the original signal.

Figure 2.14 shows the hit rate achieved when detecting a 1.6% violation in the *write_activity* signal as a function of the sample size used to recover this signal. The baseline method used for

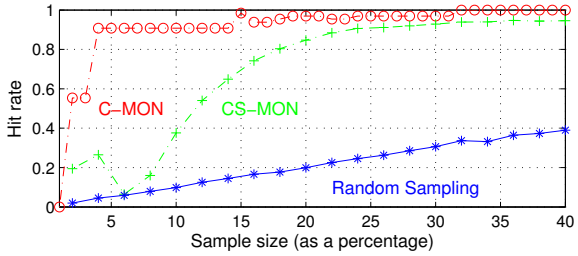


Figure 2.14: Hit rate when detecting a 1.6% violation in the *write_activity* signal, achieved by C-MON, CS-MON, and random sampling. The length of the measurement window is $N = 64$.

comparison is random sampling and its hit rate is linear with the sample size, as expected. The hit rate achieved by CS-MON is significantly better than random sampling at the same sampling rate: greater than 90% when using 25% of the original samples, for example. We observe that the spikes in the original write_activity are underestimated if a smaller sample size is used, leading to a lower hit rate. We also observe that the sample size required to detect these spikes is much lower than the sparsity of the write_activity signal, which is around 50% when the window length is 64. This result shows that when it is not essential to recover the original signal exactly, using a sample size that is smaller than the sparsity still enables us to preserve abrupt changes of the signal. In summary, both C-MON and CS-MON achieve a hit rate greater than 90%, with C-MON able to better compress the signal, using about 4% of the original signal. Note, however, that the previously discussed tradeoff between C-MON and CS-MON, in terms of data transfer, CPU, and storage overhead, applies in this scenario as well.

Detection of Trends. In the second case, we evaluate C-MON and CS-MON for trend detection, which can help detect, for example, whether a system resource is slowly being consumed to exhaustion. We use a long-running Trade6 application executing over a period of 48 hours and inject a small memory leak of about 100 KB/minute at around the 24-hour mark. Noting that there exists an increasing trend in the dataset CommittedAS, we apply C-MON and CS-MON on this dataset and use the reconstructed data for analysis.

To simplify the problem, we use time bins that each includes 1024 data points in 34 minutes. Sampling and reconstruction are applied on each time bin separately. To estimate the global trend, we use a 24-hour sliding window, which includes 40 time bins, and move the sliding window by one

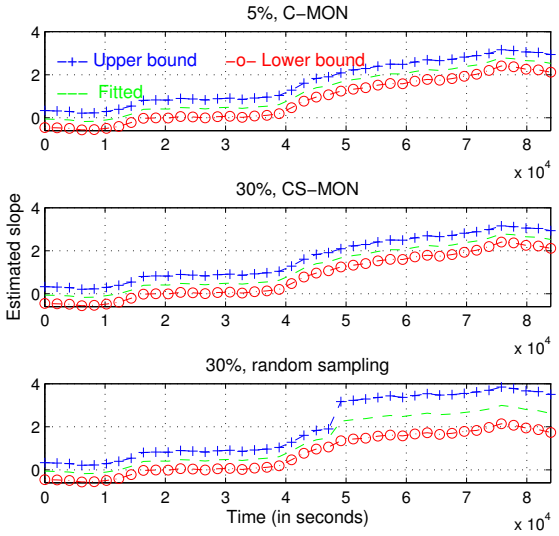


Figure 2.15: The slope estimated by C-MON, CS-MON and random sampling over 40 time windows for CommittedAS.

time bin at each step. We use the linear model $d = a \times t + b$, where d is the average of reconstructed data points within each time bin, t is the beginning time of the sliding window, a is the slope, and b is the intersection. For each sliding window, 40 (d, t) pairs are applied to this model and the slope within each sliding window is estimated along with the 95% confidence interval.

Figure 2.15 shows the estimated global slope for CommittedAS, for the first 24 hours followed by the second 24 hours, indicating that both C-MON and CS-MON perform better than random sampling: the width of the 95% confidence interval is tighter than the one obtained using random sampling, indicating greater certainty in detecting the slope. We also observe an increasing trend in the estimated slope. For CommittedAS, the estimate is around 0 at the beginning indicating no increasing trend associated with this feature. Then, the slope values increase and the lower bound on their 95% confidence interval becomes positive, indicating an increasing trend by the end of the experiment. This result shows that compressive sampling is effective for trend detection as well.

2.7 Related Work

To the best of our knowledge, compressive sampling has not been previously studied as a performance monitoring tool in the context of cloud computing except in our earlier work in [81] and

[82] where the emphasis was on studying the feasibility of using fixed-rate compressive sampling for monitoring server systems. This work has developed an adaptive-rate model that exploits any time-varying sparsity in the signal to further reduce the number of collected samples.

Tuma *et al.* study the applicability of compressive sampling for fine-grained monitoring of processor performance and evaluate its performance on signals representing micro-architecture counters within a core [72]. They show that compressive sampling can recover these signals if one can identify the bases in which the signals can be sparsely represented. Their approach bears some similarity to our work, but the measurements are obtained from hardware counters inside various micro-architectural components of the processor and the evaluation is limited to a signal-to-noise metric—same as the relative error metric used in this chapter. In contrast, this chapter combines compressive sampling with sparsity prediction for rate adjustment, and evaluates its performance in spike detection and trend detection with the reconstructed signal. Besides, rather than using traditional algorithms such as Orthogonal Matching Pursuit (OMP) for signal recovery as in [72], we employ a faster iterative Hard Thresholding Pursuit algorithm originally proposed in [33].

Work on adaptive-rate sampling in other domains includes the results reported in [29, 73, 94, 95, 96, 97, 98]. Meng *et al.* propose a method for distributed state monitoring in Cloud datacenters wherein the sampling intervals are updated dynamically according to the likelihood of detecting a threshold-based state violation [29]. Ward shows in [99] that side information of cross-validation measurements can be used to improve the recovery algorithm used in compressive sampling and reduce the sampling rate. For the similar purpose of improving signal recovery algorithms under a reduced sample rate for compressive sampling of images and videos, Stankovic *et al.* use spatial and temporal correlations within images and videos [95]. OMP is employed in [95] as the recovery algorithm. Besides, spatial correlations between non-overlapping blocks of the same image are exploited in [95] for image recovery, and temporal correlations between one frame and the previous frame of a video are exploited for video recovery. Sampling rates for compressive sampling are adaptively adjusted to collect biomedical data [96] and surveillance data [97].

One of the strategies proposed by Warnell *et al.* for sampling surveillance videos bears some similarity to our work [98]. Cross-validation error is used for sparsity estimation. However, only the sparsity of the foreground in each frame needs to be estimated, and the foreground data can be fitted

approximately by Gaussian distribution. The data used in this chapter, however, cannot be easily fitted into any known distributions. Moreover, the scheme proposed in this chapter accomplishes sparsity prediction using Kalman filter in addition to sparsity estimation.

Finally, it could be argued that any number of existing compression algorithms, especially lossless ones such as `bzip2` and `DEFLATE`, could be used to condense the information monitored at the data center before writing to disk. However, massive data acquisition followed by compression is extremely wasteful of computing, memory, and network resources. Compressive sampling, on the other hand, enables us to acquire data directly in compressed form.

2.8 Summary

The adaptive sampling strategy developed here exploits any available time-varying sparsity information within the underlying signal to reduce the number of samples collected when compared to a fixed-rate scheme. The reconstructed signal adequately preserves properties in the original signal that are useful for performance management and anomaly detection. The sensing scheme has low computation and storage complexity, making it suitable for use on the local server. It is also universally applicable—simply measuring signals via a random Gaussian sensing matrix—and does not have to be tailored in any way to the type of signal being measured or the type of anomaly being detected.

Finally, though the sensing costs are substantially reduced on the local server, the burden of reconstructing the signal falls on the monitoring station. In an effort to minimize this overhead as well, our ongoing work aims to develop techniques that detect anomalies by analyzing the compressed data directly without having to recover the signal to perform the analysis.

3. Anomaly detection in computer systems with compressed measurements

Online performance monitoring of computer systems incurs a variety of costs: the very act of monitoring a system interferes with its performance and if the information is transmitted to a monitoring station for analysis and logging, this consumes network bandwidth and disk space. Compressive sampling-based schemes can help reduce these costs on the local machine by acquiring data directly from the system in a compressed form, and in a computationally efficient way. This chapter focuses on reducing the computational cost associated with recovering the original signal from the transmitted sample set at the monitoring station for anomaly detection. Towards this end, we show that the compressed samples preserve, in an approximate form, properties such as mean, variance, as well as correlation between data points in the original full-length signal. We then use this result to detect changes in the original signal that could be indicative of an underlying anomaly such as abrupt changes in magnitude and gradual trends without the need to recover the full-length data. We illustrate the usefulness of our approach via case studies involving IBM's Trade Performance Benchmark using signals from the disk and memory subsystems. Experiments indicate that abrupt changes can be detected using a compressed sample size of 25% with a hit rate of 95% for a fixed false alarm rate of 5%; trends can be detected within a confidence interval of 95% using a sample size of only 6%. The material presented in this chapter was previously published in [100].

3.1 Introduction

Online monitoring of performance-related metrics is a necessary first step towards detecting anomalies in computer systems. Measurements may include high-level metrics such as response time and throughput as well as low-level ones such as processor utilization, disk I/O, memory, and network activity. The monitored information helps detect performance-related hotspots and bottlenecks as well as incipient faults associated with gradual resource exhaustion—the so-called software aging problem [35, 75, 101]. In the case of intermittent problems that are hard to isolate, browsing back through historical data can help identify and localize recurring problems affecting

the same portion of the computing infrastructure at different times. The data can also help detect security breaches resulting in the computers being infected by malicious software [102].

We consider a server cluster wherein software-based sensors embedded within the infrastructure measure various performance-related parameters associated with the cluster. The measured information is transmitted over a network to a monitoring station for data analysis and visualization. Online monitoring, however, incurs a variety of costs. First, the very act of monitoring an application interferes with its performance. If sensing-related code is merged with the application code, this change may interfere with the timing characteristics of the application or if sensors execute as separate processes, they contend for CPU resources along with the original application. Transmitting the monitored data over a network consumes bandwidth. Finally, logging the data for future analysis consumes disk space. So, when monitoring a large-scale computing system it is desirable to minimize the above-described costs.

Traditional methods of sampling signals use Shannon's theorem: the sampling rate must be at least twice the signal bandwidth to capture all the information content present in the signal. The theory of *compressive sampling* (*CS*) states, however, that we can recover a certain class of signals from the original measurements using far fewer samples than that used by techniques that rely on Shannon's theorem [32, 34, 87, 88]. In previous work reported in [81, 82, 83], we have developed CS-based sampling strategies for the performance monitoring of computing systems that can acquire signals of interest from the underlying system *directly* in a compressed form. The methods exploit the fact that the signals often can be sparsified—that is, encoded concisely—under an appropriate representation basis and that the sampling rate itself can be tuned as a function of sparsity. We show experimentally that the recovered signals can be used to detect, with high confidence, the existence of trends within the original signal as well as abrupt changes where the signal's magnitude exceeds some threshold value. Detection of these anomalies is achieved using a substantially reduced sample size—a reduction of more than 70% when compared to the standard fixed-rate sampling method.

Compressive sampling allows for a very simple sampling strategy on the local machine. Rather than tailoring the sensing scheme to capture specific properties in the underlying signal, a *signal-independent strategy* such as randomized sampling can be used, significantly reducing the intrusion of monitoring on application performance [83]. Also, since signals are acquired directly in com-

pressed form, the network bandwidth needed to transmit these few samples to the monitoring station is reduced and so is the hard-disk space required to store them.

When operators wish to analyze the original signal, there is a way to use numerical optimization to reconstruct the full-length signal from the sample set. The reconstructed signal allows for real-time anomaly detection and diagnosis, and also helps drive decisions of a longer-term nature such as intelligent capacity planning. This chapter focuses on reducing the computational cost incurred by the monitoring station, associated with recovering the original signal from the sample set and analyzing it for anomalies. The recovery process is typically posed as a linear programming (LP) problem and solved under some sparsity assumptions using a class of reconstruction algorithms called basis pursuit or iterative hard thresholding pursuit [33]. Though modern LP solvers are quite efficient, each monitoring station may be responsible for recovering and analyzing hundreds of signals belonging to many servers, making it important to reduce the corresponding overhead. The chapter makes the following contributions towards this goal:

- We prove from a theoretical viewpoint that the compressed samples preserve statistical properties of the original data such as variance and mean. This result allows for the detection of abrupt changes and trends by *directly analyzing* just the compressed samples without having to reconstruct the full-length signal.

- Since the sampling process is just a linear projection of the original data, we also prove that the compressed samples approximately *preserve spectral properties* such as correlation between data points, the length of the data vectors, as well as the distance between two vectors, under such a projection. This result allows for well-known anomaly detection methods such as principal component analysis (PCA) to be used directly on the compressed samples; performance is almost equivalent to the case in which the raw data is completely available.

We illustrate the usefulness of the approach via case studies using IBM's Trade Performance Benchmark (also known as Trade6). We measure signals from the disk and memory subsystems using a CS-based sampling strategy, and analyze the compressed samples for possible anomalies. The first scenario involves detecting abrupt changes in the signal during which the magnitude exceeds some nominal threshold value. In the second scenario, we wish to detect the gradual deterioration

of system performance, say over hours or days, associated with software aging by statistically analyzing the appropriate signals for the existence of trends [74, 75]. We use a long-running Trade6 application having a small memory leak and evaluate the ability of the approach to estimate a positive slope in the compressed data even in the presence of seasonal variations and periodicity in the signal. Finally, we evaluate the efficacy of applying PCA to the compressed samples to detect abrupt changes in the signal.

Abrupt changes can be detected using a sample size of 25% with a hit rate of 95% for a fixed false alarm rate of 5%; trends can be detected with a confidence interval of 95% using a sample size of 6%. Finally, the hit rate achieved by the PCA-based analysis when using the compressed samples to detect abrupt changes is higher than 95% when the false alarm is fixed at 0.5%. The corresponding sample size is about 18%. These results point to the feasibility of adopting a two-step anomaly detection process at the monitoring station: the received compressed data is examined for possible anomalies; if one is suspected, the relevant portion of the signal is fully reconstructed to localize and further analyze the anomaly.

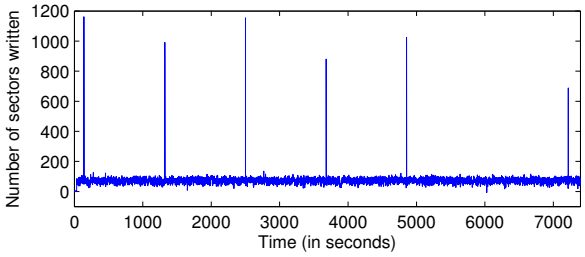
The chapter is organized as follows. Section 3.3 describes the theoretical basis behind being able to detect anomalies in compressed data and Section 3.4 presents the case studies that evaluate the performance of the approach. Section 3.5 discusses related work and Section 3.6 provides some concluding remarks.

3.2 Experimental settings

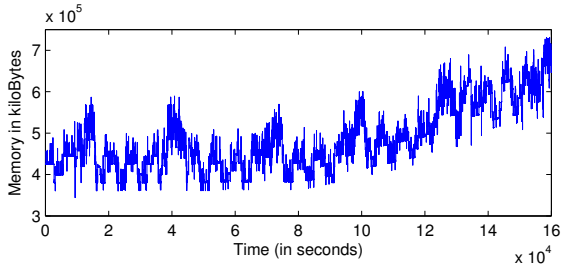
The system setup used in our experiments is the described in Chapter 2.2. We use the workload shown in Figure 2.1(b) and collect the following two features: *CommittedAS*, and *write_activity*. Figure 3.1 plots these two features during an experimental run of 48 hours. The data points are sampled once every two seconds.

3.3 Anomaly Detection using Compressed Samples

The optimization problem posed in Eq. (2.7) can be solved efficiently by modern LP solvers. However, a monitoring station may be responsible for recovering and analyzing hundreds of signals belonging to many servers. In the following, we prove from a theoretical viewpoint that key



(a) Disk sectors written, the *write_activity* signal.



(b) Memory allocated to proesses, the *CommittedAS* signal.

Figure 3.1: Signals corresponding to: (a) I/O activity collected at the database tier showing the number of sectors written to disk and (b) total amount of memory allocated by processes in the system. Note that a memory leak has been injected around the 24 hour mark.

statistical properties of the original data such as the variance, mean, and correlation between data points are preserved in approximate form within the compressed samples. We also show how these properties can be exploited to detect, with high confidence, data spikes and trends by analyzing just the compressed samples without having to reconstruct the full-length signal.

We first describe the compressive sampling process on each local server in greater detail since this knowledge is crucial to understanding the claims and the proofs presented later in this chapter. As shown in the schematic in Figure 2.4, this process involves multiplying the incoming signal $\mathbf{d}$ (treated as a vector) with a sampling matrix to form a set of compressed measurements $\mathbf{y}$. When a new data item $d(t)$ arrives at time t, it is multiplied by the entries in the sampling matrix $G(i,t), i = 1,\ldots,M$ and the partial products are accumulated into $y(i)$. After a period of length $N \times T$, where T is the sampling period, the current values of $y(i)$ are sent out as the M samples to the monitoring station, and then reset back to zero. So, effectively, $y(i) = \sum_{t=1}^{N} G(i,t)d(t)$, where $i = 1,\ldots,M$, giving us $\mathbf{y} = G\mathbf{d}$.

The sampling matrix is often designed to have random Gaussian entries [34]. Thus, the gener-

ated compressed samples are linear combinations of the original data, and these samples are able to preserve certain properties of the original data.

3.3.1 Detection of spikes from compressed samples

We show in the following that a data spike, i.e., an abrupt rise and fall in the value of a data item, within a time window leads to a significant change in the corresponding compressed-sample variance. Let $\mathbf{d}$ denote the vector of data within a window of length N exhibiting no data spikes. Let $\mathbf{d}'$ denote another length-N data series whose entries have the same magnitude as those in $\mathbf{d}$ with the exception of the n-th entry, $d(n)$, i.e., $d'(t) = d(t)$, $t = 1, \cdots, n-1, n+1, \cdots, N$, and $d'(n) = d(n)+\Delta$. If G is the $M \times N$ sampling matrix, then the compressed samples of length M corresponding to $\mathbf{d}$ and $\mathbf{d}'$ are $\mathbf{y} = G\mathbf{d}$ and $\mathbf{y}' = G\mathbf{d}'$, respectively.

Let us denote the sample mean of $\mathbf{y}$ and $\mathbf{y}'$ as $\mu(\mathbf{y})$ and $\mu(\mathbf{y}')$, respectively; denote the (unbiased) sample variance of $\mathbf{y}$ and $\mathbf{y}'$ as $\sigma^2(\mathbf{y})$ and $\sigma^2(\mathbf{y}')$, respectively. Let G_j denote the j^{th} column of G and let $\mu(G_j)$ denote the mean of the j^{th} column of G.

Let $\sigma^2(G_j)$ denote the (unbiased) sample variance of G's j-th column, that is $\sigma^2(G_i) = \frac{1}{M-1}\sum_{i=1}^{M}[G(i,j)-\mu(G_j)]^2$. Let $\sigma_G(j,t)$ denote the sample covariance between the j-th column and the t-th column of G, $\sigma_G(j,t) = \frac{1}{M-1}\sum_{i=1}^{M}[G(i,j)-\mu(G_j)][G(i,t)-\mu(G_t)]$.

In the following, we prove that the change in sample variance as a result of the data spike of size Δ in the n-th element of $\mathbf{d}$ leads to a change in the variance given by

$$\sigma^2(\mathbf{y}') - \sigma^2(\mathbf{y}) = \sigma^2(G_n)\Delta^2 + 2\Delta\sum_{t=1}^{N} d(t)\sigma_G(n,t). \tag{3.1}$$

Proof. First, we show that the sample mean of $\mathbf{d}'$ differs from that of $\mathbf{d}$ by $\mu(G_n)\Delta$.

$$\begin{aligned}
\mu(\mathbf{y}) &= \frac{1}{M}\sum_{i=1}^{M} y(i) = \frac{1}{M}\sum_{i=1}^{M}\sum_{t=1}^{N} G(i,t)d(t) \\
&= \sum_{t=1}^{N}[\frac{1}{M}\sum_{i=1}^{M} G(i,t)]d(t) = \sum_{t=1}^{N}\mu(G_t)d(t) \\
\mu(\mathbf{y}') &= \sum_{t=1}^{N}\mu(G_t)d'(t) = \sum_{t=1}^{N}\mu(G_t)d(t) - \mu(G_n)d(n) + \mu(G_n)d'(n) \\
&= \sum_{t=1}^{N}\mu(G_t)d(t) + \mu(G_n)\Delta = \mu(\mathbf{y}) + \mu(G_n)\Delta
\end{aligned}$$

In the following we show that due to the change by Δ the sample mean of $\mathbf{d}'$ differs by $\sigma^2(G_n)\Delta^2 + 2\Delta\sum_{t=1}^{N} d(t)\sigma_G(n,t)$, where $\sigma^2(G_n)$ is the sample variance of G's n-th column and $\sigma_G(n,t)$ is the sample covariance between the n-th column and t-th column of G.

Note that, since $y'(i) = y(i) + G(i,n)\Delta$, $y'(i) - \mu(\mathbf{y}')$ is equivalent to $y(i) - \mu(\mathbf{y}) + [G(i,n) - \mu(G_n)]\Delta$. This enables us to break $\sigma^2(\mathbf{y}') = \frac{1}{M-1}\sum_{i=1}^{M}[y'(i) - \mu(\mathbf{y}')]^2$ into three parts:

$$\begin{aligned}
&\frac{1}{M-1}\sum_{i=1}^{M}\{y(i) - \mu(\mathbf{y})\}^2 \\
&+\frac{1}{M-1}\sum_{i=1}^{M}[G(i,n) - \mu(G_n)]^2\Delta^2 \\
&+\frac{1}{M-1}\sum_{i=1}^{M}\left\{2\sum_{t=1}^{N}[G(i,t) - \mu(G_t)]d(t)[G(i,n) - \mu(G_n)]\Delta\right\}
\end{aligned}$$

The first part is the (unbiased) sample variance of $\mathbf{y}$, $\sigma^2(\mathbf{y})$. By the definition of $\sigma^2(G_n)$, $\sigma^2(G_n) = \frac{1}{M-1}\sum_{i=1}^{M}[G(i,n) - \mu(G_n)]^2$. Therefore, the second part is equivalent to $\sigma^2(G_n)\Delta^2$. By the definition of $\sigma_G(n,t)$, $\sigma_G(n,t) = \frac{1}{M-1}\sum_{i=1}^{M}[G(i,n) - \mu(G_n)][G(i,t) - \mu(G_t)]$; therefore, the third part equals $2\Delta\sum_{t=1}^{N} d(t)\sigma_G(n,t)$.

As a result, the following holds true:

$$\sigma^2(\mathbf{y}') = \sigma^2(\mathbf{y}) + \sigma^2(G_n)\Delta^2 + 2\Delta\sum_{t=1}^{N} d(t)\sigma_G(n,t)$$

□

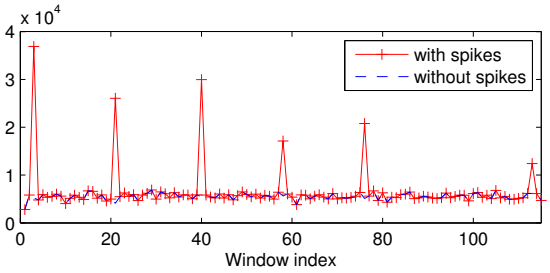


Figure 3.2: The sample variance of the compressed samples for *write_activity* data and the spike-free data in each time window.

In the proof of Eq. (3.1), we also prove that $\mu(\mathbf{y}') = \mu(\mathbf{y}) + \mu(G_n)\Delta$. Note that since entries of G are independently generated random Gaussian variables, the expected column mean of G is 0. As a result, the expected difference between $\mu(\mathbf{y})$ and $\mu(\mathbf{y}')$, i.e., $\mu(G_n)\Delta$, is 0. A large Δ, therefore, cannot be expected to cause a significant change in the sample mean.

The expected column variance of G is $1/M$. The fact that two columns of G are independent of each other leads to zero covariance between any two columns of G. As a result, the expected value of $\sigma^2(G_n)$ is $1/M$ and the expected value of $\sigma_G(n,t)$ is 0. The expected value of the difference between $\sigma^2(\mathbf{y})$ and $\sigma^2(\mathbf{y}')$, i.e., $\sigma^2(G_n)\Delta^2 + 2\Delta\sum_{t=1}^{N} d(t)\sigma_G(n,t)$, is therefore equal to Δ^2/M. Therefore, a large Δ may not lead to a significant change in the sample mean, but it will lead to a significant change in the sample variance.

This inference from Eq. (3.1) is consistent with our observations. For the *write_activity* data, we set the window length to $N = 64$ and the sample size to $M = 16$; so for every 64 data points, 16 compressed samples are generated. We then calculate the sample variance for each window. We also repeat the sampling and variance calculation on the data without any spikes. An overlay of the sample variance for the data with and without spikes is shown in Figure 3.2. Windows containing spikes have extremely large variance compared with other time windows.

Our method to detect data spikes or abrupt changes is to collect data in each time window using compressive sampling, and calculate the sample variance of each time window. We then detect a window that has a variance exceeding a threshold as the window that contains spikes or abrupt changes. To select the proper threshold for the sample variance, we learn its distribution by training a dataset that contains no spikes. After obtaining the set of variance for each time window, we

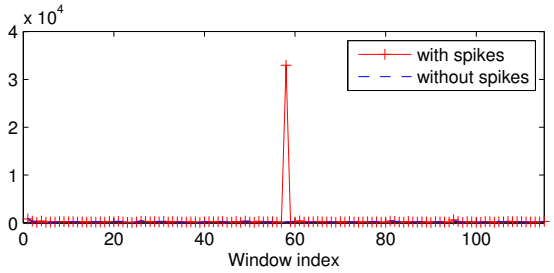


Figure 3.3: The variance of the random samples collected for *write_activity* in each time window.

obtain the sample mean μ and sample variance σ, and set the threshold to $\mu + \sigma z_{1-\alpha}$, where $z_{1-\alpha}$ is the upper $1-\alpha$ critical value of a standard normal distribution.

During the detection phase, a time window that has a sample variance above this threshold is said to contain spikes or abrupt changes. To find out the exact time stamp within the particular window for the spikes, one can use the reconstruction algorithm, such as HTP, on samples of the detected time window to recover the original data.

Note that compressive sampling preserves the data spikes better than random sampling. To compare these two sampling techniques, we use random sampling to collect the same amount of samples from the *write_activity* data, and show the sample variance in Figure 3.3. The sample variance is extremely large only for the window that contains the fourth spike. The other spikes are not selected as the samples, which is expected since a spike in the original data is selected as a sample with probability M/N.

3.3.2 Detection of trends from compressed samples

In the following, we show that the compressed samples preserve the mean of the original data. Let $\mu(G)$ be the average of all entries of G, $\mu(G) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{t=1}^{N} G(i,t)$. As before, denote the mean of $\mathbf{d}$ and $\mathbf{y}$ as $\mu(\mathbf{d})$ and $\mu(\mathbf{y})$, respectively. We prove in the following that

$$\mu(\mathbf{y}) \overset{P}{\approx} N\mu(G)\mu(\mathbf{d}), \tag{3.2}$$

where $\overset{\mathrm{P}}{\approx}$ stands for being approximately equal with high probability. In other words, the sample mean is approximately the mean of the original data scaled by $N\mu(G)$. Given this property, when

the original data increases or decreases in magnitude, the mean of the compressed samples increases or decreases proportionally.

Proof. From the proof of Equation (3.1), we know that the compressed sample mean $\mu(\mathbf{y})$ is equal to $\sum_{t=1}^{N} \mu(G_t)d(t)$. By the definition of $\mu(G)$, we have $\mu(G) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{t=1}^{N} G(i,t) = \frac{1}{N}\sum_{t=1}^{N}\mu(G_t)$, which leads to the following:

$$\frac{\mu(G_t)}{N\mu(G)} = \frac{\mu(G_t)}{\mu(G_t) + \sum_{j=1,j\neq t}^{N}\mu(G_j)} = \frac{1}{1 + \frac{\sum_{j=1,j\neq t}^{N}\mu(G_j)}{\mu(G_t)}} \tag{3.3}$$

Note that entries of G, $G(i,t)$, are independent and follow Gaussian distribution with zero mean and variance $1/M$. As a result, the t-th column mean of G, $\mu(G_t) = \frac{1}{M}\sum_{i=1}^{M} G(i,t)$, also follows a Gaussian distribution with zero mean, but with a variance of $1/M^3$ (denoted by $\mathcal{N}(0, 1/M^3)$). Therefore, $\sqrt{M^3}\mu(G_t)$ can be said to follow $\mathcal{N}(0,1)$.

Similarly, $\sum_{j=1,j\neq t}^{N}\mu(G_j)$ follows $\mathcal{N}(0,(N-1)/M^3)$. As a result, $\sqrt{M^3/(N-1)}\sum_{j=1,j\neq t}^{N}\mu(G_j)$ can be said to follow $\mathcal{N}(0,1)$.

The ratio of the above two independent variables which both follow the standard normal distribution $\mathcal{N}(0,1)$ is given by:

$$\frac{\sqrt{\frac{M^3}{N-1}}\sum_{j=1,j\neq t}^{N}\mu(G_j)}{\sqrt{M^3}\mu(G_t)} = \frac{1}{\sqrt{N-1}}\frac{\sum_{j=1,j\neq t}^{N}\mu(G_j)}{\mu(G_t)}$$

Thus, $\frac{1}{\sqrt{N-1}}\frac{\sum_{j=1,j\neq t}^{N}\mu(G_j)}{\mu(G_t)}$ follows the standard Cauchy distribution, denoted by Cauchy(0, 1).

We now apply the fact that if a random variable X follows Cauchy(μ,σ), then the linear transformation $\alpha X + \beta$ follows Cauchy$(\alpha\mu+\beta, |\alpha|\sigma)$ [103]. This implies that $1 + \frac{\sum_{j=1,j\neq t}^{N}\mu(G_j)}{\mu(G_t)}$ follows Cauchy$(1, \sqrt{N-1})$.

Similarly, we again apply another result on transformation of Cauchy distributions: if X follows Cauchy(μ,σ), then $1/X$ follows Cauchy$(\mu/c, \sigma/c)$ where $c = \mu^2 + \sigma^2$ [103]. This implies that the right hand side of (3.3) follows Cauchy$(1/N, \sqrt{N-1}/N)$.

Given (3.3) and given the probability density function of the above Cauchy distribution with mean $1/N$ and small variance, it follows that $\mu(G_t)/(N\mu(G))$ is $1/N$ or close to it with high probabil-

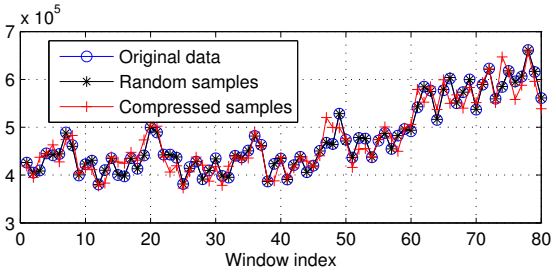


Figure 3.4: The average value of the original *CommittedAS* data in each time window overlaid with the scaled average of the compressed samples and random samples.

ity. Since $\mu(\mathbf{y})$ is equal to $\sum_{t=1}^{N} \mu(G_t)d(t)$, $\frac{\mu(\mathbf{y})}{N\mu(G)}$ is close to $\sum_{t=1}^{N} \frac{1}{N}d(t) = \mu(\mathbf{d})$ with high probability. That is, $\mu(\mathbf{y})$ is close to $N\mu(G)\mu(\mathbf{d})$ with high probability. □

The implications of Eq. (3.2) are consistent with our simulation results. For the *CommittedAS* dataset, we use time bins that each includes 1024 data points in 34 minutes. Sampling is applied on each time bin separately. The sample size for compressive sampling is 3.12%. Figure 3.4 shows the average of the original data in each time window compared with the scaled mean of the compressed samples (scaled by $1/N\mu(G)$). The two sets of average values are similar and both of them can capture the increasing trend in the *CommittedAS*. We also show the average of random samples in Figure 3.4 and find that the values are closer to the average of the original data. However, the process of random sampling loses information of the original data. The original data cannot be fully recovered using the random samples. On the other hand, the samples collected via compressive sampling can be used to reconstruct the original data with much greater fidelity.

To detect the increasing trend in *CommittedAS*, we exploit the fact that the average of the original data within each time window is preserved in scaled form by the compressed samples. To estimate the global trend, we use a 24-hour sliding window, which includes 40 time bins, and move the sliding window by one time bin at each step. We use the linear model $d = a \times t + b$, where d is the average of compressed samples within each time bin, t is the beginning time of the sliding window, a is the slope, and b is the intersection. For each sliding window, 40 (d, t) pairs are applied to this model and the slope within each sliding window is estimated along with the 95% confidence interval. We use the slope estimates to check whether an increasing trend or a decreasing trend exists in the original data.

3.3.3 PCA-based detection from compressed samples

Principal component analysis (PCA) is a dimension reduction technique that is frequently used for anomaly detection [48, 52]. It transforms a high-dimensional dataset into new bases called principal components ordered by the strength of the correlations exhibited by the data along their respective directions. As a result, the first principal component captures the strongest correlation pattern of the original data, the second principal component captures the second strongest correlation pattern, and so on [53]. The first few principal components are often chosen as the signature pattern of the data.

For anomaly detection purposes, PCA is typically applied on the original raw data. In this work, on the other hand, we apply PCA directly on the compressed samples to extract normal patterns of behavior present in the original data. These extracted features are then exploited to detect occurrences such as spikes in this data. Our method comprises two phases: training and detection. During training, we use a large dataset to obtain its signature pattern. To remove extreme values from the dataset, we replace the top 0.1% of the data with its median value. For every N data points, M samples are generated via compressive sampling. Assume the sample from the t-th time window is the length-M vector $\mathbf{s}_t$ and that the obtained samples from T windows are $[\mathbf{s}_1, \cdots, \mathbf{s}_T]$. Applying PCA on the sample sets gives us M principal components $\mathbf{p}_1, \cdots, \mathbf{p}_M$, each a length-$M$ vector.

We then study the strength of the correlation captured by each of the principal components and choose the top k among them which capture more than 95% of the correlations. We then define the normal pattern of the data using the first k principal components as $P_k = [\mathbf{p}_1, \cdots, \mathbf{p}_k]$, also referred to as the *normal subspace*. The *anomalous subspace* is the subspace orthogonal to the normal subspace.

In the detection phase, we subtract the statistical mean $\mu(\mathbf{y})$ from the compressed data $\mathbf{y}$, project it onto the normal subspace P_k, and then examine the difference between the projection and the original test data $\mathbf{y} - \mu(\mathbf{y}) - P_k P_k^T(\mathbf{y} - \mu(\mathbf{y}))$. We then compare the norm of the difference $r_y = \| \mathbf{y} - \mu(\mathbf{y}) - P_k P_k^T(\mathbf{y} - \mu(\mathbf{y})) \|^2$, which we call the *projection residual*, to a certain threshold, and issue an alert indicating an anomaly when it exceeds the threshold. We set the threshold to $\sqrt{2(N-k)(N/M+1)}z_{1-\alpha}+$

$N - k$; the choice of this threshold is explained later in this section.

Rationale behind the method. The PCA-based detection described above is based on the fact that the projection residual r_y as a result of our detection process is sufficiently similar to r_d to allow the use of r_y (in place of r_d) for anomaly detection. Here, r_d is the residual obtained using the full-length data.

Let Σ_d be the covariance matrix of the original data $\mathbf{d}$ with $\Sigma_d = U\Lambda U^T$ being the singular value decomposition of Σ_d where U is a unitary matrix and Λ is a diagonal matrix. The columns of U, $\mathbf{u}_i$, are the eigenvectors/principal components of the original data and the diagonal entries of Λ, λ_i, are the corresponding eigenvalues. Let Σ_y be the covariance matrix of the compressed samples with $\Sigma_y = V\Lambda^* V^T$, where $V = [\mathbf{v}_1, \cdots, \mathbf{v}_N]$, being the principal components of Σ_y. The diagonal entries of Λ^*, λ_i^*, are the eigenvalues.

The sampling matrix for compressive sampling is a random Gaussian matrix, the entries of which follow the Gaussian distribution of zero mean and variance $1/M$. As a result, the obtained compressed samples are linear combinations of the original data. Such linear projections should preserve some spectral properties of the original data.

Our detection method relies on one major finding. We find that when the test data is normal, key statistical properties of the distribution of r_y, the projection residual as a result of applying the subspace method directly on the compressed samples, is related to those of the distribution of r_d, the projection residual of the original data. More specifically, we prove in the following that

$$E(r_y) - E(r_d) = O(1). \tag{3.4}$$

That is, the difference between $E(r_y)$ and $E(r_d)$ is a constant independent of M or N.

Proof. First, we show that the mean of r_d is $E(r_d) = \sum_{i=k+1}^{N} \lambda_i$. Let I_N denote an $N \times N$ identity

matrix. By the definition of r_d, we have:

$$\begin{aligned} r_d &= \|(\mathbf{d} - \mu(\mathbf{d})) - U_k U_k^T (\mathbf{d} - \mu(\mathbf{d}))\|^2 \\ &= (\mathbf{d} - \mu(\mathbf{d}))^T (I_N - U_k U_k^T)(\mathbf{d} - \mu(\mathbf{d})) \\ &= (\mathbf{d} - \mu(\mathbf{d}))^T (UU^T - U_k U_k^T)(\mathbf{d} - \mu(\mathbf{d})) \\ &= (\mathbf{d} - \mu(\mathbf{d}))^T \sum_{i=k+1}^{N} u_i u_i^T (\mathbf{d} - \mu(\mathbf{d})) \\ &= \sum_{i=k+1}^{N} \|u_i^T (\mathbf{d} - \mu(\mathbf{d}))\|^2 \end{aligned}$$

Since $\|u_i^T(\mathbf{d} - \mu(\mathbf{d}))\|^2$ captures the variance of $\mathbf{d}$ along u_i^T, we have $E(r_d) = \sum_{i=k+1}^{N} E(\|u_i^T(\mathbf{d} - \mu(\mathbf{d}))\|^2) = \sum_{i=k+1}^{N} \lambda_i$. Similarly, the mean of r_y is $E(r_y) = \sum_{i=k+1}^{M} \lambda_i^*$.

Next, we show that r_d is similar to r_y by proving $E(r_y) - E(r_d)$ is a constant independent of M or N, or equivalently, $\sum_{i=k+1}^{M} \lambda_i^* - \sum_{i=k+1}^{N} \lambda_i = O(1)$. The proof is completed in two parts: (i) by proving that $E(\sum_{i=1}^{N} \lambda_i) = E(\sum_{i=1}^{M} \lambda_i^*)$, and (ii) by proving that $E(\sum_{i=1}^{k} \lambda_i^*) - E(\sum_{i=1}^{k} \lambda_i) = O(1)$.

Part (i): We first prove that $E(\sum_{i=1}^{N} \lambda_i) = E(\sum_{j=1}^{M} \lambda_j^*)$. Note that $\sum_{j=1}^{M} \lambda_j$ is also the trace of the covariance matrix Σ_d, i.e., $\text{tr}(\Sigma_d) = \sum_{i=1}^{N} \Sigma_d(i,i) = \sum_{i=1}^{N} \lambda_i$. Similarly, $\text{tr}(\Sigma_y) = \sum_{i=1}^{M} \Sigma_d^*(i,i) = \sum_{i=1}^{M} \lambda_i^*$. Recall that $\mathbf{y} = G\mathbf{d}$, we have $\Sigma_y = G\Sigma_d G^T = GU\Lambda U^T G^T$. The i-th diagonal entry of Σ_y, therefore, is given by:

$$\sum_{k=1}^{N} \lambda_k \Big(\sum_{j=1}^{N} G(i,j) u_k(j)\Big)\Big(\sum_{j=1}^{N} u_k(j) G(i,j)\Big)$$

As a result, the trace of Σ_y can be rewritten as

$$\begin{aligned} \text{tr}(\Sigma_y) &= \sum_{i=1}^{M} \sum_{k=1}^{N} \lambda_k \left(\sum_{j=1}^{N} G(i,j) u_k(j)\right)\left(\sum_{j=1}^{N} u_k(j) G(i,j)\right) \\ &= \sum_{k=1}^{N} \lambda_k \sum_{i=1}^{M} \left(\sum_{j=1}^{N} u_k(j) G(i,j)\right)\left(\sum_{j=1}^{N} G(i,j) u_k(j)\right) \\ &= \sum_{k=1}^{N} \lambda_k u_k^T G^T G u_k \end{aligned}$$

The difference between the traces of Σ_y and Σ_d is $\sum_{k=1}^{N} \lambda_k u_k^T (G^T G - I_N) u_k$, where I_N is an $N \times N$ identity matrix. Note that entries of matrix G follow $\mathcal{N}(0, \frac{1}{M})$. Therefore, $E(G^T G) = I_N$, and $E(\text{tr}(\Sigma_y)) = E(\text{tr}(\Sigma_d))$.

Part (ii): We next prove that $\sum_{i=1}^{k} \lambda_i \overset{P}{\approx} \sum_{j=1}^{k} \lambda_j^*$.

Consider a length-N vector $\mathbf{x}$ that follows a Gaussian distribution $\mathcal{N}(0, \Sigma_x)$ where $\Sigma_x = \text{diag}\{\lambda_1^{(x)}, \cdots, \lambda_N^{(x)}\}$ is a diagonal matrix that follows the spiked covariance model. It is shown in [104] that if we build $X = [\mathbf{x}_1, \cdots, \mathbf{x}_M]$ with M realizations of $\mathbf{x}$, then the i-th largest eigenvalue of matrix $S = \frac{1}{M} X X^T$, $\lambda_i^{(s)}$, satisfies:

$$\sqrt{M}(\lambda_i^{(s)} - \lambda_i^{(x)}) - \sqrt{M} \frac{\gamma \lambda_i^{(x)}}{\lambda_i^{(x)} - 1} \sim \mathcal{N}(0, 2\lambda_i^{(x)2} - \frac{\gamma 2 \lambda_i^{(x)2}}{(\lambda_i^{(x)} - 1)^2}) \tag{3.5}$$

as $M, N \to \infty$ and $\frac{N}{M} \to \gamma$. Applying the result of [104] to our case, we build X as $X = \Lambda^{\frac{1}{2}} U^T G^T$. The i-th column of X is $\mathbf{x}_i = \Lambda^{\frac{1}{2}} U^T g_i^T$. Due to randomness of $G(i, j)$, we have the mean of $\mathbf{x}_i$ $E(\mathbf{x}_i) = \Lambda^{\frac{1}{2}} U^T E(g_i^T) = 0$, and $E(\mathbf{x}_i \mathbf{x}_i^T) = \Lambda^{\frac{1}{2}} U^T E(g_i^T g_i) U \Lambda^{\frac{1}{2}} = \Lambda^{\frac{1}{2}} U^T \frac{1}{M} I_N U \Lambda^{\frac{1}{2}} = \frac{1}{M} \Lambda$. Therefore, $\lambda_i^{(x)} = \lambda_i / \sqrt{M}$. On the other hand, $S = \frac{1}{M} X X^T$ has the same eigenvalues with those of $\frac{1}{M} X^T X = \frac{1}{M} G \Sigma_d G^T = \frac{1}{M} \Sigma_y$ which implies that $\lambda_i^{(s)} = \lambda_i^* / \sqrt{M}$, $i = 1, \cdots, M$. Applying the result of [104] in our case allows us to relate λ_i^* to λ_i as:

$$(\lambda_i^* - \lambda_i) - \frac{\gamma \lambda_i}{\lambda_i - 1} \sim \mathcal{N}(0, 2\lambda_i{}^2 - \frac{\gamma 2 \lambda_i{}^2}{(\lambda_i - 1)^2}) \tag{3.6}$$

Using this relationship between λ_i^* and λ_i, we get that the difference between $E(\sum_{i=1}^{k} \lambda_i^*)$ and $E(\sum_{i=1}^{k} \lambda_i)$ is only a constant since $E(\sum_{i=1}^{k} \lambda_i^* - \sum_{i=1}^{k} \lambda_i) = E(\sum_{i=1}^{k} \gamma \lambda_i / (\lambda_i - 1)) = O(1)$.

Combining the results in Parts (i) and (ii) above, we have:

$$E(r_y) - E(r_d) = (\sum_{i=1}^{M} \lambda_i^* - \sum_{i=1}^{M} \lambda_i) - (\sum_{i=1}^{k} \lambda_i^* - \sum_{i=1}^{k} \lambda_i) = O(1)$$

□

Further, it is known from [105] that

$$\frac{\text{var}(r_y)}{\text{var}(r_d)} = 1 + \gamma + z(M), \tag{3.7}$$

where $z(M) = O(1/\sqrt{M})$, $N/M = \gamma$ for large N, $\lambda_1 > 1 + \sqrt{\gamma}$, and Λ_d follows the spiked covariance model, i.e., $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k > \lambda_{k+1} = \lambda_{k+2} = \cdots = \lambda_N = 1$.

The above results show that the mean of the two projection residuals differ by a constant and the variance of one is a scaled version of the other. As a result, it can be stated that the principal components of the compressed samples preserve some of the correlation relationship within the original data.

Under the spiked covariance model, the mean of r_d and r_y are both $\sum_{i=k+1}^{N} \lambda_i = N - k$; the variance of r_d is $\sum_{i=k+1}^{N} 2\lambda_i^2 = 2(N-k)$, and the variance of r_y is approximately $2(1 + N/M)(N-k)$. So, during the detection process, the threshold for the projection residual r_y is set to

$$\sqrt{2(N-k)(N/M+1)}z_{1-\alpha} + N - k.$$

3.4 Performance evaluation

We use the compressed samples for anomaly detection under the following two scenarios:

- The system operator wishes to detect performance-related bottlenecks or anomalies that manifest themselves as the magnitude of the signal exceeding some nominal threshold value—as spikes or abrupt changes. The performance in this regard is quantified by a hit-rate metric.

- The operator wishes to detect gradual performance deterioration, say over hours or days, associated with software aging or resource exhaustion by analyzing the signals for the existence of trends. Common causes involve resource exhaustion due to memory leaks and bloat, unreleased network sockets and file locks, and unterminated threads. Here the performance is quantified by the confidence with which the samples can be used to estimate a positive slope, if one exists.

3.4.1 Detection of Spikes and Abrupt Changes

This case study uses the *write_activity* signal shown previously in Figure 3.1(a) to evaluate how well threshold violations are detected from the compressed samples, specifically abrupt changes and

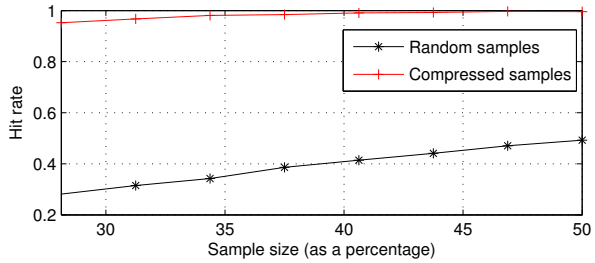


Figure 3.5: An overlay of the hit rate achieved as a result of using the compressed samples versus that of random sampling.

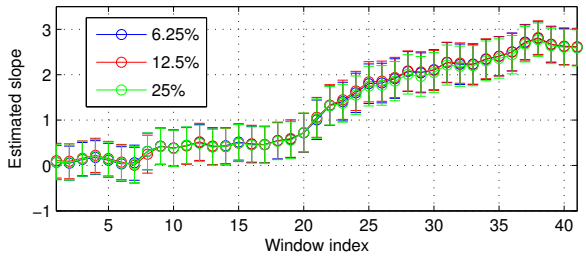
spikes present in the original full-length signal. We use a hit-rate metric to characterize the number of abrupt changes that can be detected by defining a hit as follows: given a time window t within the original data and the compressed samples, a spike occurring in the compressed samples matches a similar spike seen in the original signal.

Figure 3.5 shows the result of applying our method on the *write_activity* data as a function of the sample size. The length of each observation window is set to $N = 64$ data points. Once the sample size exceeds 28%, the hit rate is higher than 95% when the false alarm is fixed at 0.5%. (A sample size smaller than 28% would lead to a false alarm higher than 0.5%.) We find this result is significantly better than that obtained using randomly selected samples from the original signal. The hit rate achieved by random sampling is linear with the sample size, since a higher sampling rate proportionally increases the probability of sampling the spikes present in the original data.

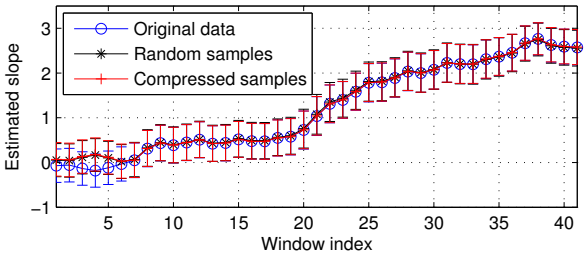
3.4.2 Detection of Trends

The second case study shows that analysis of the compressed samples can detect trends within the full-length signal that may indicate a system resource being slowly exhausted—such as memory. We use a long-running Trade6 application executing over a period of 48 hours and inject a small memory leak of about 100 KB/minute at around the 24-hour mark; referring back to Figure 3.1(b), note the increasing trend in the *CommittedAS* dataset.

Figure 3.6 summarizes the results. The quality of the slope estimate, in terms of the confidence interval, obtained using the compressed data is relatively insensitive to the sample size as it varies from 6.25% to 25%. Also, the overlay of the estimated slope values obtained using the original, compressed, and randomly sampled data, shown in Figure 3.6(b), shows only minor differences in



(a) Estimation of slope using compressed samples.



(b) Comparision of the slope estimates.

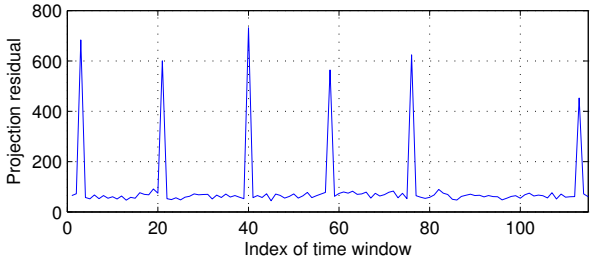
Figure 3.6: Slope estimates obtained for *CommittedAS* using the compressed data for different sample sizes; overlay of the estimated slope values obtained using the original, compressed, and randomly sampled data.

the achieved quality. The sample size for both the compressive and random sampling methods was set to 6.25%. The use of compressive sampling over random sampling is however still advantageous since it allows us to recover the full-length signal with much higher fidelity.

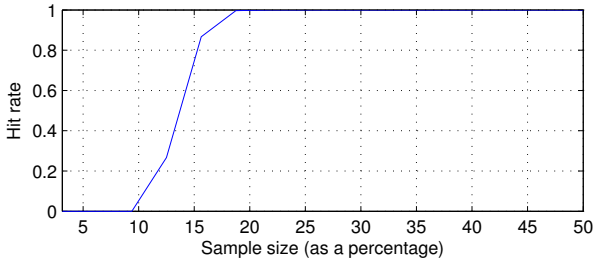
3.4.3 PCA-based Detection of Spikes and Abrupt Changes

The final case study deals with using PCA to analyze the *write_activity* dataset for spikes and abrupt changes.[1] Figure 3.7(a) shows the projection of *write_activity* on to the anomalous subspace—obtained via the previously described training process—for a sample size of 20%. Observation windows containing spikes have significantly higher projection residuals than those without spikes. Figure 3.7(b) shows the achieved hit rate as a function of sample size when the false alarm rate is fixed. With the false alarm rate fixed at 0.5%, the PCA-based method achieves a hit rate greater than 95% using a sample size of about 18%.

[1] A MATLAB implementation of our PCA-based solution for detecting spikes is available via a GitHub repository located at `https://github.com/TingshanHuang/AnomalyDetectionWithCompressedMeasurements`. The result in Figure 3.7(b) is reproducible using the provided dataset and demo script.



(a) The projection residuals.



(b) Hit rate versus sample size.

Figure 3.7: The projection residual of the *write_activity* signal is shown in (a) wherein windows containing a spike have extremely large projections on the anomalous subspace. The achieved hit rate is shown in (b) when the false alarm rate is fixed as 0.5%. The length of the measurement length is set to $N = 64$ and the sample size varies from 3% to 50%.

3.5 Related work

Compressive sampling has recently gained traction as a low-cost monitoring solution in varied fields such as wireless sensor networks [70], power grids [73], and microprocessor design [72]. Our previous research applied the concept to the performance monitoring of computing systems [81, 82, 83]. The emphasis in [81, 82] was on studying the feasibility of using fixed-rate compressive sampling for monitoring server systems. More recently. we developed an adaptive-rate model that exploits any time-varying sparsity in the signal to further reduce the number of collected samples [83]. Tuma *et al.* study the applicability of compressive sampling for fine-grained monitoring of processor performance and evaluate its performance on signals representing micro-architecture counters within a core [72]. They show that compressive sampling can recover these signals if one can identify the bases in which the signals can be sparsely represented.

Lakhina *et al.* develop a PCA-based method for real-time detection of anomalies in computer networks [49]. Here, PCA is applied to high-dimensional network-wide traffic data to extract nor-

mal traffic patterns which have a highly reduced dimension. During the detection phase, traffic that does not correspond to the normal pattern is identified as an anomaly. The patterns extracted by this approach is also called a normal subspace; so it is also referred to as the PCA-based subspace method. Recent extensions of this method for network anomaly detection include [106, 107, 108]. We have also seen PCA-based methods developed for anomaly detection in cloud computing systems [52]. Here, PCA is applied to the run-time performance data collected from each server to extract relevant features, which are then used to train decision tree classifiers for anomaly detection. The aforementioned techniques for anomaly detection have been shown to be quite effective in detecting anomalies affecting network and computing systems. However, these techniques require the full-length data stream to perform the necessary analysis.

There have been a few recent attempts focused on anomaly detection using compressed data obtained from the underlying system [105, 109]. These papers show that the performance of spectral-based methods (e.g., the PCA-based subspace method) using only knowledge of the compressed samples is similar to that of knowing the original data. Motivated by the fact that the samples should preserve more information than just the spectral properties of the original data, such as the sample mean and variance, our work extends their use to detect abrupt changes and trends in the raw data.

3.6 Summary

From a viewpoint of reducing monitoring costs, compressive sampling allows for a simple, randomized sensing strategy to acquire the signals of interest on the local server; the network bandwidth needed to transmit these samples to the monitoring station is also significantly reduced. When operators wish to analyze the original signal, there is a way to use numerical optimization to reconstruct the signal from the sample set. We have shown that the compressed samples preserve important statistical properties of the original signal such as the mean, variance, and correlation. This result can be used to reduce the processing cost associated with data analysis on the monitoring station via a two-step process: the compressed samples are first examined for characteristics of interest such as spikes, abrupt changes, and trends; if necessary, the full-length signal is then reconstructed to find out a more accurate time stamp of occurrence within a particular window.

4. Fast and Distributed Detection of Anomalies in Feature Matrices of Network Traffic

Anomalies in communication network traffic caused by malware or denial-of-service attacks manifest themselves in structural changes in the covariance matrix of traffic features. Real-time detection of anomalies in high-dimensional data demands a very efficient algorithm to identify these changes in a compact low-dimensional representation. This chapter presents an efficient algorithm for the rapid detection of structural differences between two covariance matrices, as measured by the maximum possible angle between the subspaces specified by subsets of the two sets of principal components of the matrices. We show that our algorithm achieves a significantly lower computational complexity compared to a naive approach. Finally, we apply our results to real traffic traces from Internet backbone links and show that our approach offers a substantial reduction in the computational overhead of anomaly detection. The preliminary work was published in [110].

4.1 Introduction

Security professionals who monitor communication networks for malware or denial-of-service attacks are increasingly dependent on real-time detection of anomalous behavior in the data traffic [111]. The volume of data that one has to monitor and process for effective real-time management of networks and systems, however, poses significant Big Data challenges [112]. Further, many of the anomalies are apparent only upon an examination of the correlations in the traffic data from multiple locations. State-of-the-art network management today, therefore, requires the constant monitoring of a very large number of features at hundreds of locations across a network and the ability to collect, transmit and process the data in real-time for near-instantaneous mitigation actions in case of anomalous patterns indicative of an intrusion or other threats. A key step in network management and traffic analysis, therefore, is *dimensionality reduction*, which cuts down the number of observed features or variables into a smaller set that is sufficient to capture the information essential to the goals of network management. This chapter addresses the challenge of *real-time* dimensionality reduction for anomaly detection, with a new method based on a metric called the *maximum subspace*

distance for differentiating between observed patterns in traffic streams.

Principal Component Analysis (PCA) is a widely used tool that allows us to derive a reduced set of the most significant uncorrelated features that are linear combinations of the original set of features [113]. Given N features, one selects $k \ll N$ most significant principal components out of N to define a k-dimensional subspace based on observations of normal traffic (we assume the availability of normal or some reference traffic for purposes of comparing observed traffic with it.) Then, any significant deviation in the projection of the data onto this reference subspace can be defined as an anomaly for purposes of detection [49]. There are at least two weaknesses of PCA-based subspace methods for anomaly detection: (i) the reduced number of principal components, k, is computed based on the structure of normal traffic when, actually, the structure of the changes between the observed and the normal traffic is more relevant for choosing the appropriate number of dimensions for anomaly detection, and (ii) a static determination of k is inadequate at capturing real-time changes — the right number of dimensions is often different during different periods of time depending on the structure of both the normal and observed traffic. Motivated by the need to address these weaknesses, this chapter reports results which differ from and build upon the body of work on subspace methods for anomaly detection.

Normal and anomalous traffic tend to differ in the correlations between pairs of features of the traffic being monitored. These correlations, along with the variance of each monitored feature are entires in the covariance matrix. This work is additionally motivated by two observations: (i) anomalies lead to changes in the covariance matrix of the set of traffic features being monitored, and (ii) different types of anomalies cause different types of deviations in the covariance matrix allowing a categorization of the detected anomaly and an immediate prescription of actions toward threat mitigation [77]. Besides, use of the covariance matrix requires no assumptions about the distributions of the monitored features, rendering it a general method for traffic characterization. Rapid detection of structural differences between two covariance matrices, therefore, is an important goal in modern anomaly detection and is a key component of this work. The analysis and the algorithms presented in this chapter facilitate a computationally efficient and distributed algorithm for ascertaining the appropriate number of dimensions to use in detecting changes in covariance matrices.

4.1.1 Problem Statement and Contributions

Let N denote the number of traffic features being monitored in the network. These N features could all be observations made at a single location in the network or could be from a multiplicity of locations where the traffic is being monitored. Assume that each feature sampled in the traffic corresponds to a designated time slot. We place no assumptions on the selection of traffic features.

The primary goal of this chapter is to find an appropriately reduced set of $k \ll N$ dimensions in real-time, which may be different during different time periods depending on the nature of the observed traffic at that time. As opposed to traditional PCA-based methods which determine k statically based only on the normal or reference traffic, our goal is to derive k based on the currently observed differential between the monitored traffic and the reference or normal traffic. This allows the use of only the number of dimensions necessary at any given time instead of a static determination independent of the currently observed patterns.

The primary technical contribution of this chapter is in real-time dimensionality reduction for detecting changes in patterns in order to ease the Big Data challenges of network and system management. In this chapter, we illustrate our technique of dimensionality reduction through the end goal of anomaly detection in networks — but, our contribution can be employed in all contexts where real-time dimensionality reduction can be employed to reduce the computational burden of detecting changes in data streams.

Centralized anomaly detection

In the case in which all N features are observations at a single node, the covariance matrices can be readily calculated at the node itself. Also, in the case in which the N features are observations made at multiple nodes but transmitted and collected at a central monitoring station, the covariance matrices can be calculated at this station for analysis. In either of these two cases, we can characterize network traffic as a stream of covariance matrices, one for each designated window of time, and then use the observed changes in the covariance matrices to infer changes in the system status, anomalous or otherwise. Let Σ_A and Σ_B denote the two $N \times N$ covariance matrices that need to be compared to detect changes indicative of an anomaly. These two matrices could represent real traffic

data during different time windows or one of them could be a reference matrix representing normal operation without anomalies. The problem becomes one of devising a measure of the structural difference using a parsimonious metric that allows efficient computation at reasonable accuracy while also serving as a reliable indicator of anomalies.

The contribution of this chapter begins with the use of a new metric, which we call the *maximum subspace distance*, defined as the largest possible angle, $\theta_{\max}$, between the subspace composed of the first k_A principal components of Σ_A and the subspace composed of the first k_B principal components of Σ_B, for all possible values of k_A and k_B. The maximum subspace distance is defined in more detail in Section 4.3.

Finding the subspace (i.e., all the principal components) given a covariance matrix is computationally expensive since it requires either singular value decomposition or eigenvalue decomposition. By avoiding subspace computation through use of only the most significant principal components and based on theoretical insights proved in the Section 4.4, this chapter contributes a fast approximate algorithm to estimate a reduced number of principal components that is sufficiently effective at distinguishing between the two matrices being compared. This number of principal components — which we call the *effective subspace dimension (ESD)* — is such that it describes each of the two subspaces enough to gain a close estimate of the maximum subspace distance. In Section 4.4, we present the algorithm, called GETESD, in pseudo-code along with an analysis of its computational complexity. This algorithm is an improved version of our early work in [110] with a better criterion for stopping the iterations.

Distributed anomaly detection

In the case in which the N features are sampled at multiple locations in the network, the covariance matrix of these features is not readily computable at any given location. If the feature data cannot all be transmitted to a central monitoring station due to communication costs or other reasons, we have only the raw features computed for every time slot based on which we need to estimate the maximum subspace distance. Let M denote the number of consecutive time slots for which these features have to be sampled in order to ascertain the second-order statistics of these features, i.e., the variances and correlations which populate the covariance matrix of these features.

The input to the distributed algorithm at each node, therefore, is not a stream of covariance matrices but raw feature data in the form of a stream of vectors, each of length M.

To address this case, we present a distributed algorithm, called GETESD-D, which avoids the direct computation of the covariance matrices but nevertheless returns the effective subspace dimension and an estimate of the maximum subspace distance. The participating nodes in the GETESD-D algorithm deploy the gossip-based Gaussian mixture learning mechanism to estimate principal components of the traffic features [114]. The algorithm requires only local computation and some communication between neighboring nodes. The correctness of this algorithm is proved in the Section 4.5. Section 4.5 presents this distributed algorithm along with an analysis of its computational and communication complexity. We show that the estimation result in the distributed algorithm is as good as the centralized one with the extra cost for message exchanges.

Our work in this chapter improves upon the traditional subspace methods for anomaly detection in three ways: (i) our attempt to reduce the number of dimensions is based on data from both of the two traffic streams being compared instead of depending only on some reference data for a normal traffic stream, (ii) our methods allow a dynamic real-time computation of the structural changes in network traffic features and, therefore, allow a better characterization of attack traffic, and (iii) our method makes it significantly harder for attack traffic to spoof the structure of normal traffic in order to escape detection.

The remainder of this chapter is organized as follows. Section 4.2 discusses related work in the use of covariance matrices and in subspace methods for anomaly detection. Section 4.3 describes the assumptions and definitions underlying the problem and the solution approach used in the chapter. Sections 4.4 and 4.5 describe the centralized and distributed versions of our algorithms, respectively. Section 4.6 describes simulation experiments using real traffic traces which illustrate the performance of anomaly detection using our method.

4.2 Related Work

The problem of anomaly detection often invites customized solutions particular to the type of anomaly that is of interest. For example, methods reported in [115] and [116] allow the detection of

load anomalies in voice-over-IP traffic at the IP level and at application level, respectively. Similarly, distributed denial-of-service (DDoS) attacks are the target of detection in [117], [118] and [119]. However, given that anomalies manifest themselves in multiple and sometimes uncommon or even unknown ways, a more practical solution to the problem of anomaly detection is a generic method that can detect anomalies of all kinds — common and uncommon, known and unknown.

Developing general anomaly detection tools can be challenging, largely due to the difficulty of extracting anomalous patterns from huge volumes of high-dimensional data contaminated with anomalies. Early anomaly detection methods which used artificial intelligence, machine learning, or state machine modeling are reviewed in [120]. Examples of later work on developing general anomaly detection tools include [46, 49, 77, 108, 119, 121, 122, 123]. There are two broad approaches to anomaly detection that are related to the work reported in this chapter — those using covariance matrices directly and those using subspace methods, and occasionally those which use both these approaches together.

While the covariance matrix plays a role in many anomaly detection methods, it was most directly used in [77] to detect flooding attacks based on comparisons between a covariance matrix under normal conditions (used as the reference matrix) and the observed covariance matrix. The method helps identify, characterize, and differentiate between various attacks by revealing the set of entries in the covariance matrix that change significantly under different attack scenarios. The covariance matrix is also used directly for anomaly detection in [121]. In this method, signs in the observed covariance matrix are compared with a reference covariance matrix, and the corresponding traffic samples are marked as anomalous if the number of mismatches is higher than a certain threshold. While the methods in [77] and [121] are based on detecting the changes in individual entries in the covariance matrix, they do not exploit the correlation between these changes in the entries. This chapter, on the other hand, advances the use of covariance matrices by examining the underlying correlation among the changes in the entries to offer a more refined and a more reliable approach to anomaly detection.

The subspace method, based on PCA, was first proposed for anomaly detection in [49] and later improved in [46] to explore the deviation in the network-wide traffic volume and feature distributions caused by anomalies. The scheme proposed in [49] applies PCA on training data and

separates the high-dimensional space of network traffic into two subspaces: the normal subspace and the anomalous subspace. The normal subspace is low-dimensional and captures high variance of normal traffic data, thus modeling the normal behavior of a network. The projections of measurement data onto the anomalous subspace are used to signal, identify and classify anomalies.

The limitations of the subspace method are discussed in [76]. In particular, the performance of the PCA-based method — the false positive rate in particular — is very sensitive to small changes in the number of dimensions chosen for the normal subspace. The simulation results in [124] further confirm that the effectiveness of the subspace method depends strongly on the dimension chosen for the normal subspace. In addition, excessively large anomalies can contaminate the normal subspace and weaken the performance of the detector. Later work has improved upon the training process of the subspace method [106, 107, 125, 126], but choosing the appropriate dimension for the normal subspace has remained an unmet challenge.

The method proposed in this chapter seeks to overcome some of these shortcomings of the subspace method. When comparing the covariance matrices under normal and observed conditions, our method does not rely entirely on tuning and training for the normal condition to ascertain the right number of principal components — instead, the number of dimensions the method uses is based on both the observed traffic and the normal reference traffic. In other words, instead of defining the characteristics of normal behavior only, as in [49] and [46], our method draws out the characteristics of the difference between the normal and observed behaviors to come up with two sets of principal components. Furthermore, our method makes it significantly harder for an anomaly to disguise itself as normal traffic. In the previously proposed subspace methods, an attacker with knowledge of the normal subspace used for anomaly detection can devise attack traffic that matches its principal components for that subspace. To foil the detection in our case, however, the attacker would have to additionally match the order of the principal components of the normal subspace — a significantly more difficult task.

In other related work, PCA-based methods have been decentralized for a variety of purposes including anomaly detection [79, 80, 127, 128, 129]. A distributed framework for PCA is proposed in [128] to achieve accurate detection of network anomalies through monitoring of only the local data. A distributed implementation of PCA is developed for decomposable Gaussian graphical

models in [129] to allow decentralized anomaly detection in backbone networks. Distributed gossip algorithms using only local communication for subspace estimation have been used in the context of sensor networks [130, 131, 132, 133]. Our work in this chapter extends the distributed average consensus protocol proposed in [133] to estimate the principal subspace for the context of anomaly detection in network traffic data.

4.3 The Metric

Let N denote the number of features in the dataset of interest and let Σ_A and Σ_B denote the two $N \times N$ covariance matrices to be compared. Let $\mathbf{a}_1, \ldots, \mathbf{a}_N$ and $\mathbf{b}_1, \ldots, \mathbf{b}_N$ denote the eigenvectors of Σ_A and Σ_B, respectively.

4.3.1 The subspace distance

Let $\theta_{k_A,k_B}(A, B)$ denote the angle between the subspace composed of the first k_A principal components of Σ_A, $\mathbf{a}_1, \ldots, \mathbf{a}_{k_A}$, and the subspace composed of the first k_B principal components of Σ_B, $\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}$. We refer to this angle between the subspaces as the *subspace distance*, which has a range between 0 to 90 degrees. We have:

$$\sin \theta_{k_A,k_B}(A, B) = \| T_{k_A,k_B}(A, B) \| \tag{4.1}$$

where $\| \cdot \|$ is the matrix norm and $T_{k_A,k_B}(A, B)$ is the part of $[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]$ orthogonal to $[\mathbf{a}_1, \ldots, \mathbf{a}_{k_A}]$. Therefore,

$$T_{k_A,k_B}(A, B) = (I - \sum_{i=1}^{k_A} \mathbf{a}_i \times \mathbf{a}_i')[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}] \tag{4.2}$$

$$= (\sum_{i=k_A+1}^{N} \mathbf{a}_i \times \mathbf{a}_i')[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]. \tag{4.3}$$

The sine of the subspace distance captures the norm of $[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]$ not including its projection in $[\mathbf{a}_1, \ldots, \mathbf{a}_{k_A}]$. The more distinguishable or orthogonal these two subspaces are to each other, the larger the subspace distance between them.

4.3.2 The maximum subspace distance

To compare the two matrices, we quantify the difference between Σ_A and Σ_B as the maximum angle, $\theta_{k_A,k_B}(A,B)$, where $1 \leq k_A \leq N$ and $1 \leq k_B \leq N$:

$$\theta_{\max} = \max_{1 \leq k_A, k_B \leq N} \theta_{k_A,k_B}(A,B) \tag{4.4}$$

In this chapter, we refer to $\theta_{\max}$ as the *maximum subspace distance*, which serves as our metric for anomaly detection and quantifies the difference between the two matrices.

When k_A and k_B hold values that maximize $\theta_{k_A,k_B}(A,B)$, the two sets of principal components, $[\mathbf{a}_1, \ldots, \mathbf{a}_{k_A}]$ and $[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]$, can be thought of as the distinguishing characteristics of covariance matrices Σ_A and Σ_B. Once the maximum subspace distance and the corresponding pair of dimensions are found, they can be used to characterize the two datasets with Σ_A and Σ_B as their covariance matrices, and distinguish these two datasets from each other. We show in Section 4.6 how these two sets of characteristics can be employed for anomaly detection.

4.3.3 Comparison with principal angles

Our proposed metric of subspace distance is new, and the closest related metric in the literature is the *principal angle* [134]. Let $\mathcal{A}$ be an N_A-dimensional subspace defined by the orthonormal basis $\mathbf{a}_1, \ldots, \mathbf{a}_{N_A}$, and let $\mathcal{B}$ be an N_B-dimensional subspace defined by the orthonormal basis $\mathbf{b}_1, \ldots, \mathbf{b}_{N_B}$ where $N_A \leq N_B$. The principal angles between $\mathcal{A}$ and $\mathcal{B}$ are a set of N_A angles $\{\theta_1 \leq \theta_2 \leq \cdots \leq \theta_{N_A}\}$ defined as follows. The first principal angle captures the smallest angle between any two vectors of these two subspaces:

$$\theta_1 = \min_{\mathbf{a} \in \mathcal{A}, \mathbf{b} \in \mathcal{B}} \arccos \frac{|\langle \mathbf{a}, \mathbf{b} \rangle|}{\| \mathbf{a} \| \| \mathbf{b} \|}$$

Let $\tilde{\mathbf{a}}_1$ and $\tilde{\mathbf{b}}_1$ denote the corresponding $\mathbf{a}$ and $\mathbf{b}$ for which the angle is smallest as defined above. The other principal angles, θ_k, are found similarly by using a pair of $\mathbf{a}$ and $\mathbf{b}$ that are orthogonal to

$\tilde{\mathbf{a}}_i$ and $\tilde{\mathbf{b}}_i$, $i < k$:

$$\theta_k = \min_{\substack{\mathbf{a}\in\mathcal{A},\mathbf{b}\in\mathcal{B},\\ \mathbf{a}\perp\{\tilde{\mathbf{a}}_1,\cdots,\tilde{\mathbf{a}}_{k-1}\},\\ \mathbf{b}\perp\{\tilde{\mathbf{b}}_1,\cdots,\tilde{\mathbf{b}}_{k-1}\}}} \arccos \frac{|\langle \mathbf{a}, \mathbf{b}\rangle|}{\| \mathbf{a} \| \| \mathbf{b} \|}$$

The subspace distance, as a metric, is different from principal angle in several ways. Most importantly, the subspace distance considers the importance of each principal component while the principal angle does not. For example, let $\mathcal{A}$ and $\mathcal{B}$ be identical two-dimensional subspaces, then the two principal angles between these two subspaces are both zero. As for the subspace distance, we have $\theta_{2,2} = 0$, while $\theta_{1,1} = \theta_{1,2} = \theta_{2,1}$, the value of which depends on the angle between $\mathbf{a}_1$ and $\mathbf{b}_1$, the first principal components of subspaces $\mathcal{A}$ and $\mathcal{B}$, respectively. Another example is when $\mathcal{A}$ and $\mathcal{B}$ are two two-dimensional subspaces that form an angle α. In this case, principal angle θ_1 is still 0 and $\theta_2 = \alpha$. Thus, unlike in the case of the subspace distance, the values of the principal angles are independent of the principal components $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{b}_1$ and $\mathbf{b}_2$.

The two sets of principal components used in the definition of the subspace distance capture the linear correlated pattern of the two datasets. In addition, the subspace distance depends on the order of these principal components which represents their order of importance. As a result, our metric of the maximum subspace distance serves as a dependable measure of the magnitude of changes in the pattern between any two datasets. This property makes the maximum subspace distance an appropriate metric for change detection and, therefore, for anomaly detection.

4.4 The Centralized Algorithm

Given the high computational cost of finding the maximum subspace distance, in this section, we present an algorithm which estimates the metric at sufficient accuracy for successful anomaly detection. Our solution is based on four key ideas: (i) allowing $k_A = k_B$ in our search for the maximum subspace distance, $\theta_{\max}$, (ii) reducing the problem to one of always finding only the first principal component of a matrix, (iii) using the power iteration method to approximate the first principal components, and finally, (iv) using a heuristic to approximately ascertain the value of $\theta_{\max}$.

4.4.1 The rationale behind allowing $k_A = k_B$

In the approach presented in this chapter, we limit our search for $\theta_{\max}$ to consider only the cases in which $k_A = k_B$. Our rationale is based on Theorem 1 below.

Theorem 1. *If for some k_A and k_B, $\theta_{k_A,k_B}(A,B) = \theta_{\max}$, then there exists k, $1 \leq k \leq N$, such that $\theta_{k,k}(A,B) = \theta_{\max}$.*

Proof. Without loss of generality, assume $k_A \geq k_B$. The statement of the theorem is proved if $\theta_{k_A,k_B}(A,B) \leq \theta_{k_B,k_B}(A,B)$ and $\theta_{k_A,k_B}(A,B) \leq \theta_{k_A,k_A}(A,B)$. The proofs of each of these two cases follows.

Case (i): Let $\mathbf{x}$ denote a column vector of length k_B. Using the definition of matrix norm, we have:

$$\| T_{k_A,k_B}(A,B) \|^2 = \max_{\forall \mathbf{x}, \|\mathbf{x}\|=1} \| T_{k_A,k_B}(A,B)\mathbf{x} \|^2 = \max_{\forall \mathbf{x}, \|\mathbf{x}\|=1} \sum_{i=k_A+1}^{N} \| \mathbf{a}_i'[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]\mathbf{x} \|^2$$
$$= \sum_{i=k_A+1}^{N} \| \mathbf{a}_i'[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]\mathbf{x}_{\max} \|^2$$

where $\mathbf{x}_{\max}$ is the column vector with unit norm that achieves the maximum matrix norm. Similarly, using the definition of the matrix norm again:

$$\| T_{k_B,k_B}(A,B) \|^2 = \max_{\forall \mathbf{x}, \|\mathbf{x}\|=1} \sum_{i=k_B+1}^{N} \| \mathbf{a}_i'[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]\mathbf{x} \|^2 \geq \sum_{i=k_B+1}^{N} \| \mathbf{a}_i'[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]\mathbf{x}_{\max} \|^2$$
$$\geq \sum_{i=k_A+1}^{N} \| \mathbf{a}_i'[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]\mathbf{x}_{\max} \|^2 = \| T_{k_A,k_B}(A,B) \|^2 .$$

Thus we have $\theta_{k_A,k_B}(A,B) \leq \theta_{k_B,k_B}(A,B)$.

Case (ii): Let $\mathbf{y}$ denote a column vector of length k_A. Using the definition of matrix norm:

$$\| T_{k_A,k_A}(A,B) \|^2 = \max_{\forall \mathbf{y}, \|\mathbf{y}\|=1} \sum_{i=k_A+1}^{N} \| \mathbf{a}_i'[\mathbf{b}_1, \ldots, \mathbf{b}_{k_A}]\mathbf{y} \|^2$$

Let $\mathbf{z}$ denote a column vector of length k_A with $\mathbf{x}_{\max}$ followed by $k_A - k_B$ zeroes: $\mathbf{z}' = [\mathbf{x}_{\max}\ 0 \cdots 0]'$.

This vector has unit norm and $[\mathbf{b}_1, \ldots, \mathbf{b}_{k_A}]\mathbf{z} = [\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]\mathbf{x}_{\max}$. Therefore,

$$\| T_{k_A,k_A}(A,B) \|^2 \geq \sum_{i=k_A+1}^{N} \| \mathbf{a}_i'[\mathbf{b}_1, \ldots, \mathbf{b}_{k_A}]\mathbf{z} \|^2 = \sum_{i=k_A+1}^{N} \| \mathbf{a}_i'[\mathbf{b}_1, \ldots, \mathbf{b}_{k_B}]\mathbf{x}_{\max} \|^2$$
$$= \| T_{k_A,k_B}(A,B) \|^2 .$$

Thus we have $\theta_{k_A,k_B}(A,B) \leq \theta_{k_A,k_A}(A,B)$. □

Allowing $k_A = k_B$ to find the maximum subspace distance reduces the search space from N^2 to N. We refer to the value of k for which $\theta_{k,k}(A,B)$ is the maximum subspace distance as the *optimal subspace dimension*.

4.4.2 Subspace distance and the projection matrix

We define an $N \times N$ projection matrix P from $\{\mathbf{b}_1, \ldots, \mathbf{b}_N\}$ to $\{\mathbf{a}_1, \ldots, \mathbf{a}_N\}$ as:

$$P_{i,j} = \langle \mathbf{a}_i, \mathbf{b}_j \rangle = \mathbf{a}_i'\mathbf{b}_j \tag{4.5}$$

where $\langle *, * \rangle$ represents dot product of two vectors.

According to Theorem 2 below, the smallest singular value of its submatrix $P_{1:k,1:k}$, consisting of the first k rows and k columns of P, is equal to $\cos(\theta_{k,k}(A,B))$.

Theorem 2. *If an $N \times N$ matrix P is built as in Eq. (4.5), and its submatrix $P_{1:k,1:k}$ has $\sigma_k(P_{1:k,1:k})$ as its smallest singular value, then $\sigma_k(P_{1:k,1:k}) = \cos(\theta_{k,k}(A,B))$.*

Proof. First we use $P_{i,j}$ to express $T_{k,k}(A,B)$ and $T'_{k,k}(A,B)T_{k,k}(A,B)$.

$$T_{k,k}(A,B) = (I - \sum_{i=1}^{k} \mathbf{a}_i \times \mathbf{a}_i')[\mathbf{b}_1, \ldots, \mathbf{b}_k] = [\mathbf{b}_1, \ldots, \mathbf{b}_k] - \sum_{i=1}^{k} \mathbf{a}_i \times [\mathbf{a}_i'\mathbf{b}_1, \ldots, \mathbf{a}_i'\mathbf{b}_k]$$
$$= [\mathbf{b}_1, \ldots, \mathbf{b}_k] - \sum_{i=1}^{k} \mathbf{a}_i[P_{i,1}, \ldots, P_{i,k}]$$

$T'_{k,k}(A,B)T_{k,k}(A,B)$ expressed using $P_{i,j}$ becomes

$$
\begin{aligned}
&T'_{k,k}(A,B)T_{k,k}(A,B)\\
=&[\mathbf{b}_1,\ldots,\mathbf{b}_k]'[\mathbf{b}_1,\ldots,\mathbf{b}_k] - [\mathbf{b}_1,\ldots,\mathbf{b}_k]'\sum_{i=1}^{k}\mathbf{a}_i[P_{i,1},\ldots,P_{i,k}]\\
&-\sum_{i=1}^{k}[P_{i,1},\ldots,P_{i,k}]'\mathbf{a}_i'[\mathbf{b}_1,\ldots,\mathbf{b}_k] + \sum_{i=1}^{k}[P_{i,1},\ldots,P_{i,k}]'\mathbf{a}_i'\sum_{j=1}^{k}\mathbf{a}_j[P_{j,1},\ldots,P_{j,k}]\\
=&I_{k\times k} - \sum_{i=1}^{k}[P_{i,1},\ldots,P_{i,k}]'[P_{i,1},\ldots,P_{i,k}] - \sum_{i=1}^{k}[P_{i,1},\ldots,P_{i,k}]'[P_{i,1},\ldots,P_{i,k}]\\
&+\sum_{i=1}^{k}\sum_{j=1}^{k}[P_{i,1},\ldots,P_{i,k}]'\times\mathbf{a}_i'\mathbf{a}_j\times[P_{j,1},\ldots,P_{j,k}]\\
=&I_{k\times k} - 2\sum_{i=1}^{k}[P_{i,1},\ldots,P_{i,k}]'[P_{i,1},\ldots,P_{i,k}]\\
&+\sum_{i=1}^{k}\sum_{j=1}^{k}[P_{i,1},\ldots,P_{i,k}]'\times\mathbf{a}_i'\mathbf{a}_j\times[P_{j,1},\ldots,P_{j,k}]
\end{aligned}
$$

Using the fact that $\mathbf{a}_i'\mathbf{a}_j = \delta(i-j)$, we have

$$
\begin{aligned}
&T'_{k,k}(A,B)T_{k,k}(A,B)\\
=&I_{k\times k} - 2\sum_{i=1}^{k}[P_{i,1},\ldots,P_{i,k}]'[P_{i,1},\ldots,P_{i,k}] + \sum_{i=j=1}^{k}[P_{i,1},\ldots,P_{i,k}]'\times\mathbf{a}_i'\mathbf{a}_j\times[P_{j,1},\ldots,P_{j,k}]\\
&+\sum_{j=1}^{k}\sum_{i\neq j,i=1}^{k}[P_{i,1},\ldots,P_{i,k}]'\times\mathbf{a}_i'\mathbf{a}_j\times[P_{j,1},\ldots,P_{j,k}]\\
=&I_{k\times k} - 2\sum_{i=1}^{k}[P_{i,1},\ldots,P_{i,k}]'[P_{i,1},\ldots,P_{i,k}] + \sum_{i=j=1}^{k}[P_{i,1},\ldots,P_{i,k}]'\times 1\times[P_{j,1},\ldots,P_{j,k}]\\
&+\sum_{j=1}^{k}\sum_{i\neq j,i=1}^{k}[P_{i,1},\ldots,P_{i,k}]'\times 0\times[P_{j,1},\ldots,P_{j,k}]\\
=&I_{k\times k} - \sum_{i=1}^{k}[P_{i,1},\ldots,P_{i,k}]'[P_{i,1},\ldots,P_{i,k}]\\
=&I_{k\times k} - P'_{1:k,1:k}P_{1:k,1:k}
\end{aligned}
$$

The value of $\sin(\theta_{k,k}(A,B))$ is defined earlier as $\| T_{k,k}(A,B) \|$, which is the first singular value of

$T_{k,k}(A,B)$, the square root of the largest eigenvalue of $T'_{k,k}(A,B)T_{k,k}(A,B)$. Following the previous analysis, we have

$$\begin{aligned}
&\sin^2(\theta_{k,k}(A,B)) = \| T_{k,k}(A,B) \|^2 = \sigma_1^2(T_{k,k}(A,B)) = \lambda_1(T'_{k,k}(A,B)T_{k,k}(A,B)) \\
&= \lambda_1(I_{k\times k} - P'_{1:k,1:k}P_{1:k,1:k}) = 1 - \lambda_k(P'_{1:k,1:k}P_{1:k,1:k}) \\
&= 1 - \sigma_k^2(P_{1:k,1:k})
\end{aligned}$$

where λ_i stands for the i-th eigenvalue and σ_i for the i-th singular value. As a result, the following holds true

$$\cos(\theta_{k,k}(A,B)) = \sigma_k(P_{1:k,1:k}) \tag{4.6}$$

□

To understand the result in Theorem 2, one can refer to the fact that the singular values of a projection matrix $P_{1:k,1:k}$ are the scaling factors of projecting $\mathbf{b}_1, \ldots, \mathbf{b}_k$ onto $\mathbf{a}_1, \ldots, \mathbf{a}_k$ via matrix $P_{1:k,1:k}$ along different axes. For example, the largest singular value of $P_{1:k,1:k}$ shows the largest projection from $\mathbf{b}_1, \ldots, \mathbf{b}_k$ onto $\mathbf{a}_1, \ldots, \mathbf{a}_k$, which results in the smallest angle between these two subspaces. The axis in $\mathbf{a}_1, \ldots, \mathbf{a}_k$ corresponding to the largest singular value is most parallel to $\mathbf{b}_1, \ldots, \mathbf{b}_k$. Similarly, the smallest singular value results in an axis in $\mathbf{a}_1, \ldots, \mathbf{a}_k$ that is most orthogonal to $\mathbf{b}_1, \ldots, \mathbf{b}_k$, and the resulting angle is the exact definition of the subspace angle.

It can be shown, as stated in Theorem 3 below, that by increasing the subspace dimension k, the largest projection from $\mathbf{b}_1, \ldots, \mathbf{b}_k$ onto $\mathbf{a}_1, \ldots, \mathbf{a}_k$ is non-decreasing with k and has 1 as its maximum value. In other words, the axis in $\mathbf{a}_1, \ldots, \mathbf{a}_k$ becomes more parallel to $\mathbf{b}_1, \ldots, \mathbf{b}_k$ with increasing k.

Theorem 3. *If an $N \times N$ matrix P is built as in Eq. (4.5), and its submatrix $P_{1:k,1:k}$ has $\sigma_k(P_{1:k,1:k})$ as its smallest singular value and $\sigma_1(P_{1:k,1:k})$ as its largest singular value, then:*

$$\sigma_1(P_{1:k-1,1:k-1}) \leq \sigma_1(P_{1:k,1:k}),\ k = 2, \ldots, N \tag{4.7}$$

$$\sigma_1(P_{1:k,1:k}) \leq 1,\ k = 1, \ldots, N \tag{4.8}$$

The proof of Theorem 3 relies on the following Lemma.

Lemma 1: If an $N \times N$ matrix P is built as in Eq. 4.5, and $\mathbf{u}_i$ is an $N \times 1$ vector with i-th entry nonzero,

$$\mathbf{u}_i(k) = \begin{cases} 1 & \text{if } k = i \\ 0 & \text{if } k \neq i \end{cases}$$

then we have

$$\mathbf{u}_i' P' P \mathbf{u}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

Proof.

$$\mathbf{u}_i' P' P \mathbf{u}_j = \mathbf{p}_i' \mathbf{p}_j = \sum_{k=1}^{N} P_{k,i} P_{k,j} = \sum_{k=1}^{N} \mathbf{b}_i' \mathbf{a}_k \mathbf{a}_k' \mathbf{b}_j = \mathbf{b}_i' (\sum_{k=1}^{N} \mathbf{a}_k \mathbf{a}_k') \mathbf{b}_j = \mathbf{b}_i' I_{N \times N} \mathbf{b}_j = \mathbf{b}_i' \mathbf{b}_j$$

where $\mathbf{p}_i$ is the i-th column of P, and $I_{N \times N}$ is an identity matrix of size $N \times N$. □

The proof of Theorem 3 follows.

Proof. For simplicity, we denote $P_{1:k,1:k}$ as P_k in this proof. We prove the result in Theorem 3 by first proving that $\sigma_1(P_k) > \sigma_1(P_{k-1})$ and then proving that $\sigma_1(P_k) \leq 1$.

$$P_k = \begin{bmatrix} P_{k-1} & \mathbf{u} \\ \mathbf{v}' & c \end{bmatrix} \tag{4.9}$$

where $\mathbf{u}' = [\mathbf{a}_1' \mathbf{b}_k, \ldots, \mathbf{a}_{k-1}' \mathbf{b}_k]$, $\mathbf{v}' = [\mathbf{a}_k' \mathbf{b}_1, \ldots, \mathbf{a}_k' \mathbf{b}_{k-1}]$, and $c = \mathbf{a}_k' \mathbf{b}_k$. Then $P_k' P_k$ can be constructed as follows

$$P_k' P_k = \begin{bmatrix} P_{k-1}' & \mathbf{v} \\ \mathbf{u}' & c \end{bmatrix} \begin{bmatrix} P_{k-1} & \mathbf{u} \\ \mathbf{v}' & c \end{bmatrix} = \begin{bmatrix} P_{k-1}' P_{k-1} + \mathbf{v}\mathbf{v}' & P_{k-1}' \mathbf{u} + c\mathbf{v} \\ \mathbf{u}' P_{k-1} + c\mathbf{v}' & \mathbf{u}'\mathbf{u} + c^2 \end{bmatrix}$$

$\sigma_1(P_k)$ being the first singular value of P_k implies the following: first, $\sigma_1^2(P_k)$ is an eigenvalue of $P_k' P_k$, i.e. $P_k' P_k \mathbf{z}_k = \sigma_1^2(P_k) \mathbf{z}_k$ where $\mathbf{z}_k$ is the corresponding eigenvector with unit norm; second, $\| P_k \mathbf{z} \| \leq \sigma_1(P_k)$ for any vector $\mathbf{z}$ with unit norm. Thus we have

$$\sigma_1^2(P_k) \geq \begin{bmatrix} \mathbf{z}'_{k-1} & 0 \end{bmatrix} P'_k P_k \begin{bmatrix} \mathbf{z}_{k-1} \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{z}'_{k-1} & 0 \end{bmatrix} \begin{bmatrix} P'_{k-1} P_{k-1} \mathbf{z}_{k-1} + \mathbf{v}\mathbf{v}'\mathbf{z}_{k-1} \\ \mathbf{u}' P_{k-1} \mathbf{z}_{k-1} + c\mathbf{v}'\mathbf{z}_{k-1} \end{bmatrix}$$

$$= \mathbf{z}'_{k-1} P'_{k-1} P_{k-1} \mathbf{z}_{k-1} + \mathbf{z}'_{k-1} \mathbf{v}\mathbf{v}'\mathbf{z}_{k-1} = \mathbf{z}'_{k-1} \sigma_1^2(P_{k-1}) \mathbf{z}_{k-1} + (\mathbf{v}'\mathbf{z}_{k-1})^2$$

$$\geq \sigma_1^2(P_{k-1}) \mathbf{z}'_{k-1} \mathbf{z}_{k-1} = \sigma_1^2(P_{k-1})$$

Since $\sigma_1(P_k)$ is non-negative, we have $\sigma_1(P_k) \geq \sigma_1(P_{k-1})$.

$\sigma_1(P_k) \leq 1$ can be proved as follows.

For any $\mathbf{x} \in R^{N \times 1}$ with unit norm, we can represent it as a linear combination of $\mathbf{u}_i$, $\mathbf{x} = \sum_{k=1}^{N} x_i \mathbf{u}_i$, with $\sum_{k=1}^{N} x_i^2 = 1$. Using the result from Lemma 1, we have

$$\| P\mathbf{x} \|^2 = \mathbf{x}' P' P \mathbf{x} = \left(\sum_{i=1}^{N} x_i \mathbf{u}'_i \right) P' P \sum_{j=1}^{N} x_j \mathbf{u}_j = \sum_{i=1}^{N} \left(\sum_{j=1}^{N} x_i x_j \mathbf{u}'_i P' P \mathbf{u}_j \right) = \sum_{i=1}^{N} \left(x_i^2 \right) = 1$$

Therefore, $\sigma_1(P) = \cdots = \sigma_N(P)$ and $\sigma_1(P) = \max \| P\mathbf{x} \| = 1$. As a result,

$$\sigma_1(P_k) \leq \sigma_1(P_N) = \sigma_1(P) = 1$$

□

The results in Eqs. (4.7) and (4.8) indicate that when $\sigma_1(P_{1:k,1:k})$ is already close enough to 1, further increasing the number of principal components can only slightly increase the ability to differentiate between the two subspaces using the maximum subspace distance as the metric.

4.4.3 Estimating the optimal subspace dimension

Based on our results as stated in Theorems 1–3, we develop an estimation algorithm for the optimal subspace dimension. It is straightforward to find the optimal subspace dimension by obtaining $\mathbf{a}_1, \ldots, \mathbf{a}_N$ and $\mathbf{b}_1, \ldots, \mathbf{b}_N$ first, computing $\theta_{k,k}(A, B)$ for every k from 1 to N and determining the k for which $\theta_{k,k}(A, B)$ is the maximum. However, we are interested in a less computationally expensive method that does not require full knowledge of $\mathbf{a}_1, \ldots, \mathbf{a}_N$ and $\mathbf{b}_1, \ldots, \mathbf{b}_N$. Furthermore, our

interest is in searching for a smaller k, which we refer to as the *effective subspace dimension (ESD)*, which corresponds to a subspace distance close enough to the maximum such that this value of k is sufficiently effective at distinguishing between the two subspaces.

Our algorithm, which we call the GETESD algorithm, relies on the results stated in Theorems 1-3. Firstly, because of Theorem 1, we are able to limit our search of the optimal subspace dimension to cases in which $k_A = k_B = k$. Secondly, since the values of subspace distance, $\theta_{k,k}(A, B)$, depend on singular values of submatrix $P_{1:k,1:k}$, as stated in Theorem 2, we can compute the subspace distance $\theta_{k,k}(A, B)$ with only the knowledge of the first k eigenvectors of Σ_A and Σ_B. In other words, if we have already tried values of k from 1 to K, the pair of $(K + 1)$-th eigenvectors is all the additional information we need to get $\theta_{K+1,K+1}(A, B)$.

Finally, because of the property of singular values of submatrix $P_{1:k,1:k}$, as stated in Theorem 3, $\sigma_1(P_{1:k,1:k})$ is non-decreasing to 1. The closer $\sigma_1(P_{1:k,1:k})$ is to 1, the less distinguishable these two subspaces become. As a result, we set a threshold on $\sigma_1(P_{1:k,1:k})$, and stop the algorithm when $\sigma_1(P_{1:k,1:k})$ is above the threshold $1 - \epsilon$. Of course, a higher threshold leads to a wider search range and a closer approximation of the optimal subspace dimension.

Overall, our algorithm computes an approximate value of $\theta_{k,k}(A, B)$ beginning with $k = 1$ and increases k at each step; the algorithm stops when the subspace distance, $\theta_{k,k}(A, B)$, has dropped and the largest singular value of $P_{1:k,1:k}$ is in a pre-defined neighborhood of 1. This observed $\theta_{k,k}(A, B)$ is used as the *estimated* maximum subspace distance and the corresponding k becomes the effective subspace dimension.

Figure 4.1 presents the pseudo-code of our algorithm, GETESD, which returns ESD, the effective subspace dimension, and $\theta_{\max}$, the estimated maximum subspace distance, given two covariance matrices, Σ_A and Σ_B, and ϵ, which defines the threshold at which the algorithm stops. Note that the k-th eigenvectors of Σ_A and Σ_B are not computed until we are in the k-th iteration of the `while` loop. Our implementation of this algorithm uses the power iteration method [78]. For each iteration in our algorithm, we add one row and one column to the projection matrix along with the updated pair of eigenvectors. During the k-th iteration of the `while` loop, we compute $\mathbf{a}_k$ and $\mathbf{b}_k$, i.e., the k-th eigenvectors of Σ_A and Σ_B, using the power iteration method. We then construct $P_{1:k,1:k}$ by adding row vector $[\mathbf{a}_k'\mathbf{b}_1, \ldots, \mathbf{a}_k'\mathbf{b}_{k-1}]$ to the bottom of $P_{1:k-1,1:k-1}$, and column vector $[\mathbf{a}_1'\mathbf{b}_k', \ldots, \mathbf{a}_k'\mathbf{b}_k']$ to

the right. We then use the power iteration method again to calculate the singular values $\sigma_1(P_{1:k,1:k})$ and $\sigma_k(P_{1:k,1:k})$. The algorithm is run until the aforementioned stopping criteria is met.

4.4.4 Complexity analysis

By the naive approach, we would first calculate the two sets of eigenvectors and then the subspace distance for every possible pair of (k_A, k_B) to ascertain the maximum subspace distance. The complexity of calculating the two sets of eigenvectors is $O(N^3)$. The complexity of calculating the subspace distance for every possible pair of (k_A, k_B) is $N^2 * O(N^3) = O(N^5)$. The computational complexity of the naive method which computes all eigenvectors to find the optimal subspace dimension, therefore, is $O(N^5)$.

Theorem 4 states the computational complexity of the GETESD algorithm.

Theorem 4. *The* GETESD *algorithm in Figure 4.1 for N nodes has a computational complexity of* $O(Pk^3 + k^2N)$*, where k is the resulting effective subspace dimension and P is the upper bound on the number of iterations in the power method.*

Proof. The analysis from previous subsection shows we can construct projection matrix in our search of optimal subspace dimension. First let us look at the complexity of Equations (4.9) and (4.10) for constructing $P_{1:k,1:k}$ and $P'_{1:k,1:k}P_{1:k,1:k}$ with the knowledge of $P'_{1:k-1,1:k-1}P_{1:k-1,1:k-1}$ and $\mathbf{a}_1, \ldots, \mathbf{a}_k, \mathbf{b}_1, \ldots, \mathbf{b}_k$.

The complexity for calculating $\mathbf{u}$, $\mathbf{v}$ and c is $O(kN)$. The complexity for constructing $P'_{1:k,1:k}P_{1:k,1:k}$ is $O(k^2)$. The complexity for calculating eigenvalue of $P'_{1:k,1:k}P_{1:k,1:k}$ using power iteration method is $O(Pk^2)$ with P for the number of iterations necessary for convergence. Since $k < N$, the overall complexity of algorithm GETESD in k loops is $O(Pk^3 + k^2N)$.

□

This shows that the GETESD algorithm is significantly more computationally efficient compared with the naive method. Of course, the naive method computes the exact optimal subspace dimension while the GETESD algorithm computes an approximation of it. We will demonstrate in Section 4.6 that this approximate result suffices to detect anomalies.

procedure GETESD(Σ_A, Σ_B, ϵ)
  $k \leftarrow 1$ ▷ number of Principal Components (PCs)
  $\theta \leftarrow 0$ ▷ angle between two subspaces
  $\hat{\Sigma}_A \leftarrow \Sigma_A$ ▷ projection of Σ_A on its last $N - k$ PCs
  $\hat{\Sigma}_B \leftarrow \Sigma_B$ ▷ projection of Σ_B on its last $N - k$ PCs
  $\theta_{\max} \leftarrow 0$ ▷ maximum angle observed
  $ESD \leftarrow 0$ ▷ value of k corresponding to $\theta_{\max}$
  while $(k \leq N)$ **do**
    $\mathbf{a}_k \leftarrow$ estimated first PC of $\hat{\Sigma}_A$ ▷ k-th PC of Σ_A
    $\mathbf{b}_k \leftarrow$ estimated first PC of $\hat{\Sigma}_B$ ▷ k-th PC of Σ_B
    construct $P'_{1:k,1:k}P_{1:k,1:k}$
    $\sigma_1(P_{1:k,1:k}) \leftarrow \sqrt{\lambda_1(P'_{1:k,1:k}P_{1:k,1:k})}$
    $\sigma_k(P_{1:k,1:k}) \leftarrow \sqrt{\lambda_k(P'_{1:k,1:k}P_{1:k,1:k})}$
    $\theta' \leftarrow \arccos(\sigma_k(P_{1:k,1:k}))$
    if $(\theta' < \theta \;\&\; \sigma_1(P_{1:k,1:k}) > 1 - \epsilon)$ **then**
      return $(ESD, \theta_{\max})$
    end if
    if $(\theta > \theta_{\max})$ **then**
      $\theta_{\max} \leftarrow \theta$
      $ESD \leftarrow k$
    end if
    $\hat{\Sigma}_A \leftarrow \hat{\Sigma}_A - \mathbf{a}_k \times (\mathbf{a}'_k \times \hat{\Sigma}_A)$
    $\hat{\Sigma}_B \leftarrow \Sigma_B - \mathbf{b}_k \times (\mathbf{b}'_k \times \hat{\Sigma}_B)$
    $k \leftarrow k + 1$
    $\theta \leftarrow \theta'$
  end while
  return $(ESD, \theta_{\max})$
end procedure

Figure 4.1: The GETESD algorithm: An efficient algorithm for finding an effective subspace dimension (ESD) and the corresponding estimate of the maximum subspace distance between the two given matrices, Σ_A and Σ_B.

4.5 Distributed Algorithm

When a multiplicity of points in a network are being monitored for anomalies, the data required for the computation of covariance matrices has to be sampled at multiple nodes, some of them geographically distant from each other. The centralized algorithm in Section 4.4 would require a monitoring station to collect all the data from the collection nodes, compute the covariance matrices, and then process the stream of matrices for anomalies. The prohibitive logging and communication costs in this case form the motivation behind our distributed algorithm to allow an estimation of the maximum subspace distance without a central station.

In the distributed case, no node can possess the knowledge of the entire covariance matrix (which requires all of the data from all of the nodes) but each node can update and maintain its corresponding entry in the eigenvector. As a result, each node will need to collaborate with its neighbors to perform eigenvector/eigenvalue estimation in a distributed fashion.

4.5.1 Assumptions and system model

Let N be the total number of features being monitored. In practice, different nodes will have different numbers of features to monitor. However, for purposes of simplicity and clarity, in this section we will illustrate our distributed algorithm assuming there is one feature monitored at each node — so, in this section, we will also assume that there are N nodes. As described in Section 4.1.1, let M denote the number of consecutive time slots for which each of these N features would have to be sampled to compute a covariance matrix (any two features have to be observed for a certain length of time to ascertain the correlation between them). The input to the distributed algorithm, therefore, is a stream of vectors of length M each, with each entry being a feature sampled at a certain time slot.

Collectively, these vectors form an $N \times M$ matrix, X, with data collection of the n-th feature $X(n,:)$ available only at node n. Besides, for an eigenvector $\mathbf{v}$ of X, $\mathbf{v}_n$ is estimated and stored at the n-th node.

We also assume a weighted connection matrix W associated with an undirected graph composed of these N nodes. Besides a self-loop for each node, an edge exists between any two nodes as long

as they can communicate with each other directly (we make no assumptions on the protocol layer for this communication, although the detection algorithm will achieve better performance at the IP layer than at the application layer). Each edge is associated with a weight, and the total weight of all edges connected to the same node is one. This makes the column sums and row sums of W equal to one.

4.5.2 Average consensus

Our distributed algorithm, GETESD-D, requires a distributed routine to compute the average of certain values at all nodes but one which runs only locally at each node. If each of the N nodes holds the value x_n, $n = 1, \ldots, N$, Figure 4.2 presents the pseudo-code to compute the average $m = \frac{1}{N}\sum_{n=1}^{N} x_n$ at each node locally.

The average consensus algorithm runs as follows. Let $m_n^{(k)}$ be the estimate of the average by node n in the k-th step. At the beginning, each node initializes its estimate as its current reading, $m_n^{(0)} = x_n$. At step k, each node sends its estimation to and receives estimates from all of its neighbors. It then calculates the weighted sum of its own estimate and the estimates from its neighbors as $m_n^{(k+1)} = \sum_{j=k \text{ or } j\in n\text{'s neighbor}} W(n, j)m_j^{(k)}$. This is equivalent to calculating $m_n^{(k+1)}$ as $m_n^{(k+1)} = W(n, :)[m_1^{(k)} \ldots m_N^{(k)}]'$. After several steps, the actual average can be obtained by reading $m_n^{(k)}$.

Theorem 5. *If the reading at node n is $x_n, n = 1, \ldots, N$, the distributed average consensus algorithm in Figure 4.2 returns a value which converges to $m = \frac{1}{N}\sum_{n=1}^{N} x_n$.*

Proof. The action of updating is equivalent to $[m_1^{(k+1)}, \ldots, m_N^{(k+1)}]^T = W[m_1^{(k)}, \ldots, m_N^{(k)}]^T$, and thus equivalent to $W^k[m_1^{(0)}, \ldots, m_N^{(0)}]^T = W^k[x_1, \ldots, x_N]^T$.

Suppose the singular value decomposition of W is $U\Lambda U'$, where columns of U are its eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$, and Λ is a diagonal matrix whose diagonal entries are the corresponding eigenvalues $\lambda_1, \ldots, \lambda_N$. It follows that $W^2 = U\Lambda U'U\Lambda U' = U\Lambda^2 U'$, and $W^{k+1} = U\Lambda U'W^k = U\Lambda^{k+1}U'$.

Since W is assumed to have unit row sum and column sum, we have $W\mathbf{1} = \mathbf{1}$ and $\mathbf{1}'W = \mathbf{1}'$, where $\mathbf{1}$ is an $N \times 1$ vector with 1s as its entries. As a result, $\mathbf{1}'W\mathbf{1} = N$ and the largest eigenvalue is $\lambda_1 = 1$ and the corresponding eigenvector is $\mathbf{u}_1 = \mathbf{1}/\sqrt{N}$. When k is large, diagonal entries of Λ^k, i.e., $[\lambda_1^k, \ldots, \lambda_N^k]$, converges to $[1, 0, \ldots, 0]$, and W^k converges to $U[1, 0, \ldots, 0]U' = \mathbf{u}_1\mathbf{u}_1' = \frac{1}{N}\mathbf{1}\mathbf{1}'$.

procedure AVERAGECONSENSUS(x_n,W)
  $m_n^{(0)} \leftarrow x_n$ ▷ initialize estimate
  $k \leftarrow 0$ ▷ step
  for $k = 1 : S$ **do**
    sends $m_n^{(k)}$ to its neighbors
    receives $m_i^{(k)}$ from its neighbors, $i \in$ Neighbor(n)
    $m_n^{(k+1)} \leftarrow \sum_{i \in \text{Neighbor}(n)} W_{n,i} m_i^{(k)}$
  end for
  return $m_n^{(k)}$
end procedure

Figure 4.2: The average consensus procedure.

procedure POWERITERATION(C, ϵ)
  $\mathbf{v}^{(0)} \leftarrow$ randomly chosen $N \times 1$ vector ▷ initialize
  $k \leftarrow 0$ ▷ step
  while True **do**
    $\mathbf{v}^{(k+1)} \leftarrow C\mathbf{v}^{(k)}$
    if $\| \mathbf{v}^{(k+1)} - \mathbf{v}^{(k)} \| < \epsilon$ **then**
      return $\mathbf{v}^{(k+1)} / \| \mathbf{v}^{(k+1)} \|$
    end if
    $k \leftarrow k + 1$
  end while
end procedure

Figure 4.3: The centralized power iteration procedure.

As a result, $[m_1^{(k)} \dots m_N^{(k)}]^T$ converges to $\frac{1}{N}\mathbf{1}\mathbf{1}'[m_1^{(k)} \dots m_N^{(k)}]^T = (\frac{1}{N}\sum_{n=1}^{N} x_n)\mathbf{1}$. □

4.5.3 Distributed power iteration method

The goal of a distributed power iteration method is to allow nodes that are remotely located to cooperate in their estimation of the most significant principal component of a dataset. Figure 4.3 presents the pseudo-code for a centralized power iteration method which estimates the first principal component $\mathbf{v}$ given a covariance matrix C. The method rests on the result of Theorem 6 below.

Theorem 6. *Given a matrix C, its first eigenvector can be estimated by* $\lim_{k\to\infty} \frac{C^k \boldsymbol{v}}{\|C^k \boldsymbol{v}\|}$, *where* $\boldsymbol{v}$ *is a random vector.*

Proof. Suppose principal components of Σ are $\mathbf{v}_1, \ldots, \mathbf{v}_N$, with corresponding eigenvalue $\lambda_1, \ldots, \lambda_N$. Express initialization $\mathbf{v}^{(0)}$ in terms of $\mathbf{v}_n$ as

$$\mathbf{v}^{(0)} = \sum_{n=1}^{N} c_n \mathbf{v}_n, \tag{4.10}$$

then we have the following

$$\begin{aligned}\mathbf{w}^{(k)} = \Sigma^k \mathbf{v}^{(0)} = \Sigma^k \sum_{n=1}^{N} c_n \mathbf{v}_n = \sum_{n=1}^{N} c_n \Sigma^k \mathbf{v}_n = \sum_{n=1}^{N} c_n \lambda_n^k \mathbf{v}_n \\ = \lambda_1^k (c_1 \mathbf{v}_1 + \sum_{n=2}^{N} c_n (\frac{\lambda_n}{\lambda_1})^k \mathbf{v}_n).\end{aligned}$$

When k is large enough, the second term goes to zero (for $\frac{\lambda_n}{\lambda_1} \ll 1$) and $\mathbf{w}^{(k)}$ would be in the direction of $\mathbf{v}_1$. □

In the following, we develop and describe a distributed version of the power iteration method. First, we describe how power iteration can be performed without involving the covariance matrix C. In the k-th step of the centralized power iteration method shown in Figure 4.3, $\mathbf{v}^{(k+1)}$ needs to be updated by multiplying with C. In the absence of C, $\mathbf{v}^{(k+1)}$ can be computed as

$$\begin{aligned}\mathbf{v}^{(k+1)} &= C\mathbf{v}^{(k)} \\ &= \frac{1}{M-1} XX'\mathbf{v}^{(k)} \\ &= \frac{1}{M-1} X[\mathbf{x}_1' \ldots \mathbf{x}_N']\mathbf{v}^{(k)} \\ &= \frac{1}{M-1} X \sum_{i=1}^{N} \mathbf{x}_i' v_i^{(k)}\end{aligned}$$

Here $\mathbf{x}_n$ is the n-th row of raw data X (the raw data that is collected by node n locally). The n-th entry of the estimated principal component at $(k+1)$-th step is:

$$v_n^{(k+1)} = \frac{1}{M-1} \mathbf{x}_n \mathbf{z}^{(k)} \tag{4.11}$$

where $\mathbf{z}^{(k)} = \sum_{n=1}^{N} \mathbf{x}_n' v_n^{(k)}$. Since the n-th node has access to $\mathbf{x}_n$ and $\mathbf{x}_n' v_n^{(k)}$, it is able to compute

procedure PowerIteration-D($\mathbf{x}_n$, $\hat{\mathbf{x}}_n$, ϵ)
  $v_n^{(0)} \leftarrow$ randomly chosen value ▷ initialize
  $k \leftarrow 0$ ▷ step
  while True **do**
    $\mathbf{z}^{(k)} \leftarrow N \times \text{AverageConsensus}(\mathbf{x}'_n v_n^{(k)}, W)$
    $v_n^{(k+1)} \leftarrow \frac{1}{M-1} \hat{\mathbf{x}}_n \mathbf{z}^{(k)}$
    $e_n^{(k)} \leftarrow \text{AverageConsensus}((v_n^{(k+1)} - v_n^{(k)})^2, W)$
    if $e_n^{(k)} < \epsilon$ **then**
      $l_n \leftarrow \text{AverageConsensus}((v_n^{(k+1)})^2, W)$
      return $\frac{1}{\sqrt{l_n}} v_n^{(k+1)}$
    end if
    $k \leftarrow k + 1$
  end while
end procedure

Figure 4.4: The distributed power iteration method.

$v_n^{(k+1)}$ after it uses the average consensus process to get $\mathbf{z}^{(k)}$. The norm of $v^{(k)}$ and $v^{(k+1)} - v^{(k)}$ can be achieved by applying average consensus on $(v_n^{(k)})^2$ and $(v_n^{(k+1)} - v_n^{(k)})^2$.

Our distributed version of the power iteration method is shown in Figure 4.4. Note that the input to this algorithm is $\mathbf{x}_n$, raw data visible to the node running the algorithm, with the method requiring no knowledge of covariance matrix C. Besides $\mathbf{x}_n$, the algorithm takes an additional input, $\hat{\mathbf{x}}_n$, which we explain in Section 4.5.4. For now, it suffices to know that $\hat{\mathbf{x}}_n = \mathbf{x}_n$ for the estimation of v_n. The estimated principal component is normalized before it is returned, and each node only needs to estimate one entry of $\mathbf{v}^{(k)}$ in each step.

4.5.4 The getESD-D algorithm

In this section, we present the getESD-D algorithm, the decentralized implementation of our algorithm in Figure 4.1. In each iteration of the `while` loop in this algorithm, the estimation of the first principal component and the corresponding eigenvalue is achieved by calling the distributed power iteration method.

After obtaining $\mathbf{a}_k$ and $\mathbf{b}_k$, $P_{1:k,1:k}$ can be constructed as follows. Recall that $P_{i,j} = \sum_{n=1}^{N} \mathbf{a}_i(n)\mathbf{b}_j(n)$, entries of row vector $[\mathbf{a}'_k\mathbf{b}_1, \ldots, \mathbf{a}'_k\mathbf{b}_{k-1}]$ and column vector $[\mathbf{a}'_1\mathbf{b}'_k, \ldots, \mathbf{a}'_k\mathbf{b}'_k]$ can be calculated by calling the distributed algorithm for average consensus on $\mathbf{a}_i(n)\mathbf{b}_j(n)$. Then we can add the row

vector $[\mathbf{a}_k'\mathbf{b}_1, \ldots, \mathbf{a}_k'\mathbf{b}_{k-1}]$ to the bottom of $P_{1:k-1,1:k-1}$, and column vector to its right to construct $P_{1:k,1:k}$.

Note that at the end of each `while` loop of Figure 4.1, the part corresponding to $\mathbf{a}_k$ and $\mathbf{b}_k$ are subtracted from covariance matrices $\hat{\Sigma}_A$ and $\hat{\Sigma}_B$. Without the subtraction, the eigenvector $\mathbf{v}^{(k+1)}$ can be equivalently estimated using distributed power iteration as follows. Let $\mathbf{a}_k$ be the estimate for the k-th eigenvector of dataset X, then

$$\begin{aligned}
\mathbf{v}^{(k+1)} &= (C - \mathbf{a}_k\mathbf{a}_k'C)\mathbf{v}^{(k)} \\
&= C\mathbf{v}^{(k)} - \frac{1}{M-1}\mathbf{a}_k\mathbf{a}_k'XX'\mathbf{v}^{(k)} \\
&= C\mathbf{v}^{(k)} - \frac{1}{M-1}\mathbf{a}_k\left(\sum_{i=1}^{N}\mathbf{a}_k(i)\mathbf{x}_i\right)\left(\sum_{j=1}^{N}\mathbf{x}_n'v_n^{(k)}\right) \\
&= C\mathbf{v}^{(k)} - \frac{1}{M-1}\mathbf{a}_k\left(\sum_{i=1}^{N}\mathbf{a}_k(i)\mathbf{x}_i\right)\mathbf{z}^{(k)},
\end{aligned}$$

where $\mathbf{z}^{(k)}$ is defined as in Eq. (4.11). In particular, the n-th entry of the eigenvector $\mathbf{v}^{(k+1)}$ can be estimated by:

$$\begin{aligned}
v_n^{(k+1)} &= \left(C\mathbf{v}^{(k)}\right)_n - \frac{1}{M-1}\mathbf{a}_k(n)\left(\sum_{i=1}^{N}\mathbf{a}_k(i)\mathbf{x}_i\right)\mathbf{z}^{(k)} \\
&= \frac{1}{M-1}\mathbf{x}_n\mathbf{z}^{(k)} - \frac{1}{M-1}\mathbf{a}_k(n)\left(\sum_{i=1}^{N}\mathbf{a}_k(i)\mathbf{x}_i\right)\mathbf{z}^{(k)} \\
&= \frac{1}{M-1}(\mathbf{x}_n - \mathbf{a}_k(n)f(\mathbf{a}_k, X))\mathbf{z}^{(k)}
\end{aligned}$$

where $f(\mathbf{a}_k, X) = \sum_{i=1}^{N}\mathbf{a}_k(i)\mathbf{x}_i$. As a result, a copy of the readings at node n, denoted by $\hat{\mathbf{x}}_n$, can be updated using $\mathbf{x}_n - \mathbf{a}_k(n)f(\mathbf{a}_k, X)$ to allow equivalent estimation as in Figure 4.1.

The distributed algorithm that combines the aforementioned computations is shown in Figure 4.5.

4.5.5 Complexity Analysis

In this subsection, we first analyze the computational complexity and the communication complexity of the distributed version of the power iteration method. Let k be the number of principal

procedure GETESD-DISTRIBUTED($\mathbf{x}_n$, $\mathbf{y}_n$, ϵ)
  $k \leftarrow 1$ ▷ number of Principal Components (PCs)
  $\theta \leftarrow 0$ ▷ angle between two subspaces
  $\theta_{\max} \leftarrow 0$ ▷ maximum angle observed
  $\hat{\mathbf{x}}_n \leftarrow \mathbf{x}_n$ ▷ copy of $\mathbf{x}_n$
  $\hat{\mathbf{y}}_n \leftarrow \mathbf{y}_n$ ▷ copy of $\mathbf{y}_n$
  $ESD \leftarrow 0$ ▷ value of k corresponding to $\theta_{\max}$
  while ($k \leq N$) **do**
    $a_k(n) \leftarrow$ POWERITERATION-D($\mathbf{x}_n$, $\hat{\mathbf{x}}_n$, ϵ)
    $b_k(n) \leftarrow$ POWERITERATION-D($\mathbf{y}_n$, $\hat{\mathbf{y}}_n$, ϵ)
    construct $P'_{1:k,1:k}P_{1:k,1:k}$
    $\sigma_1(P_{1:k,1:k}) \leftarrow \sqrt{\lambda_1(P'_{1:k,1:k}P_{1:k,1:k})}$
    $\sigma_k(P_{1:k,1:k}) \leftarrow \sqrt{\lambda_k(P'_{1:k,1:k}P_{1:k,1:k})}$
    $\theta' \leftarrow \arccos(\sigma_k(P_{1:k,1:k}))$
    if ($\theta' < \theta$ & $\sigma_1(P_{1:k,1:k}) > 1 - \epsilon$) **then**
      return (ESD, $\theta_{\max}$)
    end if
    if ($\theta > \theta_{\max}$) **then**
      $\theta_{\max} \leftarrow \theta$
      $ESD \leftarrow k$
    end if
    $f_X \leftarrow$ AVERAGECONSENSUS($\mathbf{a}_k(n)\mathbf{x}_n$, W)
    $\hat{\mathbf{x}}_n \leftarrow \hat{\mathbf{x}}_n - \mathbf{a}_k(n)f_X$
    $f_Y \leftarrow$ AVERAGECONSENSUS($\mathbf{b}_k(n)\mathbf{y}_n$, W)
    $\hat{\mathbf{y}}_n \leftarrow \hat{\mathbf{y}}_n - \mathbf{b}_k(n)f_Y$
    $k \leftarrow k + 1$
    $\theta \leftarrow \theta'$
  end while
  return (ESD, $\theta_{\max}$)
end procedure

Figure 4.5: The distributed algorithm, GETESD-D running at node n to find an effective subspace dimension (ESD) and the corresponding estimate of the maximum subspace distance between the two given datasets, X and Y.

components needs to be estimated, then the overall computational and communication complexity of GETESD-D is shown in the following two theorems.

Theorem 7. *The computational complexity of* GETESD-D *is* $O(kPM\Delta S)$ *for* M *measurements at each node, where* S *is an upper bound for number of steps for converging in the average consensus, and* P *is the upper bound for number of steps for converging of the method itself.*

Proof. The most computationally expensive part is in calculating $\mathbf{z}_n^{(k)}$. When the size of measurements is M, the length of $\mathbf{x}_n'\mathbf{v}_n^{(k)}$ is also M. Assuming the maximum degree of each node is Δ, then the process of average consensus in S steps has complexity of $O(\Delta S)$ for each entry of $\mathbf{x}_n'\mathbf{v}_n^{(k)}$. Overall, the computational complexity of the distributed power iteration method running for P steps is $O(PM\Delta S)$. As a result, the overall computational complexity of GETESD-D using $O(k)$ power iterations is $O(kPM\Delta S)$. □

Theorem 8. *The communication complexity of* GETESD-D *is* $O(kM\Delta + kP\Delta S)$ *for* M *measurements at* N *nodes in a network with a maximum degree of* Δ*, where* S *is an upper bound for number of steps for converging in the average consensus, and* P *is the upper bound for number of steps for converging of the method itself.*

Proof. We consider the number of messages that one node needs to send when the algorithm is running. Again, let the maximum degree of each node be Δ. At the very step, the node needs to send its observation $\mathbf{x}_n$ to its neighbors, leading to communication cost of $O(M\Delta)$. After that, this node only needs to update its neighbors about its estimate $\mathbf{v}_n^{(k)}$. The number of messages sent for each power iteration is $O(\Delta S)$. Overall, the communication complexity of the distributed power iteration method running for P steps is $O(M\Delta + P\Delta S)$. The overall communication complexity of GETESD-D using $O(k)$ power iterations is $O(kM\Delta + kP\Delta S)$. □

Notice that in a central setting, the computational cost for calculating covariance matrix is $O(N^2M)$, and is $O(PN^2)$ for P steps of the power method. The messages containing information about principal components sent from the center to all nodes would be $O(N^3)$ in the best case and $O(N^4)$ in the worst case.

It is also worth noticing that the number of steps for the average consensus process S depends on the ratio of $\lambda_2(W)/\lambda_1(W)$, the second and first eigenvalue of the weight matrix W: the smaller ratio results in a smaller S. For a sparse graph, such condition can be easily met. Similarly, the upper bounds on the number of iterations in the distributed power method P relies on the ratio of $\lambda_2(C)/\lambda_1(C)$, the second and first eigenvalues of C. In our CAIDA traffic data, the ratio could be as small as 3E–4, which means only a couple of iteration for 7000-dimensional dataset would be enough to reach a convergence.

4.6 Simulation results

4.6.1 Verification of theorems

In this subsection, we show that our theoretic results in Theorems 1–3 hold true with experiment result on randomly generated covariance matrices. Each of the two $N \times N$ random covariance matrices is built as follows: first, build a $N \times N$ random matrix R, each entry of which is an random number independently selected from normal distribution; the covariance matrix is a result of $R\prime \times R$ to ensure the symmetric property of covariance matrices.

Figure 4.6 shows the pre-defined subspace distance between $[\mathbf{a}_1, \ldots, \mathbf{a}_k]$ and $[\mathbf{b}_1, \ldots, \mathbf{b}_k]$, the smallest and the largest singular value of submatrix $P_{1:k,1:k}$ when k varies from 1 to the dimension of the matrix $N = 20$.

This simulation result supports our theoretical result. First, it is evident that the cosine of subspace distance is indeed equal to the smallest singular value of the submatrix of the projection matrix $P_{1:k,1:k}$, as we have stated in Theorem 2. Second, the largest singular value of $P_{1:k,1:k}$ is non-decreasing, with maximum equal to 1, as we have stated in Theorem 3.

4.6.2 Estimation of subspace distance

We use anonymized passive traffic traces from four monitors of CAIDA (Cooperative Association for Internet Data Analysis) connected to high-speed Internet backbone links of Tier1 ISPs: equinix-chicago-dirA, equinix-chicago-dirB, equinix-sanjose-dirA, and equinix-sanjose-dirB [135]. The following six features of each packet are selected for data analysis: source IP address, destina-

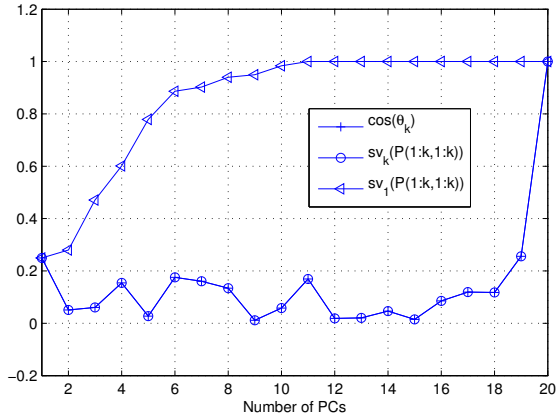


Figure 4.6: The subspace angle and singular values of $P_{1:k,1:k}$ with varying k

tion IP address, source port number, destination port number, protocol and packet size. This allows a total of $4 \times 6 = 24$ features. We compute the 24×24 covariance matrix of these features for every time window of 50 milliseconds and compare it to the covariance matrix corresponding to the previous time window.

Our results with Internet backbone traffic data show that the maximum subspace distance between two covariance matrices corresponding to consecutive time windows has a mean of 89.82 degrees and is close to 90 degrees with high probability. This shows that we can always choose two sets of principal components—one set for each covariance matrix with size equal to the optimal subspace dimension—that are almost orthogonal to each other in order to capture and successfully detect anomalous changes.

Fig. 4.7 shows that the effective subspace dimension rarely exceeds 8 with an average of 4.33, a significant reduction compared to the 24 principal components that would have to be examined to find the actual maximum subspace distance.

As also shown in Fig. 4.7, the effective subspace dimension is almost always smaller than the optimal subspace dimension, which has an average of about 16.5 and can be as high as 23. Even though it is true in the case of most real data that the first few principal components are the most significant in capturing the internal structure of the data while the last few are comparatively trivial, Fig. 4.7 shows that the optimal subspace dimension cannot often be computed by considering only a few fixed number of significant principal components. The *getESD* algorithm, therefore, offers a

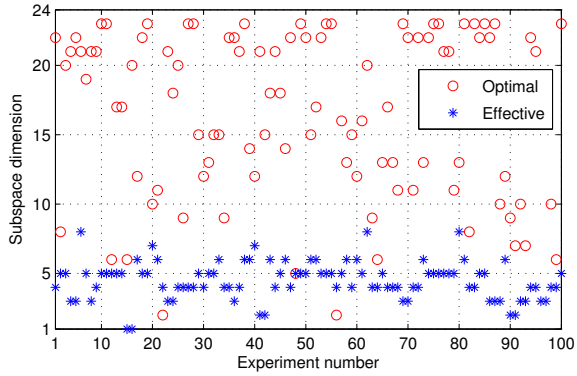


Figure 4.7: The optimal and the effective subspace dimension for estimating the maximum subspace distance between covariance matrices corresponding to consecutive time windows in Internet backbone traffic traces.

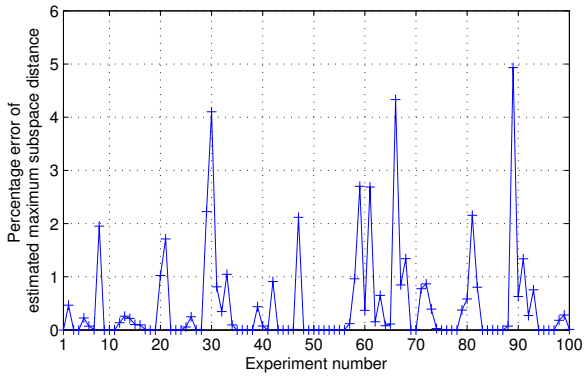


Figure 4.8: Relative percentage error of the estimated maximum subspace distance between covariance matrices corresponding to consecutive time windows in Internet backbone traffic traces.

solution to determine an effective number of principal components to obtain a good approximation of the maximum subspace distance.

Fig. 4.8 shows that the relative percentage error of the estimated maximum subspace distance computed by the *getESD* algorithm rarely exceeds 5%. In fact, the average relative percentage error is as small as 0.64%.

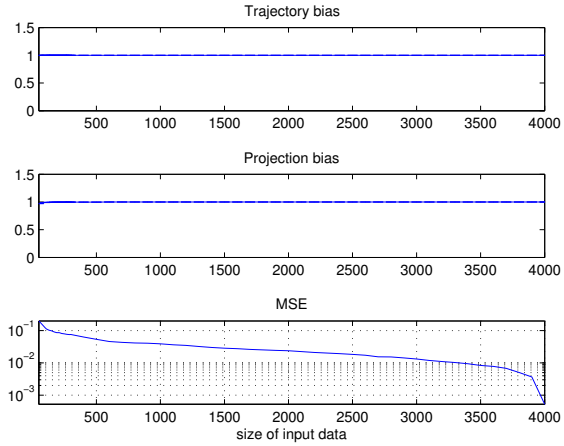


Figure 4.9: Estimated principal component using distributed power iteration

4.6.3 Evaluating the distributed power iteration method

We use three metrics to evaluate the estimation result. The first metric, *trajectory bias*, quantifies the closeness of the estimated principal component to a unit-norm vector. It is the vector norm of the estimated principal component and the trajectory should be 1 if the estimate is properly normalized. The second metric, *projection bias*, quantifies the bias of the estimate in the direction. The third metric, *MSE* or mean squared error, quantifies the closeness of the estimate to the actual principal component.

Simulation setting for result in Figure 4.9 uses 100 nodes, with sample size varying from 200 to 4000, and a maximum of 10 steps is used for average consensus. In average, the result converges after around 20 distributed power iterations. The result shows that the estimated eigenvector has unit norm and is in the same direction with the actual eigenvector; the mean squared error falls below 0.1% when the sample size is larger than 3600.

We also compare the result of MSE due to our distributed power iteration method with that of the centralized power iteration method as shown in Figure 4.10. The MSE as a result of the distributed method and the centralized method is quite close: given partial data, both methods experience similarly large MSE when the size of input data is small. These results show that our distributed version of power iteration method can achieve an estimate as good as using the centralized method.

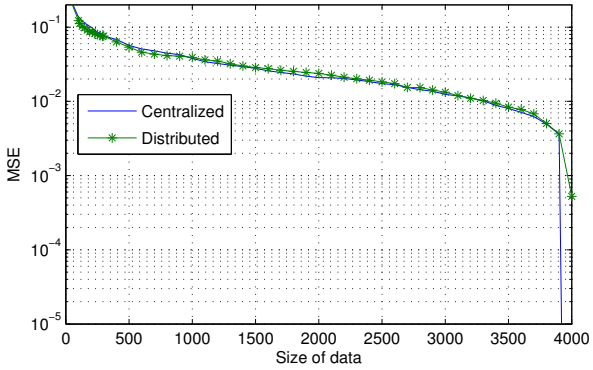


Figure 4.10: The mean square error between the actual principal component with the estimated principal component. The estimation of principal component is performed using the distributed power iteration and the centralized power iteration when the sample size increases.

4.6.4 Application on anomaly detection

For the purpose of anomaly detection, we use the effective subspace dimension and the optimal subspace dimension as a result of comparing Σ_B and Σ_A. Here, Σ_B denotes the before-traffic (dataset **BEFORE**), and is assumed to represent the normal traffic, while Σ_A is the covariance matrix of the after-traffic (dataset **AFTER**), where some anomalous traffic has been injected into the network. We then construct a normal subspace that consists of the first few principal components of Σ_B, which is assumed to be the normal subspace and distinct from that of Σ_A.

We compare our technique with one of the popular technique for network anomaly detection, the subspace method, and show that our method show similar detection ability and, more importantly, can detect anomalies that could be missed by the subspace method.

The subspace method

An underlying assumption of the subspace method is that given a dataset of network traffic, a small amount of principal components are enough to represent the characteristic of the dataset due to its low-dimension property. Since the importance of a principal component is defined by the variance it captures, in the subspace method, a normal subspace of low dimension, U, is built such that it consists of the first few principal components, i.e. the minimum set of principal components that capture some pre-defined percentage of variance in the dataset.

As a result, the projection of normal data onto the normal subspace captures most of the data,

Table 4.1: Number of principal components that captures 99.9% variance with varying λ, when the data dimension is 100.

λ	75	100	200	500
number of PC(s)	10	7	3	2

and the projection of anomalous data, whose characteristic is not in the aforementioned normal subspace, will only capture very little of the data. Once such normal subspace U has been built, the projection residual of a single vector $\mathbf{x}$ is defined as $\| (I - UU') \|$, where I is the identity matrix.

Such-defined projection residual is the residual of the data after it has been projected to the normal subspace: if the data is normal, the projection residual should be small; otherwise, an unusually large projection residual will be observed. The projection residual, therefore, shows the level of abnormality, also known as the abnormal score. After the normal subspace has been modeled, the detection process is simply thresholding on the projection residual, and projection residuals above the threshold will trigger the alarm [46].

The test data

One of the two datasets for experiment is synthetic data. Assume singular value decomposition of its covariance matrix is $\Sigma = U\Lambda V$, then $U' = V$ since covariance matrix Σ is symmetric, columns of U are eigenvectors of Σ and principal components of data that has covariance Σ, and Λ is a diagonal matrix, the k-th entry of whose diagonal shows the amount of variance that the k-th principal component captures. We set the diagonal values of Λ to be a set of values that are exponentially decaying, $\Lambda(k,k) = \exp(\lambda - \lambda * k)$, and set a set of random orthonormal vectors to be columns of U. The synthetic dataset is drawn from random Gaussian distribution with zero mean and Σ as its covariance matrix. One example of the diagonal entries of Λ built with $\lambda = 75$ is shown in Figure 4.11, where 10% of the principal components captures more than 99.9% of the total variance. Of course, a larger λ leads to a smaller set of principal component that captures the same level of variance, as presented in Table 4.1.

The other dataset is the anonymized passive traffic traces from one of CAIDA's monitors on high-speed Internet backbone links, equinix-chicago-dirA [13]. We consider the histogram of

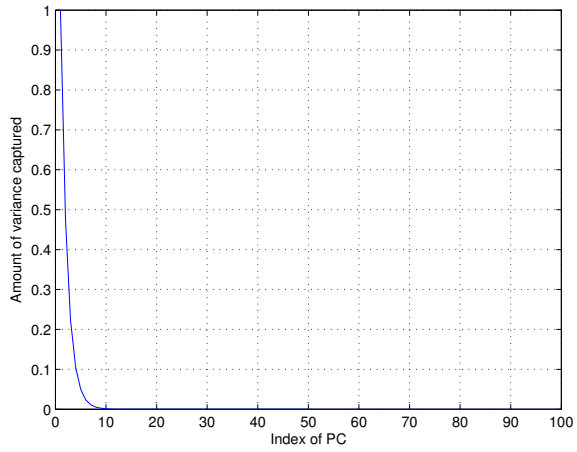


Figure 4.11: The amount variance captured by each principal components of the test data model

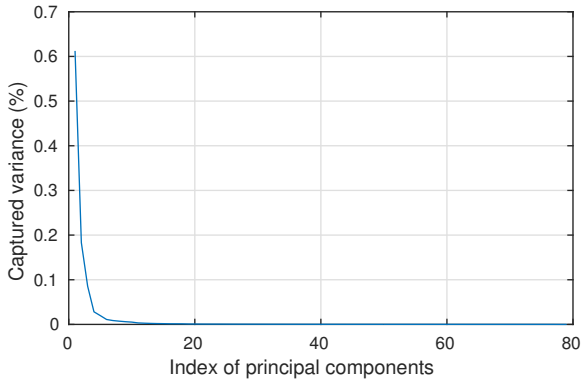


Figure 4.12: The amount variance captured by each principal components of the histogram data

packet sizes and protocol of the packets flowing over a single link within 25ms. The number of bins is 75 for packet sizes and 5 for protocol types, leading to a feature vector of length 80. For the trace lasting 1 minute, we obtain around 2400 examples of the feature vectors and use them to calculate a covariance matrix.

The amount of variance captured by each principal component is shown in Figure 4.12. As we can see, the histogram of packets of the real network traffic has the same property with the synthetic dataset generated with large λ : a very small set of principal components captures most of the variance in a high-dimensional dataset, which is also assumed by the subspace method.

4.6.5 Anomaly detection using Projection Residual

For the purpose of anomaly detection, we use the effective subspace dimension to construct a normal subspace that consists of the first few principal components of Σ_B (covariance matrix of dataset BEFORE), which is assumed to be normal. We then project dataset BEFORE and dataset AFTER onto the normal subspace, and use the projection residual to indicate the level of abnormality. The projection residual of the dataset BEFORE and AFTER is shown in the top of Figure 4.13. It is clear that the dataset BEFORE has small projection residuals, which is expected; dataset AFTER has large projection residuals, indicating it has different property in terms of covariance matrix.

We also compare this result with the subspace method, where the dimension of normal subspace captures 99.5% of the variance in the dataset, as shown at the bottom of Figure 4.13. The projection residual result of subspace method is similar to our algorithm getESD.

We show in Figure 4.14 the receiver operating characteristic (ROC) curve showing the relationship between hit rate and false alarm rate when the threshold is set differently. Given a threshold for the projection residual, the hit rate is defined as the percentage of datapoints in dataset AFTER whose projection residual is above the threshold, and the false alarm rate is defined as the percentage of datapoints in dataset BEFORE whose projection is above the threshold. The ROC curve is a result of varying threshold from 0 to 4 with step size equal to $1e-3$. The two methods, subspace method and getESD, have similar performance in detecting dataset AFTER being different from dataset BEFORE: the best performance of subspace method in detecting dataset AFTER is hit rate 99.9% with false alarm rate 0.1% when the threshold is around 0.29; while the getESD has 100% hit rate with 0% false alarm rate when the threshold is set between 0.1 and 0.26. This result shows that the performance of anomaly detection using the effective subspace dimension to construct normal subspace is as good as that using the dimension defined by variance.

4.6.6 Anomaly detection against normal traffic spoofing

We define normal traffic spoofing as follows: suppose the anomaly detection of a system relies on some publicly known profile of normal traffic, i.e. the normal subspace, then inserting anomalous traffic that resides in the normal subspace is called normal traffic spoofing.

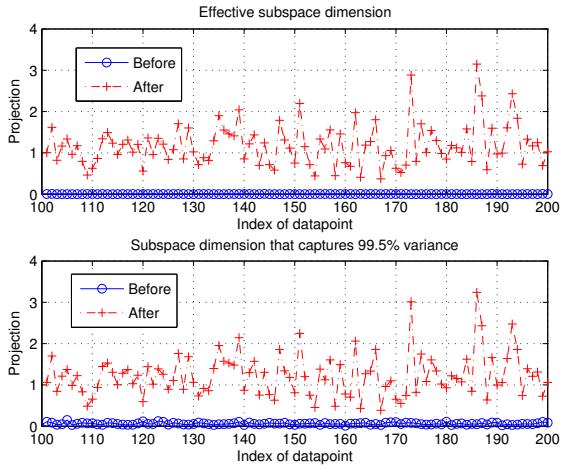


Figure 4.13: Projection residual of dataset BEFORE and AFTER onto normal subspace. Top: the dimension of normal subspace is a result of our algorithm, getESD. Bottom: the dimension of normal subspace is the minimum number of principal components that captures 99.5% of the variance in the dataset.

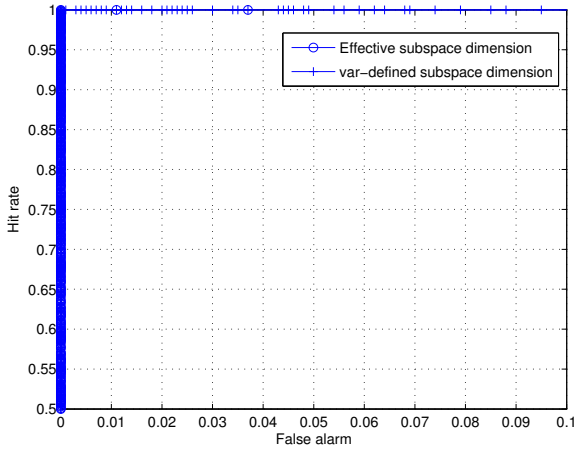


Figure 4.14: Receiver operating characteristic curve as a result of getESD and the subspace method.

In this subsection, we consider a particular case where the set of principal components for the normal traffic is the same as the set for the anomalous traffic, while the order of the principal components is changed because of the attack data. A real case could be that an attacker learns the normal subspace and designs the traffic data following the normal pattern. Suppose an anomaly detection system uses the histogram of packet sizes and protocols as traffic features. In order to launch a denial of service attack that evades detection, the attacker can design the size of attack packets such that the attack traffic follows the direction of one principal component in the normal

subspace. As a result, the order of principal components of the attack traffic can differ from that of the normal traffic.

Suppose the normal dataset has $N \times N$ covariance matrix $\Sigma_B = U\Lambda U'$, where $U = V' = [\mathbf{u}_1, \ldots, \mathbf{u}_N]$ with random orthonormal columns, and $\Lambda(k,k) = \exp(\lambda - \lambda * k)$. Assume the injected anomalous traffic behaves as the 4-th principal component in the normal subspace, and the resulting covariance matrix becomes $\Sigma_A = V\Lambda_A V'$, with $V = [\mathbf{v}_1, \ldots, \mathbf{v}_N]$, and

$$\mathbf{v}_n = \begin{cases} \mathbf{u}_4 & \text{if } n = 3 \\ \mathbf{u}_3 & \text{if } n = 4 \\ \mathbf{u}_n & \text{o.w.} \end{cases}$$

In other words, the injected anomalous traffic changes the order of importance of these two principal components, $\mathbf{u}_3$ and $\mathbf{u}_4$, both of which resides in the normal subspace.

Under this setting, we find that the effective subspace dimension is 3, while the dimension for the normal subspace that captures 99.5% variance is 8. We present in Figure 4.15 the projection residual before and after such anomalous traffic has been injected. One thing we observe from Figure 4.15 is that because the anomalous data resides in the normal subspace, there is no difference between the projection residual of normal data and anomalous data when the normal subspace dimension is defined by the variance. It indicates that the variance-defined subspace is not able to detect the anomaly that spoofed as normal traffic.

The projection residual when the normal subspace is defined by the effective subspace dimension is high for both BEFORE and AFTER datasets. At the first sight, this result alone may indicate high detection rate with high false alarm rate. However, the result of our getESD method shows in Figure 4.16 that the subspace distance between two subsets, one for the normal traffic and one for the anomalous traffic, reaches a local maximum when subspace dimension is only 3, indicating that the difference between these two datasets can be told by the first three principal components. It also explains the larger projection residuals we observe in Figure 4.15. A close examination of output of the getESD algorithm shows that the order of the first 4 principal components of the attack traffic has been changed, and the anomaly resides along the 4th principal components of the normal data. It takes other actions such as examining the traffic that resides along the 4th principal components

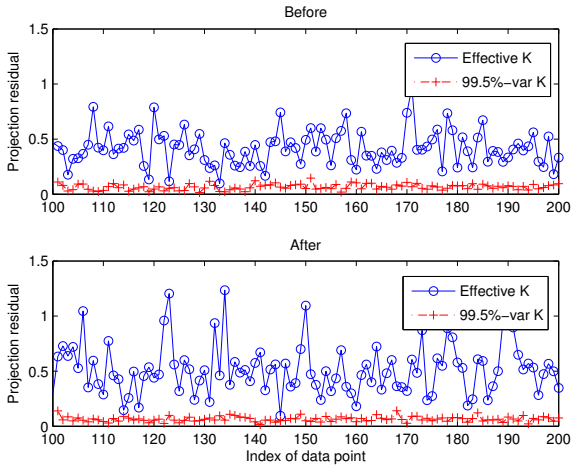


Figure 4.15: Projection residual of test data in anomalous subspace, the dimension of which is defined by our method and the subspace method. Top: dataset BEFORE. Bottom: dataset AFTER

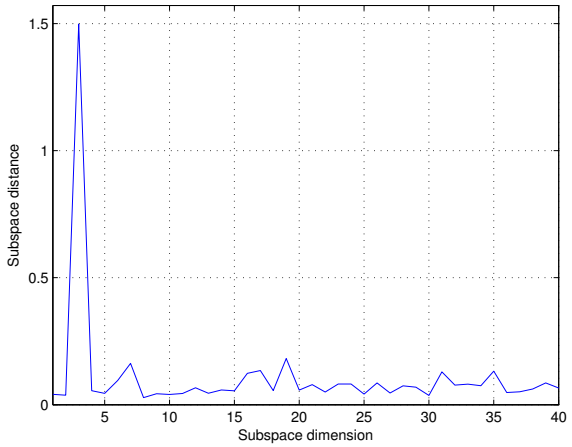


Figure 4.16: Subspace distance using different number of principal components

of the normal data in order to filter out the attack traffic.

As a result, we conclude that the getESD algorithm is not under the impact of normal traffic spoofing as the subspace does. Instead, our algorithm can be applied to tell which one or set of principal components have been spoofed in the anomalous data.

4.6.7 Running time comparison

The computational complexity of the subspace method is $O(N^3)$, which is much larger than the complexity of getESD, $O(Pk^3 + k^2N)$, when both P and k is small.

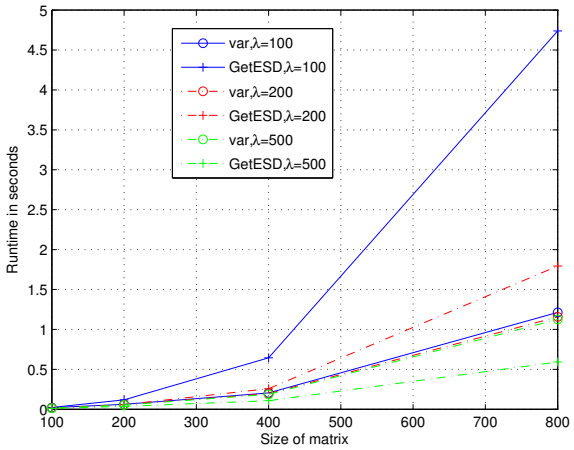


Figure 4.17: Comparing running time of GetESD with that of the subspace method.

Figure 4.17 compares the running time of our algorithm with that of the subspace method on the synthetic data. We observe that a larger λ leads to shorter running time for the getESD algorithm. This is expected since steeper decaying variance leads to faster converges of the power method. We observe that with varying λ, the running time of the subspace method does not vary much, since the subspace method need to compute the whole set of principal components before it tries to analyze the amount of variance captured by each principal component. When λ is 500, the running time of our algorithm is only half of what takes the subspace method.

4.7 Concluding Remarks

The two algorithms presented in this chapter, GETESD and GETESD-D, reduce the computational overhead of comparing two sets of traffic features for real-time anomaly detection. Meanwhile, it also retains an ability to distinguish between two datasets for traffic features that is close to what an optimal algorithm would accomplish. These two algorithms combined can be a general-purpose tool for comparing structural differences between evolving traffic patterns. They are applicable in almost any scenario that involves either processing a stream of matrices for centralized real-time detection of developing changes or processing a stream of raw measurements for distributed real-time detection.

5. Conclusions and future work

5.1 System monitoring

The concept of system monitoring is nothing new, for there exist numerous products in practice for monitoring purposes. However, the current challenge for system monitoring arises from the increasing complexity of a system as well as its vulnerability to increasingly sophisticated attacks. During monitoring, the type of measurements as well as the data collection method need to be carefully designed to reflect the overall status of the system. Furthermore, one cannot ignore the costs associated with monitoring which can interfere with the normal operation of a system and deplete the supply of resources available for the system.

To overcome the aforementioned challenges, we propose two monitoring tools in this dissertation. Both of these tools are efficient in incurring reduced monitoring costs while effective in preserving useful information of the data to be collected. Our approaches exploit the sparsity of the data with an underlying assumption that most of the measurements are sparse. This assumption implies that if we transform the data into some basis, we only need a small amount of its representation to keep most of its information.

5.1.1 C-MON

The first of our proposed monitoring tools is based on compression, referred to as C-MON. In C-MON, performance-related measurements are first transformed into a basis, and the most significant entries of the resulting representation in the basis are kept and then used for data recovery and analysis. The major contribution of our work is the best-basis algorithm employed in C-MON. This algorithm automatically adapts the transform basis to the structure of the measurements, achieving an extremely high compression ratio. Our simulation results on measurements of a data center show that using C-MON, a reconstruction error lower than 1% is achieved when the sample size is only 1%. We find C-MON effective in requiring a small data size while capturing most content of the orignial data.

5.1.2 CS-MON

Our second monitoring tool, termed as CS-MON, is based on compressive sampling. The technique of compressive sampling is a non-adaptive technique originally developed for image compression, and our work is among the first few to apply compressive sampling for system monitoring. In CS-MON, performance-related measurements are sampled using compressive sampling, and the resulting samples are kept and then used for data analysis. When further analysis is required, linear programming algorithms can be applied to recover the full-length data. The technique of compressive sampling significantly reduces the costs associated with information collection by sampling data directly into a compressed form. Its low-complexity sampling process allows CS-MON applicable for any system that asks for a low cost associated with data collection.

We also contribute to the adaptive-rate mechanism used in CS-MON. In our approach, the sample size is automatically adjusted to the structure of the measurements such that a larger sample size is used for measurements that contain more information. The goal is to use a sample size large enough to capture all information without incurring too much cost. We show that the reconstructed data preserves useful information for the detection of abrupt changes and trends in measurements collected from a data center, achieving 90% hit rate when the sample size is 25%.

5.2 Anomaly detection

Employment of anomaly detection is critical to ensure the reliability and security of a system. Traditional approaches for fault detection rely on signatures of known conditions such as system failures or malicious attacks. These signature-based approaches, however, can fail when confronted with unknown conditions due to new circumstances or new attacks. On the contrary, anomaly detection techniques do not rely on a dictionary of known signatures, but treat conditions of interest as outliers not obeying a normal behavior.

Developing a pervasively effective technique for anomaly detection is challenging. The complexity of a system adds to the difficulty of deriving a normal behavior, while the increasingly sophisticated attacks make it crucial to define an outlier. In this dissertation, we tackle the problem of anomaly detection from different angles. In our first approach for anomaly detection, we propose

a new possibility for the definition of normal behavior and outliers by showing that anomaly detection can be applied directly on compressed measurements. The other two approaches improve upon state-of-the-art techniques in defining the normal behavior. In our second approach, we characterize performance-related measurements as a stream of covariance matrices, one for each designated window of time, and then use observed changes in the covariance matrices to infer anomalies in the system. In our third approach, anomalies in a system are detected using a distributed algorithm when only steams of raw measurement vectors, one for each time window, are available and distributed among multiple locations.

5.2.1 Anomaly detection with compressed measurements

In our first approach for anomaly detection, we propose to apply state-of-the-art anomaly detection techniques directly on the compressed measurements. We use the low-complexity data collection of compressive sampling with a reduced cost for data reconstruction. The data collection is similar to CS-MON, where performance-related measurements are processed using compressive sampling. However, instead of being used only for recovering the original data, the compressed measurements are analyzed directly for anomaly detection. Our theoretical results show that important statistical properties of the raw data—in particular, mean, variance, and correlation changes—are well preserved in the compressed measurements, allowing us to exploit these properties of the compressed measurements for anomaly detection (step one). Once an anomaly is detected, we can use the compressed measurements to recover the full-length data for further analysis (step two). Our simulation results on measurements of a data center show that detection performance using our two-step detection mechanism is promising: the achieved hit rate is greater than 95% when the sample size is only 18% and the false alarm rate is fixed at 0.5%.

5.2.2 Centralized anomaly detection using covariance matrices

In our second approach we detect anomalies in a system by detecting changes in the correlation pattern of its features. We use covariance matrices of performance-related measurements to summarize system status. In this way, we place no assumption on the distribution of the measurements. Moreover, covariance matrices provide the correlations between each pair of features, the exact in-

formation we need to detect changes in the correlation pattern of a system. To quantify correlation changes between two covariance matrices, we introduce a new metric called the maximum subspace distance. We also develop an efficient algorithm to estimate this metric with reduced complexity. The estimate by our algorithm, termed the effective subspace dimension, is then employed to define the dimension of the normal behavior for anomaly detection. The simulation results on backbone network traffic show that the effective subspace dimension reduces the dimension by more than 70% while the average relative percentage error is as small as 0.64%. Results of anomaly detection using the effective subspace dimension is similar to that of state-of-the-art techniques. However, with the reduced dimension employed for detection, our method has improved efficiency.

5.2.3 Distributed anomaly detection using raw measurements

Our third approach also defines the dimension of normal subspace using the effective subspace dimension, an estimate of the maximum subspace distance. However, we consider a special case where the covariance matrices are not available for the estimation. In particular, raw measurements that can be used for computing the covariance matrices are distributed among several locations. Under this condition, we propose a low-complexity distributed algorithm for the estimation of the effective subspace dimension, where neighboring locations exchange their measurements and their own estimates of the subspace to update their estimates. The converged estimate of the subspace help each location to determine the effective subspace dimension. We show that in the distributed scenario, the detection performance of our estimate is as effective as that using the maximum subspace distance and that of the centralized algorithm.

5.3 Future work

With regards to system monitoring using compressive sampling, our work in Chapter 2 has limited the measurements to numerical values. Many applications of system monitoring, however, deal with non-numerical values such as program execution traces for malware detection. An extension of our approach to work with non-numerical measurements can broaden the applicability of compressive sampling for system monitoring. One solution is to map non-integer values into integers before

applying compressive sampling. However, the mapping should be appropriately designed such that both the underlying correlation and sparsity of the original values are well preserved in their mapped values. For example, for natural language processing, a simple mapping from words to numbers can lose important correlations between letters in the same word or correlations between two words in a sentence. As a result, the solution for compressive sampling with non-numerical datasets would acquire sophisticated techniques specific to its application to preserve the properties of sparsity and correlations in the non-numerical data.

In Chapter 2 we propose the best basis algorithm to automatically adjust the basis to the structure of the underlying data. In our current implementation, the best basis is selected from a dictionary of bases that are persuasively used for data compression. Instead of using a dictionary of bases, we can learn the data structure during the monitoring process, and use the learned information to derive new bases in which the data has extremely low sparsity. Incorporating the data structure into the basis design for compressive sampling is a potential area of research that has not yet been investigated.

We have solved in Chapter 3 the relationship between the correlation of compressed measurements and that of the original data, and have shown how the properties of compressed measurements can be exploited to detect anomalies effectively. Most of current work on compressive sampling, including ours, uses random Gaussian matrices for sampling. So far, we have not investigated the influence of different sampling matrices on the detection results. As a matter of fact, the design of sampling matrix for compressive sampling other than using random matrices is brand-new. However, we expect the sampling matrix to be able to extract information in the original data useful for anomaly detection. Future work should include novel designs for a sampling matrix that leads to a smaller amount of compressed measurements but still captures sufficient information for anomaly detection.

Bibliography

[1] Z. Bu and R. Rachwald, "Ghost-hunting with anti-virus," *https://goo.gl/vFjKTv*, 2014.

[2] D. Yadron, "Symantec develops new attack on cyberhacking," *http://goo.gl/x0yiAf*, 2014.

[3] E. Kovacs, "Malware and DDoS were the most common attack types in 2014: IBM," *http://goo.gl/54BCGz*, 2015.

[4] Symantec Corporation, "Internet security threat report," *Symantec Corporation Technical Report*, vol. 20, 2015.

[5] A. Roy and J. Das, "FireEye revenue jumps 150%," *http://goo.gl/Q2wSDh*, 2015.

[6] A. Winn, "Have printers become a gateway for malware?" *https://goo.gl/h2dSBX*, 2014.

[7] O. Younis and S. Fahmy, "Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach," in *IEEE Conference on Computer Communications (INFOCOM)*, 2004.

[8] V. Mhatre and C. Rosenberg, "Design guidelines for wireless sensor networks: communication, clustering and aggregation," *Ad Hoc Networks*, vol. 2, no. 1, pp. 45–63, 2004.

[9] S. Bandyopadhyay and E. J. Coyle, "An energy efficient hierarchical clustering algorithm for wireless sensor networks," in *IEEE Conference on Computer Communications (INFOCOM)*, vol. 3, Mar. 2003, pp. 1713–1723.

[10] ——, "Minimizing communication costs in hierarchically-clustered networks of wireless sensors," *Computer Networks*, vol. 44, no. 1, pp. 1–16, Jan. 2004.

[11] K. Sohrabi, J. Gao, V. Ailawadhi, and G. Pottie, "Protocols for self-organization of a wireless sensor network," *IEEE Personal Communications*, vol. 7, no. 5, pp. 16–27, Oct. 2000.

[12] J. Stankovic, T. Abdelzaher, C. Lu, L. Sha, and J. Hou, "Real-time communication and coordination in embedded sensor networks," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1002–1022, July 2003.

[13] M. Bhardwaj and A. Chandrakasan, "Bounding the lifetime of sensor networks via optimal role assignments," in *IEEE Conference on Computer Communications (INFOCOM)*, vol. 3, 2002, pp. 1587–1596.

[14] P. Ogren, E. Fiorelli, and N. Leonard, "Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment," *IEEE Transactions on Automatic Control*, vol. 49, no. 8, pp. 1292–1302, Aug. 2004.

[15] X.-Y. Li, X. Xu, S. Wang, S. Tang, G. Dai, J. Zhao, and Y. Qi, "Efficient data aggregation in multi-hop wireless sensor networks under physical interference model," in *IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, Oct. 2009, pp. 353–362.

[16] B. Krishnamachari, D. Estrin, and S. Wicker, "The impact of data aggregation in wireless sensor networks," in *IEEE International Conference on Distributed Computing Systems Workshops (ICDCS)*, 2002, pp. 575–578.

[17] J. Kusuma, L. Doherty, and K. Ramchandran, "Distributed compression for sensor networks," in *IEEE International Conference on Image Processing*, vol. 1, 2001, pp. 82–85.

[18] S. Pradhan, J. Kusuma, and K. Ramchandran, "Distributed compression in a dense microsensor network," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 51–60, Mar. 2002.

[19] T. Cui, L. Chen, T. Ho, S. Low, and L. Andrew, "Opportunistic source coding for data gathering in wireless sensor networks," in *IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, Oct. 2007, pp. 1–11.

[20] S. Pradhan and K. Ramchandran, "Distributed source coding: symmetric rates and applications to sensor networks," in *Data Compression Conference*, 2000, pp. 363–372.

[21] E. K. Lee, H. Viswanathan, and D. Pompili, "Silence: Distributed adaptive sampling for sensor-based autonomic systems," in *ACM International Conference on Autonomic Computing (ICAC)*, 2011, pp. 61–70.

[22] ——, "Distributed data-centric adaptive sampling for cyber-physical systems," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 9, no. 4, pp. 1–27, Jan. 2015.

[23] J. Wilkes, "More Google cluster data," *http://googleresearch.blogspot.com*, Nov. 2011.

[24] I. Cozzani and S. Giordano, "Traffic sampling methods for end-to-end QoS evaluation in large heterogeneous networks," *Computer Networks and ISDN Systems*, vol. 30, no. 16–18, pp. 1697–1706, Sep. 1998.

[25] N. Duffield, "Sampling for passive Internet measurement: A review," *Statistical Science*, pp. 472–498, 2004.

[26] D. S. Starnes, D. Yates, and D. S. Moore, *The practice of statistics*. Macmillan, 2010.

[27] C. Hu, S. Wang, J. Tian, B. Liu, Y. Cheng, and Y. Chen, "Accurate and efficient traffic monitoring using adaptive non-linear sampling method," in *IEEE Conference on Computer Communications (INFOCOM)*, 2008.

[28] S. Ali, I. U. Haq, S. Rizvi, N. Rasheed, U. Sarfraz, S. A. Khayam, and F. Mirza, "On mitigating sampling-induced accuracy loss in traffic anomaly detection systems," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 3, pp. 4–16, 2010.

[29] S. Meng, A. K. Iyengar, I. M. Rouvellou, and L. Liu, "Volley: Violation likelihood based state monitoring for datacenters," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2013, pp. 1–10.

[30] X. Wang, X. Li, and D. Loguinov, "Modeling residual-geometric flow sampling," *IEEE/ACM Transactions on Networking*, vol. 21, no. 4, pp. 1090–1103, Aug. 2013.

[31] G. Wade, *Signal coding and processing*. Cambridge University Press, 1994.

[32] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, pp. 1289–1306, 2006.

[33] S. Foucart, "Hard thresholding pursuit: An algorithm for compressive sensing," *SIAM Journal on Numerical Analysis*, vol. 49, no. 6, pp. 2543–2563, Dec. 2011.

[34] E. Candes and M. Wakin, "An introduction to compressive sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21–30, Mar. 2008.

[35] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, 2009.

[36] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42–57, 2013.

[37] F. Feather, D. Siewiorek, and R. Maxion, "Fault detection in an ethernet network using anomaly signature matching," *ACM SIGCOMM Computer Communication Review*, vol. 23, no. 4, pp. 279–288, Oct. 1993.

[38] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring Internet denial-of-service activity," *ACM Transactions on Computer Systems*, vol. 24, no. 2, pp. 115–139, May 2006.

[39] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23–24, pp. 2435–2463, Dec. 1999.

[40] M. Roesch, "Snort: Lightweight intrusion detection for networks." in *LISA*, vol. 99, no. 1, 1999, pp. 229–238.

[41] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proceedings of ACM SIGCOMM Workshop on Internet measurment*, Nov. 2002, pp. 71–82.

[42] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *ACM SIGCOMM Conference on Internet Measurement*, 2003, pp. 234–247.

[43] J. D. Brutlag, "Aberrant behavior detection in time series for network monitoring," in *USENIX Conference on System Administration*, 2000, pp. 139–146.

[44] M. Roughan, T. Griffin, M. Mao, A. Greenberg, and B. Freeman, "Combining routing and traffic data for detection of IP forwarding anomalies," *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, pp. 416–417, Jun. 2004.

[45] O. S. Vallis, J. Hochenbaum, and A. Kejariwal, "Twitter AnomalyDetection R package," *https://github.com/twitter/AnomalyDetection*, 2014.

[46] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 217–228, Aug. 2005.

[47] Z. Lan, Z. Zheng, and Y. Li, "Toward automated anomaly identification in large-scale systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 2, pp. 174–187, 2010.

[48] Q. Guan and S. Fu, "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures," in *IEEE International Symposium on Reliable Distributed Systems*, Sept. 2013, pp. 205–214.

[49] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 219–230, Aug. 2004.

[50] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang, "Principal component-based anomaly detection scheme," in *Foundations and Novel Approaches in Data Mining*. Springer, 2006, pp. 311–329.

[51] D. Brauckhoff, K. Salamatian, and M. May, "Applying PCA for traffic anomaly detection: Problems and solutions," in *IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2009, pp. 2866–2870.

[52] S. Fu, "Performance metric selection for autonomic anomaly detection on cloud computing systems," in *IEEE Global Communications Conference, Exhibition & Indudstry Forum (GLOBECOM)*, 2011, pp. 1–5.

[53] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[54] S. Mallat, *A wavelet tour of signal processing*. Academic press, 1999.

[55] E. Axell, G. Leus, E. G. Larsson, and H. V. Poor, "Spectrum sensing for cognitive radio: State-of-the-art and recent advances," *IEEE Signal Processing Magazine*, vol. 29, no. 3, pp. 101–116, 2012.

[56] Y. L. Polo, Y. Wang, A. Pandharipande, and G. Leus, "Compressive wide-band spectrum sensing," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2009, pp. 2337–2340.

[57] J. A. Bazerque and G. B. Giannakis, "Distributed spectrum sensing for cognitive radio networks by exploiting sparsity," *IEEE Transactions on Signal Processing*, vol. 58, no. 3, pp. 1847–1862, 2010.

[58] Z. Zhang, H. Li, D. Yang, and C. Pei, "Space-time bayesian compressed spectrum sensing for wideband cognitive radio networks," in *IEEE Symposium on New Frontiers in Dynamic spectrum*, 2010, pp. 1–11.

[59] M. Lustig, D. Donoho, and J. M. Pauly, "Sparse MRI: The application of compressed sensing for rapid MR imaging," *Magnetic resonance in medicine*, vol. 58, no. 6, pp. 1182–1195, 2007.

[60] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly, "Compressed sensing MRI," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 72–82, 2008.

[61] U. Gamper, P. Boesiger, and S. Kozerke, "Compressed sensing in dynamic MRI," *Magnetic Resonance in Medicine*, vol. 59, no. 2, pp. 365–373, 2008.

[62] M. Murphy, M. Alley, J. Demmel, K. Keutzer, S. Vasanawala, and M. Lustig, "Fast-SPIRiT compressed sensing parallel imaging MRI: scalable parallel implementation and clinically feasible runtime," *IEEE Transactions on Medical Imaging*, vol. 31, no. 6, pp. 1250–1262, 2012.

[63] M. Tello Alonso, P. López-Dekker, and J. J. Mallorquí, "A novel strategy for radar imaging based on compressive sensing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 12, pp. 4285–4295, 2010.

[64] L. C. Potter, E. Ertin, J. T. Parker, and M. Cetin, "Sparsity and compressed sensing in radar imaging," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1006–1020, 2010.

[65] V. M. Patel, G. R. Easley, D. M. Healy Jr, and R. Chellappa, "Compressed synthetic aperture radar," *IEEE Journal on Selected Topics in Signal Processing*, vol. 4, no. 2, pp. 244–254, 2010.

[66] M. A. Herman and T. Strohmer, "High-resolution radar via compressed sensing," *IEEE Transactions on Signal Processing*, vol. 57, no. 6, pp. 2275–2284, 2009.

[67] J. Prades-Nebot, Y. Ma, and T. Huang, "Distributed video coding using compressive sampling," in *IEEE Picture Coding Symposium*, 2009, pp. 1–4.

[68] L.-W. Kang and C.-S. Lu, "Distributed compressive video sensing," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2009, pp. 1169–1172.

[69] S. Pudlewski and T. Melodia, "On the performance of compressive video streaming for wireless multimedia sensor networks," in *IEEE International Conference on Communications (ICC)*, 2010, pp. 1–5.

[70] C. Caione, D. Brunelli, and L. Benini, "Distributed compressive sampling for lifetime optimization in dense wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 1, pp. 30–40, 2012.

[71] S. Das and T. Sidhu, "Application of compressive sampling in computer based monitoring of power systems," *Advances in Computer Engineering*, vol. 2014, 2014.

[72] T. Tuma, S. Rooney, and P. Hurley, "On the applicability of compressive sampling in fine grained processor performance monitoring," in *IEEE International Conference on Engineering of Complex Computer Systems*, 2009, pp. 210–219.

[73] R. Ward, "Compressed sensing with cross validation," *IEEE Transactions on Information Theory*, vol. 55, no. 12, pp. 5773–5782, 2009.

[74] L. Li, K. Vaidyanathan, and K. Trivedi, "An approach for estimation of software aging in a web server," in *IEEE International Symposium on Empirical Software Engineering*, 2002, pp. 91–100.

[75] A. Avritzer, A. Bondi, M. Grottke, K. S. Trivedi, and E. J. Weyuker, "Performance assurance via software rejuvenation: Monitoring, statistics and algorithms," in *IEEE International Conference on Dependable Systems and Networks (DSN)*, 2006, pp. 435–444.

[76] H. Ringberg, A. Soule, J. Rexford, and C. Diot, "Sensitivity of PCA for traffic anomaly detection," *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1, pp. 109–120, Jun. 2007.

[77] D. Yeung, S. Jin, and X. Wang, "Covariance-matrix modeling and detecting various flooding attacks," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 37, no. 2, pp. 157–169, 2007.

[78] G. H. Golub and C. F. Van Loan, *Matrix computations*, 3rd ed. Johns Hopkins University Press, 2012.

[79] M. Jelasity, G. Canright, and K. Engø-Monsen, "Asynchronous distributed power iteration with gossip-based normalization," in *Euro-Par Parallel Processing*. Springer, 2007, pp. 514–525.

[80] A. Bertrand and M. Moonen, "Power iteration-based distributed total least squares estimation in ad hoc sensor networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 2669–2672.

[81] T. Huang, N. Kandasamy, and H. Sethu, "Evaluating compressive sampling strategies for performance monitoring of data centers," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2012, pp. 655–658.

[82] ——, "Evaluating compressive sampling strategies for performance monitoring of data centers," in *ACM International Conference on Autonomic Computing (ICAC)*, 2012, pp. 201–210.

[83] T. Huang, N. Kandasamy, H. Sethu, and M. C. Stamm, "An efficient strategy for online performance monitoring of datacenters via adaptive sampling," ECE Department, Drexel University, Tech. Rep., May 2015, Can be accessed at www.ece.drexel.edu/kandasamy/adaptive_cs.pdf.

[84] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "Automated anomaly detection and performance modeling of enterprise applications," *ACM Transactions on Computer Systems*, vol. 27, no. 3, pp. 1–32, 2009.

[85] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf, "Monalytics: Online monitoring and analytics for managing large scale data centers," *ACM International Conference on Autonomic Computing (ICAC)*, pp. 141–150, 2010.

[86] G. Lanfranchi, P. D. Peruta, A. Perrone, and D. Calvanese, "Toward a new landscape of systems management in an autonomic computing environment," *IBM Systems Journal*, vol. 42, no. 1, pp. 119–128, 2003.

[87] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on Information Theory*, vol. 52, no. 2, pp. 489–509, 2006.

[88] E. J. Candès and T. Tao, "Near optimal signal recovery from random projections: Universal coding strategies?" *IEEE Transactions on Information Theory*, vol. 52, no. 12, pp. 5406–5425, 2006.

[89] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive management of software aging," in *IBM Journal of Research and Development*, vol. 45, no. 2, 2001, pp. 311–332.

[90] D. Mosberger and T. Jin, "httperf: A tool for measuring web server performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[91] J. S. Walker, *A Primer on Wavelets and their Scientific Applications*, 2nd ed. Chapman and Hall, 2008.

[92] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mappings into a hilbert space," in *Contemp. Math.*, vol. 26, no. 1, 1984, pp. 189–206.

[93] D. Simon, *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[94] P. Boufounos, M. Duarte, and R. Baraniuk, "Sparse signal reconstruction from noisy compressive measurements using cross validation," in *IEEE/SP Workshop on Statistical Signal Processing (SSP)*, 2007, pp. 299–303.

[95] V. Stankovic, L. Stankovic, and S. Cheng, "Compressive image sampling with side information," in *IEEE International Conference on Image Processing (ICIP)*, 2009, pp. 3037–3040.

[96] A. Watkins *et al.*, "Adaptive compressive sensing for low power wireless sensors," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2014, pp. 99–104.

[97] Y. Tan, X. Wang, and K. Lin, "Adaptive image sequence reduction in surveillance using region enhancement block compressive sensing," in *IEEE World Congress Intelligent Control and Automatio (WCICA)*, 2014, pp. 3375–3380.

[98] G. Warnell, D. Reddy, and R. Chellappa, "Adaptive rate compressive sensing for background subtraction," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 1477–1480.

[99] R. Ward, "Compressed sensing with cross validation," *IEEE Transactions on Information Theory*, vol. 55, no. 12, pp. 5773–5782, 2009.

[100] T. Huang, N. Kandasamy, and H. Sethu, "Anomaly detection in computer systems using compressed measurements," in *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2015, pp. 1–12.

[101] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 124–137, April 2005.

[102] R. Canzanese, M. Kam, and S. Mancoridis, "Toward an automatic, online behavioral malware classification system," in *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2013, pp. 111–120.

[103] A. Papoulis and S. U. Pillai, "Probability, random variables, and stochastic processes," *Tata McGraw-Hill Education*, 2002.

[104] D. Paul, "Asymptotics of sample eigenstructure for a large dimensional spiked covariance model," *Statistica Sinica*, vol. 17, no. 4, p. 1617, 2007.

[105] Q. Ding and E. D. Kolaczyk, "A compressed PCA subspace method for anomaly detection in high-dimensional data," *IEEE Transactions on Information Theory*, vol. 59, no. 11, pp. 7419–7433, 2013.

[106] C. Pascoal, M. Rosario de Oliveira, R. Valadas, P. Filzmoser, P. Salvador, and A. Pacheco, "Robust feature selection and robust PCA for Internet traffic anomaly detection," in *IEEE International Conference on Computer Communications (INFOCOM)*, Mar. 2012, pp. 1755–1763.

[107] T. Kudo, T. Morita, T. Matsuda, and T. Takine, "PCA-based robust anomaly detection using periodic traffic behavior," *IEEE International Conference on Communications (ICC)*, pp. 1330–1334, June 2013.

[108] C. Callegari, L. Gazzarrini, S. Giordano, M. Pagano, and T. Pepe, "Improving PCA-based anomaly detection by using multiple time scale analysis and Kullback–Leibler divergence," *International Journal of Communication Systems*, vol. 27, no. 10, pp. 1731–1751, 2014.

[109] D.-S. Pham, S. Venkatesh, M. Lazarescu, and S. Budhaditya, "Anomaly detection in large-scale data stream networks," *Data Mining and Knowledge Discovery*, vol. 28, no. 1, pp. 145–189, 2014.

[110] T. Huang, H. Sethu, and N. Kandasamy, "A fast algorithm for detecting anomalous changes in network traffic," in *IEEE International Conference on Network and Service Management (CNSM)*, 2015.

[111] M. Bhuyan, D. Bhattacharyya, and J. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[112] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *ACM Annual Computer Security Applications Conference*, 2013, pp. 199–208.

[113] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. New York: Springer, 2002.

[114] N. Vlassis, Y. Sfakianakis, and W. Kowalczyk, "Gossip-based greedy Gaussian mixture learning," in *Advances in Informatics*. Springer, 2005, pp. 349–359.

[115] M. Mandjes, I. Saniee, and A. L. Stolyar, "Load characterization and anomaly detection for voice over IP traffic," *IEEE Transactions on Neural Networks*, vol. 16, no. 5, pp. 1019–1026, 2005.

[116] E. Freire, A. Ziviani, and R. Salles, "Detecting VoIP calls hidden in web traffic," *IEEE Transactions on Network and Service Management*, vol. 5, no. 4, pp. 204–214, December 2008.

[117] V. Thing, M. Sloman, and N. Dulay, "Locating network domain entry and exit point/path for DDoS attack traffic," *IEEE Transactions on Network and Service Management*, vol. 6, no. 3, pp. 163–174, September 2009.

[118] Y. Xie and S. zheng Yu, "A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, pp. 54–65, Feb 2009.

[119] I. Paschalidis and G. Smaragdakis, "Spatio-temporal network anomaly detection by assessing deviations of empirical measures," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 685–697, June 2009.

[120] M. Thottan and C. Ji, "Anomaly detection in IP networks," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2191–2204, Aug. 2003.

[121] M. Tavallaee, W. Lu, S. A. Iqbal, A. Ghorbani *et al.*, "A novel covariance matrix based approach for detecting network anomalies," in *Communication Networks and Services Research Conference*. IEEE, 2008, pp. 75–81.

[122] A. Kind, M. Stoecklin, and X. Dimitropoulos, "Histogram-based traffic anomaly detection," *IEEE Transactions on Network and Service Management*, vol. 6, no. 2, pp. 110–121, June 2009.

[123] A. D. Alconzo, A. Coluccia, F. Ricciato, and P. Romirer-Maierhofer, "A distribution-based approach to anomaly detection and application to 3G mobile traffic," in *Global Telecommunications Conference*, 2009, pp. 1–8.

[124] K. Nyalkalkar, S. Sinha, M. Bailey, and F. Jahanian, "A comparative study of two network-based anomaly detection methods," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2011, pp. 176–180.

[125] V. W. Bandara and A. P. Jayasumana, "Extracting baseline patterns in Internet traffic using robust principal components," in *IEEE Conference on Local Computer Networks*, 2011, pp. 407–415.

[126] G. Mateos and G. B. Giannakis, "Robust PCA as bilinear decomposition with outlier-sparsity regularization," *IEEE Transactions on Signal Processing*, vol. 60, no. 10, pp. 5176–5190, 2012.

[127] Z. Meng, A. Wiesel, and A. O. Hero III, "Distributed principal component analysis on networks via directed graphical models," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 2877–2880.

[128] L. Huang, X. Nguyen, M. Garofalakis, M. I. Jordan, A. Joseph, and N. Taft, "In-network PCA and anomaly detection," in *Advances in Neural Information Processing Systems*, 2006, pp. 617–624.

[129] A. Wiesel and A. O. Hero, "Decomposable principal component analysis," *IEEE Transactions on Signal Processing*, vol. 57, no. 11, pp. 4369–4377, 2009.

[130] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2508–2530, 2006.

[131] M. Gastpar, P. L. Dragotti, and M. Vetterli, "The distributed Karhunen–Loève transform," *IEEE Transactions on Information Theory*, vol. 52, no. 12, pp. 5177–5196, 2006.

[132] A. G. Dimakis, S. Kar, J. M. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1847–1864, 2010.

[133] L. Li, A. Scaglione, and J. H. Manton, "Distributed principal subspace estimation in wireless sensor networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 4, pp. 725–738, 2011.

[134] A. Björck and G. H. Golub, "Numerical methods for computing angles between linear subspaces," *Mathematics of computation*, vol. 27, no. 123, pp. 579–594, 1973.

[135] CAIDA, "The CAIDA Anonymized Internet Traces 2013." [Online]. Available: http://www.caida.org/data/passive/passive_2013_dataset.xml

Vita

Tingshan Huang was born in Nanchang, China. She graduated from Shanghai Jiao Tong University (Shanghai, China) in 2009 with Bachelor of Science degree in Information Engineering and received her Master of Science degree in Electrical and Computer Engineering from Drexel University in June 2012. She joined the Department of Electrical and Computer Engineering at Drexel University as a graduate student in September 2009, and has been under the supervision of Drs. Nagarajan Kandasamy and Harish Sethu since July 2010. Her current research interest spans from adaptive sampling, data compression and bottleneck analysis for performance monitoring, to pattern modeling and anomaly detection in networked systems. She is a recipient of Dean's Fellowship and Travel Subsidy Award of Drexel University. During her years at Drexel, Tingshan has served as a teaching assistant for a variety of courses in Electrical and Computer Engineering department at both graduate and undergraduate levels. She is a student member of IEEE and ACM.