Detecting Incipient Faults in Software Systems: A Compressed Sampling-based Approach

Salvador DeCelles*, Tingshan Huang†, Matthew C. Stamm*, Nagarajan Kandasamy*
*Electrical and Computer Engineering Department, Drexel University, Philadelphia, PA
{sid28, mstamm, kandasamy}@drexel.edu
†Akamai Technologies, Cambridge, MA
tingshan.huang@akamai.com

***Abstract*—The volume of data to be collected and processed for effective real-time monitoring of large-scale computing systems and networks poses significant Big Data challenges, and a scalable solution requires a systematic approach to dimensionality reduction during the data collection, transmission, and analysis phases. Compressive sampling can reduce the dimensionality of the data collected at the source prior to transmission to the monitoring station. Exploiting the fact that the compressed samples preserve in approximate form, the correlation information between data points in the original full-length signal, we develop a low-cost anomaly detection technique based on principal component analysis (PCA) aimed at incipient faults such as software aging—the key idea being PCA is performed directly on the compressed samples without having to reconstruct the original signal. Using case studies involving long-running enterprise benchmark applications, Trade6 and RuBBoS, with injected memory leaks, we show that the performance of the PCA-based detector when using just the compressed data is almost equivalent to the case in which the raw data is completely available, but achieved using significantly fewer samples with a compression rate exceeding 75%.**

***Index Terms*—Online monitoring, anomaly detection, principal component analysis, compressive sampling**

I. Introduction

Online performance monitoring of computing systems and network infrastructure is vital to ensuring efficient operation [1], [2]. The monitored information has a variety of uses including: (i) anomaly detection, diagnosis, and mitigation; (ii) browsing back through historical data to help support cyber-incident event reconstruction and analysis; and (iii) providing real-time detection of anomalous behavior for operators who monitor communication networks for malware or denial-of-service attacks [3]. This paper considers a setting wherein software-based sensors embedded within the IT infrastructure measure performance-related parameters associated with server and network performance.[1] Measurements can include high-level metrics such as response time and throughput as well as low-level metrics such as processor utilization, disk I/O, memory, and network activity. Online monitoring of such a system incurs various costs:

- The very act of monitoring a system 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.
- The information collected by the sensors is typically transmitted over the network to a central monitoring station for analysis and visualization—especially to detect anomalies that become apparent only upon an examination of correlations in the data acquired from multiple locations. Transmitting large amounts of monitored data over a network consumes bandwidth.
- Logging the data for future use such as analysis aimed at capacity planning consumes disk space.

The volume of data that one has to monitor and process for effective real-time monitoring of systems and networks poses significant Big Data challenges in collection, analysis, and storage [4]. Current state-of-the-art performance analysis invariably deals with high-dimensional datasets of increasingly larger size. Thus, it is important to derive a low-dimensional structure as a compact or parsimonious representation of the original dataset—by taking a systematic approach to dimensionality reduction during the data collection and transmission phases, as well as during the analysis phase.

From the viewpoint of reducing the dimensionality of the data collected right at the source—server, router, or end point—before transmitting to the monitoring station, *compressive sampling* (CS) allows us to exploit any inherent sparsity in the signal being sampled to reduce the dimensionality of the collected data. The fundamental idea is that a signal can be sparsified, that is, have a concise representation when expressed in the proper basis; this property can be 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. In previous work, we have shown that various signals from processor, memory, network, and disk sub-systems can be acquired from servers running enterprise benchmark applications in compressed form, and that the recovered signals can be used to detect, with high confidence, the existence of trends as well as abrupt changes within the original signal [5]. Detection is achieved using a substantially reduced sample size—a 70% reduction when compared to traditional fixed-rate sampling.

[1]The work reported in this paper was performed while the second author, T. Huang, was a student at Drexel University.

2159-6190/16 $31.00 © 2016 IEEE
DOI 10.1109/CLOUD.2016.46

CS offers key advantages during the data collection and transmission phases: (i) rather than tailoring the sensing scheme to the specific signal being measured, a simple signal-independent strategy such as randomized sampling can be used, significantly reducing the intrusion of monitoring on application performance; and (ii) since signals are acquired directly in compressed form, the network bandwidth required to transmit these few samples is reduced and so is the space required to store them.

When operators wish to recover the original full-length signal from the sample set at the monitoring station, the process is posed as a linear programming (LP) problem and solved under some sparsity assumptions using a class of reconstruction algorithms called basis pursuit [6]. Though modern solvers are quite efficient at processing large LP problems, the computational burden of recovering and analyzing a large number of signals from multiple servers at the monitoring station may become substantial. In recent work, Huang *et al.* [7] proved from a theoretical viewpoint that the data samples acquired via compressive sampling preserve, in an approximate form, properties such as mean and variance, as well as spectral properties such as correlation between data points. So, instead of having to reconstruct the original signal—and incurring the associated computational cost—this result points to the feasibility of applying well-known anomaly detection methods such as principal component analysis (PCA) directly on the compressed samples.

This paper builds on the theoretical results reported in Huang *et al.* as well as previous work by DeCelles *et al.* [8] and makes the following contributions:

- An anomaly detection process aimed at incipient faults such as software aging, that is gradual resource exhaustion due to memory leaks and bloat [9]–[11]. Our approach is one of PCA-based multi-variate analysis on features capturing memory performance that are compressively sampled in real time. The key point is that the PCA is performed *directly* on compressed samples.
- We analyze the performance of the detector using case studies involving two long-running enterprise benchmark applications: IBM's Trade Performance Benchmark (also known as Trade6) and RuBBoS. The specific experiments are aimed at detecting memory leaks within these applications. Experimental results show that the detector performance using just the compressed feature data is *almost equivalent* to the case in which the raw data is completely available. Furthermore, this performance is achieved using significantly fewer samples with a compression rate exceeding 75%.

The paper is organized as follows. Section II discusses related work in the area of compressive sampling and PCA-based anomaly detection. Section III describes the experimental testbed and Section IV develops the anomaly detection scheme and its performance is analyzed in Section V. We conclude the paper in Section VI.

II. Related Work

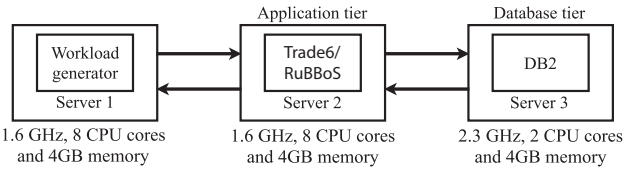
The variance-based subspace method, based on PCA, was first proposed for anomaly detection in [12] and later improved in [13] to explore the deviation in the network-wide traffic volume and feature distributions caused by anomalies. The scheme proposed in [12] applies PCA on training data and separates 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.

Chitradevi *et al.* [14] propose a mechanism to detect anomalies in wireless sensor networks through the use of a PCA-based data model. This model depicting normal behavior of sensor data is generated from principal components of nominal data; similar to our work, it detects anomalies via PCA residual from real-time data. To enhance the PCA model avoiding its sensitivity to outlier values, the model employs a robust estimator in the form of either the minimum volume ellipsoid or the minimum covariance determinant. On the matter of cost, while it keeps a modest computational complexity, it does not make any strides to reduce communication overhead.

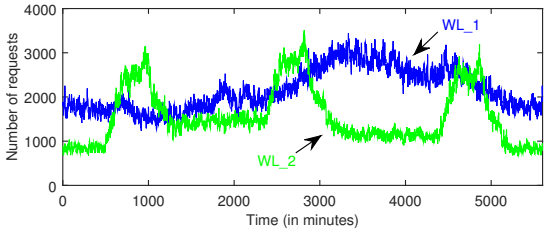
To improve their computational efficiency, PCA-based methods have been decentralized for a variety of purposes including anomaly detection [15]–[19]. A distributed framework for PCA is proposed in [18] to achieve accurate detection of network anomalies through monitoring of only the local data. A distributed implementation of PCA is developed for decomposable Gaussian graphical models in [19] 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 [20], [21].

Kung *et al.* [22] use compressive sampling to reduce the data-collection cost incurred in monitoring MapReduce applications for "stragglers" in a datacenter setting. Zhang *et al.* [23] use it as a compressed storage technique for distributed data analytics. Rather than store the complete data vector, each storage node maintains a compressed snapshot of this vector. The reconstructed data provides approximate, but useful, results for top-K queries, outlier and major mode detection. A key difference is that in our work, anomaly detection is performed using just the compressed samples; reconstructing the original data is not necessary. There have been a few recent attempts focused on anomaly detection using compressed data obtained from the underlying system [24], [25]. These papers also show that the performance of spectral-based methods using only knowledge of the compressed samples is similar to that of knowing the original data.

We build on the theoretical results reported in [7], [24] to develop a low-cost detector aimed at incipient faults in software systems operating in a cloud-based setting. Note that Our previous work reported in [8] also focused on detecting



(a) The testbed comprising Trade6 and RuBBoS applications.



(b) Dynamic workload trace provided to the applications.

Fig. 1. The system hosting the enterprise applications, and examples of workload traces, labeled WL_1 and WL_2, provided to the testbed in our experiments. Incoming requests are plotted in granularity of 30 seconds.

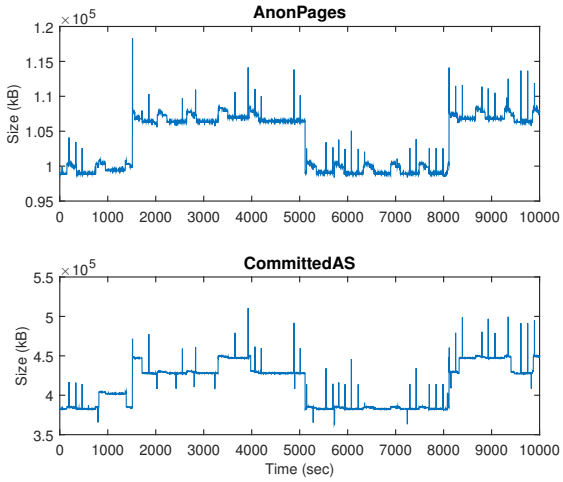


Fig. 2. Measurements corresponding to the AnonPages and CommittedAS signals collected over a 24-hour operating period.

these types of faults using a combination of entropy and PCA—the key difference being the use of the full-length data set rather than the compressed samples.

III. Testbed Description

Figure 1a shows the system setup 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 system hosts the following applications:

- 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 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. It maintains 500 user accounts and information for 3500 stocks.
- A servlet implementation of RuBBoS, a bulletin-board benchmark similar to Slashdot with the capability to browse for, and post messages. We host RuBBoS on an Apache Tomcat application server with DB2 as the database component.

We use httperf [26], an open-loop workload generator, to send a 50% mix of buy/browse and browse/post transactions to the Trade6 and RuBBoS applications, respectively, over a period of 24 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. Example workload traces are shown in Fig. 1b where each data point represents the aggregated workload in a 30-second interval.

The anomaly detector uses the following features contained within the `/proc` pseudo file system at the application tier, specifically the contents of `/proc/meminfo` that report real-time information about memory usage in Linux systems:

- *MemFree*. This quantity reflects the amount of physical memory left unused in the system.
- *CommittedAS*. This quantity 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.
- *PageTables*. This quantity reflects the amount of memory dedicated to the lowest level of page tables.
- *AnonPages*. This quantity tracks the amount of anonymous or non-file backed pages mapped to page tables responsible for the user space.

The above-listed features were chosen to support the case study involving the detection of memory leaks. Here we have chosen low-level metrics that are most likely impacted by this fault. Figure 2 plots two of the features collected during an experimental run of the system lasting 24 hours. The data points are sampled once every two seconds.

IV. Anomaly Detection Using Compressed Samples

Figure 3 shows the flow of the proposed anomaly detection method which can be broken down into two distinct phases: (i) compressed sampling of features at the local server; and (ii) data analysis at the monitoring station to obtain the error residuals. These are discussed in greater detail in this section.

A. Data Sampling and Transmission

In the data preparation phase, we isolate potential anomalous information from the raw feature data before reducing its size for transfer efficiency. More specifically, first from the observed feature data, we construct a model of generally normal behavior, implemented through the use of a simple threshold-based filter, which compares the differences between sequential observations. If the difference exceeds a set threshold, it passes; otherwise, the value is set to zero. Our rationale

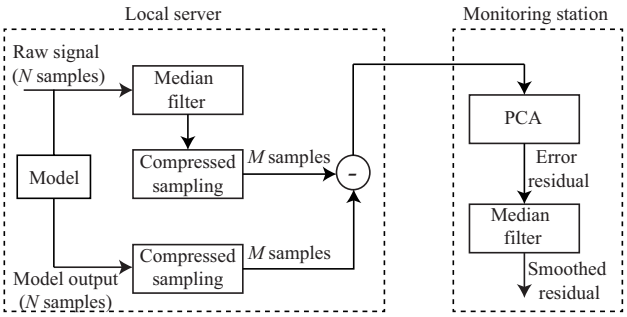


Fig. 3. N data samples corresponding to each feature of interest are compressively sampled to generate M samples for transmission. Here $M << N$.

is that under normal circumstances we should not observe small trickling changes in the data. As for the threshold, we select a value approximately equal to one standard deviation of the respective feature data taken in a fault-free environment. Next we take the difference between the observed data and our model, separating out potential anomalies. Regarding the overall approach, one might ask why it is necessary to model the feature data. Our goal in employing the modeling process is to eliminate the periodic behavior of the features so that we may directly analyze the prediction residuals alone. Our model is designed to isolate the underlying structure of our sampled features. As such, by taking the difference between our actual and modeled features, we may extract just the underlying prediction errors for analysis.

The following brief discussion familiarizes the reader with the basic concepts underlying compressive sampling. We refer the reader to [27] for a more detailed (and gentle) introduction. The fundamental premise behind compression is that a signal can be sparsified, that is, have a concise representation when expressed in the proper basis; this sparsity determines the quality of the subsequent reconstruction. Denote the data to be sampled as **d**, a vector of length *N*, and its representation in basis **B** as **x**. In other words, $\mathbf{d} = \sum_{i=1}^{N} x_i \mathbf{b}_i = \mathbf{B}\mathbf{x}$, where $\mathbf{b}_i$ denotes the i^{th} row in the $N \times N$ matrix **B**. For example, if **B** is selected to be the Haar wavelet basis, the elements of the vector **x** represent the coefficients obtained after wavelet decomposition of the signal **d**. If the chosen basis **B** concisely represents **d**, then a large number of the coefficient entries in **x** will be zero or close to zero and can be ignored when reconstructing the data. More precisely, if at most *S* entries in **x** are nonzero, then **x** is called an *S*-sparse vector; if *S* is small, **d** is said to be sparsely represented in the basis **B**.

Previously, we have shown that various members of the Daubechies wavelet family can capture abrupt changes as well as trends in various signals measured from computing systems [5]. The signals were measured from a server executing enterprise applications and included CPU utilization, disk sectors read/written, network packets sent/received, and memory and page table allocation. Moreover, the bases in the wavelet family were able to sparsify the signal substantially while maintaining good reconstruction quality: less than 2% of the coefficients were needed to maintain the relative error between the original and reconstructed signals within 1%.

When the signal can be represented sparsely in an appropriate basis, it can be acquired directly in a compressed form rather than first collecting a number of samples and then compressing them. The sampling strategy is quite simple and is independent of the signal being measured. Prior to sample collection, we generate an $M \times N$ Gaussian random matrix $\mathbf{G}$ as the underlying sampling matrix, where N is the length of the input data and M is the desired number of samples. Elements in the matrix are independently chosen from a standard Gaussian distribution of zero mean and variance $1/M$. Suppose we wish to convert an $N \times 1$ vector $\mathbf{x}$ to an $M \times 1$ vector of samples $\mathbf{y}$. The data is multiplied with $\mathbf{G}$ such that $\mathbf{y} = \mathbf{GD} = \mathbf{GBx} = \mathbf{Ax}$, where $\mathbf{A}$ is a $M \times N$ matrix.

B. Data Analysis

If we choose to reconstruct the original data **d** at the monitoring station, this inverse problem must be solved: given the vector **y** and matrix **A**, find a sparse vector $\tilde{\mathbf{x}}$ of length *N* such that $\mathbf{y} = \mathbf{A}\tilde{\mathbf{x}}$; i.e., we are looking for $\tilde{\mathbf{x}}$ as a solution to

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

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 under-constrained since the matrix A has more columns than rows; there are infinitely many candidate signals $\mathbf{b}$ for which $A\mathbf{b} = \mathbf{y}$. To solve this under-determined system, the constraint of sparsity is added, allowing only solutions which have a small number of nonzero coefficients. If there is a unique sparse solution, then the compressive sampling framework allows for the recovery of that solution using basis pursuit strategies such as hard thresholding pursuit.

Our method forgoes the above-described reconstruction process and applies PCA directly on the compressed samples. The key result that allows for this approach is explained later in the section. 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 [28]. Given K features, one selects $k \ll K$ most significant principal components out of K to define a k-dimensional subspace based on observations of normal data patterns. An underlying assumption in this approach is that a very small number of principal components capture most of the variance in the data. As a result, this approach typically chooses k as the number of principal components which capture a pre-defined percentage (say, 99%) of the variance in the normal data. Then, a significant deviation in the projection of the K-dimensional observed data onto this k-dimensional reference (normal) subspace can be defined as an anomaly for purposes of detection [12]. This approach is referred to as the variance-based subspace method.

Recall that for every N data points associated with each feature, M samples are generated via compressive sampling. For a system with K features, the data in the K-dimensional space is "mean centered" by subtracting the mean from

individual data points—a necessary step to ensure that the first principal component describes the direction of maximum variance and not to the mean of the data. The eigenvectors of the covariance matrix corresponding to the mean-centered data are then calculated, defining the pattern of data dispersion across the dimensions; of these eigenvectors, the one with the highest eigenvalue would be the principal component which best defines the data trend. Applying PCA on the sample sets gives us K principal components $\mathbf{p}_1, \cdots, \mathbf{p}_K$, each a length-$M$ vector. The $K \times K$ matrix of eigenvectors $\mathbf{V}$ serves to map the data to a new coordinate system as per $\hat{\mathbf{D}} = \mathbf{V}(\mathbf{D} - \mu(\mathbf{D}))$, such that the greatest variance obtained by any projection of the data lies on the first coordinate $\mathbf{p}_1$, the second greatest variance on $\mathbf{p}_2$, and so on. Here, $\mathbf{D}$ denotes the original $K \times M$ data set, $\mu(\mathbf{D})$ its mean, and $\hat{\mathbf{D}}$ its transformation. Moreover, since the above equation describes a linear transformation, the original data can be restored or reconstructed via the inverse operation $\tilde{\mathbf{D}} = \mathbf{V}^T\hat{\mathbf{D}} + \mu(\mathbf{D})$, where $\tilde{\mathbf{D}}$ is the restored data and $\mathbf{V} = \mathbf{V}^{-T}$. Now, an incomplete matrix $\mathbf{V}$ in which one or more eigenvectors are omitted can also be plugged into the above equations to obtain $\tilde{\mathbf{D}}$. This restored data will differ from the original data in that the influence of the component corresponding to the omitted eigenvectors will be completely neutralized. The error or the projection residual between the original and reconstructed signals is computed as $\epsilon = \sum_{i=1}^{M} ||d_i - \tilde{d}_i||$, where d_i and $\hat{d}_i$ denote the i^{th} data item within the original and reconstruction signal, respectively. During each iteration of the analysis step, we measure the residual from an M-sized block received per feature from the server side. With the anomalies here manifesting as rises in the data, we arrive at our detector signal by removing outliers and smoothing our results via median filter.

The PCA-based detection described above is based on the fact that the projection residual as a result of our detection process is sufficiently similar to the residual obtained using the full-length data to allow the use of ϵ for anomaly detection. More specifically, we find that when the test data is normal, key statistical properties of the distribution of ϵ, the projection residual as a result of applying PCA directly on the compressed samples, is related to those of the distribution of the projection residual of the original full-length data [7].

Finally, we would like to make the case for multi-variate analysis using PCA for detecting incipient faults. Figure 4 shows the difficulty of detecting faults by simply analyzing the behavior of the raw low-level metrics. The plot focuses on the metrics collected from the RuBBoS application between 8 and 16 hours of a 48-hour execution run; a 100 KB/min leak is injected around the 12-hour mark. The PageTables signal shows no discernable difference whereas AnonPages and CommittedAS trend upwards ever so slightly. Note also that these signals exhibit a periodic or seasonal pattern which makes detecting faults using simple trend detection even more difficult. Therefore, we believe that a multivariate analysis technique that focuses on correlations between these metrics, such as the one presented in this paper, is necessary for effective detection of small, incipient faults. PCA can

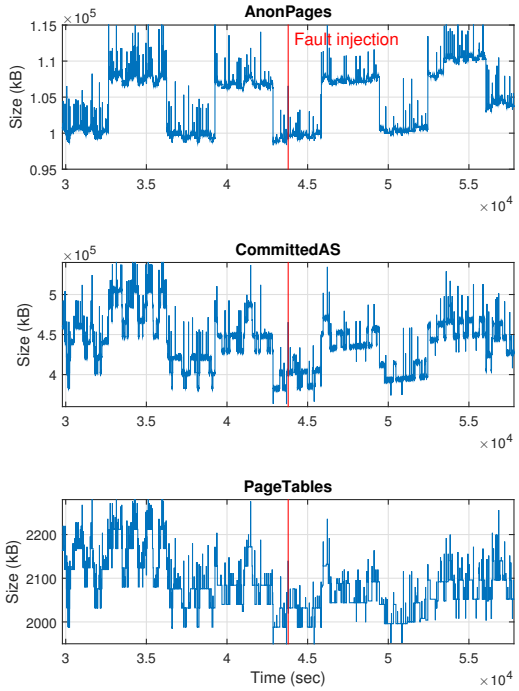


Fig. 4. Effect of the memory leak on the features of interest for the RuBBoS application. Plots show the full-length signals corresponding to each feature.

detect underlying patterns and correlations present in the M-dimensional data set of compressed values. Here, the hypothesis is that under nominal operating conditions, these values remain correlated and stable, whereas as the fault starts to manifest itself, they become less correlated and more erratic. PCA is able to detect this (subtle) distortion in correlation.

V. Performance Evaluation

We consider long-running instances of the Trade6 and RuBBoS applications over a 24-hour period, and our experiments are aimed at detecting memory leaks within these applications. We analyze data from Trade6 and RuBBoS under the two workload traces, WL_1 and WL_2, shown previously in Fig. 1b. All experiments described in this section were conducted with the following settings. In the preparation phase the raw feature data undergoes median filtering for denoising purposes. This filter is set with a width of 5 units. In the analysis phase the PCA residual, ϵ, is passed through a median filter to clean and smooth the result for interpretation. For these experiments, the post-process median filter was set with a width of 30 units. In the course of experimentation, we tested various compression rates, ranging from 50%, that is 1/2 the base sampling rate, to 98.4%, 1/64 the sampling rate. Additionally, we experimented with three distinct block sizes as inputs to the compressive sampling block: $N = 128$, $N = 256$, and $N = 512$.

A. Baseline Comparison

As a point of reference, consider residuals generated under three different approaches. Figure 5 shows our baseline case;

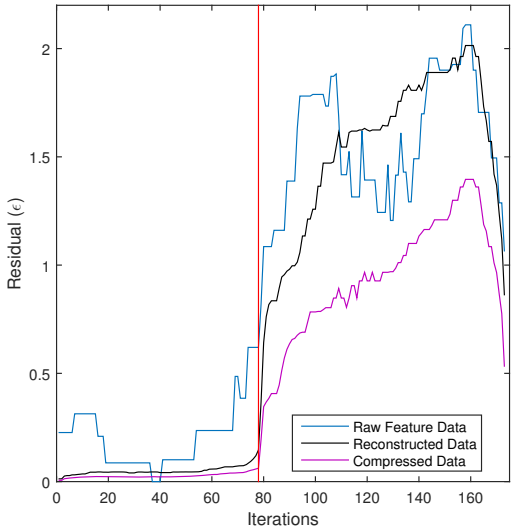


Fig. 5. Projection residuals corresponding to analysis of the raw features, reconstructed features and compressed features respectively using a Trade6 application. Effective fault-injection point is indicated.

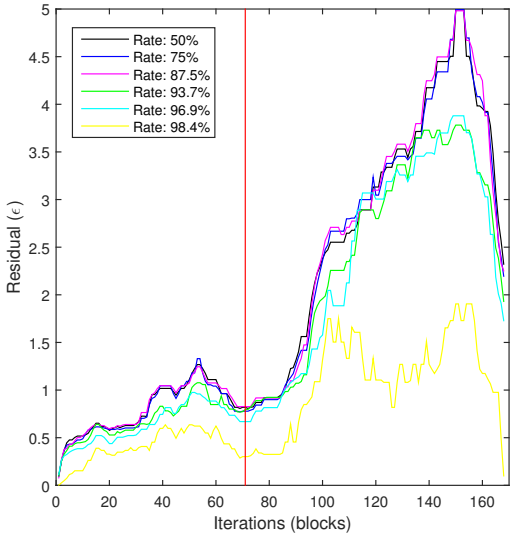


Fig. 6. Effect of varying compression rate on the residual corresponding to workload WL_1 on RuBBoS. Block size is fixed at 256 units. Fault injection point is indicated.

this residual results from PCA being directly applied to the raw feature data. Also present is the middle-ground case; here the residual is generated via an approach through which the feature data is condensed using compressive sampling then reconstructed prior to PCA. Finally there is the approach discussed in this paper; for this residual, PCA is applied to the compressed data without reconstruction. Due to the 75% compression rate applied to the data and consequent loss of information, this residual magnitude is a bit diminished in comparison to the other two. In all three cases, the residual forms a clear crest indicative of a fault's presence. With a properly set threshold, one can detect anomalies via any of these approaches without much difficulty.

These approaches differ when it comes to overall costs. In the baseline case, we analyze the undisturbed feature data; this is ideal for spotting any or all presence of faults, but also requires the full cost of transmitting feature data from the local server to the monitor. On the other hand, the middle-ground case mitigates this transmission cost by reducing the data to be transmitted then reconstructing it on the other side. On the matter of detection, this may result in some loss of information; though sufficient critical data should remain. Unfortunately, depending on method used, the reconstruction process could prove to be costly to the system. This brings us to the focal approach of our paper. By applying PCA directly on the compressively sampled data, which retains the fault behavior information, we remove reconstruction from our overall system. As such, this approach avoids both a high cost of transmission and the full cost of reconstruction, while maintaining sufficient detectability.

B. Sensitivity to Compression Rate

In the transmission phase, we reduce feature data through compressive sampling. Greater reductions of data in this manner, of course, directly lead to fewer necessary transmissions. This improved transmission efficiency is ideal for practical use and as such, we would aim to maximize compression rate whenever possible.

A potential limiting factor in our choice of compression rate would be the ease of detection of possible faults from the reduced data. Would fault indicators in the feature data remain as detectable no matter how much compression is applied? One would expect eventual loss as the rate increases. By the very nature of compression, not all information can be maintained; generally, this would imply that at greater compression rates, there is an increased likelihood of key data being lost within the converted form. Now one may argue that due to the nature of degradation from software aging, rather than having key data indicating the fault, all data would effectively convey fault presence. This line of thought would imply that compression would have no impact on detectability; however, while true on some level, there is a point at which data loss is so great that it becomes impossible to differentiate fault indicators from noise. Furthermore, PCA is limited by the quantity of analyzed data. A residual cannot be produced when fewer than three data points are available. With this in mind, we analyzed changes in the residual for six different compression rates: 50%, 75%, 87.5%, 93.7%, 96.9% and 98.4%. We tested these rates on performance data taken from systems running workload WL_1 or WL_2 on one of the two applications. In all scenarios, we experimented on a nominal case wherein no fault is present through the full run and a fault-influenced case in which a 100 KB per minute leak is injected at the midpoint of a 24-hour run.

Figure 6 shows the impact of compression rate on detection with a block size of $N = 256$ data points. One can see that generally detection magnitude appears to taper off around the 96.9% and higher compression rates as was expected. With a

few exceptions, the residual seems to maintain its magnitude in levels with 50% and 75% generating the greatest residual, 87.5% and 93.7% yielding a lesser residual and so forth. In the exceptional cases, the residual for rates such as 87.5% and 93.7% vary in a way with some larger and some weaker magnitudes. One should note that the nominal case residual in RuBBoS under WL_1 reaches a height close to if not surpassing cases in which a fault is present. This certainly could pose a problem for designing a proper detector as depending on choice of threshold, either false alarms or misses in detection may result.

C. Sensitivity to Sample-Block Size

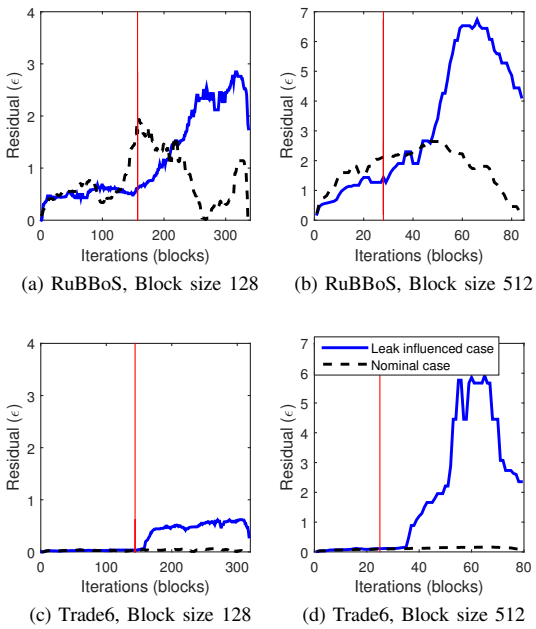
We select two block sizes for comparison, $N = 128$ and $N = 512$ data points. One should note that changes in block size will directly effect the time interval between samples of the resulting residual. This simply follows from the fact that each block encompasses time-series data sampled at a uniform rate; a block of half the size would naturally represent a period of half the duration.

When block size is reduced to 128 data points, there is overall no significant difference in the potential for detection between these results and that from the 256 block size; however, there is a general dip in magnitude of the residual. For the most part, all potential false alarms and misses present in the 256 block size residuals are retained in the 128 unit variation. Due to changes in time-scale and post-process filtering, one may speculate that potential latencies would be reduced in these cases. Figure 7 shows residuals for a block size of 512 units. Again, in general behavior there is not much difference in this from the 128 and 256 variants. In terms of magnitude, there is an overall increase in the residuals; moreover, this rise in magnitude is much more profound in the leak-indicative crests than the static rises under nominal conditions (as seen in Figs. 7a and 7b). This change in the effective range between nominal and fault-indicative peak allows for better overall detection resulting in fewer potential misses and false alarms with the larger block size. In trade-off, latency to detection might be greater for these cases.

D. Detector Performance

We now analyze performance of the detection scheme for the previously discussed settings in terms of latency of detection (shown as number of analyzed blocks) as well as false alarms and misses. To this end, we test the detector with an appropriate range of threshold values. We measure latency for each of the tested block sizes; however for purposes of being concise, we only consider a single compression rate (75%), as we established a general trend of diminishing detection with higher compressions.

Figures 8 through 10 show latency times in addition to indicating whether or not the particular experiment resulted in a false alarm or missed detection. Different block sizes correspond to different effective time durations in terms of the analysis latency. In the majority of tested cases with block sizes of 128 and 256 (see Figs. 8 and 9), the residual used



(a) RuBBoS, Block size 128 (b) RuBBoS, Block size 512

(c) Trade6, Block size 128 (d) Trade6, Block size 512

Fig. 7. Residual used for detection under workload WL_1. Both fault indicative and nominal cases shown. Fault injection is indicated.

	Trade6		RuBBoS	
Threshold	WL_1	WL_2	WL_1	WL_2
0.25	16	41	FA	FA
0.5	39	157	FA	20
1.0	miss	miss	FA	59
1.5	miss	miss	FA	miss
2.0	miss	miss	81	miss

Fig. 8. Detection latency in blocks (4 min) as a function of threshold. Block size and compression rate are set to 128 units and 75% respectively.

for detection does not exceed a magnitude of 1 or 2 (in the respective cases of 128 and 256 block sizes) resulting in misses for higher thresholds. This becomes particularly problematic when one considers the fact that the WL_1 workload using RuBBoS under the same settings can generate residual of 2 or higher magnitude under nominal conditions. As such, for these block settings, regardless of chosen threshold, there may be a likely occurrence of false alarm or missed detection. This problem is, of course, mitigated with the selection of a higher block size, such as 512. With this block size (see Fig. 10), the analysis at each iteration sufficiently indicates the presence of the fault in terms of residual magnitude for each case (not simply the RuBBoS WL_1 case). As such, we see from the figure a clear range around which we may set our threshold to minimize the chance of false alarm and missed detection. Note that given the direct impact that compression rate has on all residual magnitude, proper threshold range becomes a function of compression rate. In our experiment, we see that for a rate of 75%, the ideal threshold range would be between a residual magnitude of 3 and 4. For a block size of 512, this range would yield a latency of around approximately 21 and 35 blocks, in which a block would correspond to 17 minutes of collected data.

	Trade6		RuBBoS	
Threshold	WL_1	WL_2	WL_1	WL_2
0.5	17	22	FA	FA
1.0	31	61	FA	21
1.5	77	miss	FA	35
2.0	miss	miss	FA	78
2.5	miss	miss	31	miss

Fig. 9. Detection latency in blocks (8.5 min) as a function of threshold. Block size and compression rate are set to 256 units and 75% respectively.

	Trade6		RuBBoS	
Threshold	WL_1	WL_2	WL_1	WL_2
0.5	11	14	FA	FA
2.0	25	25	FA	22
3.0	27	30	21	25
3.5	28	34	22	26
4.5	28	miss	26	30

Fig. 10. Detection latency in blocks (17 min) as a function of threshold. Block size and compression rate are set to $N = 512$ units and 75%, respectively.

E. Computational Complexity

Our approach performs compressive sampling on both the feature data and its model. This process can be reduced to a simple matrix multiplication converting N values into M for each of K features, resulting in a cost of $\mathcal{O}(NMK)$. Additionally, we compute a residual via PCA on a set of compressed data blocks originating from our features. With M data points per block and K features, we have a cost of $\mathcal{O}(MK^2 + K^3)$ based on covariance matrix computation and eigenvalue decomposition. The remaining operations in our approach, including median filtering and generating the model output, have a cost of $\mathcal{O}(n)$ making their impact on the whole negligible. As such, the overall complexity becomes $\mathcal{O}(NMK + MK^2 + K^3)$. This cost is fairly minimal when one considers the fact that the variable of largest growth, the feature count, is generally kept small in these experiments.

VI. Conclusions

Building on theoretical results that compressively sampled data preserves in approximate form, the correlation information between data points in the original full-length signal, we have developed an anomaly detection technique based on PCA aimed at incipient faults such as software aging. Using case studies involving the Trade6 and RuBBoS benchmarks, we showed that the PCA-based detector performance using just the compressed data is almost equivalent to the case in which the raw data is completely available, but achieved using significantly fewer samples with a compression rate exceeding 75%. Using higher compression rates we gain significant reduction in communication overhead at the cost of detection accuracy. We have also shown that a stable detector in the sense of reduced false alarms and misses can be achieved by appropriately tuning the block size and threshold parameters.

References

[1] L. Cherkasova *et al.*, "Automated anomaly detection and performance modeling of enterprise applications," *ACM Trans. Comput. Syst.*, vol. 27, no. 3, pp. 1–32, 2009.

[2] M. Kutare *et al.*, "Monalytics: Online monitoring and analytics for managing large scale data centers," *Proc. IEEE/ACM Conf. Auton. Comput. (ICAC)*, pp. 141–150, 2010.

[3] 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.

[4] T.-F. Yen *et al.*, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proceedings of the 29th Annual Computer Security Applications Conference*, 2013, pp. 199–208.

[5] T. Huang, N. Kandasamy, and H. Sethu, "Evaluating compressive sampling strategies for performance monitoring of data centers," in *Proc. ACM 9th Int'l conf. Autonomic computing*, 2012, pp. 201–210.

[6] S. Foucart, "Hard thresholding pursuit: An algorithm for compressive sensing," *SIAM J. Numer. Anal.*, vol. 49, no. 6, pp. 2543–2563, 2011.

[7] T. Huang, N. Kandasamy, and H. Sethu, "Anomaly detection in computer systems using compressed measurements," in *Proc. IEEE Symp. Software Reliability Engineering (ISSRE)*, November 2015.

[8] S. DeCelles and N. Kandasamy, "Entropy-based detection of incipient faults in software systems," in *Proc. 18th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, November 2012.

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

[10] A. Avritzer *et al.*, "Performance assurance via software rejuvenation: Monitoring, statistics and algorithms," in *Proc. IEEE Conf. Dependable Syst. Netw. (DSN)*, 2006, pp. 435–444.

[11] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *IEEE Trans. Dependable Secur. Comput.*, vol. 2, no. 2, pp. 124–137, April 2005.

[12] 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.

[13] ——, "Mining anomalies using traffic feature distributions," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 217–228, Aug. 2005.

[14] N. Chitradevi, K. Baskaran, V. Palanisamy, and D. Aswini, "Designing an efficient pca based data model for wireless sensor networks," in *Proc. 1st Int'l Conf. Wireless Technologies for Humanitarian Relief*, ser. ACWR, 2011, pp. 147–154.

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

[16] 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)*. IEEE, 2012, pp. 2669–2672.

[17] 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)*. IEEE, 2012, pp. 2877–2880.

[18] 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.

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

[20] 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.

[21] 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.

[22] H. T. Kung, C.-K. Lin, and D. Vlah, "Cloudsense: Continuous fine-grain cloud monitoring with compressive sensing," in *USENIX Hot Topics Cloud Computing*, 2011.

[23] J. Zhang *et al.*, "Impression store: Compressive sensing-based storage for big data analytics," in *USENIX Hot Topics Cloud Computing*, 2014.

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

[25] 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.

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

[27] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Proc. Mag.*, vol. 25, no. 2, pp. 21–30, 2008.

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