Efficient Online Performance Monitoring of Computing Systems using Predictive Models

Salvador DeCelles, Matthew C. Stamm, Nagarajan Kandasamy

ECE Department, Drexel University
Philadelphia, PA 19104, USA
Email: {sid28, mstamm, kandasamy}@drexel.edu

***Abstract*—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, we develop and validate a strategy based on exploiting the underlying structure of the signal being monitored to sparsify it prior to transmission to a monitoring station for analysis and logging. Specifically, predictive models are designed to estimate the signals of interest. These models are then used to obtain prediction errors—the error between the signal and the corresponding estimate—that are then treated as a sparse representation of the original signal while retaining key information. This transformation allows for far less data to be transmitted to the monitoring station, at which point the signal is reconstructed by simply using the prediction errors. We show that classical techniques such as principal component analysis (PCA) can be applied to the reconstructed signal for anomaly detection. Experimental results using the Trade6 and RuBBoS benchmarks indicate a significant reduction in overall transmission costs—greater that 95% in some cases—while retaining sufficient detection accuracy.**

***Index Terms*—Online monitoring, anomaly detection, predictive models, principal component analysis.**

I. Introduction

Online monitoring of performance-related metrics—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—is a necessary first step towards detecting anomalies in computing systems. These measurements can help detect performance-related hotspots and bottlenecks as well as incipient faults associated with gradual resource exhaustion—the so-called software aging problem [1]–[3]. The data can also help detect security breaches resulting in the computers being infected by malicious software [4].

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.

With the goal of reducing the above-described costs, we develop and validate a strategy based on exploiting the underlying structure of the signal being monitored to *sparsify* it, that is encode it concisely, prior to transmission to a monitoring station for analysis and logging. Specifically, predictive models are designed to estimate the signals of interest. These models are then used to obtain prediction errors—the error between the signal and the corresponding estimate—that are then treated as a sparse representation of the original signal while retaining key information. This transformation allows for far less data to be transmitted to the monitoring station, at which point the signal is reconstructed by simply using the prediction errors.

The proposed method borrows from the *dual-prediction scheme* used in MPEG encoding of video frames [5].[1] Since a scene typically changes very little over a short period of time, a great deal of redundancy exists between video frames. The MPEG video encoder exploits this redundancy by predicting each video frame from other frames, then storing only the necessary information to predict each frame, along with the prediction error. To reconstruct the video, the decoder creates a prediction of each video frame from previously decoded frames using the same predictor employed by the encoder. Since the encoder and decoder use the same predictive model, their prediction errors will be identical. This allows the decoder to then reconstruct the video frame by adding the prediction error stored by the encoder to the decoder's predicted frame.

Sparsifying the data acquired at a local server prior to transmission to the monitoring station can be performed in one of two ways: the collected features, that is the raw data, can be sparsified, or one can sparsify an appropriate signal that is derived from this raw data—in our case, the entropy of the data. We show that using the entropy of the raw data results in significant transmission savings. The very process of calculating entropy involves summarizing the information

[1] The method is also commonly known as differential pulse code modulation or DPCM.

978-0-7695-5697-0/15 $31.00 © 2015 IEEE
DOI 10.1109/UCC.2015.31

contained within a larger data set by a single value; sparsifying the resulting signal results in further savings. We also show that these entropy signals capture enough information about the raw data to be useful in detecting incipient faults.

We illustrate the advantages of the proposed approach in terms of the reduction in data transmission with respect to the baseline case in which the entirety of the data is transmitted to the monitoring station. Our case studies involve two long-running enterprise benchmark applications, IBM's Trade Performance Benchmark (also known as Trade6) and RuBBoS, and the specific experiments are aimed at detecting memory leaks within these applications. We are able to show that the sparsified signals can be recovered at the monitoring station with sufficient fidelity for PCA-based methods to be successfully used for anomaly detection. Current state-of-the-art performance analysis techniques invariably deal with high-dimensional datasets of increasingly larger size—thus, it is important to derive a low-dimensional structure as a compact representation of the original dataset. Principal component analysis (PCA) allows us to examine the linear relationship between features of interest and derive a reduced set of unrelated features that are linear combinations of the original features [6]. The high-dimensional dataset is transformed into new bases called *principal components* that are ordered by the strength of the correlations exhibited by the data along their respective directions.

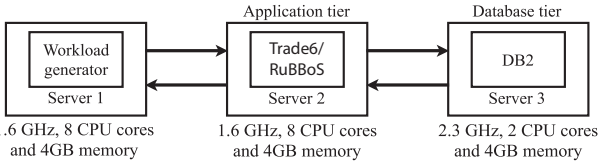
We quantify our results in terms of transmission savings and detection latency. Through the sparsification process using proper threshold settings, we achieve transmission savings of more than 95% when compared to the baseline case for both Trade6 and RuBBoS. Regarding the detection of anomalies, we demonstrate that with appropriately tuned parameters, we can minimize the occurrence of false alarms and misses. The approach is capable of successfully detecting the presence of a fault in under thirty minutes for both applications used in the case study.

The paper is organized as follows. Section II discusses the experimental testbed. Section III describes the sparsification process and Section IV discusses the application of PCA-based anomaly detection on the reconstructed data. Section V presents the case studies. Section VI discusses related work and Section VII provides some concluding remarks.

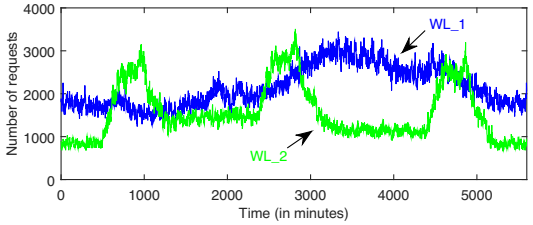
II. Testbed Description

Fig. 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 operating system on the virtual machine (VM) is the SUSE Enterprise Linux Server Edition. 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 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.
- 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.



(a) The testbed comprising Trade6 and RuBBoS applications.



(b) Dynamic workload trace provided to the applications.

Fig. 1. The system hosting the Trade6 and RuBBoS 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.

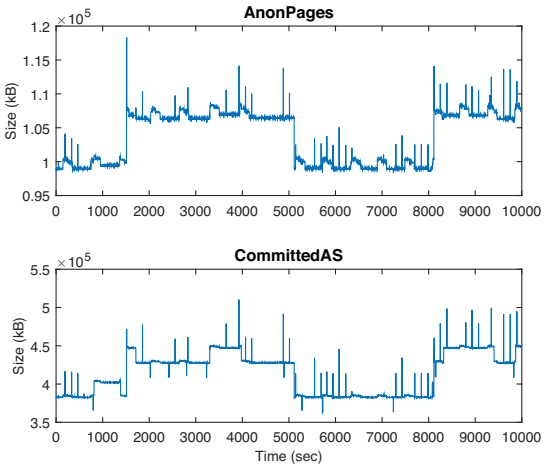


Fig. 2. Measurements corresponding to the AnonPages and CommittedAS signals collected over a 24-hour operating period.

We use httperf [7], an open-loop workload generator, to send a 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

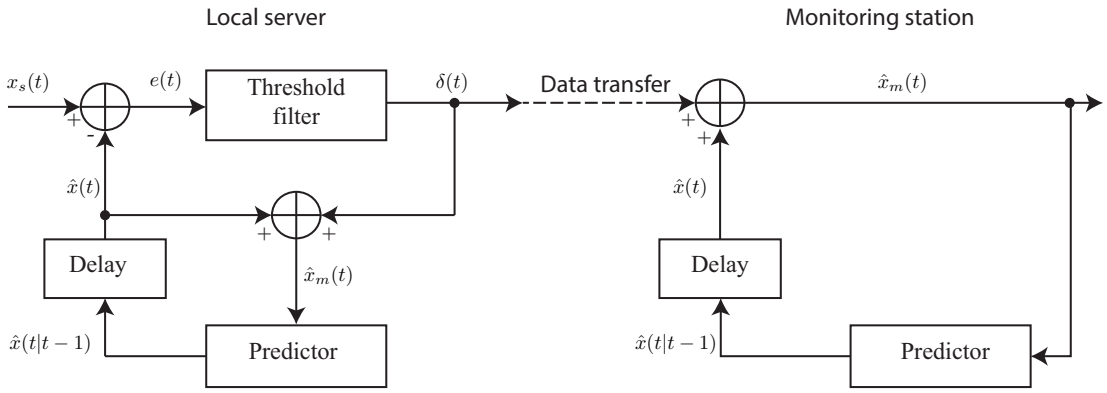


Fig. 3. The dual-prediction scheme used to sparsify the signal.

within short time periods. Some Sample workload traces are shown in Fig. 1b. Each data point in the figure represents the aggregated workload in a 30-second interval.

Our experiments use the following metrics 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. 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.
- *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.

III. Sparsifying Signals via Dual Prediction

We describe the spasification process, starting with acquiring the signal of interest at the local server to its reconstruction at the monitoring station. Figure 3 illustrates this process in its entirety, and we use the AnonPages signal in Fig. 2 as the running example to explain the theory behind it.

We periodically sample the noise influenced signal $y(t)$ on the server side as

$$y(t) = x(t) + n(t),$$

where $x(t)$ is the true measurement and $n(t)$ is the noise process corrupting it. The observed signal is then passed through a denoising filter to produce a close estimate of our signal, shown as $x_s(t)$ in Fig. 3. Simultaneously, we predict the signal's value at time t, $\hat{x}(t)$, via a feedback loop as a function of the k previous values as

$$\hat{x}(t|t-1) = g\big(\hat{x}_m(t-1), \ldots, \hat{x}_m(t-k)\big),$$

where g is the predictor function. Here, $\hat{x}(t|t-1)$ denotes the predicted value of $x(t)$ obtained at time $t-1$. We then delay this prediction to the next iteration of the process. Next we compute the prediction error $e(t)$ as the difference between $\hat{x}(t)$ and $x_s(t)$:

$$e(t) = x_s(t) - \hat{x}(t).$$

At the monitoring station, using a prediction loop mirroring that in the server, we calculate $\hat{x}_m(t)$ in parallel with the generation of $x_s(t)$. Finally, to reconstruct the signal at the fusion center, we add to $\hat{x}_m(t)$ the filtered prediction error $\delta(t)$ received from the server:

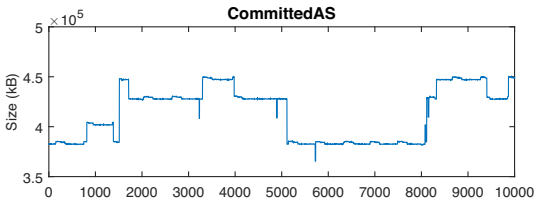
$$\hat{x}_m(t) = \hat{x}(t) + \delta(t).$$

The prediction error, as the deviation between the signal and its corresponding prediction, tends to be rather sparse, that is to say many of its values are zero or close to zero. Furthermore, one may omit values near zero without losing any critical information, depending on the scale of the featured signal. As such, for our purposes, the prediction error is the ideal signal to transfer from the server to the monitoring center. To reduce overall communication, we subject $e(t)$ to a threshold filter, only passing deviations which are sufficiently large, say greater than a threshold value η. The resultant signal $\delta(t)$ is then transmitted to the fusion center so long as the value is not zero.

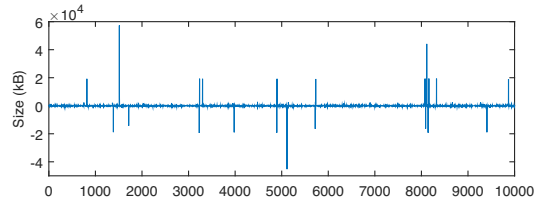
$$\delta(t) = \begin{cases} e(t), & \text{if } e(t) \geq \eta \\ 0, & \text{if } e(t) < \eta \end{cases}$$

During iterations when $\delta(t)$ is zero no information is transmitted; $\delta(t)$ at the monitoring station, synchronized with the iteration time of the server, takes the value of zero by default.

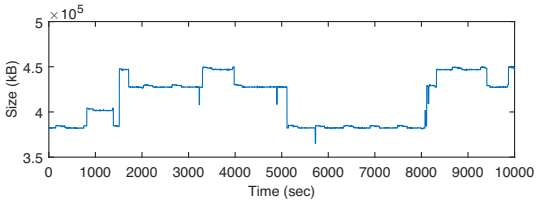
To illustrate the sparsification process, we apply the above discussed technique to the CommittedAS signal, previously



(a) Estimated signal on the local server, $x_s(t)$.



(b) The prediction error, $\delta(t)$.



(c) The reconstructed signal at the monitoring center, $\hat{x}_m(t)$.

Fig. 4. Application of SPARSE_R method on the CommittedAS signal.

shown in Fig. 2, which is sampled once every two seconds from our system. One may notice the presence of data-spikes throughout the signal. To remove these so that we could work with a much cleaner signal, we denoise it using a simple median filter as

$$x_s(t) = \text{med}_\ell\big(y(t+\tfrac{\ell}{2}), \ldots, y(t-\tfrac{\ell}{2})\big),$$

where l is the filter width. The resulting signal is shown in Fig. 4a. Also, common among the features related to the memory subsystem is a behavior where the signal would remain steady for lengthy durations interspersed with brief jumps in value. Given this pattern, we employ a simple predictor which returns the previous value in anticipation that no change has occurred as

$$\hat{x}(t|t-1) = \hat{x}_m(t-1).$$

As such, we effectively reduce the prediction error to the difference between the current and previous signal value.

$$e(t) = x_s(t) - \hat{x}(t) = x_s(t) - \hat{x}_m(t-1)$$

By filtering out the $e(t)$ values that do not exceed the specified threshold, we produce $\Delta(t)$, seen in Fig. 4b, which need only be transmitted at moderate changes of the signal. At the monitoring center, we reconstruct $\hat{x}_m(t)$ with the received $\delta(t)$. With our predictor simply passing the previous value, $\delta(t)$ is effectively the difference between consecutive values. We reconstruct $\hat{x}_m(t)$ as a cumulative sum of $\delta(t)$:

$$\hat{x}_m(t) = \hat{x}(t) + \delta(t) = \hat{x}_m(t-1) + \delta(t)$$

The resulting reconstruction is shown in Fig. 4c.

This paper addresses two variants of the above-described process, termed as SPARSE_R and SPARSE_E. These two methods differ in the point at which the sparsification process is performed in the overall flow. In SPARSE_R, the raw feature data is sparsified and reconstructed—as was just discussed using the CommittedAS signal. In this particular example, with the threshold set at $\eta = 10$, we gain 68.2% in transmission savings compared to the base case in which the entire data is transferred to the monitor.

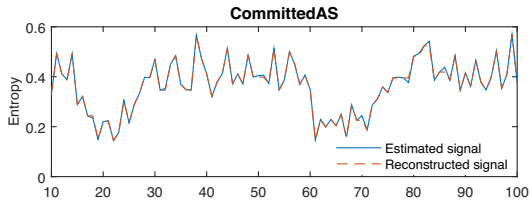
The SPARSE_E method, as a preprocessing step, summarizes the expected value of the information associated with the data set using Shannon entropy. This step reduces the amount of data to be transferred, even prior to the sparsification process, since a larger data set is now reduced a single value. It also reduces the computational burden incurred by the anomaly detection process at the monitor by reducing the amount of data to be processed. A data set comprising discrete values—for example, CommittedAS—can be binned within a histogram which is a graphical representation of the data set's distribution where each bin is weighed by how often a data item falls within it. Using the concept of *entropy* which measures the spread of a distribution, we can further summarize the contents of the histogram to a single scalar value: the amount of information that it represents. Given a histogram comprising k equally-sized bins with values $\{b_0, b_1, \ldots, b_{k-1}\}$, the corresponding entropy is defined as

$$H = -\sum_{i=0}^{k-1} p(b_i) \log p(b_i),$$

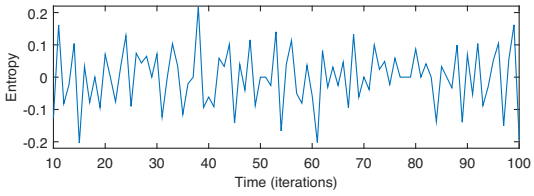
where $p(b_i) = b_i / \sum_{j=0}^{k-1} b_j$. Given the above definition, H lies within 0 and $\log n$, and so the normalized entropy value that lies between 0 and 1 is calculated as $H/\log n$. A low entropy value corresponds to a skewed distribution highly focused on just a few values and a high entropy value corresponds to a uniform distribution with a minimal focus on any bin value.

When applying SPARSE_E to the CommittedAS signal, we use the already compressed entropy data shown in Fig. 5 as the starting point for the sparsification process. In this example, we compute entropy on distinct groups, 312 data points in size. Using this preprocessing step, we accumulate transmission savings of 99.68%. With threshold set at $\eta = 0.02$, we achieve an additional 15.4% in transmission savings by sparsifying this signal for a combined savings of 99.73%. Generally for the signals we test, savings in SPARSE_R far surpass those in SPARSE_E, since the entropy is often changing, leading to more frequent transmission of prediction-error signals; however, for the purposes of transmission, the savings in SPARSE_E stack with the consolidation of entropy rather than compete against it as with the case of SPARSE_R.

Regarding matters of reconstruction, note that $\hat{x}_m(t)$ will always match $x_s(t)$ with a variance based mostly on the chosen transmission threshold. So long as the threshold is kept sufficiently low, the predictor, no matter how accurate or poor, will not cause any great deviation between $\hat{x}_m(t)$ and $x_s(t)$.



(a) Estimated and reconstructed signals, $x_s(t)$ and $\hat{x}_m(t)$.



(b) The prediction error, $\delta(t)$.

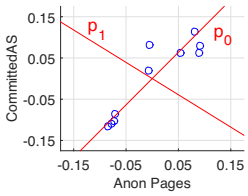
Fig. 5. Application of SPARSE_E on the CommittedAS signal.

This is because the predictors at both the server and monitoring station are identical and process the same data concurrently. As such, the accuracy of the predictor will not impact the overall detection scheme; rather, predictor accuracy directly leads to superior transmission efficiency. Nevertheless, for our purposes, a simple previous-value predictor will suffice. More advanced predictors based on Kalman filters or neural networks can certainly be introduced within the dual-prediction scheme with the aim of further reducing data transfer cost.

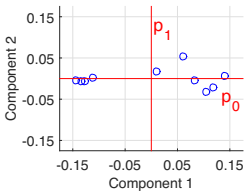
IV. PCA-based Anomaly Detection

Principal component analysis (PCA) is a dimension reduction technique frequently used for anomaly detection [8], [9]. It transforms a high-dimensional dataset into new bases called *principal components* that are 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 [6]. The first few principal components are often chosen as the signature pattern of the data.

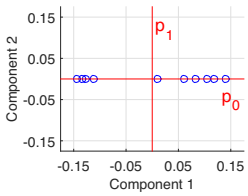
For a system with m features and n data points per feature, we clarify the key steps involved in the PCA process using a simple example in which $m = 2$ and $n = 10$. Figure 6a shows data points corresponding to the CommittedAS and AnonPages signals in a 2-dimensional feature space. The data is "mean 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. Figure 6a also shows the two principal components in which the first principal component p_0 describes the direction of the largest variance in the data set and the second principal component p_1 describes the next



(a) Mean-centered data with the corresponding principal components.



(b) Data projected on to the new bases.



(c) Complete removal of the minor component from the transformed data.

(d) Data restored to its original basis after removal of the second principal component.

Fig. 6. The major steps comprising PCA-based anomaly detection.

largest variance in the orthogonal direction. The $m \times m$ matrix of eigenvectors V serves to map the data to a new coordinate system as per

$$\hat{D} = V(D - \mu(D)),$$

such that the greatest variance obtained by any projection of the data lies on the first coordinate p_0, the second greatest variance on p_1, and so on. Here, D denotes the original $m \times n$ data set, $\mu(D)$ its mean, and $\hat{D}$ its transformation. Figure 6b shows the transformed data within the new coordinate system. Moreover, since the above equation describes a linear transformation, the original data can be restored or reconstructed via the inverse operation

$$\tilde{D} = V^T \hat{D} + \mu(D),$$

where $\tilde{D}$ is the restored data and $V = V^{-T}$. Now, an incomplete matrix V in which one or more eigenvectors are omitted can also be plugged into the above equations to obtain $\tilde{D}$. 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. Figure 6c shows the effect of removing the minor component on the transformed data, and Fig. 6d shows an overlay of the restored data (after mapping back to the original coordinate system and adjusting its mean) along with the original data.

The error between the original and reconstructed signals is computed as

$$\epsilon = \sum_{i=1}^{N} ||d_i - \hat{d}_i||,$$

where d_i and $\hat{d}_i$ denote the i^{th} data item within the original and reconstruction signal, respectively. Our detector, ultimately, is a measurement of slope, calculated as a difference in ϵ over a period of time, that is $d\epsilon/dt$. We employ this slope

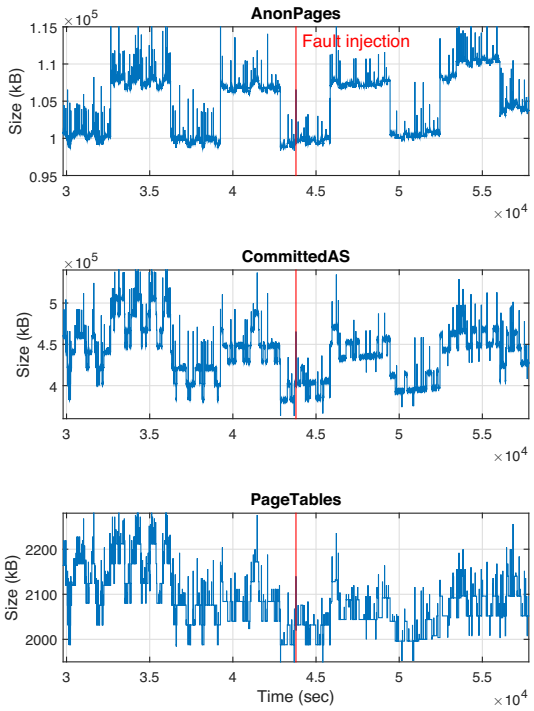


Fig. 7. Effect of the memory leak on the features of interest.

measurement as our detector, since anomalies in our system are often characterized by a rising slope in ϵ.

The overall anomaly detection scheme is iterative and uses a sliding-window based method wherein during each iteration we select consecutively sampled feature data in a structure comprised of n subwindows of size w each. For each subwindow, we compute the entropy of the contained data. With the resultant n entropy values, we generate ϵ and consequently the detector through PCA. Between iterations we slide the n-by-w window along the feature data to select the next batch of values. Once the set is ready, we repeat the process.

V. Performance Evaluation

A detailed case study illustrating the advantages of SPARSE_R and SPARSE_E in terms of reducing data transmission cost is now presented. In the case of SPARSE_R, the raw data is sparsified at the server and transmitted to the monitor where it's reconstructed and the entropy calculated prior to applying PCA; whereas for SPARSE_E, the entropy signal is calculated at the server, sparsified, and sent to the monitor which performs the analysis on the reconstructed data. As such the local server takes upon itself the computational cost of the entropy calculation; however, in exchange, we reduce the number of transmissions by the rate at which entropy compresses the data.

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. Figure 7 shows the impact on the individual features—a slightly discernible upward trend—when a small memory leak

	Trade6			RuBBoS		
η	Anon	Comm	Page	Anon	Comm	Page
0	0.036	0.231	0.004	0.286	0.284	0.013
10	0.017	0.194	0.004	0.254	0.248	0.013
20	0.013	0.188	0.002	0.243	0.246	0.012
30	0.009	0.183	0.002	0.163	0.244	0.011

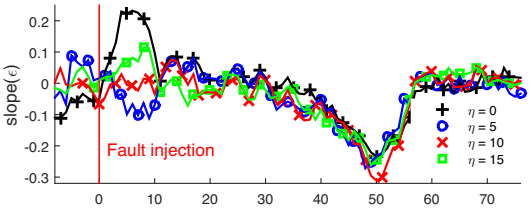
Fig. 8. Average transmission rate per unit sample as a function of η for the signals, Anon(Pages), Comm(ittedAS), and Page(Tables).

or bloat of about 100 KB/minute is injected at around the 12-hour mark. We analyze data from Trade6 and RuBBoS under the two workload traces, WL_1 and WL_2, shown previously in Fig. 1b. For our denoiser, we use a median filter with a filter length of 5 units. For the detector, we compute the slope of ϵ over a period of approximately one hour.

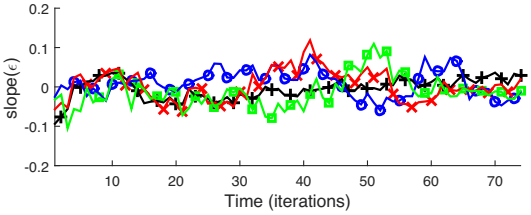
A. Effect of the Transmission Threshold

The threshold η sets the minimum limit in magnitude for transferable prediction-error values. Higher the value, the more restrictive the transmitter becomes, and the smaller the overall cost of transmissions incurred. Generally η serves as a filter that eliminates from the prediction error, minor static variance, which contributes in no way to fault detection, and thus is unnecessary to preserve. This variance differs from feature to feature; as such, η needs to be carefully determined for each feature. Figure 8 shows the cost incurred by SPARSE_R as a rate of data transfer per unit sample for the three features of interest. Transmission rate is calculated as the ratio of transmitted data over total data. As anticipated, overall, higher thresholds result in diminished costs. Efficiency is particularly high for PageTables and to a lesser extent AnonPages. CommittedAS, on the other hand, has transmission cost of 1 every 3 samples, as a result of its scale and consequentially high variance. Overall transmission costs are greater for data associated with RuBBoS over that of Trade6. This is a consequence of features in RuBBoS, which are more sensitive to workloads, being much less uniform and a bit more complex.

While a larger η value achieves greater transmission efficiency, it may also diminish detection quality. In general, a memory leak in the system may manifest as a trend in each of the features, which in the sparsified prediction-error signal is captured as a collection of impulses; and η, if large enough, may impact these impulses, effecting the fault's representation. This, of course, does not completely eliminate all indication of the fault in the data, as the trend remains through the conversion of many small impulses into fewer larger spikes; however, while not destroying evidence of the fault's presence in the data, this change may diminish the detector's overall quality. Figure 9 shows detector performance under both faulty and nominal cases for AnonPages when $w = 300$ and $n = 50$. One can see the overall, gradual deterioration of detection quality as η is made larger. Of particular note is the fact that AnonPages under the Trade6 application can be rather sensitive to this effect such that an increase in η may significantly damage the detector. This can be seen in Fig. 9a, where if $5 < \eta < 10$, this completely negates the fault-indicative crest; likewise under the nominal conditions, $\eta > 10$



(a) Trade6 subjected to workload WL_2 with the fault injected.



(b) Trade6 subjected to workload WL_1 under nominal conditions.

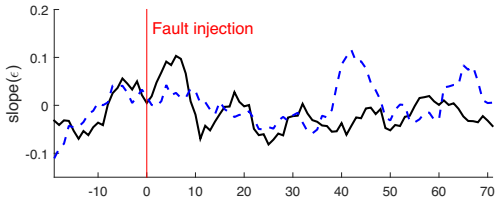
Fig. 9. Effect of varying η on detector performance. The red line indicates the approximate time of fault injection.

results in higher detector variance, increasing the likelihood of false alarms. This generally results from the elimination of certain behavior-defining data, which in AnonPages can be rather small. As such, it is prudent to keep η settings relatively low. For each feature there are regions at which transmission rate drops abruptly, signifying a loss of characterizing data. As a general rule for choosing η, one should set the value at approximately 2σ, where σ refers to the static variance in the respective feature. For the duration of this paper we maintain the η settings at 0, 10, 10, for AnonPages, CommittedAS, and PageTables, respectively.

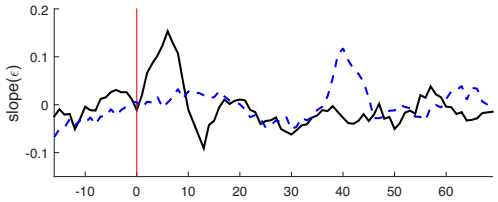
B. Detector Stability and Fault Detection Latency

The subwindow size w is a parameter which determines the number of data points used per entropy-value calculation. As a parameter of the analysis phase which follows transmission, w doesn't impact transmission cost. Figure 10 shows the detector performance with η and n fixed at (0, 10, 10), and 50 respectively for the RuBBoS application. Here we demonstrate a critical point in the setting of w. In this particular case, we lose detectability at lower settings of w (less than 300) in-so-much as a false alarm would overshadow the fault-indicative crest in magnitude. For higher settings of w, the detector generally achieves greater stability, resulting in a more reliable fault-indicative crest while reducing potential false alarms; however, by increasing the number of data-points required for each analysis step, we effectively establish a time-buffer, forcing longer minimum detection-latency periods. Similarly, with so large an evaluation window, we require a longer start-up period.

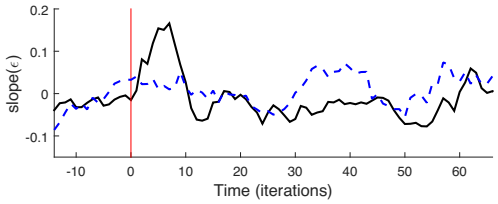
The subwindow count n is a parameter which sets the number of entropy points used for a single PCA calculation and has no bearing on transmission efficiency. Larger values of n would result in better detector stability. The expanded window length will not increase latency by virtue of the growth



(a) Subwindow size, w = 290



(b) Subwindow size, w = 300

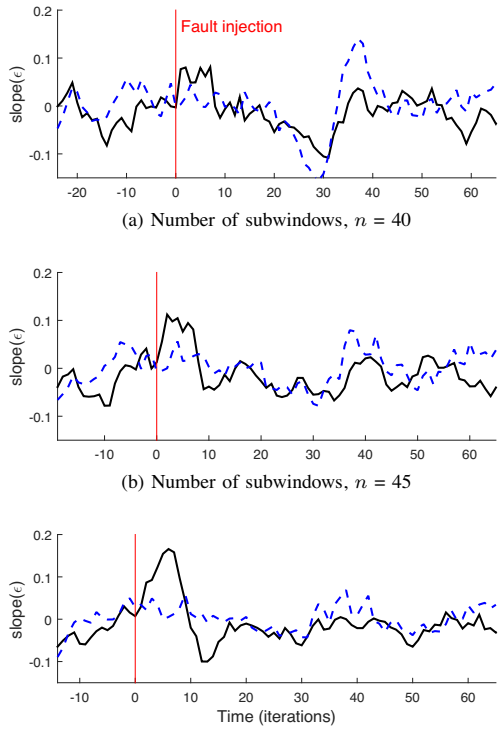


(c) Subwindow size, w = 310

Fig. 10. Effect of varying w on RuBBoS data running WL_1. Solid line depicts the leak-influenced detector; broken line represents the nominal case.

in data points required for the analysis window due to each analyzer step only adding a portion to the full window; however, growth in the total number of points used in PCA does effectively result in weakened influence of each individual point. As such, the presence of a fault may take longer to manifest in the detector, ultimately causing a greater detection latency. Figure 11 shows the change in the detector for varied values of n with transmission threshold and w fixed at (0, 10, 10) and 312 respectively. Like the subwindow size, n has a direct relationship with detector stability. Here we demonstrate a critical point around $n = 45$: below this region, our detector would fail; above this, its detectability would suffice.

Now while increases in either w or n should provide detector stability, growth in the w-by-n analysis window may lead to a major problem in our system: the larger the analysis window, the greater duration of time it encompasses. To generate ϵ, we would require a minimum of nw data samples. With a sampling rate of once every two seconds, at the higher ranges of w and n, the observation window will extend beyond a duration of 10 hours. This means that before we can make any detection, the system must collect 10 or more hours of data. The problem occurs in the scenario where a fault arises during this training period. Due to the nature of our detection via fault-indicative crest at the onset of the anomaly, a fault which manifests in the training period may go unnoticed. Later we will discuss a method to minimize this start-up cost. Nevertheless, it is ideal to keep both w, n and the analysis



(a) Number of subwindows, n = 40

(b) Number of subwindows, n = 45

(c) Number of subwindows, n = 50

Fig. 11. Effect of varying n on RuBBoS data running WL_1. Solid line depicts the leak-influenced detector; broken line represents the nominal case.

window by extension as small as possible while maintaining a sufficient level of detectability. For this reason, we maintain w and n at 312 and 50 respectively.

Detection latency is measured as the duration of time from the moment the fault enters the system to the point at which the detector exceeds a specified detection threshold. The detector used in our system is constructed from the effective slope (calculated as a difference over a set time) of the ϵ signal produced through PCA; as such, when evaluating differing shifts of the analysis window, we accommodate the increase in iterations per period resulting from the transitional entropy calculations while maintaining equivalent periods by applying a difference inversely proportional to the shift count. Figure 12 summarizes the detection latencies for two scenarios with varying detection thresholds including the results when reducing shift. Please note that the remaining cases have also been tested, yielding similar results. Changes in shift do not appear to result in any definitive trends in regard to both stability and latency. Though one might expect shift reduction to result in a decrease in latency since increasing the effective checking rate should result in detection at the same moment or sooner, this is not quite so, as other factors in the detection process may interfere. Overall, detection of the fault is generally quick with most latencies well under an hour.

One should note that in tuning our parameters w and n, we aim to minimize detection latency as well as maximize the feasible threshold range by strengthening the detection crest

Detection threshold	Trade6 under WL_1			RuBBoS under WL_2		
	w	$w/2$	$w/3$	w	$w/2$	$w/3$
0.06	0.347	0.433	0.578	FA	FA	FA
0.10	0.52	0.607	0.636	0.347	0.26	0.289
0.14	0.693	0.693	0.693	0.52	0.347	0.347
0.18	0.693	0.78	0.867	0.52	0.433	0.462

Fig. 12. Latency (in hours) as a function of detection threshold, for shifts of w, $w/2$ and $w/3$ units under SPARSE_R. A false alarm is indicated as FA.

	Trade6 under WL_1			RuBBoS under WL_2		
η	w	$w/2$	$w/3$	w	$w/2$	$w/3$
0	0.993	0.996	0.998	0.993	0.996	0.998
0.02	0.764	0.576	0.478	0.729	0.599	0.501
0.04	0.573	0.366	0.272	0.522	0.372	0.298
0.06	0.494	0.277	0.202	0.395	0.254	0.200
Base	0.003	0.006	0.01	0.003	0.006	0.01

Fig. 13. Average transmission rate per entropy sample for CommittedAS when using SPARSE_E. Also shown is the base reduction achieved by transmitting just the entropy values. We test the system for subwindow shifts of w, $w/2$, and $w/3$, where $w = 312$.

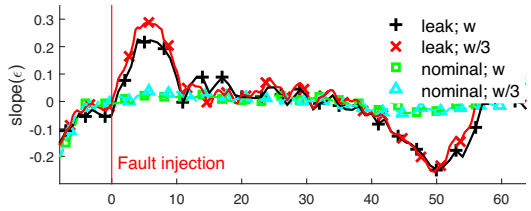
and diminishing any noise-produced rises. In certain scenarios the detection crest may not be particularly large; in these cases in order to prevent missed detection of a fault, one must keep the threshold sufficiently small in line with the crest. Likewise, at points in the detector, certain rises hold the potential of creating false positives; to avoid these false alarms, we need to select a threshold sufficiently high, bypassing the rises. For the tested scenarios, we achieve a fair threshold range with viable settings between 0.08 and 0.155. Below 0.08, false alarms may rise significantly; above 0.155, our hit rate will suffer.

C. Performance of SPARSE_E

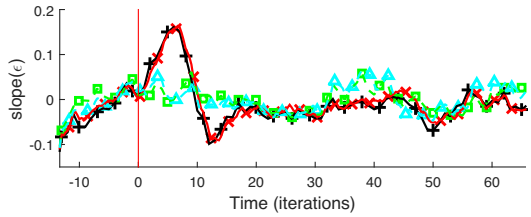
Recall under SPARSE_E, we now transmit entropy values from the server rather than the original feature data. So, we no longer contend with potentially high-variance data as the entropy values will naturally be somewhat normalized. As such we opt to use a η value consistent for each feature. Figures 13 and 14 list transmission efficiencies for the CommittedAS and PageTables signals as a function of η. Consider column two in Fig. 13 that lists transmission efficiencies for Trade6 when the window shift equals w. The base transmission rate is shown as 0.3% of the original raw data since only the entropy values are to be transmitted to the monitor. When $\eta = 0.06$, we achieve a further reduction in transmission cost of 49% over this base rate due to sparsifying the entropy signal. Additionally we test the impact on transmission when reducing the subwindow shift—effectively increasing the entropy data to be transmitted. In general, increase in the number of values will improve relative efficiency but decrease overall transmission efficiency when considering entropy savings; however, the loss of efficiency will be relatively small for higher thresholds and in return, the increased quantity of entropy data may potentially be exploited for cleaner detection and better latency. Comparing overall efficiencies listed in Fig. 13 and 14 to that in Fig. 8, one can see that the gain from SPARSE_E is rather significant. In only one case—pageTables under Trade6—does SPARSE_R actually surpass the alternate in overall efficiency.

η	Trade6 under WL_1			RuBBoS under WL_2		
	w	$w/2$	$w/3$	w	$w/2$	$w/3$
0	0.688	0.532	0.464	0.955	0.906	0.879
0.02	0.656	0.492	0.413	0.837	0.757	0.694
0.04	0.567	0.428	0.357	0.725	0.625	0.556
0.06	0.522	0.397	0.307	0.653	0.525	0.429
Base	0.003	0.006	0.01	0.003	0.006	0.01

Fig. 14. Average rate of transmission per entropy sample for PageTables under SPARSE_E. As before, we test the system for various subwindow shifts where $w = 312$.



(a) Trade6 subjected to workload trace WL_2.



(b) RuBBoS subjected to workload trace WL_1.

Fig. 15. Detector performance under SPARSE_E when using subwindow shifts of w and $w/3$. Solid lines depict the leak-influenced detectors; broken lines represent the nominal cases.

Considering performance under SPARSE_E, the detector should not deviate greatly from that in SPARSE_R. Basically the only difference as far as PCA is concerned would be whether the feature data (in SPARSE_R) or the entropy data (in SPARSE_E) is filtered through the conditional transmission system. The data ultimately processed via PCA should mostly be the same regardless of approach. We evaluate detector performance with the inclusion of transitional entropy calculations—produced by reducing the sliding window's sample shift per calculation. For these cases, rather than evaluating n consecutively computed entropy values, we analyze the entropy data corresponding to nw consecutive feature values; per iteration we effectively shift this nw sized window. While this technique allows us to generate ϵ over smaller intervals of time, some smoothing is required to keep the detector at optimal stability. To this end, we apply a trailing mean filter with length corresponding to the transitional rate.

Figure 15 shows detector performance under different shift values. We set our window parameters to ideal arrangements: $w = 312$ and $n = 50$. The transmission threshold, η, as with the original approach will deteriorate the detector at high settings. In this approach, the transmission rate versus η relationship is smooth and lacking abrupt drops; as such, in selecting η, we choose the knee-point, at which savings per increase in the threshold value begin to diminish. For our purposes, we evaluate the detector with $\eta = 0.02$—a modest setting—for each of the three features. Each plot shows the fault-influenced and nominal cases. By decreasing the shift value, we effectively increase the resolution of the detector (note that in Fig. 15, the detector curves corresponding to the reduced shift cases are scaled in respect to the quantity of iterations to be comparable time-wise). This can, in some cases, improve the general stability of the detector; however, for the most part, the change is not very significant, as one can see from the plots.

Detection Threshold	Trade6: WL_1			RuBBoS: WL_2		
	w	$w/2$	$w/3$	w	$w/2$	$w/3$
0.06	0.347	0.347	0.462	FA	FA	FA
0.10	0.52	0.607	0.578	0.347	0.26	0.289
0.14	0.693	0.607	0.636	0.347	0.347	0.347
0.18	0.693	0.607	0.809	0.52	0.52	0.462

Fig. 16. Table summarizing detection latency for various levels of detection threshold with shifts of w, $w/2$ and $w/3$ units under SPARSE_E. The latency is measured in hours. A false alarm is indicated by FA.

Figure 16 summarizes the detection latencies for two workload scenarios under SPARSE_E. Note that the remaining cases have also been tested, yielding like results. Similar to SPARSE_R, reductions in shift, though often reducing overall latency, do not always yield beneficial results. Comparing these results to those of the original approach, we see that neither is definitively superior in regards to detection. In general, the detection latencies are equivalent with slight variance, most keeping under an hour in time. Under SPARSE_E, the feasible threshold range remains fair. With thresholds at or above 0.1, most false alarms are kept under check. While holding a 0.16 or lower threshold, we maintain crest detection.

VI. Related Work

We discuss related research in two areas: adaptive sampling of data using predictive models, and PCA-based anomaly detection in computer networks and systems. The work reported in this paper differs from prior work in the follow aspects. We apply the dual-prediction approach towards sparsifing features one expects to measure in a typical datacenter setting. We also experimentally evaluate if the recovered data retains critical correlation information for subsequent PCA-based anomaly detection methods to be successfully applied.

Sampling of data using predictive models has been studied previously in the context of sensor networks where the objective is to reduce transmission costs between the source (sensors) and the sink (the base station). Le Borgne *et al*. [10], for example, achieve efficient transmission from sensor to sink using a predictive model as well as adaptive model selection. Specifically, on both ends of the system, a model is used to predict the signal's behavior. So long as the model remains accurate to the true signal's pattern, no transmission is necessary; however, if the true value deviates from the prediction, the model is then updated to one that better fits the new behavior, and the data summarizing the updated model is transmitted to the sink resulting in reduced communication.

Goel *et al*. [11] develop an energy-efficient framework for large sensor networks, again using predictive models. Here, the

monitoring station collects data from the sensor set and with it generates prediction models for these sensors. These models exploit the spatial and temporal correlation present when dealing with adjacent sensor nodes in order to predict behavior. After receiving their respective models from the monitor, the sensors, for the duration of the model's lifetime, transmit updates only when the sensed data sufficiently deviates from the model. In this manner, energy is conserved at the sensor through a reduction of necessary transmissions at the expense of greater costs on the monitoring end.

Lakhina *et al*. develop a PCA-based method for real-time detection of anomalies in computer networks [12]. Here, PCA is applied to high-dimensional network-wide traffic data to extract normal 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 comprise the so-called normal subspace; so it is also referred to as the PCA-based subspace method. Recent extensions of this method for network anomaly detection include [13]–[15]. We have also seen PCA-based methods developed for anomaly detection in cloud computing systems [8]. 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.

Finally, the authors have previously designed a method to detect incipient faults in software systems, combining tools from information theory and statistics such as entropy and PCA [16]. We have shown that memory leaks can be detected under dynamic workload patterns quickly and with a low false alarm rate. The work reported here builds on these results, specifically in terms of sparsifying signals of interest for more efficient transmission to the monitoring station, and analyzing the consequences of sparsification on the anomaly detection capability of PCA-based methods.

VII. Discussion

We have established a technique capable of online anomaly detection via the use of statistical tools such as entropy and PCA. This technique uses a dual-predictor scheme for the sparsification and efficient transmission of data, and in this context, we proposed two approaches based on the position of the transmitter in the overall flow of the system. Both approaches are viable and comparable in terms of detectability and latency; however, the second approach, SPARSE_E, surpasses SPARSE_R in overall transmission savings, making it the superior approach.

In the interest of providing design guidelines to system operators, we studied the effects of tuning the parameters, n, w, and η, as well as shift size, on detector performance. Both n and w serve to maintain stability; however, these parameters must be kept only as large as necessary to prevent an overly complex analysis window. Also, η yields transmission efficiency at the expense of stability, so this parameter must be kept generally low. Reducing the shift size increases the resolution of the detector; however, as its benefits are not definitive, application of this parameter does not merit it's implementation cost.

Returning to the matter of start-up cost, a large n-by-w analysis window can be detrimental to the overall worth of the detector due to an increased likelihood of missing faults that emerge in its start-up period; however, there are ways to mitigate this cost without even reducing window size. Under nominal conditions, the features exhibit fairly periodic behavior which could be modeled. If the system were to store prerecorded feature information from known nominal data, this information could be used to buffer the analysis window, effectively simulating a fault-free start. Given the nature of these models, we only need wait for the first delta-based impulse to synchronize the prerecorded data to the sampled data. This should reduce start-up cost to that of at most half the signal period.

References

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

[2] 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.

[3] 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.

[4] R. Canzanese, M. Kam, and S. Mancoridis, "Toward an automatic, online behavioral malware classification system," in *Proc. IEEE 7th Int'l Conf. Self-Adaptive and Self-Organizing Systems (SASO)*, 2013, pp. 111–120.

[5] T. Sikora, "The mpeg-4 video standard verification model," *IEEE Trans. Circuits & Systems Video Technology*, vol. 7, no. 1, pp. 19–31, 1997.

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

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

[8] 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.

[9] Q. Guan and S. Fu, "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures," in *IEEE Int'l Symp. Reliable Distributed Systems*, Sept 2013, pp. 205–214.

[10] Y.-A. Le Borgne, S. Santini, and G. Bontempi, "Adaptive model selection for time series prediction in wireless sensor networks," *Signal Process*., vol. 87, no. 12, pp. 3010–3020, Dec. 2007.

[11] S. Goel and T. Imielinski, "Prediction-based monitoring in sensor networks: Taking lessons from mpeg," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 5, pp. 82–98, Oct. 2001.

[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] 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 *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1755–1763.

[14] T. Kudo, T. Morita, T. Matsuda, and T. Takine, "PCA-based robust anomaly detection using periodic traffic behavior," *Proc. IEEE Int'l Conf. on Communications*, pp. 1330–1334, June 2013.

[15] 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.

[16] S. DeCelles and N. Kandasamy, "Entropy-based detection of incipient faults in software systems," in *IEEE Pacific Rim Int'l Symp. Dependable Computing (PRDC)*, Nov 2012, pp. 70–79.