Evaluating Compressive Sampling Strategies for Performance Monitoring of Data Centers

Tingshan Huang
ECE Department
Drexel University
Philadelphia, PA 19104, USA
th423@drexel.edu

Nagarajan Kandasamy
ECE Department
Drexel University
Philadelphia, PA 19104, USA
kandasamy@drexel.edu

Harish Sethu
ECE Department
Drexel University
Philadelphia, PA 19104, USA
sethu@drexel.edu

ABSTRACT

Performance monitoring of data centers provides vital information for dynamic resource provisioning, fault diagnosis, and capacity planning decisions. Online monitoring, however, incurs a variety of costs—the very act of monitoring a system interferes with its performance, and if the information is transmitted to a monitoring station for analysis and logging, this consumes network bandwidth and disk space. This paper proposes a low-cost monitoring solution using compressive sampling—a technique that allows certain classes of signals to be recovered from the original measurements using far fewer samples than traditional approaches—and evaluates its ability to measure typical parameters or signals generated in a data-center setting using a testbed comprising the Trade6 enterprise application. Experiments indicate that by using the compressive sampling mechanism, the recovered signal adequately preserves the spikes and other abrupt changes present in the original. The results, therefore, open up the possibility of using low-cost compressive sampling techniques to detect performance bottlenecks and anomalies in data centers that manifest themselves as abrupt changes exceeding operator-defined threshold values in the underlying signals.

Categories and Subject Descriptors

C.4 [**Performance of systems**]: Design studies, modeling techniques, fault tolerance

General Terms

Algorithms, Performance, Management, Reliability

Keywords

Performance management, online monitoring, compressive sampling

1. INTRODUCTION

Online performance monitoring of both the IT infrastructure as well as the physical facility is vital to ensuring the effective and efficient operation of data centers [5,9]. Examples of monitoring solutions include the Tivoli Monitoring software from IBM for the IT infrastructure [10] and the Data Center Environmental Edge from HP that monitors temperature, humidity, and state of the power network within the data center. The monitored information has a variety of uses. It drives real-time performance management decisions such as dynamic provisioning of IT resources to match the incoming workload, detection and mitigation of performance-related hotspots/bottlenecks, and fault diagnosis. In the case of intermittent problems that are hard to isolate, browsing back through historical data can help identify and localize recurring problems affecting the same portion of the IT infrastructure at different times. The information also drives decisions of a longer-term nature: intelligent capacity planning that identifies resources that are over-utilized (under-utilized) and aims to improve overall facility utilization by adding (removing) appropriate resources.

We consider a server cluster wherein software-based sensors embedded within the IT infrastructure measure various performance-related parameters associated with the cluster—high-level metrics such as response time and throughput as well as low-level metrics such as processor utilization, I/O activity (disk reads and writes), and network activity (packets sent and received). The information collected by the sensors is transmitted over a network to a monitoring station for data analysis and visualization. Online monitoring, however, incurs a variety of costs. First, the very act of monitoring an application interferes with its performance; if sensing-related code is merged with the application code, this change may interfere with the timing characteristics of the application or if sensors execute as separate processes, they contend for CPU resources along with the original application. Transmitting the monitored data over a network consumes bandwidth. Finally, logging the data for future use (such as analysis aimed at capacity planning) consumes disk space. So, when monitoring a large-scale computing system, it is desirable to minimize the above-described costs, which is the focus of this paper.[1]

Traditional methods of sampling signals use Shannon's theorem: the sampling rate must be at least twice the signal bandwidth to capture all the information content present in the signal. The theory of compressive sampling, a recent development in signal processing, states, however, that we can recover a certain class of signals from the original measurements using far fewer samples than techniques that use Shannon's theorem [2–4, 7]. Compressive sampling contends that many natural signals are sparse in that they have concise representations when expressed in the proper basis. This property is used to capture the useful information content embedded in the

[1]A preliminary version of this paper appeared as a poster in the IEEE/IFIP Network Operations and Management Symposium (NOMS), 2012.

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

This paper investigates the feasibility of using compressive sampling to measure signals typically generated in an enterprise-level data center and compares data sampling, encoding, and recovery strategies. We use IBM's Trade6 benchmark, a stock-trading service which allows users to browse, buy, and sell stocks, as our testbed and subject it to a self-similar workload while measuring these signals: response time, CPU utilization, and disk I/O activity in terms of sectors read and written. The measurements are acquired directly in terms of fewer number of samples by encoding the data in an appropriate representation basis. For our experiments, we choose four representation bases—the Fourier basis, and the Haar, Daubechies-2, and Daubechies-4 wavelet bases—to determine the sparsity or conciseness of the data when encoded in each of the basis functions. At the monitoring station, the process of recovering the original signal from these samples is posed as a linear programming problem and solved as such. We assess the quality of the reconstructed signal using two performance metrics: relative error that captures the normalized error between the original and reconstructed signals, and receiver operating characteristics (ROC) that characterizes the number of spikes that can be detected using the reconstructed signal.

A major benefit offered by compressive sampling in the context of data center operations is reducing the overhead of generating, storing, and using performance log files for offline analysis tasks such as capacity planning, bottleneck detection, and long-term trend forecasting. When analyzing the data, if the operator wishes to detect performance-related bottlenecks or anomalies that manifest themselves as spikes or abrupt changes in the signal exceeding some nominal threshold value, then the signal reconstructed via compressive sampling must preserve the spikes observed in the original trace. This performance is quantified by the ROC curve and in this regard compressive sampling performs quite well for all signals considered in this paper. By selecting the threshold value appropriately, a hit rate of 93% can be achieved with a false alarm rate of only 0.1%, using about 30% of the samples from the original signals. The results reported in the paper open up the possibility of using compressive sampling as a low-cost online monitoring tool in data centers, especially for anomaly detection and long-term capacity planning.

The paper is organized as follows. Section 2 describes our experimental testbed. Section 3 discusses the compressive sampling of signals generated by the testbed and Section 4 presents experimental results evaluating the performance of this sampling scheme in recovering the original signals. Section 5 discusses related work and Section 6 concludes the paper.

2. EXPERIMENTAL SETUP

Fig. 1 shows the computing system used in our experiments, comprising three servers networked via a gigabit switch. The system for server hosts IBM's Trade6 benchmark, a stock-trading application which allows users to browse, buy, and sell stocks. So, 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 virtual machine on the server `Demeter` within the application tier. Virtualization of this system is enabled by VMWare's ESX Server 3.5 running a Linux RedHat kernel. The operating system on the virtual machine is the SUSE Enterprise Linux Server Edition. The database component is DB2 which is hosted on the server `Ares` running SUSE Enterprise Linux. The database maintains 10,000 user accounts and information for 20,000 stocks.

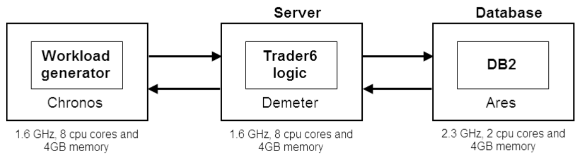

Server
Database
Workload generator
Chronos
Trader6 logic
Demeter
DB2
Ares
1.6 GHz, 8 cpu cores and 4GB memory
1.6 GHz, 8 cpu cores and 4GB memory
2.3 GHz, 2 cpu cores and 4GB memory

Figure 1: The overall system architecture hosting the Trade6 service.

We use Httperf, an open-loop workload generator, to send a mix of browse, buy, and sell requests to the Trade6 application [11]. Based on available evidence on web access traffic patterns [6], the workload follows a self-similar distribution as shown in Fig. 2(a), with an arrival rate of 100 requests per second with a 50/50 mix of buy to browse transactions. Every 100 milliseconds, we measure the average end-to-end response time incurred by the requests (*response_time*). At the database tier, we collect data every 30 seconds corresponding to CPU utilization (*cpu_util*), and the number of disk sectors read and written (*read_activity* and *write_activity*). Fig. 2(b) shows a sample set of parameters/signals collected during an experimental run of the system.

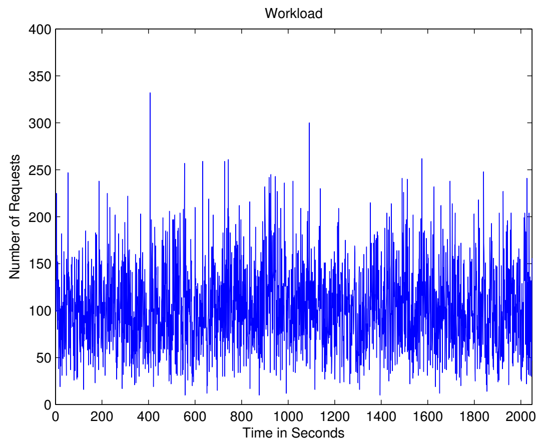
3. COMPRESSIVE SAMPLING OF SYSTEM PARAMETERS

This section describes how to acquire the signals of interest from the testbed directly in a compressed form. First, we familiarize the reader with the two key conditions underlying compressive sampling: sparsity and incoherence. The first condition applies to the signals themselves and the second condition affects the way we sample these signals. We then discuss how the original signal can be recovered from a small set of samples.

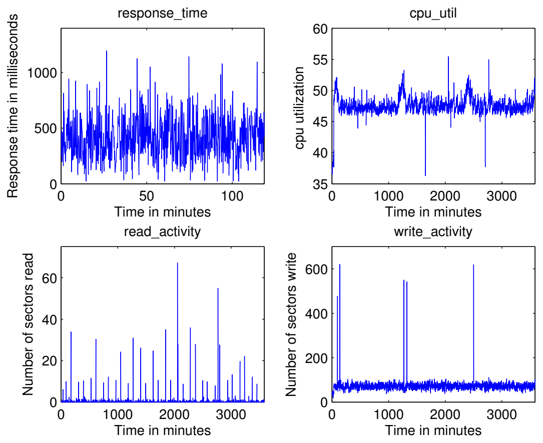
3.1 Sparse Representation of Signals

Assume that the data to be sampled $\mathbf{d}$ is a vector of length N and its representation in some basis B is $\mathbf{x}$. In other words, $d(t) = \sum_{i=1}^{N} x_i b_i(t) = B\mathbf{x}$, where $B = [b_1, b_2, \ldots, b_N]$. For example, if B is selected to be the Fourier basis, the elements of the vector $\mathbf{x}$ are Fourier coefficients corresponding to the signal $\mathbf{d}$. Also, if at most S entries in $\mathbf{x}$ are nonzero, then $\mathbf{x}$ is called an S−sparse vector and $\mathbf{d}$ is said to be sparsely represented in the basis B.

Using the data collected from our testbed, we now aim to find a basis B in which this data can be most concisely represented. We consider the following four basis functions: the Fourier basis, the Haar wavelet basis, the Daubechies-2 (or db2) wavelet basis, and the Daubechies-4 (or db4) wavelet basis [14]. The waveforms corresponding to these functions are shown in Fig. 3. These four bases are common for time-frequency analysis on time series data:

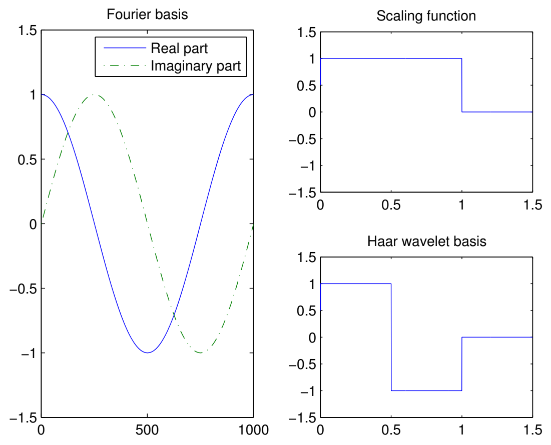


(a)

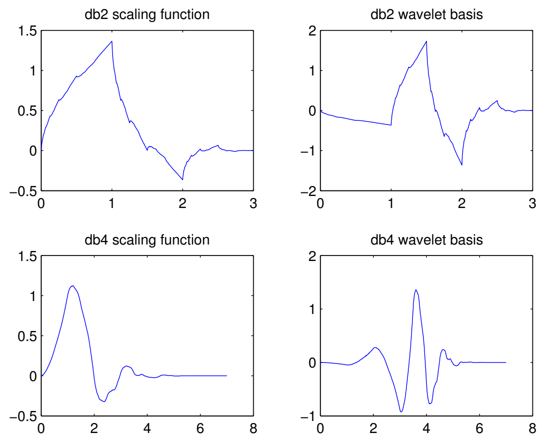


(b)

Figure 2: (a) The self-similar workload presented to the computing system shown in Fig. 1; (b) the various signals collected from the system during an experimental run.



(a) The Fourier and Haar wavelet basis functions.



(b) The Daubechies-2 and Daubechies-4 basis functions.

Figure 3: The waveforms corresponding to the four basis functions considered in this paper.

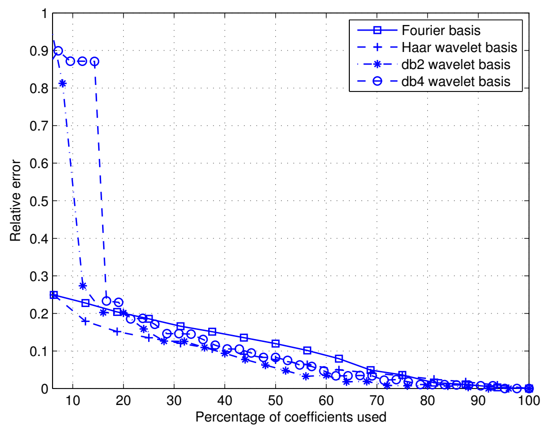
Fourier basis is the traditional tool for analyzing the frequency characteristics of a signal while wavelet bases show signal characteristics in both time and frequency domains. Considering the advantages of using wavelets to capture sharp or abrupt changes in the signal, we focus on three commonly used wavelets: the Haar, the simplest wavelet that captures discontinuities in the data; db2, a more complex waveform with more similarity to the data collected from the testbed; and db4, a wavelet that also shows similarity with our data but has a longer waveform, leading to better frequency resolution. A full introduction to the waveforms corresponding to these wavelets may be found in [13].

We analyze the above-described basis functions in terms of how concisely they encode the data collected from our system. We first perform a signal transform on each data set to find the corresponding coefficients within the chosen basis and arrange them in decreasing order of their magnitude. Then, we use only the first n coefficients, $1 \leq n \leq N$, set all the other coefficients to 0, and reconstruct a new signal $\tilde{\mathbf{d}}(n)$. A relative-error metric captures the difference between the original and reconstructed signals as

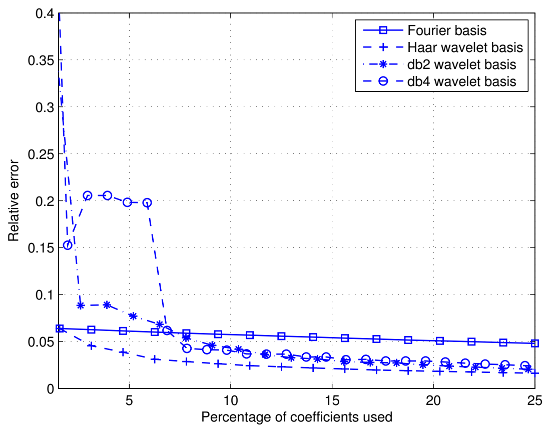
$$e(n) = \frac{\|\tilde{\mathbf{d}}(n) - \mathbf{d}\|}{\|\mathbf{d}\|}. \quad (1)$$

Finally, we examine how this relative error changes as the value of n is increased. For each set of data, we repeat the analysis for each of the four basis functions. The basis in which the relative error decreases to zero fastest represents the signal most concisely.

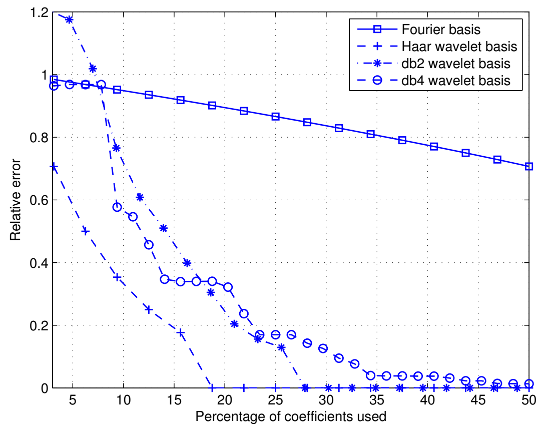
Fig 4 shows that the relative error, when using the Haar wavelet, decays most quickly for *cpu_util* and *read_activity* signals and that the representations of *response_activity* and *write_activity* are not sparse enough within any of the selected bases. The relative error achieved by the Fourier basis is the worst, especially for the *read_activity* and *cpu_activity* signals. This is because these data sets exhibit marked discontinuities with numerous spikes, and thus their representation in the Fourier basis is less concise than those in other bases. For the same reason, we find that the Haar and db2 wavelets represent *cpu_util* and *read_activity* quite concisely. Note that *read_activity* is the sparsest signal in that it requires the smallest percentage of the coefficients to reconstruct the signal within a small relative error, achieving relative error of less than 0.1% when using only 18% coefficients in Haar wavelet basis; *cpu_util* is also sparse since it can be reconstructed with relative error around 2% using 25% of the coefficients under all bases except for Fourier; for the *response_time* and *write_activity* signals, we find that more than 50% of the coefficients are needed to reconstruct them with small



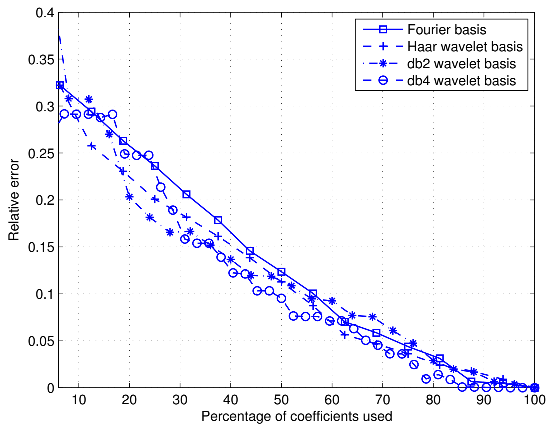
(a) Relative error for the *response_time* signal.



(b) Relative error for the *cpu_util* signal.



(c) Relative error for the *read_activity* signal.



(d) Relative error for the *write_activity* signal.

Figure 4: Plots showing the change in relative error for the *response_time*, *cpu_util*, *read_activity* and *write_activity* signals as a function of the percentage of coefficients used during reconstruction.

Table 1: Percentage of coefficients needed to maintain the relative error within 5%.

Data	Haar	db2	db4	Fourier
response_time	62.5%	51.6%	58.9%	68.8%
cpu_data	2.7%	8.4%	7.5%	21.8%
read_data	19.9%	27.0%	33.9%	99.1%
write_data	66.4%	75.4%	66.6%	72.4%

relative error of 5%, irrespective of the basis. Table 1 summarizes the percentage of coefficients needed to maintain the relative error within 5% for each of the bases.

Section 4 quantifies the effect of using the different bases, in terms of relative error, when the original signal is reconstructed using the samples obtained via compressive sampling. We also use another metric, ROC, in Section 4 to characterize the number of spikes that can be detected in the reconstructed signal.

3.2 Incoherent Sampling of the Signal

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

$$\mu(\Psi, \Phi) = \sqrt{N} \max_{1 \le k, j \le N} \left| \langle \phi_k, \psi_j \rangle \right|, \qquad (2)$$

where $\langle \phi_k, \psi_j \rangle$ is the dot product of vectors ϕ_k and ψ_j.

Usually, the coherence between two bases lies in the range of $\left[1, \sqrt{N}\right]$ and when the value of coherence is small we consider the two bases to be uncorrelated or incoherent. When the sensing basis and the representation basis have a small coherence value, and thus uncorrelated, then a signal that is represented as a spike in one basis will be represented as a spread-out waveform in the other. For example, the Dirac delta basis and Fourier basis have a coherence of one; a signal shown as a spike in the Dirac basis is spread out when represented using the Fourier basis. This property allows us to capture the complete information present in the original data using a small number of samples obtained by incoherent sampling.

As a sampling strategy to collect measurements from our testbed, we choose Gaussian random matrices that have a low coherence of $\sqrt{2 \log N}$ relative to any representation matrix with high probability. Table 2 shows the average coherence between an $M \times N$ Gaussian sensing basis and the various representation bases of interest where N is the length of the input data and M is the desired number of samples. As expected, the coherence values are quite low.

Table 2: Average coherence between an $M \times N$ random Gaussian sensing basis and the different representation bases when $N = 2048$.

Representation basis	Haar	db2	db4	Fourier
$M = 0.1N$	3.67	4.81	4.77	4.82
$M = 0.3N$	3.81	5.05	5.04	5.07
$M = 0.5N$	3.90	5.11	5.12	5.10

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

For example, assume the data to be sampled is an 8×1 vector

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

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

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

$$= G \times \mathbf{d} = G \times B \times \mathbf{x} = A \times \mathbf{x},$$

where $A = G \times B$ is a 4×8 matrix.

Fig. 5 shows the implementation of compressive sampling in our system in which the incoming signal $\mathbf{d}$ is acquired directly in a compressed form $\mathbf{y}$. When a new data item $d(t)$ arrives at time t, it is multiplied by the entries in the sampling matrix $G(j,t), j = 1, \ldots, M$, and the partial products are accumulated into $y(j)$. After a period of length $N \times T$, where T is the sampling period, the current values of $y(j)$ are sent out as the M samples and then reset back to zero. Effectively, $y(j) = \sum_{j=1}^{M} G(j,t)d(t)$, where $t = 1, \ldots, N$, and thus $\mathbf{y} = G \times \mathbf{d}$.

3.3 Recovering the Original Signal

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

$$\min_{\mathbf{b} \in \mathcal{R}} \ \|\mathbf{b}\|_{l_0} \quad \text{subject to: } \mathbf{y} = A\mathbf{b}, \tag{3}$$

where $\|\mathbf{b}\|_{l_0}$ is 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 $A\mathbf{b} = \mathbf{y}$. The problem of minimizing the l_0 norm is a computationally expensive nonlinear optimization problem. So, one of the following classes of reconstruction algorithms is typically used for computational efficiency: basis pursuit and iterative algorithms based on hard thresholding pursuit.

The basis pursuit technique solves the problem in (3) by minimizing the l_1 norm, which is the linear programming problem posed

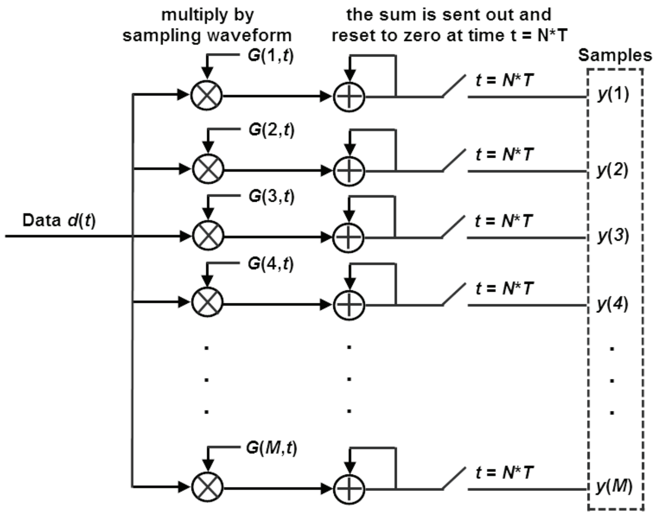

multiply by sampling waveform
the sum is sent out and reset to zero at time t = N*T
Samples
G(1,t)
G(2,t)
G(3,t)
G(4,t)
G(M,t)
Data d(t)
t = N*T
y(1)
y(2)
y(3)
y(4)
y(M)

Figure 5: Implementation of compressive sampling in our system that takes N data items over a time period as input and returns M samples.

as follows:

$$\min_{\mathbf{b} \in \mathcal{R}} \ \|\mathbf{b}\|_{l_1} \quad \text{subject to: } \mathbf{y} = A\mathbf{b}. \tag{4}$$

Reconstruction of the original signal using (4) is considered to be exact with probability exceeding $1 - \delta$, where δ is a very small constant, if the number of samples

$$M \geq C\mu(\Psi, \Phi)^2 S \log \frac{N}{\delta}, \tag{5}$$

where C is some positive constant [1]. The direct consequence of (5) is that when the coherence between the representation and sensing bases, μ, as well as the sparsity metric, S, is small, we need only a few samples to recover the original signal exactly with high probability.

In iterative algorithms, the problem in (3) is solved by iteratively selecting an sparse vector $\mathbf{b}$ based on the previous selection. In our work, we use a method termed the hard thresholding pursuit previously proposed by Foucart [8]. Let the initial value of the vector be $\mathbf{b} = \mathbf{0}$ and let the S−sparse vector selected at step n be $\mathbf{b}^n$, the iteration scheme involves two steps:

$$I^{n+1} = \{\text{Indices of S largest entries in } \mathbf{b}^n + A^*(\mathbf{y} - A\mathbf{b}^n)\}$$

$$\mathbf{b}^{n+1} = \arg\min\{\|\mathbf{y} - A\mathbf{b}\|_2, \ \text{supp}(b) \subseteq I^{n+1}\}$$

The iterations continue until $I^{n+1} = I^n$. This algorithm has been validated to perform faster than other threshold-based methods such as orthogonal matching pursuit.

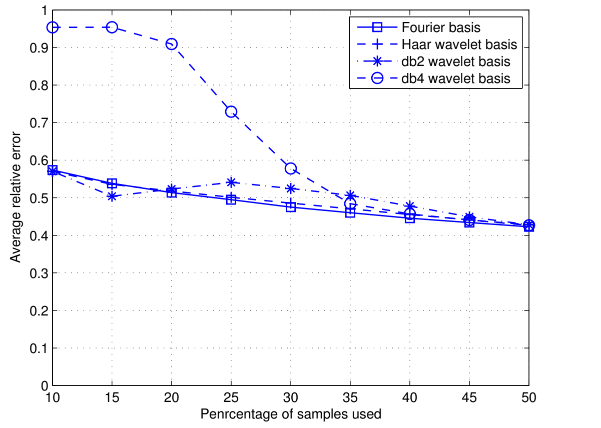
4. PERFORMANCE EVALUATION

This section evaluates the performance of compressive sampling in recovering the various signals obtained from the testbed using the Fourier basis and the Haar, db2, and db4 wavelet bases. The goal of these experiments is to determine the most suitable representation basis to use when reconstructing the original signal.

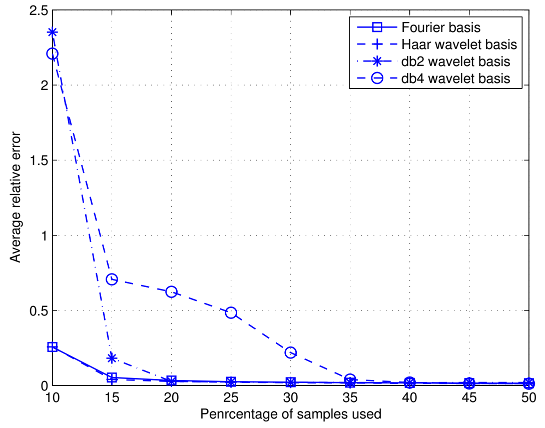
4.1 Performance Metrics

We use the following metrics to assess the quality of the signal recovered from its sampled form.

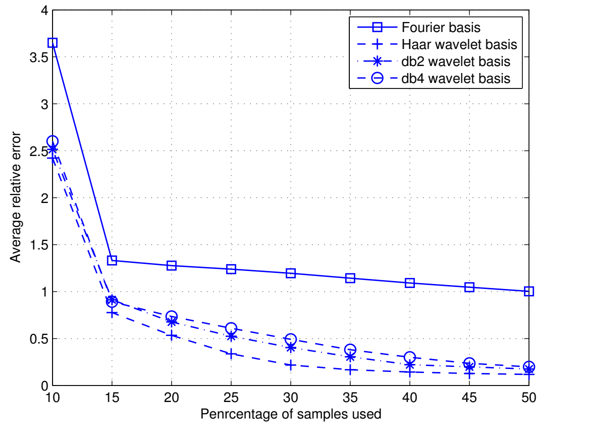
Relative error. This metric (defined in Section 3.1) expresses the normalized error between the original and recovered signals.



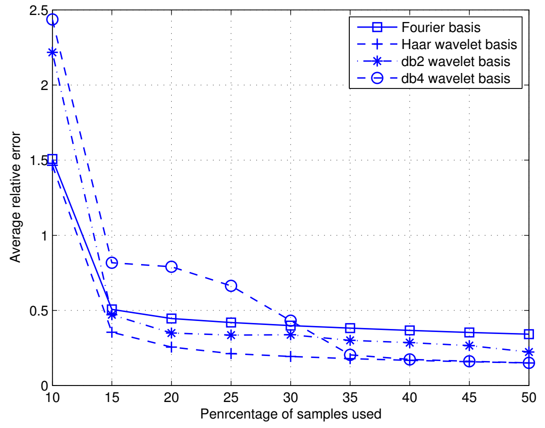
(a) Relative error achieved by the recovered *response_time* signal.



(b) Relative error achieved by the recovered *cpu_util* signal.



(c) Relative error achieved by the recovered *read_activity* signal.



(d) Relative error achieved by the recovered *write_activity* signal.

Figure 6: The relative error achieved by the recovered signals as a function of the number of samples used during the reconstruction.

Receiver operating characteristic (*ROC*). Considering the number of spikes present in the *cpu_util*, *read_activity*, and *write_activity* data sets, we use the ROC metric to characterize the number of spikes that can be detected using the reconstructed signal. We define a hit as follows: at time t within the original and recovered signals, a spike occurring in the recovered signal matches a similar spike in the original signal. We define a false alarm as follows: at time t, a spike occurring in the recovered signal has no match in the original signal. The ROC plots the hit rate against the false alarm rate. The ROC becomes relevant in situations in which it may not be essential for compressive sampling to recover the original signal exactly, such as when the operator is mainly interested in detecting performance bottlenecks, hot spots, or anomalies affecting the computing system, some of which manifest themselves as spikes or abrupt changes in the signals being monitored.

4.2 Characterizing Signal Recovery via Relative Error

Figure 6 summarizes the relative errors achieved by the various representation bases when recovering the signals as a function of the percentage of samples used. Figure 6(a) shows the relative error achieved when reconstructing the *response_time* signal. Even when 50% of the samples are used, the relative error is over 40% for all the bases. This result is expected since, as shown in Fig. 4(a), none of the bases can concisely represent the signal in terms of the number of required coefficients. For example, we find that at least 50% of the coefficients are needed in the db2 basis to capture most of the signal energy, and so the sparsity of the signal in db2 basis is more than $0.5N$, where N is the length of the signal. According to (5), we require $4 \times S \times \mu^2(\Phi, \Psi)$ samples for exact reconstruction. For example, when using the db2 wavelet as the representation basis Ψ, the coherence $\mu(\Phi, \Psi)$ is around 5.1 (from Table 2) and to reconstruct *response_time* exactly, we need around $4 \times 0.5N \times 5.1^2 = 52.02N$ samples—a sample size fifty times larger than the original data. When *response_time* is represented using the other bases, the sample size required for exact reconstruction is even larger. This demonstrates why compressive sampling is not very effective on data sets that cannot be represented as a sparse vector within a certain basis.

Figure 6(b) summarizes the relative error achieved by the recovered *cpu_util* signal; when 20% of the samples are used, the relative error is less than 4% using the wavelet bases. The signal is quite sparse and when represented in any of the wavelet bases, *cpu_util* can be reconstructed with less than 5% relative error using at most 20% of the wavelet coefficients. Furthermore, if we approximate the sparsity of *cpu_util* in the Haar wavelet basis by

$0.02N$ and use 3.67 as the coherence value, then we require about $4 \times 0.02N \times 3.67^2 = 1.13N$ samples for exact signal reconstruction with high probability. However, we find a better result from our experiment: when 15% of the samples are used, the relative error corresponding to each of the three wavelet bases falls below 3%.

Figure 6(c) shows that when 50% of the samples are used, the relative error achieved by the reconstructed *read_activity* signal is around 20% under the wavelet bases. The relative error never falls below 100% when the Fourier basis is used, implying that this basis is not suitable for the compressive sampling of *read_activity*. Figure 6(d) summarizes the relative error achieved for the *write_activity* signal. Under the db4 or Haar wavelets, the relative error falls below 25% when about 35% of the samples are used. Conversely, even when 50% of the samples are used, the relative error is around 25% for the db2 wavelet basis and 40% for the Fourier basis. The result for the Fourier basis is as expected. Figure 4 indicates that a very large percentage of the Fourier coefficients are needed to maintain a 5% relative error and the coherence between our sampling waveform and the Fourier basis is about 5.1 (from Table 2). We find that the sparsity of *write_activity* under the Fourier basis is about $0.75N$ and for an exact signal recovery, we need a sample size $4 \times 0.75N \times 5.1^2 = 78.03N$; that is, even a sample set the same size as the original data is not sufficient for reconstruction. So, *write_activity* is not appropriate for compressive sampling using the Fourier basis. We find similar results with the Haar, db2, and db4 wavelet bases as well. As discussed in Section 3.1, more than 70% of coefficients are necessary to maintain a 5% relative error using these bases; the sparsity of *write_activity* when represented in these bases is at least $S = 0.7N$. Since the coherence between the Gaussian sampling matrix and the Haar, db2, or db4 bases is at least 3.67 (from Table 2), the sample size required for exact recovery has to be $4 \times 0.7N \times 3.67^2 = 37.71N$. As a result, we can conclude that *write_activity* is less appropriate for compressive sampling using the Haar, db2, or db4 bases—at least for exact recovery.

We summarize the results as follows: for the data sets considered in this paper, the Haar wavelet basis achieves, on average, the best performance in terms of the relative error metric. The Fourier basis performs well for *response_time* and *cpu_util*, but is the worst for *read_activity* and *write_activity* signals. If the Haar wavelet is used as the representation basis, the basis vector for the *cpu_util* signal is sparser than those for the *response_time*, *read_activity* and *write_activity* signals. Fig. 7 shows the performance of the Haar basis as the sample size is varied; when 30% of the samples are used, the relative error is nearly 2.5% for the recovered *cpu_util* signal. For the *read_activity* and *write_activity* signals, however, we require at least 50% of the samples to lower the relative error to 15% and the relative error for the *response_time* signal is almost 43% even if we use 50% of the samples.

4.3 Characterizing Signal Recovery via Receiver Operating Characteristics

As noted earlier in the paper, there are situations in which it is not essential for compressive sampling to recover the original signal exactly. If the operator is mainly interested in detecting performance bottlenecks, hot spots, or anomalies affecting the computing system that manifest themselves as spikes or abrupt changes in the signals being monitored, it is more important that the reconstructed signal preserve these characteristics.

Fig. 8 shows the original and recovered signals for *write_activity*, *cpu_util*, and *read_activity*, overlayed on each other. The signals are mostly limited to narrow bands of values and compressive sampling does an adequate job of reconstruction using about 30% of the

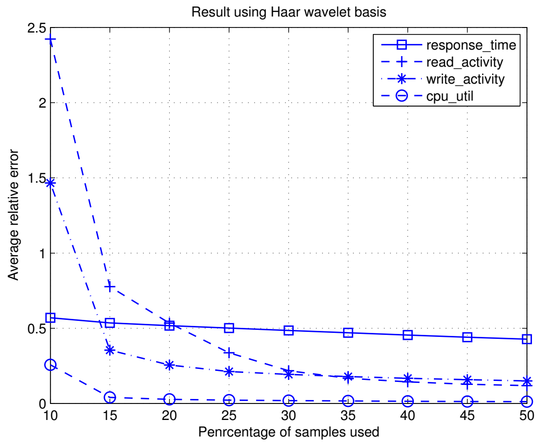


Figure 7: The relative error achieved by the Haar wavelet basis for the various data sets as a function of the number of samples used for signal reconstruction.

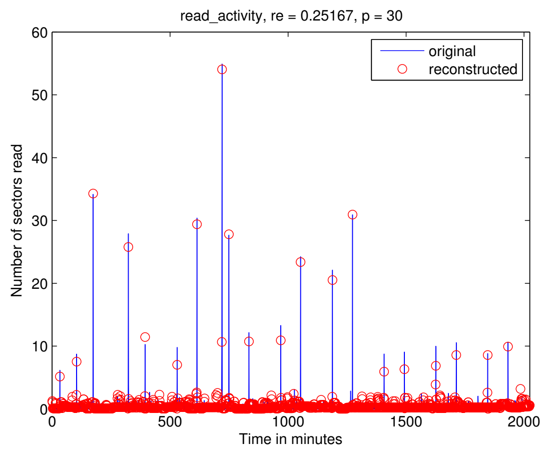
Table 3: False alarm and hit rates associated with different threshold levels for the reconstructed *write_activity* and *cpu_util* signals.

Data	read_activity	write_activity	cpu_util
Threshold level	6	132	43.8%
False alarm rate	0.05	0.05%	0.05%
Hit rate	94.64	97.59%	93.96%

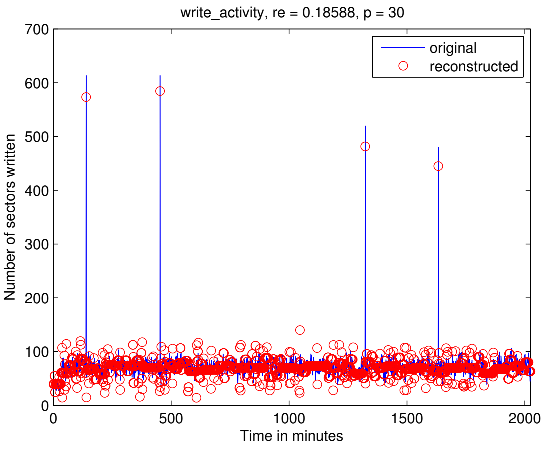
original signal when using Haar wavelet basis even though the relative error are larger than 15% for *read_activity* and *write_activity*. More importantly, the recovered signal preserves the spikes found in the original quite well, preserving the abrupt bursts in the *read_activity* and *write_activity* signals, and sudden (steep) decreases in *cpu_util*. Figs. 9(a), 9(b) and 9(c) graph the ROC curves for the *read_activity*, *write_activity* and *cpu_util* signals reconstructed under the different representation bases. The plots are generated using 30% of the samples from the original *write_activity*, *write_activity* and *cpu_util* signals, respectively. We change the threshold level that defines a spike, and for each threshold obtain and plot a pair of values: the hit rate and the corresponding false alarm rate.

Table 3 lists the false alarm rates and hit rates with their corresponding threshold-level settings for the reconstructed *read_activity*, *write_activity* and *cpu_util* signals. The original signals were encoded using the Haar wavelet basis. The performance, in terms of ROC, indicates that compressive sampling still helps to reconstruct the spikes of interest. For example, from the viewpoint of detecting anomalies affecting *write_activity*, a hit rate of more than 95% can be achieved with a corresponding false alarm rate of less than 0.1% by setting the threshold to about 132 sectors written.

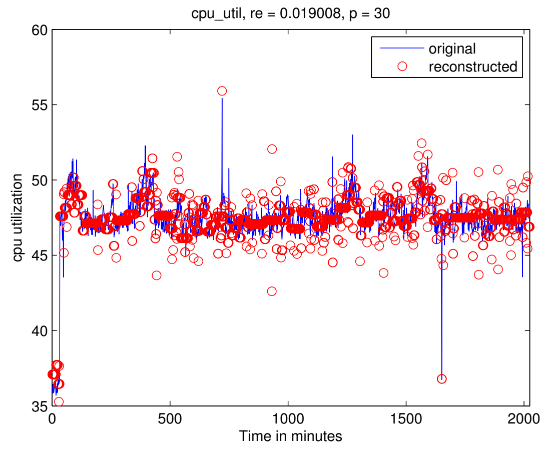
Generally, both the false alarm rate and the hit rate should decrease as the threshold is increased since fewer false spikes occur in the reconstructed signal. However, we see in Fig. 9(a) that the hit rate corresponding to the db2 basis increases as the threshold is increased whereas the corresponding false alarm decreases from 0.2% to 0.05%. We see a similar result as well for the db4 basis in Fig. 9(b) and for all the bases in Fig. 9(c). To explain this unusual result, we examine the reconstructed signal for the *read_activity* when using db2 in conjunction with the original signal (shown in the overlay plot in Fig. 10). We find that spikes larger than 7 in the *read_activity* signal are all detected but under-estimated in the reconstructed signal, i.e., the reconstructed spikes match those in



(a) Overlay of the original and recovered *read_activity* signals.

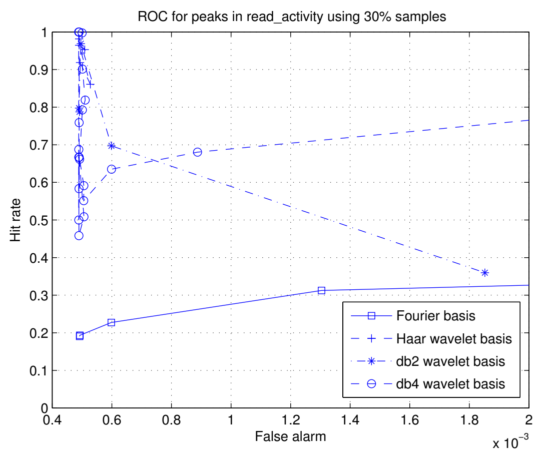


(b) Overlay of the original and recovered *write_activity* signals.

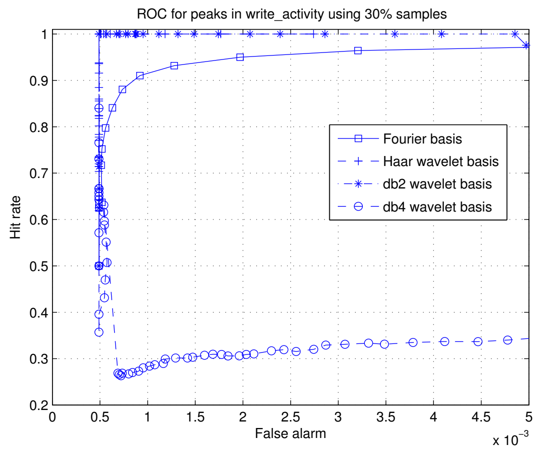


(c) Overlay of the original and recovered *cpu_util* signals.

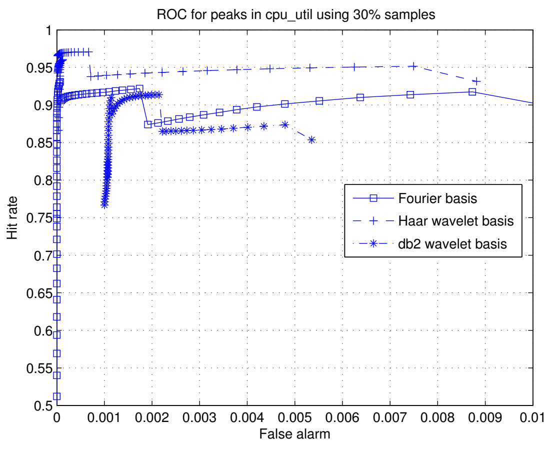
Figure 8: Overlay plots comparing the recovered *read_activity*, *write_activity* and *cpu_util* signals using Haar wavelet basis with the originals. Compressive sampling preserves the spikes present in the original signals quite well.



(a) ROC curve generated by the recovered *read_activity* signal.



(b) ROC curve generated by the recovered *write_activity* signal.



(c) ROC curve generated by the recovered *cpu_util* signal.

Figure 9: Plots showing the ROC curves for the read_activity, write_activity and cpu_util signals reconstructed under the four representation bases.

the original signal but have smaller values. Besides, we find that smaller spikes in the original signal within the range [1, 7] are not detected and there are small spikes in the reconstructed signal without a match in the original signal. For these reasons, as the threshold is increased within the [1, 7] range the number of de-

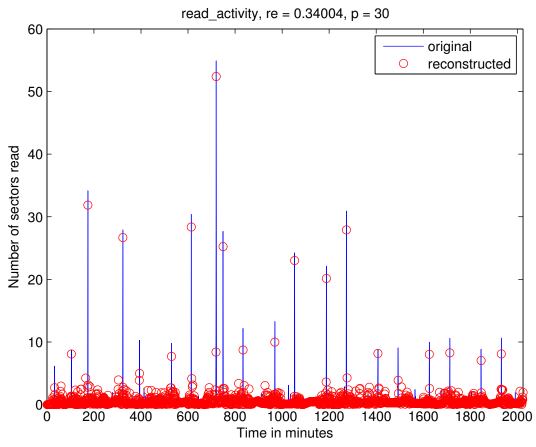


Figure 10: Overlay plot comparing the recovered *read_activity* signal using db2 wavelet basis with the original. Spikes present in the original signals are underestimated in the reconstructed signal.

Table 4: Execution Time (in milliseconds) for the incoherent sampling process as a function of sample size N.

Percentage of Samples	10%	20%	30%
$N = 1024$	0.14	0.29	0.38
$N = 2048$	0.52	0.96	1.40
$N = 4096$	1.90	3.90	5.50

Table 5: Execution Time (in seconds) incurred by the signal recovery process as a function of sample size N.

Percentage of Samples	10%	20%	30%
$N = 1024$	0.3	0.3	0.6
$N = 2048$	0.9	1.8	1.9
$N = 4096$	5.1	9.1	11.7

tected spikes that are above the threshold stays the same, whereas the number of spikes above the threshold in the original signal decreases. As a result, the hit rate increases. Furthermore, as the threshold increases within this range, there are fewer reconstructed spikes above the threshold without a match in the original and the false alarm decreases as expected. The unusual trend in the other plots can be explained similarly. This explanation also shows the importance of selecting both the proper basis for compressive sampling and threshold values to detect anomalies.

4.4 Computational Complexity

We quantify the execution-time overhead incurred by both the sampling and signal recovery processes as a function of sample size. The algorithms were implemented in MATLAB and executed on a server equipped with an AMD Athlon II 3.0 GHz processor. For presentation purposes, we denote the length of original data to be sampled as N; the length of compressed samples as M; and the average execution time incurred by the sampling and recovery processes as $t_s(N, M)$ and $t_r(N, M)$, respectively.

The execution time for incoherent sampling is summarized in Table 4 as a function of sample size. Generally, this process is not time-consuming: when $N = 4096$ (corresponding to approximately 48 hours of monitoring time in the case of *cpu_util*, *read_activity*, and *write_activity* signals) and when we wish to compress this data to 30% of its original size (or $M/N \times 100\% = 30\%$) the running time is about 6 ms. The average value for $t_s(N, 2M)/t_s(N, M)$ is 1.99 and the average value for $t_r(N, 3M)/t_r(N, M)$ is 2.77, which indicates that the execution time grows sublinearly with M. Assume the execution time for the sampling and the recovery to be exponential with N as the exponent,

$$t_s(N, M) = c_s \times N_s^r,$$

$$t_r(N, M) = c_r \times N_r^r,$$

where the values for c_s and c_r depend on M. Then the value for order r_s and r_r can be calculated by

$$c_s = \frac{\log(t_s(2N, M)) - \log(t_s(N, M))}{\log(2)}.$$

$$c_r = \frac{\log(t_r(2N, M)) - \log(t_r(N, M))}{\log(2)}.$$

From Table 4, we find the average value for order c_s is 1.89, indicating that the running time grows sublinearly with N^2. Since the running time increases linearly as M/N increases and quadruples as N is doubled, the computational complexity associated with incoherent sampling is $O(N^2M)$. In our implementation, the process of incoherent sampling is simply the multiplication of an $M \times N$ matrix with an $N \times 1$ vector, and the complexity is actually $M \times (N \times \text{multiplication} + (N-1) \times \text{addition}) = O(NM)$.

The algorithm used for the signal recovery is a recent iterative algorithm termed Hard Thresholding Pursuit proposed by Foucart [8]. Table 5 lists the execution times incurred by the recovery algorithm as a function of N. The average value for $t_r(N, 2M)/t_r(N, M)$ is 1.59 and for $t_r(N, 3M)/t_r(N, M)$, it is 2.14, indicating that the running time grows sublinearly with M; the average value for order c_r is 2.22, showing that the running time grows linearly with N^2. Therefore, the computational complexity associated with the signal recovery process is $O(N^2M)$.

5. RELATED WORK

To the best of our knowledge compressive sampling has not been perviously studied as a technique for monitoring the performance of enterprise computing systems. It has recently been validated for monitoring fine-grained processor performance [12] and for the detection of Internet traffic anomalies [15].

Tuma *et al.* study the applicability of compressive sampling to fine-grained monitoring of processor performance and propose the method as a means of simplifying the complex processes of sensing and transferring signals related to micro-architecture performance [12]. The authors evaluate the performance of compressive sampling on signals representing one or more micro-architectural counters within a processor core, and show that compressive sampling can recover these signals if one can identify the bases in which the signals can be sparsely represented. Their approach bears some similarity to our work, but the measurements are obtained from hardware counters inside various micro-architectural components of the processor and the evaluation is limited to a signal-to-noise (SNR) metric—same as the relative error metric used in this paper. Our data is obtained from a server platform and includes both high-level performance metrics (response time) as well as low-level ones (CPU utilization and disk I/O activity). We also evaluate compressive sampling in terms of the ROC metric to evaluate how well spikes can be recovered within the reconstructed signal. Besides, rather than using traditional algorithms such as Orthogonal Matching Pursuit for signal recovery as in [12], we use a faster iterative Hard Thresholding Pursuit algorithm originally proposed in [8].

In the area of computer network monitoring, Zhang *et al.* use the spatial-temporal compressive sampling framework for traffic matrix interpolation [15]. This technique uses an algorithm called sparsity regularized matrix factorization that uses both the global low-rank property of traffic in a network and spatio-temporal properties of the local traffic, and the authors focus on building a model that includes both spatial and temporal properties of the underlying traffic matrix. They claim that the normal component of the traffic matrix can be captured via a low-rank matrix, and try to determine a low-rank matrix that best estimates the original traffic matrix. Their work shows that over 90% of the missing values in traffic matrix can be inferred with reasonable accuracy using compressive sampling.

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

6. CONCLUSIONS

This paper has proposed and evaluated a low-cost monitoring solution for data centers based on the concept of compressive sampling that allows certain classes of signals to be recovered from the original measurements using far fewer samples than traditional approaches. Using the Trade6 application as our testbed, we showed how to acquire measurements corresponding to response time, CPU utilization, and disk I/O activity directly in a compressed form, and how to reconstruct the full-length signal at the monitoring station using a few number of samples. We have experimented with four different basis functions to determine the conciseness of the data when encoded in each of these functions and assessed the resulting quality of the reconstructed signal. The performance of compressive sampling is quite promising in terms of the ROC—the recovered signal adequately preserves the spikes and other abrupt changes present in the original signal. By selecting the threshold value appropriately, a hit rate of 93% can be achieved with a false alarm rate of less than 0.1%.

Acknowledgments

This work was partially funded by the Office of Naval Research (ONR) through grant number N00014-10-1-0625.

7. REFERENCES

[1] E. J. Candès and J. Romberg. Sparsity and incoherence in compressive sampling. *Inverse Prob.*, 23(3):969–985, 2007.

[2] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inform. Theory*, 52(2):489–509, 2006.

[3] E. J. Candès and T. Tao. Near optimal signal recovery from random projections: Universal coding strategies? *IEEE Trans. Inform. Theory*, 52(12):5406–5425, 2006.

[4] E. J. Candès and M. B. Wakin. An introduction to compressive sampling. *IEEE Signal Proc. Mag.*, 25(2):21–30, 2008.

[5] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni. Automated anomaly detection and performance modeling of enterprise applications. *ACM Trans. Comput. Syst.*, 27:6:1–6:32, Nov. 2009.

[6] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE Trans. Networking*, 5(6):835–846, 1997.

[7] D. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52(4):1289–1306, 2006.

[8] S. Foucart. Hard thresholding pursuit: An algorithm for compressive sensing. *preprint*, 2010.

[9] M. Kutare et al. Monalytics: Online monitoring and analytics for managing large scale data centers. *Proc. ACM ICAC*, 2010.

[10] G. Lanfranchi, P. D. Peruta, A. Perrone, and D. Calvanese. Toward a new landscape of systems management in an autonomic computing environment. *IBM Systems Journal*, 42(1):119–128, 2003.

[11] D. Mosberger and T. Jin. httperf: A tool for measuring web server performance. *Perf. Eval. Review*, 26:31–37, 1998.

[12] T. Tuma, S. Rooney, and P. Hurley. On the applicability of compressive sampling in fine grained processor performance monitoring. *Proc. IEEE Int'l Conf. on Engineering of Complex Computer Systems*, pages 210–219, 2009.

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

[14] G. G. Walter and X. Shen. *Wavelets and Other Orthogonal Systems*. CRC Press, 2 edition, 2000.

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