This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

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

Tingshan Huang, *Member, IEEE*, Nagarajan Kandasamy, *Senior Member, IEEE*, Harish Sethu, *Member, IEEE*, Matthew C. Stamm *Member, IEEE*

***Abstract*—Performance monitoring of datacenters provides vital information for dynamic resource provisioning, anomaly detection, and capacity planning 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, this paper develops and validates a strategy based on adaptive-rate compressive sampling. It exploits the fact that the signals of interest often can be sparsified under an appropriate representation basis and that the sampling rate can be tuned as a function of sparsity. We use the Trade6 application as our experimental platform and measure the signals of interest—in our case, signals pertaining to memory and disk I/O activity—using adaptive sampling. We then evaluate whether the reconstructed signals can be used for trend detection to track the gradual deterioration of system performance associated with software aging. Our experiments show that the signals recovered by our methods can be used to detect, with high confidence, the existence of trends within the original signal. We also evaluate the reconstructed signals for threshold-violation detection wherein the magnitude of the signal exceeds a preset value. Our experiments show that performance bottlenecks and anomalies that manifest themselves in portions of the signal where its magnitude exceeds a threshold value can also be detected using the reconstructed signals. Most importantly, detection of these anomalies is achieved using a substantially reduced sample size—a reduction of more than 70% when compared to the standard fixed-rate sampling method.**

***Index Terms*—Online monitoring, anomaly detection, adaptive sampling, compressive sampling.**

I. Introduction

Online performance monitoring of computing systems and network infrastructure within a datacenter is vital to ensuring efficient operation [1]–[3]. The monitored information has a variety of uses: it drives real-time performance management decisions such as dynamic provisioning of resources to match the incoming workload as well as anomaly detection, diagnosis, and mitigation. The monitored information also drives decisions of a longer-term nature; for example, capacity planning that identifies resources that are over-utilized or under-utilized and aims to improve utilization by adding or removing appropriate resources. In terms of system security, operators who monitor communication networks for malware or denial-of-service attacks are increasingly dependent on real-time detection of anomalous behavior in the traffic data [4].

T. Huang is with Akamai Technologies, Cambridge, MA 02142. E-mail: tingshan.huang@akamai.com. N. Kandasamy, H. Sethu, and M. C. Stamm are with the Electrical and Computer Engineering Department, Drexel University, Philadelphia, PA 19104. E-mail: kandasamy@drexel.edu, sethu@drexel.edu, mstamm@coe.drexel.edu.

This work was partially funded by the Office of Naval Research (ONR) through grant number N00014-10-1-0625 and by the NSF Award 1228847.

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

The volume of data that one has to collect and process for effective monitoring of datacenters poses significant Big Data challenges in collection, analysis, and storage [5]. In this paper, we focus on the data collection phase and develop sampling methods to derive a low-dimensional structure as a compact or parsimonious representation of the original dataset at the local server prior to transmission to the monitoring station. Using *compressive sampling* (*CS*) [6]–[9] as the theoretical foundation, our methods are designed to exploit any inherent sparsity in the signal being sampled to reduce the dimensionality of the collected data. A signal or portion thereof is termed *sparse* if it can be concisely represented in the proper basis function; this property can be used to capture the useful information content embedded in the signal and condense it into a small amount of data. In other words, one can acquire these signals from the underlying system *directly* in a compressed form.

From a viewpoint of reducing the costs associated with monitoring, compressive sampling allows for a very simple sensing strategy—rather than tailoring the sensing scheme to the specific signal being measured, a signal-independent strategy such as randomized sampling can be used, significantly reducing the intrusion of monitoring on application performance. Also, since signals are acquired directly in compressed form, the network bandwidth required to transmit these few samples to the monitoring station is reduced and so is the hard-disk space required to store them. When operators wish to analyze the original signal, there is a way to use numerical optimization to reconstruct the full-length signal from the sample set. The recovery process is typically posed as a linear programming problem and solved using the hard

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

thresholding pursuit (HTP) approach [10].

We focus on two major challenges related to the practical use of CS as a monitoring tool in a datacenter setting. First, having to manually select the appropriate representation basis under which the measured signal is most sparse and thus achieves the best reconstruction imposes a substantial burden on the system operator and must be automated. Secondly, the notion that signal sparsity—the information content—is constant or bounded across its entire length rarely holds in practice. Therefore, to maximize the overall information content captured from the signal while simultaneously reducing the requisite number of samples, it is best to follow an adaptive rather than fixed rate sampling strategy.

We address the above challenges by developing a new adaptive sampling strategy within the CS framework for online monitoring, exploiting the fact that the signals of interest often can be sparsified under an appropriate representation basis and that the sampling rate itself can be tuned as a function of sparsity. The following two new data sampling and encoding strategies are developed and evaluated:

- We develop a basis selection algorithm called *Best Basis* that automatically adapts the representation basis to the structure of the underlying signal being sampled such that the information can be most concisely represented. Using the best-basis algorithm, we develop a strategy called C-MON that takes advantage of signal compressibility to reduce the data transfer and storage costs associated with online monitoring.
- We develop *adaptive compressive sampling* where the key idea is to dynamically tune the sampling rate as the signal sparsity changes: in time windows where the signal is sparse we reduce the sampling rate and in windows where the signal is less concise the rate is increased, all the while ensuring that the chosen sampling rate guarantees a user-defined signal recovery quality. We term this strategy for online monitoring as CS-MON.

A major benefit offered by adaptive sampling in the context of datacenter operations is in reducing the overhead of generating, storing, and using performance log files for analysis tasks such as capacity planning, anomaly detection, and trend forecasting. Here we evaluate the efficacy of detecting threshold violations and trends using the signals reconstructed via C-MON and CS-MON. We use IBM's Trade6 benchmark, a stock-trading service which allows users to browse, buy, and sell stocks, as our testbed and subject it to a dynamic workload while measuring signals pertaining to the memory and disk I/O subsystems. Under the first scenario, the operator wishes to detect anomalies that manifest themselves as the magnitude of the signal exceeding some nominal threshold value. The performance of the C-MON and CS-MON strategies in this regard is quantified by a hit-rate metric; we show that by selecting the threshold value appropriately, both strategies achieve a hit rate of 90% with a false alarm rate of only 0.1%. In the second scenario, the operator wishes to detect the gradual deterioration of system performance, say over hours or days, associated with software aging [11]. Common causes involve resource exhaustion due to memory leaks and bloat, unreleased network sockets and file locks, and unterminated threads. One way of determining if software aging exists is to statistically analyze the appropriate signals for the existence of trends [12], [13]. We use a long-running Trade6 application having a small memory leak of about 100 KB/minute, and evaluate the performance of C-MON and CS-MON in terms of their ability to estimate a positive slope in the reconstructed data even in the presence of seasonal variations and periodicity in the signal.

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

The paper is organized as follows. Section II discusses related work and Section III describes our experimental testbed. Section IV demonstrates C-MON, the compression of signals generated by the testbed using the Best Basis algorithm, and Sections V and VI discuss CS-MON, the compressive sampling of signals using an adaptive rate strategy. Section VII presents experimental results evaluating the performance of these strategies in recovering the original signals, and in detecting threshold violation and trends. Section VIII provides some concluding remarks.

II. Related Work

To the best of our knowledge, compressive sampling has not been previously studied as a performance monitoring tool in the context of cloud computing except in our earlier work in [14] and [15] where the emphasis was on studying the feasibility of using fixed-rate compressive sampling for monitoring server systems. This work has developed an adaptive-rate model that exploits any time-varying sparsity in the signal to further reduce the number of collected samples.

Tuma *et al.* study the applicability of compressive sampling for fine-grained monitoring of processor performance and evaluate its performance on signals representing micro-architecture counters within a core [16]. They show that compressive sampling can recover these signals if one can identify the bases in which the signals can be sparsely represented. Their approach bears some similarity to our work, but the measurements are obtained from hardware counters inside various micro-architectural components of the processor and the evaluation is limited to a signal-to-noise metric—same as the relative error metric used in this paper. In contrast, this paper combines compressive sampling with sparsity prediction for rate adjustment, and evaluates its performance in spike detection and trend detection with the reconstructed signal. Besides, rather than using traditional algorithms such as Orthogonal Matching Pursuit (OMP) for signal recovery as in [16], we employ a faster iterative Hard Thresholding Pursuit algorithm originally proposed in [10].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

Work on adaptive-rate sampling in other domains includes the results reported in [17]–[23]. Meng *et al.* propose a method for distributed state monitoring in Cloud datacenters wherein the sampling intervals are updated dynamically according to the likelihood of detecting a threshold-based state violation [17]. Ward shows in [18] that side information of cross-validation measurements can be used to improve the recovery algorithm used in compressive sampling and reduce the sampling rate. For the similar purpose of improving signal recovery algorithms under a reduced sample rate for compressive sampling of images and videos, Stankovic *et al.* use spatial and temporal correlations within images and videos [20]. OMP is employed in [20] as the recovery algorithm. Besides, spatial correlations between non-overlapping blocks of the same image are exploited in [20] for image recovery, and temporal correlations between one frame and the previous frame of a video are exploited for video recovery. Sampling rates for compressive sampling are adaptively adjusted to collect biomedical data [21] and surveillance data [22].

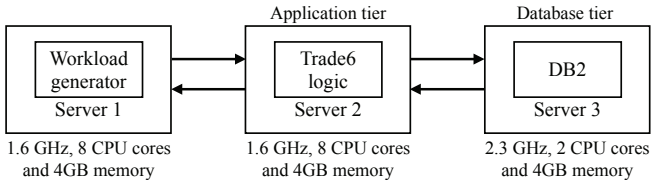
One of the strategies proposed by Warnell *et al.* for sampling surveillance videos bears some similarity to our work [23]. Cross-validation error is used for sparsity estimation. However, only the sparsity of the foreground in each frame needs to be estimated, and the foreground data can be fitted approximately by Gaussian distribution. The data used in this paper, however, cannot be easily fitted into any known distributions. Moreover, the scheme proposed in this paper accomplishes sparsity prediction using Kalman filter in addition to sparsity estimation.

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

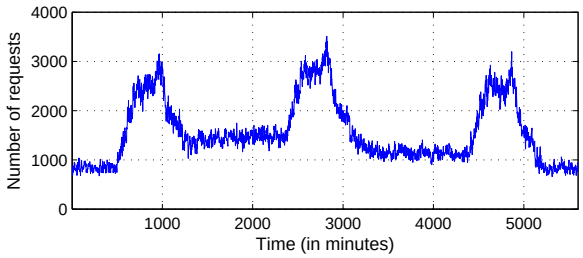
III. Experimental Setting

Fig. 1(a) shows the system used in our experiments, comprising three servers networked via a gigabit switch. Virtualization of this system is enabled by VMWare's ESX Server running a Linux RedHat kernel. The operating system on the virtual machine (VM) is the SUSE Enterprise Linux Server Edition. The system hosts IBM's Trade6 benchmark, a stock-trading application which allows users to browse, buy, and sell stocks. Users can perform dynamic content retrieval as well as transaction commitments, requiring database reads and writes, respectively. The application logic for Trade6 resides within the IBM WebSphere Application Server, which in turn is hosted by the VM on the server within the application tier. The database component is DB2, hosted on the server running SUSE Enterprise Linux. The database maintains 500 user accounts and information for 3500 stocks.

We use httperf [24], an open-loop workload generator, to send a mix of buy/browse transactions to the Trade6 application over a period of 48 hours. The workload traces are synthesized to reflect realistic operating scenarios such as time-of-day variations as well as bursty traffic where request



(a) The Trade6 application.



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

Fig. 1. (a) The overall system architecture hosting the Trade6 application which implements a stock-trading service that allows users to browse, buy, and sell stocks. (b) Example of workload provided to the testbed in our experiments, plotted in granularity of 30 seconds.

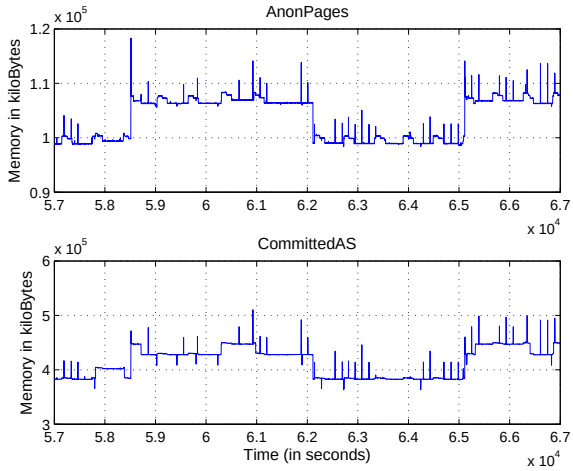


Fig. 2. Measurements corresponding to the AnonPages and CommittedAS signals collected over a 48-hour operating period.

rates vary significantly within short time periods. A sample workload is shown in Fig. 1(b), having an average arrival rate of 50 requests per second with a 50/50 mix of buy/browse transactions. Each data point in the figure represents the aggregated workload in a 30-second interval.

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

- *Free memory*. This quantity, termed MemFree, reflects the amount of physical memory left unused in the system.
- *Committed memory*. This quantity, termed CommittedAS, reflects the total amount of memory allocated by processes in the system using `malloc()` calls, even if the memory has not been used by them as of yet. For example, a process may allocate 1 GB of memory but only touch 100 MB of it. Although the current memory usage is only 100 MB, the 1 GB allocation is memory

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

TABLE I
THE LIST OF SYMBOLS USED IN THE PAPER AND THEIR DEFINITIONS.

Notation	Definition
t	Index of time window.
$\mathbf{d}_t$	Data to be sampled within time window t.
N	Length of the acquired data $\mathbf{d}$ in each time window.
$B^\star$	Representation basis selected by the Best Basis algorithm.
$\mathbf{x}_t$	Representation of $\mathbf{d}_t$ in basis B.
M_t	Length of collected samples in time window t.
$\mathbf{G}_t$	Sampling matrix of size $M_t \times N$.
$\mathbf{y}_t$	Compressed samples of $\mathbf{d}_t$ using $\mathbf{G}_t$.
$\hat{s}_t$	The *a priori* sparsity estimate for time window t.
$\tilde{s}_t$	The *a posteriori* sparsity estimate for time window t.
e_t^{cv}	Cross-validation error of time window t.

that has been committed by the memory subsystem to the process and can be used at any time by the process.

- *Page tables*. This quantity, termed PageTables, reflects the amount of memory dedicated to the lowest level of page tables.
- *Anonymous pages*. This quantity, termed AnonPages, tracks the amount of non-file backed pages mapped to page tables responsible for the user space.

In addition to these features, we also use disk I/O activity measurements (sectors read/written) as part of our evaluation experiments. Our goal is to show that the reconstructed signals can be used to detect various system-level faults. The above-listed features were chosen to support one of the case studies discussed in this paper: detection of memory leaks. Here we have chosen low-level metrics that are most likely impacted by this fault. Fig. 2 plots a subset of the features collected during an experimental run of the system lasting 48 hours. The data points are sampled once every two seconds.

IV. COMPRESSIBILITY OF SIGNALS

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

A. Sparse Representation of Signals

Denote the data to be sampled as $\mathbf{d}$, a vector of length N, and its representation in basis $\mathbf{B}$ as $\mathbf{x}$. In other words, $\mathbf{d} = \sum_{i=1}^{N} x_i \mathbf{b}_i = \mathbf{B}\mathbf{x}$, where $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_N]$. Here $\mathbf{b}_i$ denotes the i^{th} column in the $N \times N$ matrix $\mathbf{B}$. For example, if $\mathbf{B}$ is selected to be the Haar wavelet basis, the elements of the vector $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ are coefficients of wavelet decomposition for signal $\mathbf{d}$. Also, if at most S entries in $\mathbf{x}$ are nonzero, then $\mathbf{x}$ is called an S-sparse vector; if S is small, $\mathbf{d}$ is said to be sparsely represented in the basis $\mathbf{B}$.

As possible basis functions, we consider the following members of the Daubechies wavelet family that can capture signal characteristics in both time and frequency domains: db1 (also known as Haar), db2, and db4 wavelet basis.[1] These

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

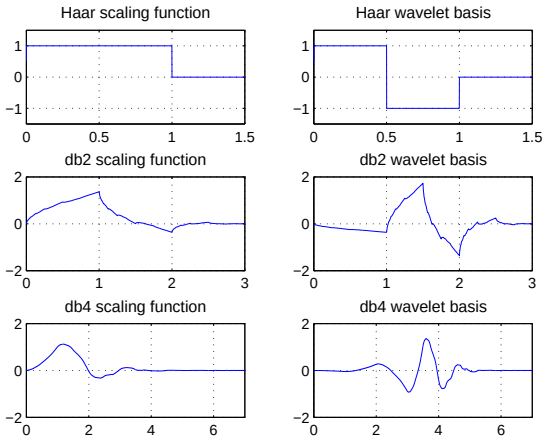


Fig. 3. Waveforms corresponding to three members of the Daubechies wavelet family. The Haar is the simplest wavelet that captures discontinuities in the data; db2 and db4 also show similarity with our data but have longer waveforms, leading to better frequency resolution.

TABLE II
PERCENTAGE OF COEFFICIENTS NEEDED TO KEEP THE RELATIVE ERROR WITHIN 1% UNDER EACH REPRESENTATION BASIS.

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

wavelets are able to capture sharp or abrupt changes in the signal. Fig. 3 shows the waveforms for these wavelets. We refer the reader to Walker for a primer on wavelets and their scientific applications [25]. We analyze the basis functions in terms of how concisely they encode the data collected from our system. We first perform a signal transform on each data set to find the corresponding coefficients within the chosen basis and sort them in decreasing order of their magnitude. Then, we use the n largest coefficients where $1 \leq n \leq N$, set all the other coefficients to 0, and reconstruct a new signal $\tilde{\mathbf{d}}(n)$. A relative error metric captures the difference between the original and reconstructed signals as $e(n) = \|\tilde{\mathbf{d}}(n) - \mathbf{d}\|/\|\mathbf{d}\|$, where $\| * \|$ denotes the l_2 norm. Finally, we examine how this relative error changes as the value of n is increased. Table II summarizes the percentage of coefficients needed to maintain the relative error within 1% for each of the bases. In this respect, the Haar wavelet represents our data most concisely.

The foregoing discussion makes it clear that a signal can be sparsified when expressed under a different basis—that is, the transformed signal contains many coefficients close to or equal to zero. This is the insight used by both the C-MON and CS-MON approaches. We now develop a basis selection strategy that automatically adapts the representation basis to the structure of the sampled signal such that it can be most concisely represented.

B. The Best Basis Algorithm

Given a set of possible basis functions, we use a thresholding technique to find the basis in which the signal is most sparse [26]. Assume we have a choice between K possible bases and let the representation of the signal in the k^{th} basis $\mathbf{B}_k$ be $\mathbf{x}^{(k)} = [x_1^{(k)}, \cdots, x_N^{(k)}]$ where $x_i^{(k)}$ is the value of the i^{th}

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

coefficient under basis k. Let the magnitude of the $(M+1)^{\text{th}}$ largest coefficient in $\mathbf{B}_k$ be t_k, and denote the recovered signal using coefficients with magnitude larger than t_k in basis $\mathbf{B}_k$ as $\mathbf{d}_k$. Let us define a soft-thresholding function as

$$\gamma_{t_k}(x) = \begin{cases} x^2/2, & \text{if } x < t_k \\ t_k x - {t_k}^2/2, & \text{otherwise.} \end{cases} \quad (1)$$

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

$$R(\mathbf{B}_k) = \sum_{i=1}^{N} \gamma_{t_k}(|x_i^{(k)}|) \quad (2)$$

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

$$\frac{1}{2} \parallel \mathbf{d} - \mathbf{d}_k \parallel^2 + t_k \parallel \mathbf{B}_k^T \mathbf{d}_k \parallel_0 \quad (3)$$

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

$$\mathbf{B}^* = arg \min_{k=1,\cdots,K} R(\mathbf{B}_k) \quad (4)$$

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

C. The C-MON Strategy for Signal Compression

The foregoing discussion suggests a straightforward way of using signal compressibility to reduce the data transfer and storage costs associated with online monitoring. Under C-MON, we collect N samples for each signal of interest locally at each server or VM at some user-specified sampling rate. After the best-basis transformation as per (4), say using wavelet decomposition, on these measurements, we obtain a set of N coefficients for the signal and choose the M largest coefficients in terms of magnitude, where $M \ll N$. These M coefficients along with their positions within the larger set of N coefficients are sent to the monitoring station where an approximation of the original measurements is recovered. The drawback with C-MON is the CPU overhead imposed on the local machine due to the sampling and compression process: once the N measurements are obtained, the best-basis algorithm must be invoked followed by a sorting routine to arrange the coefficients in order of decreasing magnitude. This overhead, quantified in Section VII, can interfere with the execution of other applications running on the server.

V. Compressive Sampling of System Measurements

When the signal can be represented sparsely in an appropriate basis, it can be acquired from the system *directly* in a compressed form rather than first collecting a number of samples and then compressing them, as was done in C-MON. This section familiarizes the reader with the concept of *incoherence*, a key condition underlying compressive sampling that affects the way we sample signals. We then discuss how the original signal is recovered from a small set of samples.

A. 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:

$$\mu(\Psi, \Phi) = \sqrt{N} \max_{1 \le k,j \le N} \left| \langle \phi_k, \psi_j \rangle \right|, \quad (5)$$

where $\langle \phi_k, \psi_j \rangle$ is the dot product of the vectors ϕ_k and ψ_j [6]. Typically the coherence between the two bases lies between 1 and $\sqrt{N}$, and when the value of coherence is small we consider the two bases to be uncorrelated or incoherent. When the sensing and representation bases are uncorrelated, a spike in one basis will be represented as a spread-out waveform in the other. This property allows us to capture the complete information present in the original data using a small number of samples obtained by incoherent sampling.

As a sampling strategy to collect measurements from our testbed, we choose Gaussian random matrices that have a low coherence of $\sqrt{2 \log N}$ relative to any representation matrix with high probability. Prior to sample collection, we generate an $M \times N$ Gaussian random matrix $\mathbf{G}$ as the underlying sampling matrix. Elements in the matrix are independently chosen from a standard Gaussian distribution of zero mean and variance $1/M$. To obtain the samples from the input data, we simply multiply this matrix $\mathbf{G}$ by the data vector $\mathbf{d}$. For example, assume the data to be sampled is an $N \times 1$ vector

$$\mathbf{d} = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,N} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ B_{N,1} & B_{N,2} & \cdots & B_{N,N} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \mathbf{Bx},$$

where $\mathbf{B}$ is an $N \times N$ matrix corresponding to the Haar wavelet basis and $\mathbf{x}$ is the representation of $\mathbf{d}$ in that basis. Suppose we wish to obtain a $M \times 1$ vector of samples $\mathbf{y}$. The data is multiplied with a $M \times N$ Gaussian matrix $\mathbf{G}$ such that $\mathbf{y} = \mathbf{Gd} = \mathbf{GBx} = \mathbf{Ax}$, where $\mathbf{A} = \mathbf{GB}$ is a $M \times N$ matrix.

B. Recovering the Original Signal

To reconstruct the original data $\mathbf{d}$, we must solve this inverse problem: given a vector $\mathbf{y}$ of length M and matrix $\mathbf{A}$ of size $M \times N$ where $M \ll N$, find a sparse vector $\tilde{\mathbf{x}}$ of length N such that $\mathbf{y} = \mathbf{A}\tilde{\mathbf{x}}$—that is, we are looking for $\tilde{\mathbf{x}}$ as a solution to

$$\min_{\mathbf{b} \in \mathcal{R}^N} \parallel \mathbf{b} \parallel_0 \quad \text{subject to: } \mathbf{y} = \mathbf{Ab}, \quad (6)$$

where $\|\mathbf{b}\|_0$ is the l_0 norm of $\mathbf{b}$, i.e. the number of nonzero entries in $\mathbf{b}$. This problem is under-constrained since the matrix $\mathbf{A}$ has more columns than rows; there are infinitely many candidate signals $\mathbf{b}$ for which $\mathbf{Ab} = \mathbf{y}$. To solve this under-determined system, the constraint of sparsity is added, allowing only solutions which have a small number of nonzero coefficients. If there is a unique sparse solution, then the CS framework allows the recovery of that solution.

Since minimizing the l_0 norm is a computationally expensive nonlinear optimization problem, a class of reconstruction algorithms called basis pursuit or iterative hard thresholding

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

pursuit (HTP) is used. Here the problem is recast as one of minimizing the l_1 norm, which is the LP problem:

$$\min_{\mathbf{b} \in \mathcal{R}^N} \|\mathbf{b}\|_1 \quad \text{subject to: } \mathbf{y} = \mathbf{Ab}. \tag{7}$$

We use the iterative hard thresholding pursuit technique previously proposed by Foucart [10] to solve (7). If $\mathbf{b}^{(k)}$ denotes the S-sparse vector obtained after the k^{th} iteration, this method performs the following two steps to obtain the next approximation of $\mathbf{b}$:

1) Perform the operation $\mathbf{b}^{(k)} + \mathbf{A}^T(\mathbf{y} - \mathbf{Ab}^{(k)})$ and identify the indices of the S largest coefficients in the resulting $N \times 1$ vector; store these indices in a set $I^{(k+1)}$.
2) Solve $\mathbf{b}^{(k+1)} = \arg\min_{\mathbf{b}^N \in \mathcal{R}} \|\mathbf{y} - \mathbf{Ab}^{(k)}\|_2$, $\text{supp}(\mathbf{b}) \subseteq I^{(k+1)}$. Here, only the coefficients residing in the indices previously stored in $I^{(k+1)}$ are tuned so as to minimize the cost function. The other coefficients are zeros.

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

$$M \geq C\mu(\Psi, \Phi)^2 S \log \frac{N}{\delta}, \tag{8}$$

where δ is a small constant and C is some positive constant [27]. The implication of (8) is that when the coherence between the representation and sensing bases, μ, as well as the sparsity metric, S, are small, we need only a few samples to recover the original signal exactly with high probability.

C. A CS-based Online Monitoring Strategy

Fig. 4 shows the implementation of the CS-based method within each local server in which the incoming signal $\mathbf{d}$ is acquired directly in a compressed form $\mathbf{y}$. When a new data item $d(t)$ arrives at time t it is multiplied by the entries in the sampling matrix $G(j,t)$, $j = 1, \ldots, M$ and the partial products are accumulated into $y(j)$. After a period of length $N \times T$, where T is the sampling period, the current values of $y(j)$ are sent out as the M samples to the monitoring station, and then reset back to zero. Effectively, $y(j) = \sum_{t=1}^{N} G(j,t)d(t)$, where $j = 1, \ldots, M$, and thus $\mathbf{y} = \mathbf{Gd}$.

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

VI. Adaptive-rate compressive sampling

The previous section assumes that the signal sparsity when represented in the underlying basis is constant or bounded across time windows. Under this assumption, we can use a fixed rate for sampling the signal. This assumption of a constant signal sparsity within each time window, however, rarely holds in practice. For example, Fig. 5 plots the sparsity level in terms of the number of coefficients needed in the Haar wavelet basis to capture 99% of the information contained within the CommittedAS and AnonPages signals (previously shown in Fig. 2) within different time windows.

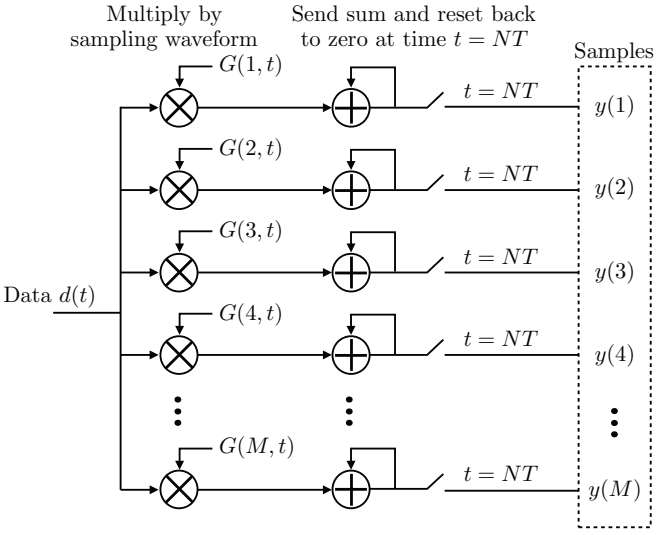


Fig. 4. Implementation of compressive sampling in our system that takes N data items over a time period as input and returns M samples, where $M \ll N$.

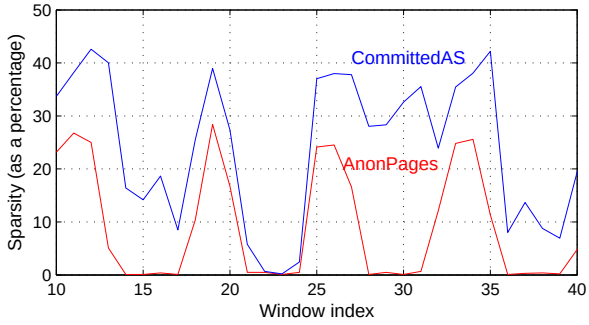


Fig. 5. Change in the sparsity level over time for the AnonPages and CommittedAS signals.

The sparsities change significantly over time. So a fixed-rate sampling strategy based on an upper bound on the sparsity level—between 45-50% in this particular case—will collect 100% of the samples at all times. We can perform much better in terms of reducing the number of samples needed by dynamically changing the sampling rate as the signal sparsity changes. This section adds a rate adjustment component to the aforementioned CS strategy. We show that by collecting some relevant side information from the signal—which could have been under or over sampled during any particular time window—we can estimate the underlying sparsity and predict it for some succeeding time windows. The number of samples collected over these windows can then be adjusted based on the predicted sparsity values. The goal is to achieve an overall reduction in sample size without compromising the reconstruction quality specified in terms of relative error.

A. Overview of the CS-MON Strategy

Fig. 6 outlines the CS-MON strategy for adaptive-rate sampling, comprising three major algorithmic components: a compressive sampling block; a cross validation block to estimate the signal sparsity; and a Kalman filter block to predict the sparsity over future time windows. The side information is collected by the cross validation block once every K time

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

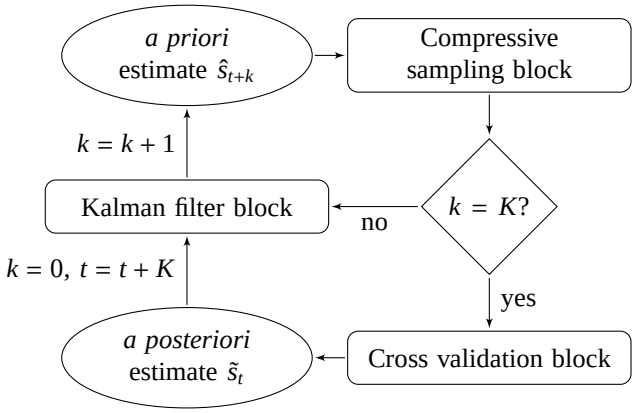


Fig. 6. Workflow for the adaptive-rate compressive sampling strategy. The workflow comprises three major steps: compressive sampling, cross validation, and prediction of signal sparsity.

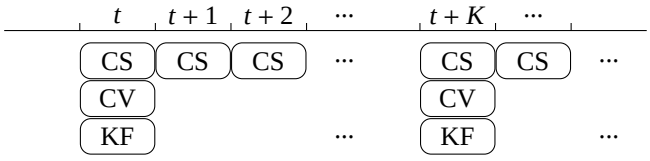


Fig. 7. The timeline for the execution of each block in the adaptive-rate sampling strategy. Here CS denotes the compressive sampling block, CV, the cross validation block, and KF, the Kalman filter.

windows and the sparsity is predicted for every time window.

As per Fig. 6, an *a priori* estimate of the signal sparsity within the t^{th} time window, $\hat{s}_t$, is obtained (or initialized at start up) and the sampling rate is appropriately adjusted for the subsequent time window based on this estimate. Knowing $\tilde{s}_t$, we estimate the sparsity for the next K time windows, $\hat{s}_{t+1}, \hat{s}_{t+2}, \ldots, \hat{s}_{t+K}$, using a Kalman Filter. These serve as the *a priori* estimates for time windows $(t+1)$ through $(t+K)$ and determine the compressive sampling rates within each of the windows. Once every K time widows, cross-validation measurements are collected from the signal and used to generate *a posteriori* estimate of the sparsity, $\tilde{s}_t$, via maximum-likelihood estimation. This estimate forms the basis for the next set of predictions.

Fig. 7 shows the execution timeline for each block in our system. The cross-validation and the Kalman filter block are executed once every K time windows whereas the compressive sampling block is executed every time window.

B. The Compressive Sampling Block

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

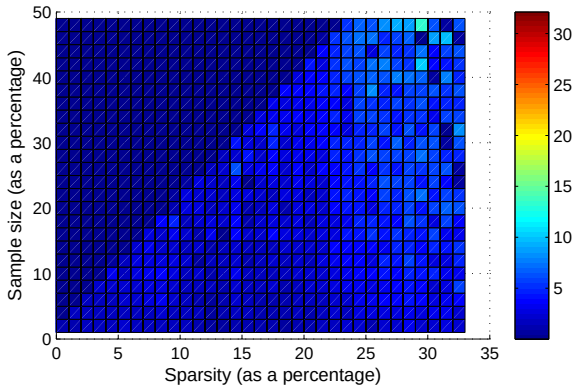


Fig. 8. The relative error achieved by various sample sizes, given different sparsity levels within the AnonPages signal. Best viewed in color.

Given that signal sparsity can be estimated at each time window, we can use this information to adjust the sample size such that the desired reconstruction quality is achieved. To achieve this objective, a quantitative relationship between sparsity and sample size needs to be established—empirically, in our case. Consider the AnonPages signal whose sparsity varies over time (see Fig. 5). Here, sparsity is defined as the percentage of coefficients that capture 99.5% of the original data. For data belonging to each sparsity level, we try different sample sizes, reconstruct the data and evaluate the result using the relative error metric. Fig. 8 shows the relative error achieved using various sample sizes, given different sparsity levels within the AnonPages signal. As expected, portions of the signal which are less sparse require a larger sample size to achieve the same relative error. The figure also implies a linear relationship between sparsity and sample size: for data of length N with sparsity s, a sample size of $M(s) = 5s$ guarantees a reconstruction error below 5% with confidence exceeding 93%. Therefore, if $\hat{s}_t$ is the sparsity for time window t, the corresponding sample size $M_t(\hat{s}_t)$ is chosen as $5\hat{s}_t$.

C. The Cross-Validation Block

Recall from Fig. 6 that the actual sparsity of the underlying data being sampled must be measured periodically, as best as possible, so that an *a posteriori* estimate can be obtained to update the Kalman filter. The cross validation block, executed once every K windows, serves this purpose by performing the following operations:

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

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

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

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

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

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

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

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

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

The relative error between the original and the reconstructed signals is determined by the exact sparsity and the value assumed during the signal recovery process. First we show that the relative error obtained using the cross-validation measurements is a good approximation of the optimal relative error between the original and reconstructed signals. Given the representation of the signal in the t-th time window, $\mathbf{x}_t$, and an arbitrary $\hat{s}$ as the assumed sparsity, we can define the optimal $\hat{s}$-sparse approximation error as $e_t^{\text{opt}}(\hat{s}) = \min_{\|\mathbf{x}\|_0 \leq \hat{s}} \| \mathbf{x}_t - \mathbf{x} \|$, where the search space includes all vectors with sparsity $\leq \hat{s}$. Let s be the exact sparsity that captures $1 - \epsilon$ of the signal. If $\hat{s} = s$, then the optimal $\hat{s}$-sparse approximation error $e_t^{\text{opt}}(\hat{s}) = \epsilon \| \mathbf{x}_t \|$, which is the upper bound on the absolute error. If $\hat{s} < s$, then $e_t^{\text{opt}}(\hat{s}) > \epsilon \| \mathbf{x}_t \|$ since the signal will be under-sampled, losing information in the process; similarly if $\hat{s} > s$, then $e_t^{\text{opt}}(\hat{s}) < \epsilon \| \mathbf{x}_t \|$ due to over sampling of the signal. The value of $e_t^{\text{opt}}(\hat{s})$ is indicative of the difference between the actual and assumed sparsities, and therefore we should be able to estimate s by observing $e_t^{\text{opt}}(\hat{s})$. Unfortunately, the $e_t^{\text{opt}}(\hat{s})$ values are unobservable due to the lack of ground truth for $\mathbf{x}_t$, since we do not sample at full rate when CS-MON is operating. However, we can show that the cross-validation error can be used to obtain a tight upper and lower bounds on $e_t^{\text{opt}}(\hat{s})$, thereby replacing $e_t^{\text{opt}}(\hat{s})$ during sparsity estimation. The cross-validation error e_t^{cv} is observable since both $\mathbf{z}_t$ and $\hat{\mathbf{z}}_t$ are known, and its relationship with $e_t^{\text{opt}}(\hat{s}_t)$,

$$\frac{1}{h(1+\epsilon)} e_t^{\text{cv}}(\hat{s}_t) \leq e_t^{\text{opt}}(\hat{s}_t) \leq \frac{1}{1-\epsilon} e_t^{\text{cv}}(\hat{s}_t) \tag{9}$$

holds with high probability. Here h is a constant and ϵ is the upper bound on the relative error. A direct consequence of (9) is that the observed $e_t^{\text{cv}}(\hat{s}_t)$ can be used in place of $e_t^{\text{opt}}(\hat{s}_t)$ for the sparsity estimation.

We now provide the proof that $e_t^{\text{opt}}(\hat{s}_t)$ can be bounded by the cross-validation error as stated in (9) using the Johnson-Lindenstrauss lemma [28]. First we define an approximation error that quantifies the reconstruction quality achieved by the compressive sampling block as

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

where $\hat{\mathbf{x}}_t$ is the representation of $\hat{\mathbf{d}}_t$ in basis $\mathbf{B}$. We can provide upper and lower bounds for $e_t^{\text{app}}(\hat{s}_t)$ in terms of $e_t^{\text{opt}}(\hat{s}_t)$ with high probability for some constant h [18]:

$$e_t^{\text{opt}}(\hat{s}_t) \leq e_t^{\text{app}}(\hat{s}_t) \leq h e_t^{\text{opt}}(\hat{s}_t) \tag{10}$$

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

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

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

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

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

$$\frac{1}{1+\epsilon} e_t^{\text{cv}}(\hat{s}_t) \leq e_t^{\text{app}}(\hat{s}_t) \leq \frac{1}{1-\epsilon} e_t^{\text{cv}}(\hat{s}_t). \tag{11}$$

Equation (9) follows directly from (10) and (11), proving that $e_t^{\text{cv}}(\hat{s}_t)$ can indeed be used to estimate the actual sparsity with high confidence.

[2]The cross-validation error only depends on the actual and the assumed sparsity values of the signal and not on the individual data values comprising the signal. The training dataset includes data corresponding to different sparsity levels and thus can be used to derive a general model for such dependency that is representative of all variations of data (which may vary from those in the training set). In other words, we sweep through the entire range of sparsity values to build our model using the training dataset. So, this model is representative of the sparsity values that one might see in the signals captured at run time.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

D. The Kalman Filter block

We treat the *a posteriori* estimates of the signal sparsity $\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_t$ obtained from the cross validation block as time-series data and use an appropriate forecasting model to estimate the sparsity value $\hat{s}_{t+1}$ for the next time window and beyond using these past observations. We use the Holt-Winter model which uses a moving average filter to capture the mean of the observed data and a slope component to capture the trend. A state-space form of this model is developed within the framework of a Kalman Filter. The prediction step of the filter uses the state estimate from the previous time step to produce an estimate of the state at the current time step. In the update step, the current prediction is combined with current observation $\tilde{s}_t$ to further refine the state estimate. Knowledge of the Kalman state is used to predict the future values. We refer the interested reader to Harvey for more details on how the Kalman Filter can be used for time-series forecasting [29].

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

E. Summary of the Adaptive-Rate Model Operation

Let the initial condition be that $\mathbf{d}_t$, the data to be sampled during time window t, has some unknown sparsity. To compressively sample this data, we assume its sparsity to be some initial value $\hat{s}_t$ and use an $M(\hat{s}_t) \times N$ matrix $\mathbf{G}(\hat{s}_t)$ to collect $M_t(\hat{s}_t)$ samples, resulting in measurements $\mathbf{y}_t(\hat{s}_t) = \mathbf{G}(\hat{s}_t)\mathbf{d}_t = \mathbf{G}(\hat{s}_t)\mathbf{B}\mathbf{x}_t$. Recall that $M(\hat{s}_t) = 5\hat{s}_t$. The data in this time window is then reconstructed as $\hat{\mathbf{d}}_t$ as described in Section V by the compressive sampling block.

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

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

VII. Performance Evaluation

We evaluate the performance of the C-MON and CS-MON strategies in terms of their efficacy in: (1) reducing the sample size while guaranteeing a specified signal reconstruction quality; and (2) using the recovered signals to detect threshold violations and trends that could be indicative of performance bottlenecks or degradation in the system. The CPU and storage overhead incurred by these methods is also quantified. The results reported here use data collected from our testbed, described earlier in Section III.

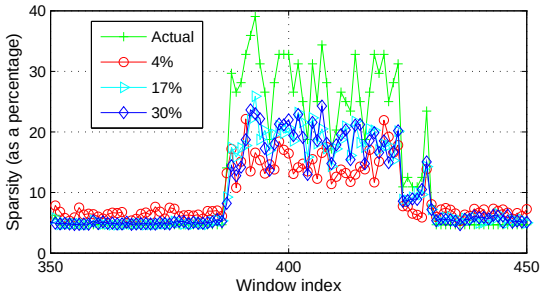


Fig. 9. Overlay of the actual sparsity of the AnonPages signal with *a posteriori* values $\tilde{s}$ obtained by the cross validation block. Here sparsity is defined as the number of coefficients needed to capture 99% of the original signal. The plots show the $\tilde{s}$ values obtained using the following initial guesses for the sparsity levels: 4%, 17%, and 30%.

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

- The datacenter operator wishes to detect performance-related bottlenecks or anomalies that manifest themselves as the magnitude of the signal exceeding some nominal threshold value.
- The operator wishes to detect the gradual performance deterioration associated with software aging by analyzing the signals for the existence of trends.

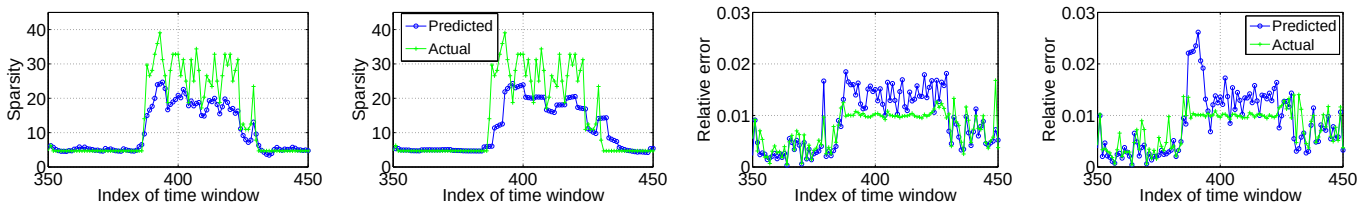
A. Signal Reconstruction Quality using CS-MON

We use the signals obtained from our testbed to evaluate the performance of the CS-MON strategy in terms of overall signal reconstruction quality, focusing on the performance of the key blocks comprising the workflow shown in Fig. 6.

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

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing



(a) Predicted sparsity, $K = 1$. (b) Predicted sparsity, $K = 4$. (c) Relative error, $K = 1$. (d) Relative error, $K = 4$.

Fig. 10. Predictions for sparsity values provided by the Kalman filter for the AnonPages signal when the filter's state is updated: (a) during each time window, that is $K = 1$, and (b) once every four windows, $K = 4$. The relative error between the actual and reconstructed signals when the filter state is updated each window and once every four windows is shown in (c) and (d), respectively.

Fig. 9 shows an overlay of the actual signal sparsity (the green plot) with the *a posteriori* values $\tilde{s}_t$ obtained by the cross validation block for various initial sparsity guesses. Values along the y-axis correspond to the number of coefficients needed to capture 99% of the AnonPages signal—our definition of sparsity here. The length of each time window is set to $N = 64$. Let us focus on the plot in red, illustrating the case in which during the start of each time window, the compressive sampling block reconstructs the underlying signal assuming a sparsity of 4%, that is $\hat{s}_t = 0.04N$. In this case, the *a priori* value $\hat{s}_t$ used for compressive sampling severely underestimates the actual sparsity during time windows 380 through 425. However, by the end of each of these time windows, the cross validation block is able to correct this estimate and obtain a reasonably close approximation of the actual sparsity. Also, the *a posteriori* values obtained under different *a priori* sparsity assumptions lie fairly close to each other, and they are not overly sensitive to the *a priori* value $\hat{s}_t$. However, we do observe that guesses closer to the actual value lead to more accurate estimates. The average estimation error is within 20% of the actual sparsity value regardless of the initial guess.

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

Figs. 10(a) and 10(b) show the Kalman predictions in relation to the actual sparsity for the AnonPages signal. As before, the window size is set to $N = 64$ and the sparsity is defined as the number of coefficients needed to capture 99% of the data. Fig. 10(a) shows the case in which the cross validation block is invoked every time window. Fig. 10(b) shows the case in which the frequency is once every four windows and the Kalman filter is required to predict the values for the windows in between. Since a linear model is used in our filter implementation, the prediction process underestimates the actual sparsity in cases of sudden fluctuations in the sparsity. However, our results show that the predictions track the overall trend exhibited by the actual values, and the average prediction error is within 20%.

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

TABLE III
THE AVERAGE SAMPLE SIZE USED BY CS-MON TO RECONSTRUCT THE ANONPAGES AND COMMITTEAS SIGNALS FOR DIFFERENT SPARSITY LEVELS.

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

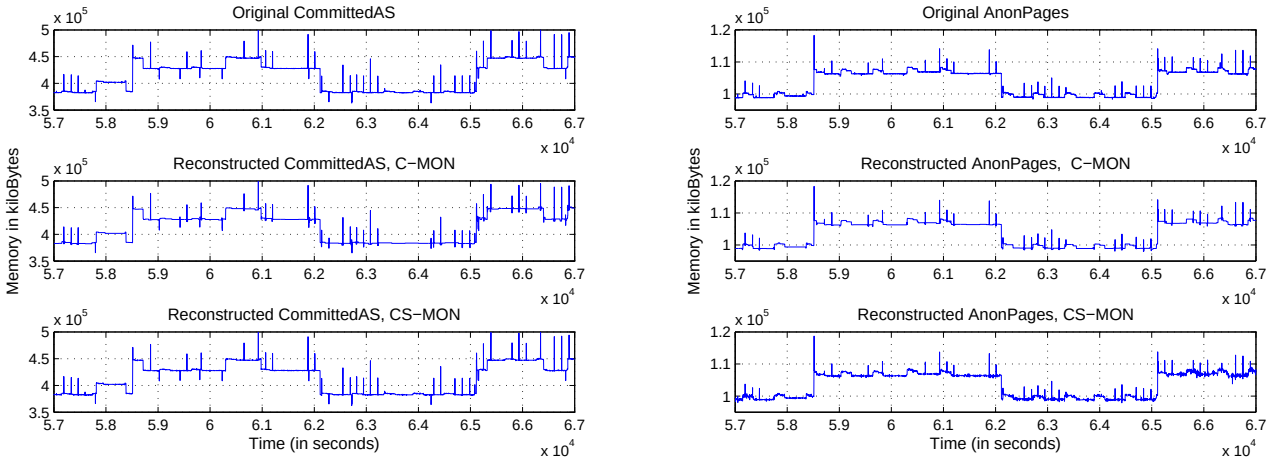
Note that in the ideal case, the relative error achieved as a result of using the actual sparsity is around 1%, which is to be expected since the sparsity accounts for 99% of the data. The relative error between the actual and reconstructed signals is shown in Figs. 10(c) and 10(d) when using the actual sparsity and the sparsity predictions provided by the filter, respectively. When sparsity is underestimated, the recovered signal incurs a higher relative error, whereas the error is low if more than the necessary number of samples are collected due to an overestimation of the sparsity. We observe that, although we underestimate the sparsity after a sudden increase in the actual value, the resulting reconstruction penalty is small: the relative error as a result of using predicted sparsity is rarely higher than 1.7%.

Reduction in sample size: Finally, we demonstrate that the sample size needed to reconstruct the signal while achieving the desired quality is substantially reduced using the adaptive-rate model. Our result shows that the adaptive-rate model achieves the same reconstruction quality while reducing the sample size by 70% when compared with fixed-rate compressive sampling.

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

Table III shows that the adaptive-rate strategy guarantees a reconstruction quality with 5% relative error using substantially small sample sizes for the signals considered in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing



(a) CommittedAS (b) AnonPages

Fig. 11. The original CommittedAS and AnonPages signals, and the same signals recovered by C-MON using 5% of the coefficients in the basis found by the best basis algorithm. The signal recovered by CS-MON with 15% error tolerance is also shown for comparison purposes.

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

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

Table III also shows that to achieve a low error tolerance, more samples are needed. We also observe a difference in sample size when K, the frequency at which the cross validation block is invoked, is set to one versus four time windows. This difference is primarily caused by prediction results that either under or overestimate the signal sparsity. For example, when $K = 4$, the Kalman filter provides sparsity predictions without any input from the cross validation block for three time windows leading to prediction errors—mostly underestimating the sudden increase in the sparsity. Consequently, the under-sampling during those time windows results in a smaller sample size.

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

TABLE IV
RELATIVE ERROR BETWEEN THE ORIGINAL ANONPAGES AND COMMITTEDAS SIGNALS, AND THE CORRESPONDING RECONSTRUCTIONS, ACHIEVED BY C-MON.

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

B. Comparing the CS-MON and C-MON Strategies

We now compare the adaptive sampling strategy to an alternative method also aimed at compressing the data collected at the server—the C-MON strategy discussed in Section IV. We find that for varying values of N, C-MON compresses the signal much better than CS-MON, resulting in a lower data transfer overhead to the monitoring station. Table IV summarizes the relative error achieved by C-MON as a function of sample size. Here the original data set comprises $N = 64$ samples and the signal is reconstructed using M coefficients from the wavelet decomposition. The results show that the Best Basis (BB) algorithm successfully identifies the basis under which the signal can be most concisely represented—which happens to be the Haar wavelet for the AnonPages and CommittedAS signals—thereby achieving good reconstruction quality while using very few coefficients. Fig. 11(a) plots the CommittedAS signal recovered by C-MON using 5% of the coefficients. The CS-MON reconstruction with 15% error tolerance is also shown for comparison purposes. Fig. 11(b) shows the same set of results for the AnonPages signal. One may conclude from the results summarized in Tables III and IV that C-MON exploits the compressibility of the underlying signals better than CS-MON.

We quantify the computation and storage costs imposed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

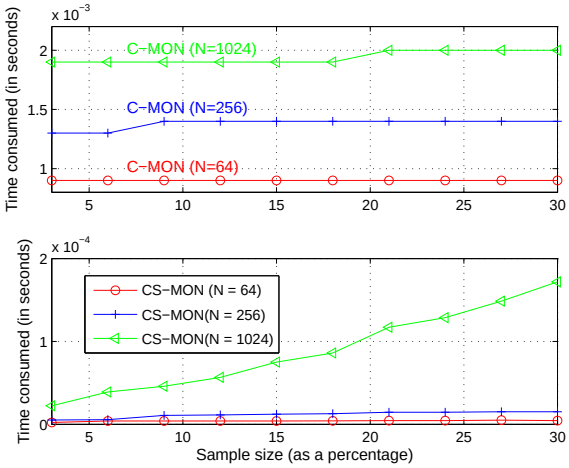


Fig. 12. The CPU overhead incurred by the sampling and encoding processes associated with the C-MON and CS-MON strategies on the local machine. The signal reconstruction overhead incurred by the monitoring station is not included here. Both strategies were implemented in MATLAB and executed on an AMD Athlon II 3.0 GHz processor.

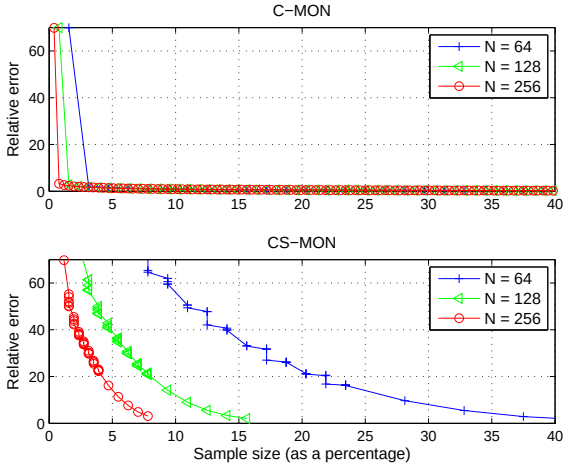


Fig. 13. The relative error achieved by C-MON and CS-MON when using smaller measurement windows.

TABLE V
STORAGE COSTS INCURRED BY C-MON AND CS-MON UNDER DIFFERENT SPARSITY LEVELS WHEN SAMPLING THE ANONPAGES SIGNAL.

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

on the local server by the two strategies to achieve their respective compression levels. Fig. 12 quantifies the CPU overhead incurred on a per-window basis by the sampling and encoding processes associated with C-MON and CS-MON on the local machine for various values of N. C-MON incurs a higher CPU overhead since the encoding process requires a wavelet transformation on the N values followed by a sorting routine to extract the largest M coefficients. In the CS-MON implementation, the process of incoherent sampling is simply the multiplication of an $M \times N$ matrix with an $N \times 1$ vector, as shown in Fig. 4. So, CS-MON incurs considerably less CPU overhead, running about an order of magnitude faster than C-MON. Moreover, since this overhead is incurred on a per-signal basis, CS-MON can be quite CPU-efficient when monitoring large numbers of signals on a single server.

C-MON also incurs an increased storage cost over CS-MON, needing a buffer size of $O(N)$ to accommodate N data items whereas a buffer size of $M \leq N$ is required for CS-MON. Table V shows the storage overhead when using C-MON and CS-MON on the AnonPages signal for a window size of $N = 64$. The overhead due to C-MON includes the buffers needed to store the measurements obtained using the length-N time window and to store the sorted coefficients after the wavelet transformation in the Daubechies basis. However, if the data is transformed in the Haar basis, this can be implemented entirely in place—on the measurement-buffer itself. So, the storage required in this specific case is around $0.5KB$. On the other hand, the CS-MON implementation shown in Fig. 4 requires a buffer to simply store the compressed data that is generated as the measurements stream in.

TABLE VI
THE PACKETIZATION DELAY INCURRED IN SECONDS FOR VARIOUS LENGTHS OF THE MEASUREMENT WINDOW AND SAMPLING RATE.

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

Another equally interesting aspect to consider is how the size of the measurement window—set to $N = 64$ to obtain the results reported in Tables III and IV—affects the monitoring operation. We test C-MON and CS-MON on measurement windows of various lengths and plot, in Fig. 13, the relative error achieved by these two strategies as a function of sample size for small measurement windows for the AnonPages signal. The advantage of C-MON and CS-MON in exploiting the compressibility of the signal shrinks with window size.

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

The foregoing discussion exposes some interesting tradeoffs between the C-MON and CS-MON strategies, particularly those related to the choice of the measurement window size N. In choosing the appropriate N, one must consider the tradeoff between the CPU and storage overhead due to these strategies and the resulting network traffic. If responsive operation is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

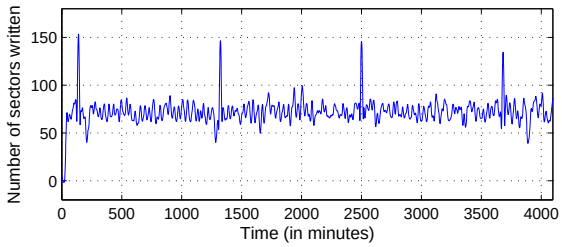


Fig. 14. The write_activity signal collected from the testbed that measures the number of disk sectors written.

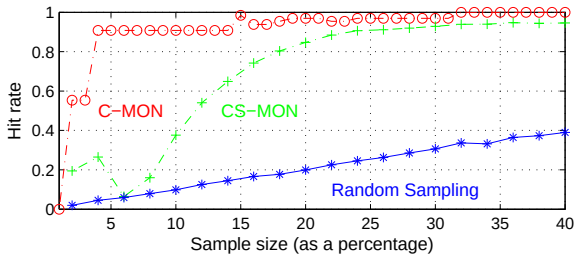


Fig. 15. Hit rate when detecting a 1.6% violation in the *write_activity* signal, achieved by C-MON, CS-MON, and random sampling. The length of the measurement window is $N = 64$.

desired then N must be small. However, when monitoring the system for faults that manifest themselves over many minutes or hours, such as memory leaks, or with the intent of simply logging the data for off-line analysis, N can be made larger to better exploit the compressibility of the underlying signals.

C. Case Studies

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

Detection of Threshold Violations. As discussed earlier, there are situations in which it is not essential to recover the original signal exactly. If the datacenter operator is mainly interested in detecting performance bottlenecks, hot spots, or anomalies affecting the computing system that manifest themselves as abrupt changes in the signals being monitored, it is more important that the reconstructed signal preserve these characteristics. We evaluate how well threshold violations are detected by C-MON and CS-MON using as an example the *write_activity* signal shown in Fig. 14. This signal was collected from our testbed during an experimental run and the signal measures the number of sectors written to the hard disk. Fundamentally, we are interested in how well the reconstructed *write_activity* signal preserves the abrupt changes and spikes present in the original. More formally, if $\mathbf{d}$ is the signal of interest, we consider $\mathbf{d}(i)$, the i^{th} data point within the signal, a p% violation if it satisfies $\mathbf{d}(i) \geq Q(\mathbf{d}, 100 - p)$, where $Q(\mathbf{d}, 100-p)$ is the $(100-p)$-th percentile of observations in $\mathbf{d}$. We use a *hit-rate* metric to characterize the number of abrupt changes that can be detected using the reconstructed signal by defining a hit as follows: at time t within the original and recovered signals, a spike occurring in the recovered signal matches a similar spike in the original signal.

Fig. 15 shows the hit rate achieved when detecting a 1.6% violation in the *write_activity* signal as a function of the sample size used to recover this signal. The baseline method used for comparison is random sampling and its hit rate is linear with the sample size, as expected. The hit rate achieved by CS-MON is significantly better than random sampling at the same sampling rate: greater than 90% when using 25% of the original samples, for example. We observe that the spikes in the original write_activity are underestimated if a smaller sample size is used, leading to a lower hit rate. We also observe that the sample size required to detect these spikes is much lower than the sparsity of the write_activity signal, which is around 50% when the window length is 64. This result shows that when it is not essential to recover the original signal exactly, using a sample size that is smaller than the sparsity still enables us to preserve abrupt changes of the signal. In summary, both C-MON and CS-MON achieve a hit rate greater than 90%, with C-MON able to better compress the signal, using about 4% of the original signal. Note, however, that the previously discussed tradeoff between C-MON and CS-MON, in terms of data transfer, CPU, and storage overhead, applies in this scenario as well.

Detection of Trends. In the second case, we evaluate C-MON and CS-MON for trend detection, which can help detect, for example, whether a system resource is slowly being consumed to exhaustion. We use a long-running Trade6 application executing over a period of 48 hours and inject a small memory leak of about 100 KB/minute at around the 24-hour mark.

To simplify the problem, we use time bins that each includes 1024 data points in 34 minutes. Sampling and reconstruction are applied on each time bin separately. To estimate the global trend, we use a 24-hour sliding window, which includes 40 time bins, and move the sliding window by one time bin at each step. We use the linear model $d = a \times t + b$, where d is the average of reconstructed data points within each time bin, t is the beginning time of the sliding window, a is the slope, and b is the intersection. For each sliding window, 40 (d, t) pairs are applied to this model and the slope within each sliding window is estimated along with the 95% confidence interval.

Fig. 16 shows the estimated global slope for CommittedAS, for the first 24 hours followed by the second 24 hours, indicating that both C-MON and CS-MON perform better than random sampling: the width of the 95% confidence interval is tighter than the one obtained using random sampling, indicating greater certainty in detecting the slope. We also observe an increasing trend in the estimated slope. For CommittedAS, the estimate is around 0 at the beginning indicating no increasing trend associated with this feature. Then, the slope values increase and the lower bound on their 95% confidence interval becomes positive, indicating an increasing trend by the end of the experiment.

VIII. Conclusions

The adaptive sampling strategy developed here exploits any available time-varying sparsity information within the underlying signal to reduce the number of samples collected when compared to a fixed-rate scheme. The reconstructed signal adequately preserves properties in the original signal that are useful for performance management and anomaly detection. The sensing scheme has low computation and storage

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2603473, IEEE Transactions on Cloud Computing

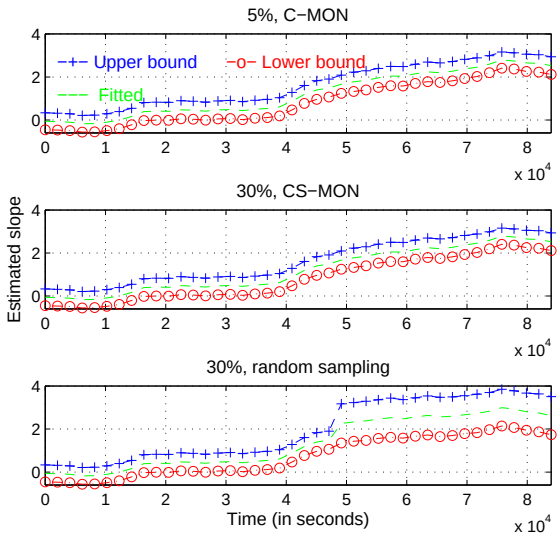


Fig. 16. The slope estimated by C-MON, CS-MON and random sampling over 40 time windows for CommittedAS.

complexity, making it suitable for use on the local server. It is also universally applicable—simply measuring signals via a random Gaussian sensing matrix—and does not have to be tailored in any way to the type of signal being measured or the type of anomaly being detected.

Finally, though the proposed sampling strategies substantially reduce data transfer costs, the burden of reconstructing the signal falls on the monitoring station. In recent work, however, we have shown that the data samples acquired via compressive sampling preserve, in an approximate form, properties such as mean and variance, as well as spectral properties such as correlation between data points [30]. So, instead of having to reconstruct the original signal, this result points to the feasibility of performing anomaly detection directly on the compressed samples themselves.

References

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

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

[3] G. Lanfranchi *et al.*, "Toward a new landscape of systems management in an autonomic computing environment," *IBM Syst. J.*, vol. 42, no. 1, pp. 119–128, 2003.

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

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

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

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

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

[9] D. Donoho, "Compressed sensing," *IEEE Trans. Inform. Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

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

[11] V. Castelli *et al.*, "Proactive management of software aging," in *IBM J. Res. Develop.*, vol. 45, no. 2, 2001, pp. 311–332.

[12] L. Li, K. Vaidyanathan, and K. Trivedi, "An approach for estimation of software aging in a web server," in *Proc. Symp. Empirical Softw. Eng.*, 2002, pp. 91–100.

[13] 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.

[14] T. Huang, N. Kandasamy, and H. Sethu, "Evaluating compressive sampling strategies for performance monitoring of data centers," in *IEEE/IFIP Netw. Oper. Manag. Symp. (NOMS)*, 2012, pp. 655–658.

[15] ——, "Evaluating compressive sampling strategies for performance monitoring of data centers," in *Proc. IEEE/ACM Conf. Auton. Comput. (ICAC)*, 2012, pp. 201–210.

[16] T. Tuma, S. Rooney, and P. Hurley, "On the applicability of compressive sampling in fine grained processor performance monitoring," *Proc. IEEE Int'l Conf. Eng. Complex Comput. Syst. (ECCS)*, pp. 210–219, 2009.

[17] S. Meng *et al.*, "Volley: Violation likelihood based state monitoring for datacenters," in *IEEE Int'l Conf. Distrib. Comput. Syst. (ICDCS)*, 2013, pp. 1–10.

[18] R. Ward, "Compressed sensing with cross validation," *IEEE Trans. Inform. Theory*, vol. 55, no. 12, pp. 5773–5782, 2009.

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

[20] V. Stankovic, L. Stankovic, and S. Cheng, "Compressive image sampling with side information," in *IEEE Int'l Conf. Imag. Proces. (ICIP)*, 2009, pp. 3037–3040.

[21] A. Watkins *et al.*, "Adaptive compressive sensing for low power wireless sensors," in *ACM Great Lakes Symp. VLSI*, 2014, pp. 99–104.

[22] Y. Tan, X. Wang, and K. Lin, "Adaptive image sequence reduction in surveillance using region enhancement block compressive sensing," in *Proc. IEEE World Congress Intell. Control Autom.*, 2014, pp. 3375–80.

[23] G. Warnell, D. Reddy, and R. Chellappa, "Adaptive rate compressive sensing for background subtraction," in *Proc. IEEE Int'l Conf. Acoust., Speech, Sign. Proces. (ICASSP)*, 2012, pp. 1477–1480.

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

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

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

[27] E. J. Candès and J. Romberg, "Sparsity and incoherence in compressive sampling," *Inverse Prob.*, vol. 23, no. 3, pp. 969–985, 2007.

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

[29] A. C. Harvey, *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, 1990.

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

Tingshan Huang is a Senior Performance Engineer at Akamai Technologies, Inc. She received her B.Sc. in Information Engineering from Shanghai Jiao Tong University, Shanghai, China in 2009, and her Ph.D. in Electrical Engineering from Drexel University in 2015. Her research interests include adaptive sampling, data compression, and pattern modeling for performance monitoring and anomaly detection in networked systems.

Nagarajan Kandasamy received his Ph.D. in Computer Science and Engineering from the University of Michigan in 2003. His current research interests lie in the areas of performance management, parallel processing, and embedded and real-time systems.

Harish Sethu obtained his B.Tech. in Electronics and Communication Engineering from Indian Institute of Technology (IIT), Chennai, in 1988. He received his Ph.D. in Electrical Engineering from Lehigh University with a doctoral dissertation in computer engineering in 1992. Prior to joining Drexel University, he was an Advisory Development Engineer/Scientist at IBM Corporation. His current research and teaching interests lie in the areas of network science, web performance, web security, computer networks and data science.

Matthew C. Stamm received his Ph.D. from the University of Maryland in 2012. He teaches and conducts research on signal processing and information security with a focus on multimedia forensics and anti forensics.