

RainMon: An Integrated Approach to Mining Bursty Timeseries Monitoring Data

Ilari Shafer, Kai Ren, Vishnu Naresh Boddeti, Yoshihisa Abe, Gregory R. Ganger, Christos Faloutsos
Carnegie Mellon University

ABSTRACT

Metrics like disk activity and network traffic are widespread sources of diagnosis and monitoring information in datacenters and networks. However, as the scale of these systems increases, examining the raw data yields diminishing insight. We present RainMon, a novel end-to-end approach for mining timeseries monitoring data designed to handle its size and unique characteristics. Our system is able to (a) mine large, bursty, real-world monitoring data, (b) find significant trends and anomalies in the data, (c) compress the raw data effectively, and (d) estimate trends to make forecasts. Furthermore, RainMon integrates the full analysis process from data storage to the user interface to provide accessible long-term diagnosis. We apply RainMon to three real-world datasets from production systems and show its utility in discovering anomalous machines and time periods.

Categories and Subject Descriptors

K.6.2 [Installation Management]: Performance and usage measurement; H.2.8 [Database Applications]: Data mining

General Terms

Algorithms, Design, Management, Performance

Keywords

System Monitoring, PCA, Bursty Data

1. INTRODUCTION

Many modern computing clusters consist of dozens to thousands of machines that work together to perform a variety of tasks. The size and complexity of these systems has created a burden for administrators. Additionally, a move towards commodity hardware has produced more frequent failures that must be diagnosed and repaired [3]. These challenges have inspired considerable interest in monitoring tools catered towards system administrators, who are often also faced with monitoring external network links in addition to the datacenter itself.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD'12, August 12–16, 2012, Beijing, China.

Copyright 2012 ACM 978-1-4503-1462-6/12/08 ...\$10.00.

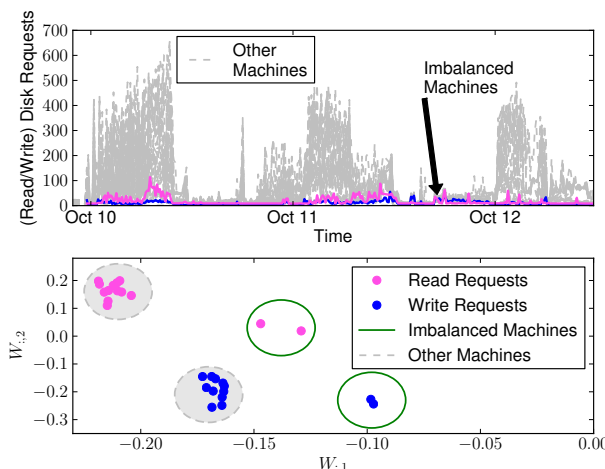


Figure 1: *Two anomalous machines discovered with RainMon.* The system can summarize bursty timeseries monitoring data (top plot) and allows easy discovery of groups of machines that share the same behavior: the four dots separated from the others in the bottom plot correspond to the imbalanced machines at top. Many tools used in current practice only offer the top view—without the helpful coloring. For details of the analysis, see Sec. 5.1.

Complicating matters further, analysis at the level of a single machine is often insufficient. To provide a concrete example that we analyze in this paper, applications like Hadoop [1] distribute work across multiple machines. Significant imbalances in monitoring observations between groups of machines are a problem worthy of human attention. A task of growing importance is understanding when problems like these occur, and determining which tasks, machines, or network links are responsible. Furthermore, administrators also would like to know about problems and potential capacity shortages as soon as possible [9], and predictions of future trends would be valuable.

To provide visibility into their behavior, systems emit a variety of timeseries streams — for example, CPU utilization, disk traffic, and network transfers. Such streams are widely used for datacenter monitoring through tools like Ganglia [24], Nagios [25], Zenoss [36], and Tivoli [13]. Typically, these tools provide summaries of performance through averages or roll-ups of streams across machines, or require administrators to look at machine status individually. Anomaly

and outlier detection are often accomplished by thresholding individual metric values.

Unfortunately, the increased number of incoming monitoring streams due to the scale of modern systems makes diagnosis with these tools difficult. For example, simply overlaying monitoring data from many machines produces an unintelligible view (for example, at top in Fig. 1), yet similar views are common in the tools used in practice [24, 25, 36]. Another significant obstacle to human-intelligible monitoring is the burstiness of many of these streams: not only is the raw data visually noisy, it also poses difficulties for many timeseries analyses, such as modeling techniques that assume a smooth time evolution of data [17]. For example, alerts based on thresholds can produce false positives.

Despite the wide variety of anomaly detection and summarization approaches that have been proposed (we survey a few in Sec. 2), there exists a need for approaches that handle real-world data sources, focus on bursty data, and integrate the analysis process. To meet those goals, RainMon is an *end-to-end* system for mining anomalies and trends from bursty streams, compressing monitoring data, and forecasting trends. We have integrated RainMon with multiple real data streams produced by complex real systems to produce insight into datacenter behavior. It has isolated problems with machines and tasks like the ones shown in Fig. 1 in the Hadoop framework and unearthed network glitches. It can compress data more effectively than a non-integrated approach and can estimate future state. These applications are not disjoint, but rather the result of judicious combination of a few techniques from the literature into a knowledge discovery tool.

Contributions: We make three primary contributions. First, we describe a novel multi-stage analysis technique catered towards bursty timeseries monitoring streams from datacenters and networks. Second, we show its utility through a series of case studies on real-world monitoring streams. Third, we describe our end-to-end system that incorporates storage, modeling, and visualization.

2. RELATED WORK

A variety of data mining techniques have been applied to timeseries monitoring, many existing monitoring tools provide data infrastructure, and some consider effective visualization of the output [8]. Here we focus on RainMon’s relation to the wide body of related work on stream anomaly detection and forecasting as applied to system monitoring. More background on the techniques we use is provided in Sec. 3; broader surveys of anomaly detection [7] and timeseries forecasting [6] are available.

Multiple data mining approaches have been proposed for summarizing relatively smooth portions of timeseries monitoring data. Patnaik et al. have developed an approach to cooling a datacenter based on finding frequent “motifs” [28]. The Intemon tool explores the use of dimensionality reduction to monitoring timeseries [12], and a case study considers applying the system to finding anomalies in environmental monitoring data. We use the same core algorithm (SPIRIT [27, 35]) as a technique for mining correlations and trends, and expand its applicability to cope with the bursty aspect of systems data. The symbolic SAX representation is also promising for anomaly detection and visualization [21].

Considerable work on automated detection of anomalies and bursts in timeseries data has resulted in a variety of tech-

niques, such as wavelet decomposition [37], changepoint detection [10, 26], incremental nearest-neighbor computation [4], and others. PCA and ICA have been applied to monitoring data for a variety of features (e.g., by [16]). Other forms of matrix decomposition have also been applied to network monitoring data to find anomalies, though evaluations often focus on small or synthetic datasets [11]. Many other automated approaches (e.g., [5, 14]) complement this work: rather than defining what constitutes a violation of a trend, we focus on modeling and presenting the data.

Forecasting of timeseries monitoring data has often been examined independently of the mining techniques above. For example, ThermoCast [18] uses a specialized model for predicting timeseries temperature data in a datacenter. Some techniques like DynaMMo [19] and PLiF [20] learn linear dynamical systems for multiple time sequences for the purposes of both forecasting and summarization, and the latter is evaluated in part on network monitoring data (though for the purposes of clustering). Linear Gaussian models like the Kalman filter used here are surveyed in [31].

3. APPROACH

We address our three goals described in Sec. 1 in the following manner. First, in order to achieve efficient compression of monitoring data and facilitate the generation of its intelligible summaries, we decompose the raw data into spike data and streams that are amenable to these two objectives. Then, actual creation of summaries is performed using incremental PCA, which produces a lower-dimensional representation of the original data. Finally, we predict future system state by modeling the variables in the lower-dimensional representation. This process is illustrated in Fig. 2. Also, the core steps of the problem are formally defined as follows:

Problem Statement: Given N labeled timeseries streams, we have at each of T time ticks a vector of observations $y_t = [y_{t,1}, \dots, y_{t,N}]$. Each reported value $y_{t,i} \in \mathbb{R}_{\geq 0}$. We seek to find $M < N$ streams $s_t = s_{t,1} \dots s_{t,M}$ at each time tick that form a summary of the data, and other model parameters that can be displayed to indicate outliers and anomalies. Additionally, we seek to forecast s_{T+f} for some $f > 0$ — that is, predict future system trends.

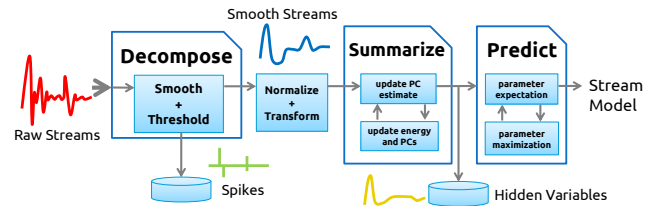


Figure 2: Multi-stage RainMon data flow. Raw streams are decomposed and then modeled.

In the following sections, we describe each step in detail. In each of these sections, we refer to the input of each stage as y_t and its output as x_t . Note that the decomposition and summarization stages of the analysis are *streaming* algorithms; that is, they can produce x_{t+1} given y_{t+1} and the model parameters estimated from $y_1 \dots y_t$. This aspect is important in a monitoring setting, since data arrives in an incremental fashion as systems produce it; streaming analyses allow for efficiency through incremental updates.

3.1 Decomposition

One of the domain challenges of modeling datacenter time-series is the burstiness of many system metrics, such as network and disk I/O [17]. Much of this burstiness is irrelevant for diagnosis, but significant bursts and long-term trends are useful features. This motivates us to borrow ideas from Cypress [30], a framework for decomposing datacenter time-series data into smoothed trends, spiky bursts, and residual data. We show concretely how decomposition can be used to effectively compress relevant data in Sec. 5.5.

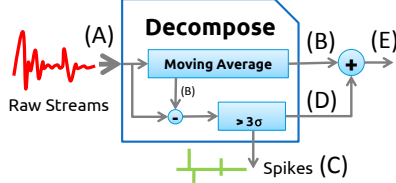


Figure 3: Stream decomposition: data is low-pass filtered; spikes that are 3σ larger than the residuals are segregated and stored separately.

In order to obtain a smoothed representation of the signal, the raw data (A in Fig. 3) is passed through a low-pass filter with cut off frequency $f_s/2m$ [29], where f_s is the sampling frequency of the timeseries and m is an application-specific parameter that is tunable based on the nature of the data streams. We use an exponential moving-average filter:

$$x_t = \alpha y_t + (1 - \alpha)x_{t-1} \quad \alpha = \frac{\Delta t}{m\Delta t/\pi + \Delta t}$$

where Δt is the interval between time ticks, and m can be experimentally determined by performing spectrum analysis on examples of the data streams [30]. Currently, we simply use $m = 60$ sec in the filtering of testing data.

The presence of significant spikes can be useful for time-based anomaly detection. In this kind of analysis, the most pronounced spikes in the signal are the most relevant. To detect these spikes, we apply a threshold to the “noise,” which is the signal obtained by subtracting the band-limited signal (B) from the original signal (A). We choose 3σ as the threshold, where σ is the standard deviation of the “noise.”

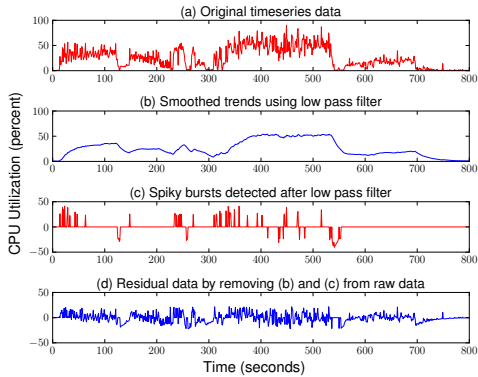


Figure 4: Sample of timeseries data decomposition. Observe the burstiness of the CPU monitoring data.

Fig. 4 illustrates a timeseries signal (CPU utilization) at each of the stages described above; the letters correspond to those in Fig. 3. We then pass the smoothed signal (E) with amplitude-constrained residuals to the next stage — summarization through dimensionality reduction.

3.2 Summarization

For the purposes of producing summaries that capture the overall behavior of the system, we use incremental PCA. We use the hidden variables it produces as timeseries summaries and its weight matrix to help identify anomalous streams. Additionally, the algorithm we use (SPIRIT [27]) adapts the number of hidden variables; addition or removal can signify a change in behavior.

The underlying model is that the N -dimensional data are well-modeled by $M \leq N$ hidden variables. As each new vector of observations y_t ((E) in Sec. 3.1) arrives, we update the model with the technique shown in Algorithm 1. We choose SPIRIT for two primary reasons. First, it is incremental, with complexity $O(MN)$. Second, in the datasets under examination, we expect linear correlations between streams in any case (e.g., the similar cluster resource utilization patterns in Sec. 5). The output x_t from this stage is used as the summary s_t .

Algorithm 1 SPIRIT update

Require: A number of principal components M_t , weight matrix W , new observation y_{t+1} of dimensionality N , energy vector d , energy thresholds f_E and F_E , previous energies E_x, E_y , $\lambda = 0.99$

Ensure: Updated W , d , M , reconstruction \hat{y}_{t+1} , hidden variables x_{t+1}

```

 $r \leftarrow y_{t+1}$ 
for  $i \leftarrow 1 \dots M$  do
     $z = W_i^T \cdot r$  (where  $W_i$  is the  $i^{th}$  column of  $W$ )
     $d_i \leftarrow \lambda d_i + z^2$ 
     $W_i \leftarrow W_i + \frac{z(r - zW_i)}{d_i}$ 
     $r \leftarrow r - zW_i$ 
end for
 $W \leftarrow \text{orthonormalize}(W)$ 
 $x_{t+1} \leftarrow W^T \cdot y_{t+1}$ 
 $\hat{y}_{t+1} \leftarrow W \cdot x_{t+1}$ 
 $E_x \leftarrow \lambda E_x + \|x_{t+1}\|^2$ 
 $E_y \leftarrow \lambda E_y + \|\hat{y}_{t+1}\|^2$ 
if  $E_x < f_E E_y$  then
     $M \leftarrow \max(0, M - 1)$ 
else if  $E_y > F_E E_x$  then
     $M \leftarrow \min(N, M + 1)$ 
end if

```

PCA-based algorithms like SPIRIT function best when the magnitudes of the features are approximately equal. This is decidedly not true of monitoring data; metrics like network bytes written are on the order of 10^7 , while CPU usage is delivered as a value between 0 and 100. For batch analysis, we simply normalize the data. For streaming analysis, we use domain knowledge of the data to associate a “maximum” value with each stream. This maximum is only accurate to an order of magnitude, but looking ahead in a stream to find the true maximum runs counter to the philosophy of incremental updates. We use a linear transform for data that is less bursty, and use a logarithmic transform for I/O-related metrics. For parameters R_{min} and R_{max} , which define the typical order-of-magnitude range of a value, the transform for a value v is:

$$\begin{aligned} \text{Linear:} \quad & f(v) = \frac{v - R_{min}}{R_{max} - R_{min}} \\ \text{Logarithmic:} \quad & f(v) = \frac{\ln(v + 1)}{\ln(R_{max})} \end{aligned}$$

3.3 Prediction

To forecast future system trends, we estimate future values of the streams output by the summarization stage. This problem amounts to multivariate timeseries forecasting, for which there is a wide body of work. We select a Kalman filter since it generalizes recursive linear regression. Other studies have compared forecasting models for datacenter event timeseries [33]. Alternative approaches like ARIMA [6] could be applied in this stage.

Given the response of SPIRIT (x_t in Sec. 3.2, denoted y_t here), we learn a state evolution model with hidden states x_t as follows:

$$\begin{aligned} x_{t+1} &= Ax_t + w \\ y_t &= Cx_t + v \end{aligned}$$

where A is the state transition matrix, C captures the observation model, and $w \sim N(0, Q)$ and $v \sim N(0, R)$ are the state evolution and observation noise models, respectively. See Fig. 5 for a pictorial representation of the model evolution. The two main sub-problems associated with learning these models are *inference* and *parameter learning*. Inference deals with issues of estimating the unknown hidden variables given some observations and a given fixed model parameters. Parameter learning pertains to estimating the model parameters given only the observations.

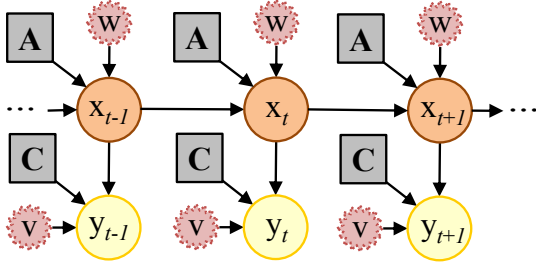


Figure 5: Kalman filter model. Hidden variables from dimensionality reduction (y) are further modeled with hidden states x .

With every observation y_t , we update the model parameters as detailed in Appendix A.1. Then, we smooth the predictions by iterating backwards (see Appendix A.2). With the forward predictions and smoothed reverse predictions, we use the standard Expectation-Maximization (EM) procedure both to learn the parameters and to predict the latent state variables (see Appendix A.3). That is, given the observations, we use the current model parameters to predict the latent state variables. Then given these latent state parameters and the observations, we determine the model parameters by maximum likelihood estimation.

4. IMPLEMENTATION

RainMon is an *end-to-end* system, accommodating the knowledge discovery process from storage through analysis to visualization. An overview of the system design is shown in Fig. 6. The tool can obtain data directly from existing monitoring systems using RRDtool [24, 32] and a database that stores terabyte-scale timeseries data [34]. Since both these data sources provide data in ranges, we run both streaming algorithms in batch fashion. The analysis process in RainMon is distributed; although we currently only run the analysis on a single machine, it can be parti-

tioned and run in parallel. A web-based interface (shown in Fig. 7) accesses the analysis results and can display analysis results to multiple users simultaneously.

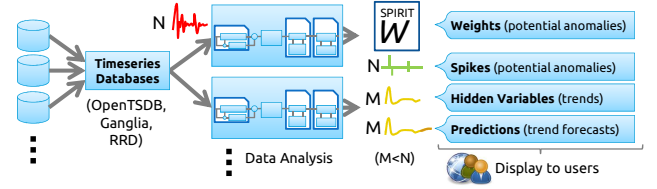


Figure 6: End-to-end, scalable architecture. RainMon can access multiple widely-deployed backing databases [24, 32], and is designed for scalability.

We have made RainMon open-source; additionally, we are releasing the CMU.net dataset, which we used as part of our evaluation, to the community. Both are available at <http://code.rainmon.com>.

5. APPLICATIONS

To assess the utility of RainMon, we demonstrate through a series of case studies how it was used to find problematic machines and time periods in real-world data. Note that the plots that follow were generated for clarity on paper, but all of the visualizations can be created in RainMon’s interface. Using the techniques described in Sec. 3.1-3.3 and this tool, RainMon has been applied to three scenarios:

Hadoop Machine-level metrics from the OpenCloud Hadoop cluster at Carnegie Mellon University (CMU). Machines are numbered cloud1-cloud64.

CMU.net Time taken for a packet to make a round trip from CMU to other locations in the world, as measured by the ping program.

Abilene Network flows measured from the Abilene network backbone of Internet2 (courtesy [15])

Dataset	Nodes	Metrics	Size
Hadoop System Metrics	64	58	220GB
CMU.net Ping Times	6	18	17.1MB
Abilene Network Flows	121	1	5.7MB

Table 1: Datasets used in case studies.

In a cluster where all machines are shared across the same scheduler, such as many Hadoop clusters, a typical large job will span all machines in the cluster. Similar resource utilization patterns on all machines indicate that a job is performing normally, whereas a single machine or groups of machines that behave differently indicate anomalies or imbalance. We first consider the task of isolating a machine or small group with abnormal behavior.

For the analyses below, we do not provide summary statistics on the precision and recall of detection since determining whether a deviation is anomalous through root-cause diagnosis is a question in itself; further work is needed to establish complete ground truth and anomaly criteria for datasets like the Hadoop system metrics. Given a definition of an anomaly, a variety of outlier detection approaches can be applied (e.g., [5]). Here we present case studies for which further investigation has shown that anomalous behavior did indeed exist in the cluster or network.



Figure 7: *RainMon* interface. Overlaid in black boxes is the workflow of the tool. At bottom are the scatterplot of $W_{:,2}$ vs $W_{:,1}$ and a machine “energy” heatmap. In the central pane are linked, zoomable, customizable timeseries plots. An interactive demonstration of the interface is available at <http://demo.rainmon.com>.

5.1 Outlier Machine Isolation

Caused by Machine: After selecting a region of data to analyze ((1) in Fig. 7), a useful visualization for quickly identifying machines that do not behave like others is a plot of the projection coefficients of the first two hidden variables to each smoothed data stream, i.e., $W_{:,2}$ vs $W_{:,1}$ of the PCA weight matrix W (from Algorithm 1, (2) in Fig. 7). If disk reads were closely correlated over time on all machines, and disk writes were also correlated, we would see two clusters on the plot — one for each metric. For example, in the case shown in Fig. 8, we see that this is the situation for almost all of the machines (points), but that the metrics for one machine (*cloud11*) were far from those for the others.

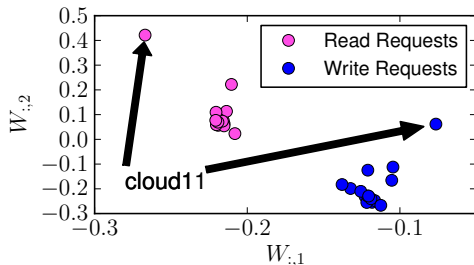


Figure 8: *Outlier node discovery*. *RainMon*’s scatterplot of $W_{:,2}$ vs $W_{:,1}$ shows that the coefficients of *cloud11* are separated from the other machines.

From this point, the tool provides the ability to easily construct overlaid timeseries plots of multiple metrics ((3) in Fig. 7). An examination of the disk read behavior of *cloud11* compared with two other cluster members is shown in Fig. 9. Observe that during periods of heavy read activity on other machines it was idle (around Oct. 12), and that

the collection process on the machine failed (the flat line from Oct. 13 onwards). Additionally, from the spike data we found that a burst of read activity on *cloud11* occurred before a burst on most other nodes.

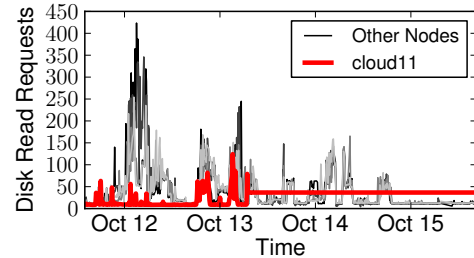


Figure 9: *Individual machine examination*. After identifying candidate outliers, *RainMon* allows users to build custom plots and overlay timeseries data.

Caused by Task: As another example of identifying outlier nodes in a representative cloud workload, we explicitly introduced load imbalance to a job running in the Hadoop cluster. The job is an email classification task running on a Hadoop-based machine learning framework [2]. Specifically, in the example input dataset, we generated a number of duplicated emails under a specific newsgroup topic, so that the total amount of input data is approximately an order of magnitude larger than that of the other topics. This caused the Mapper processes handling the topic to last longer and consume more resources than the others, because of the uneven input distribution.

Fig. 10 compares three different ways of visualizing CPU-related metrics of the Mahout workload in *RainMon*. The top graph shows raw user-level CPU usage of a set of ma-

chines that processed the task, and the middle graph shows spikes of this data extracted by decomposition. The bottom graph shows the first and second hidden variables computed from the set of CPU-related metrics including user-level CPU usage. The injected excessive input was processed by cloud9 and cloud28, increasing their resource usage. Even though the outliers are visible in the original data, they consist of multiple datapoints and require a threshold to distinguish from other behavior. In the extracted spike data, each single datapoint greater than zero can be treated as a potential anomaly, and general trends can be observed in the hidden variables hv0 and hv1. Note that spikes can also be negative, further emphasizing that domain knowledge is important to interpret the meaning. For example, a sudden workload increase may be witnessed by positive CPU spikes, whereas a brief network problem may manifest through negative network spikes.

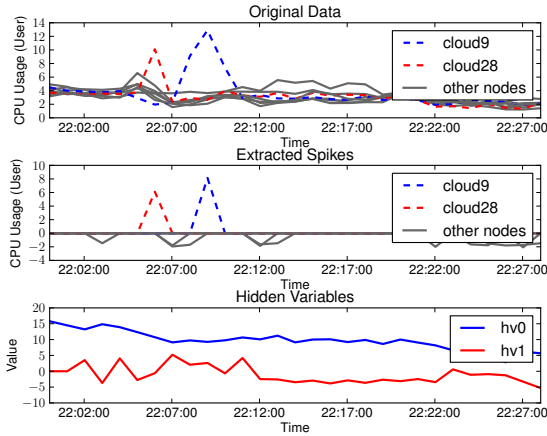


Figure 10: Separation of trends and spikes. Anomalous timeseries spikes caused by overburdened tasks and overall trends are distinguished by RainMon.

5.2 Machine Group Detection

It is not always the case that a single machine is an outlier. In many cases of uneven workload distribution across machines in a cluster, groups of machines may behave differently. For example, we identified a case with RainMon where all machines were *reading* data as expected, but there were two groups of *writers*—only about half the machines were writing heavily. Fig. 11, the plot of $W_{:,2}$ against $W_{:,1}$, illustrates the two groups of machines, which we have highlighted with gray rectangles. This was caused by a large Hadoop job whose tasks did not generate even output loads across machines. Some of those tasks generated more output data than other tasks, causing the machines running these tasks to experience higher disk write traffic. A programmer can fix this problem by partitioning the data more evenly across machines.

5.3 Detecting Correlated Routers

To demonstrate the applicability of RainMon to network monitoring, as well as within-datacenter monitoring, we analyze the packet flow count in the Abilene network [15]. Abilene is a nationwide high-speed data network for research and higher education. This public dataset was collected from 11 core routers with packet flow data between every

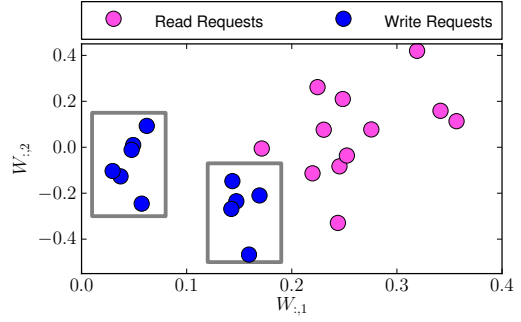


Figure 11: Machine group discovery. From the scatterplot, RainMon can also identify *groups* of machines with different behavior.

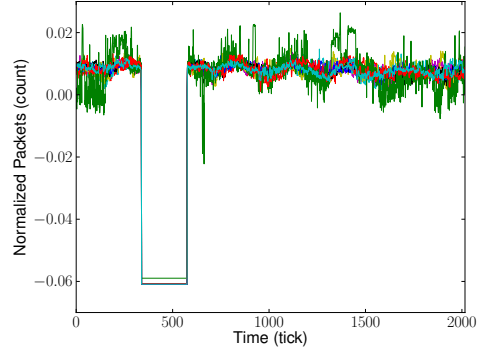
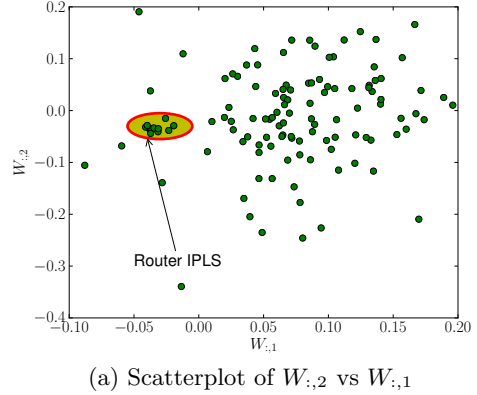


Figure 12: Correlated anomaly identification. RainMon’s scatterplot illustrates a cluster of nodes and allows visualization of the anomalous streams. All streams shown in (b) correspond to a single router named “IPLS.”

pair of nodes for a total of 121 packet flow streams. We analyzed a week’s worth of data binned at 5 minute intervals for a total of 2016 time ticks. Visualizing this data via the scatterplot of $W_{:,2}$ vs $W_{:,1}$ reveals some tightly clustered streams (see Fig. 12(a)) that interestingly all correspond to packet flows from a single router named “IPLS.” Examining in detail the corresponding data streams, this correlation between “IPLS” and other routers becomes quite evident (see Fig. 12(b)) in the abnormal behavior on its part around time tick 400.

5.4 Time Interval Anomaly Detection

In addition to finding anomalous machines, RainMon can be useful in detecting anomalous time intervals. The trends observed in hidden variables provide an overall picture of trends in smoothed data and anomalies across a multi-tick timescale. This can be helpful in finding unusual behavior in time from inspection of only a single trend.

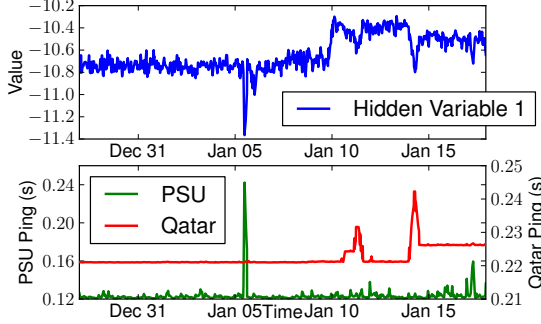


Figure 13: Trend discovery. Change in ping time behavior (Qatar) and sustained increase (PSU) found from the first hidden variable. Absolute values on the “Value” axis should not be interpreted.

When applying the tool to one year of data from the CMU.net dataset, we could rapidly identify changes in a single ping destination by examining the first hidden variable. Fig. 13 illustrates one of the cases: we observed a sudden point change in the trend that was retained after decomposition around Jan. 5, 2012, and an increase starting at approximately Jan. 10. From inspection of the weight coefficients of the first hidden variable, we isolated two unusual behaviors on two of the links. First, a point increase in ping time is clearly observable in the PSU timeseries. Second, the changes towards the end of the time window shown in the figure helped identify a change in the “normal” ping time to Qatar. This time was nearly constant at 0.221 sec before Jan. 10, and became 0.226 sec in the period after Jan. 15, a change difficult to localize from the traditional display currently used for the data. A network administrator confirmed that these sustained changes were legitimate and likely occur due to maintenance by a telecommunications provider.

5.5 Compression

To further show concretely how decomposition is effective, we show how RainMon can compress data, since the summary from dimensionality reduction can be projected back into the space of the original data. That is, by retaining hidden variables s_t and the weight matrix W , RainMon can store a lossy version of the timeseries data — but, by keeping spike data, it can also maintain potential anomalies with full fidelity. Since W is adapted over time in Algorithm 1, one would need to store this matrix at multiple points in time. Therefore, for this evaluation, we perform PCA over an interval of time, rather than use incremental PCA, and use a fixed number of hidden variables.

Fig. 14 shows the total compressed size of the data segment when stored with a combination of hidden variables and weight matrix (in red) and spike data (black). For comparison, we show two alternative compression approaches: using dimensionality reduction on the non-decomposed input (blue bars) and storing the original data (green line).

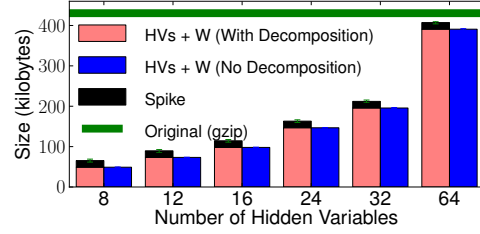


Figure 14: Space savings: storing hidden variables uses less space than the baseline lossless compression approach (green “Original” line).

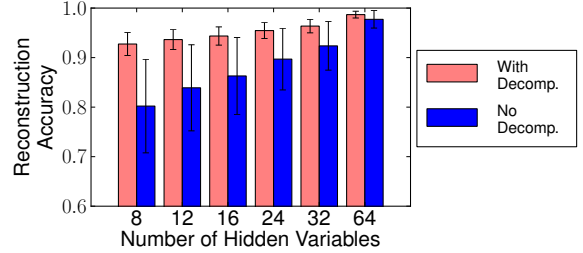


Figure 15: Greater accuracy through decomposition: Storing lossless spike data separately provides improved accuracy relative to no decomposition.

For fairness, we apply generic (**gzip**) compression to the data in all cases, which is a standard baseline approach to storing timeseries monitoring data [22]. For $M = 16$ hidden variables and eleven 56-hour intervals of 118 streams of disk access data from the Hadoop dataset, the reduced-dimensionality data and spikes together occupy 114 ± 5 KB. This is 26.5% the size of the **gzip**-compressed original data (430 ± 80 KB), and only requires 16KB (14%) of additional storage for the sparse, highly-compressible spike data.

Though spike data requires slightly more space, it yields better accuracy than using non-decomposed input. For the same data shown in Fig. 14, Fig. 15 shows the reconstruction accuracy of smoothed+spike data (“With Decomp.”) and PCA as run on the original data (“No Decomp.”)—the former has both higher mean accuracy and lower variance¹. Additionally, since spikes are stored with full fidelity, anomaly detection and alerting algorithms that require precise values could operate without loss of correctness.

5.6 Prediction

We further consider how effectively the trends captured in hidden variables can be forecasted. If projected back into the original space, future machine utilization can be useful for synthetic workload generation and short-term capacity planning [9]. RainMon uses a Kalman filter as described in Sec. 3.3 to produce adaptive predictions. To provide an indirect comparison to other approaches, we instead force a fixed look-ahead and also run vector autoregression (VAR) [23] and trivial last-value prediction (Constant) for the same

¹Reconstruction accuracy A is computed as the geometric mean of the coefficient of determination R^2 for each timeseries; we compare data where A is defined for both models:

$$A = \left(\prod_{i=1}^N 1 - \frac{\sum_{t=1}^T (y_{t,i} - \hat{y}_{t,i})^2}{\sum_{t=1}^T (y_{t,i} - \bar{y}_i)^2} \right)^{1/N}$$

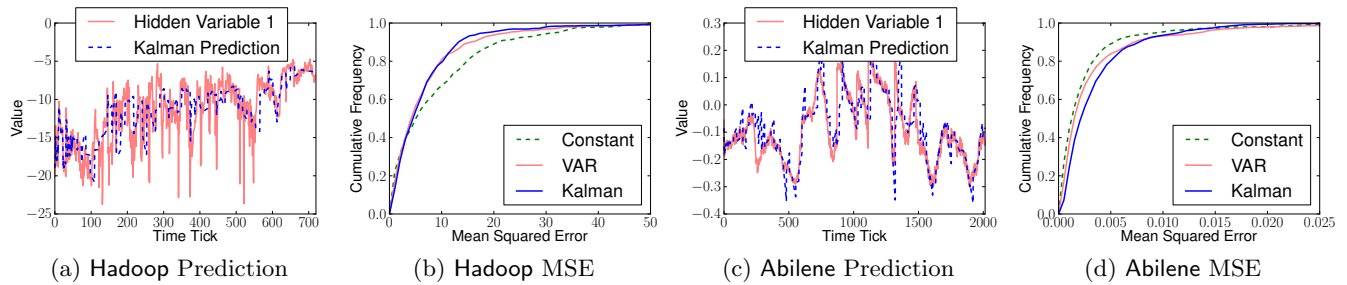


Figure 16: Kalman filter predictive performance. A Kalman filter model was used to predict summarized hidden variables two hours into the future. Fig. (a) shows the prediction overlaid on the first hidden variable for a 5-day slice of memory and disk metrics on the **Hadoop** dataset; (b) shows a CDF of the mean squared error between the prediction and data as compared to vector autoregression and simple constant prediction. Figs. (c) and (d) show the same for the **Abilene** dataset.

look-ahead. As seen in Fig. 16, on a slice of **Hadoop** data, the Kalman filter slightly outperforms VAR, whereas both VAR and the Kalman filter have less predictive power than constant forecast for the **Abilene** data. Though further work is needed to fully characterize these datasets, this highlights the limitation of RainMon’s model-based predictor on particularly self-similar or difficult data.

6. CONCLUSION AND DISCUSSION

Understanding usage and behavior in computing clusters is a significant and growing management problem. Integrating decomposition, dimensionality reduction, and prediction in an end-to-end system has given us enhanced ability to analyze monitoring data from real-world systems. RainMon has enabled exploration of cluster data at the granularity of the overall cluster and that of individual nodes, and allowed for diagnosis of anomalous machines and time periods.

Our experience with RainMon has helped to define its limitations, and we highlight three directions of ongoing work here. First, parameter selection is both involved and necessary given the diversity of datacenter streams, and as briefly glimpsed in Sec. 5.6 the ability to mine monitoring data varies across environments. Mining other datasets using similar techniques, or optimizing them for the ones we study, is an ongoing and future challenge.

Second, understanding how analysis results are interpreted is key to improving their presentation. Initial feedback from industry visitors and a few administrators has resulted in changes to RainMon’s interface, but active deployment of RainMon on more systems and studies with administrators and cluster users are needed to enhance its utility.

Third, scaling these techniques to larger systems will require further towards performance tuning. Currently, the downsampling and slicing capabilities of our storage approach have enabled us to handle large datasets like the **Hadoop** streams. However, robustness at even greater scales remains to be demonstrated. Larger systems also complicate collection, aggregation, and fault-tolerance.

7. ACKNOWLEDGEMENTS

We greatly appreciate the CMU network operations team for enabling the use of RainMon on the **CMU.net** data and the assistance of OpenCloud administrators. Feedback from Lei Li, Raja Sambasivan, and the anonymous reviewers was very helpful in improving this paper. We thank the members and companies of the PDL Consortium (including Actifio,

APC, EMC, Emulex, Facebook, Fusion-io, Google, Hewlett-Packard Labs, Hitachi, Intel, Microsoft Research, NEC Laboratories, NetApp, Oracle, Panasas, Riverbed, Samsung, Seagate, STEC, Symantec, VMware, and Western Digital) for their interest, insights, feedback, and support. This research was sponsored in part by Intel via the Intel Science and Technology Center for Cloud Computing (ISTC-CC). Ilari Shafer is supported in part by an NDSEG Fellowship, which is sponsored by the Department of Defense.

8. REFERENCES

- [1] Apache Hadoop, <http://hadoop.apache.org/>.
- [2] Apache Mahout: Scalable machine learning and data mining, <http://mahout.apache.org/>.
- [3] L. Barroso and U. Hözl. *The datacenter as a computer: An introduction to the design of warehouse-scale machines*. Morgan & Claypool Publishers, San Rafael, CA, 2009.
- [4] K. Bhaduri, K. Das, and B. Matthews. Detecting abnormal machine characteristics in cloud infrastructures. In *Proc. ICDMW’11*, Vancouver, Canada.
- [5] M. Breunig, H. Kriegel, R. Ng, and J. Sander. LOF: identifying density-based local outliers. In *Proc. SIGMOD’00*, Dallas, Texas. ACM.
- [6] P. J. Brockwell and R. A. Davis. *Introduction to time series and forecasting*. Springer-Verlag, 2002.
- [7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.
- [8] D. Fisher, D. Maltz, A. Greenberg, X. Wang, H. Warncke, G. Robertson, and M. Czerwinski. Using visualization to support network and application management in a data center. In *Proc. INM’08*, Orlando, FL.
- [9] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Proc. IISWC’07*, Boston, MA.
- [10] V. Guralnik and J. Srivastava. Event detection from time series data. In *Proc. KDD’99*, San Diego, CA.
- [11] S. Hirose, K. Yamanishi, T. Nakata, and R. Fujimaki. Network anomaly detection based on eigen equation compression. In *Proc. KDD’09*, Paris, France.
- [12] E. Hoke, J. Sun, J. Strunk, G. Ganger, and C. Faloutsos. InteMon: continuous mining of sensor

- data in large-scale self-infrastructure. *ACM SIGOPS Operating Systems Review*, 40(3):38–44, 2006.
- [13] IBM Tivoli, <http://www.ibm.com/developerworks/tivoli/>.
- [14] M. P. Kasick, J. Tan, R. Gandhi, and P. Narasimhan. Black-box problem diagnosis in parallel file systems. In *Proc. FAST'10*, San Jose, CA.
- [15] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proc. SIGCOMM'04*, Portland, OR.
- [16] Z. Lan, Z. Zheng, and Y. Li. Toward automated anomaly identification in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 21(2):174–187, Feb. 2010.
- [17] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic. In *Proc. SIGCOMM'93*, San Francisco, CA.
- [18] L. Li, C. Liang, J. Liu, S. Nath, A. Terzis, and C. Faloutsos. ThermoCast: A cyber-physical forecasting model for data centers. In *Proc. KDD'11*, San Diego, CA.
- [19] L. Li, J. McCann, N. Pollard, and C. Faloutsos. DynaMMo: Mining and summarization of coevent sequences with missing values. In *Proc. KDD'09*, Paris, France.
- [20] L. Li, B. A. Prakash, and C. Faloutsos. Parsimonious linear fingerprinting for time series. In *Proc. VLDB'10*, Singapore.
- [21] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15(2):107–144, Oct. 2007.
- [22] C. Lobo, S. Smyl, and S. Nath. DataGarage: Warehousing massive performance data on commodity servers. In *Proc. VLDB'10*, Singapore.
- [23] H. Lutkepohl. *New Introduction to Multiple Time Series Analysis*. Springer, 2007.
- [24] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, July 2004.
- [25] Nagios, <http://www.nagios.org>.
- [26] H. Nguyen, Y. Tan, and X. Gu. PAL: Propagation-aware anomaly localization for cloud hosted distributed applications. In *Proc. SLAML'11*, Cascais, Portugal.
- [27] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *Proc. VLDB'05*, Trondheim, Norway.
- [28] D. Patnaik, M. Marwah, R. Sharma, and N. Ramakrishnan. Sustainable operation and management of data center chillers using temporal data mining. In *Proc. KDD'09*, Paris, France.
- [29] J. G. Proakis and M. Salehi. *Fundamentals of Communication Systems*. Prentice Hall, 2004.
- [30] G. Reeves, J. Liu, S. Nath, and F. Zhao. Cypress: Managing massive time series streams with multi-scale compressed trickles. In *Proc. VLDB'09*, Lyon, France.
- [31] S. Roweis and Z. Ghahramani. A Unifying Review of Linear Gaussian Models. *Neural Computation*, 11(2):305–345, Feb. 1999.
- [32] RRDtool, <http://www.mrtg.org/rrdtool/>.
- [33] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E.

Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proc. KDD'03*, New York, NY.

- [34] B. Sigoure. OpenTSDB: The distributed, scalable time series database. In *Proc. OSCON'11*, Portland, OR.
- [35] J. Sun, S. Papadimitriou, and C. Faloutsos. Distributed pattern discovery in multiple streams. In *Proc. PAKDD'06*, Singapore.
- [36] Zenoss, <http://www.zenoss.com/>.
- [37] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *Proc. KDD'03*, Washington, DC.

APPENDIX

A. KALMAN FILTER DETAILS

A.1 Prediction

Given the initial model parameters ($\mu_{t|t}$ and $P_{t|t}$) for the linear dynamical system and the current observation (y_t), we update the model parameters to compute $\mu_{t+1|t+1}$ and $P_{t+1|t+1}$ as follows:

$$\begin{aligned} V_t &= AP_{t|t}A^T + \Sigma_w \\ K_{t+1} &= V_t C^T (CV_t C^T + \Sigma_v)^{-1} \\ \hat{\mu}_{t+1|t+1} &= A\hat{\mu}_{t|t} + K_{t+1}(y_{t+1} - CA\hat{\mu}_{t|t}) \\ P_{t+1|t+1} &= V_t - V_t C^T (CV_t C^T + \Sigma_v)^{-1} CV_t \end{aligned}$$

A.2 Smoothing

Given the predictions $\hat{x}_{t|t}$ and $\hat{P}_{t|t}$ from above that were computed over the past T time samples, we compute the smoothed predictions using predictions from the future:

$$\begin{aligned} A_t^s &= P_{t|t}A^T(V_t)^{-1} \\ \hat{\mu}_t^s &= \hat{\mu}_{t|t} + A_t^s(x_{t+1}^s - A\hat{x}_{t|t}) \\ P_t^s &= P_{t|t} + A_t^s(P_{t+1}^s - V_t)A_t^{sT} \end{aligned}$$

where $\hat{\mu}_t^s$ and P_t^s are the smoothed estimates of the model parameters computed by using the observations in the future. We start at $t = T$ and go backwards to $t = 1$. These are useful for parameter learning as described next.

A.3 Parameter Learning

Given the forward Kalman filter predictions and the smoothed reverse predictions we estimate all the model parameters as follows, where n refers to the EM iteration index:

$$\begin{aligned} E[x_t] &= \hat{\mu}_t^s \\ E[x_t x_{t-1}^T] &= P_t^s A_{t-1}^T + \hat{\mu}_t^s \hat{\mu}_{t-1}^s \\ E[x_t x_t^T] &= P_t^s + \hat{\mu}_t^s \hat{\mu}_t^{sT} \\ A^n &= (\sum_{t=2}^T E[x_t x_{t-1}^T]) (\sum_{t=1}^T E[x_t x_t^T])^{-1} \\ \Sigma_w^n &= \frac{\sum_{t=2}^T E[x_t x_t^T] + A^n E[x_{t-1} x_{t-1}^T] A^{nT}}{T-1} - \frac{\sum_{t=2}^T E[x_t x_{t-1}^T] A^{nT} + A^n E[x_{t-1} x_t^T]}{T-1} \\ C^n &= (\sum_{t=1}^T y_t E[x_t^T]) (\sum_{t=1}^T E[x_t x_t^T])^{-1} \\ \Sigma_v^n &= \frac{\sum_{t=1}^T y_t y_t^T + C^n E[x_t x_t^T] C^{nT}}{T} - \frac{y_t x_t^{sT} C^{nT} + C^n x_t^s y_t^T}{T} \\ \mu_0^n &= E[x_t] \\ \Sigma_0^n &= E[x_t x_t^T] - E[x_t] E[x_t]^T \end{aligned}$$