Catchy Title

John Smith

Dept. of Computer Science

Carnegie Mellon University

jsmith@cs.cmu.edu

Tom Johnson

Dept. of Metaphysics

Confusion State University

tj@bla.csu.edu

October 11, 2011

Abstract

We solved an important problem, outperforming all previous attempts.

1 Introduction

Timeseries data are prevalent in large-scale computing centers. Systems often capture sampled metrics of performance, utilization, and even sensor data like temperature. These streams are used for monitoring, placement, optimization, and more: for example, task assignment algorithms use computer utilization to determine where to place jobs.

We propose development of a model of timeseries utilization data that incorporates correlations across time and across multiple streams. Such a model provides the potential to address a number of practical advances for datacenter efficiency:

- Improve alerting and placement algorithms by predicting future usage. Even a short-term view of future usage can be valuable for decision-making. By playing the model forward, we can obtain this data.

- Reduce the storage requirement of streams through compression. By storing only the model parameters and occasional original datapoints, potentially large timeseries data can be effectively summarized.

- Detecting potential anomalies or alert conditions. Dramatic changes in model parameters can be predictors of problems or abnormalities.

The scope of potential applications is considerable, and there is prior art in models for these operations on timeseries data (Sec. **??**). We plan to focus our efforts on applying and tuning existing algorithms from the literature on the different type of timeseries data we have: rather than target relatively small sensor datasets as many have considered, we seek to apply these techniques to lengthy, bursty utilization data. This focus motivates our emphasis on evaluating multiple models, preprocessing, and examining scalability (Sec. **??**).

2   Survey

2.1   Models

SPIRIT [**?**] is an algorithm for discovering patterns in multiple streams of time-series data. This algorithm effectively tries to find correlations between the different data streams by performing principal component analysis on the the data covariance matrix. These principal components are modified and updated as each data sample is received at each time step while forgetting the samples. An interesting aspect of this algorithm is, anomaly detection which is done by detecting the changes in the number of principal components required to capture most of the energy in the signal. In [**?**] the authors describe a unifying model for Linear Gaussian Models which includes PCA, Kalman Filters, Gaussian Mixture Models, ICA etc. They describe a general linear state transition model along with a linear observation model corrupted by independent Gaussian noise. They describe general algorithms to do inference and also learn the parameters of the model.

DynaMMo is an algorithm for reconstructing missing timeseries data by learning a linear dynamical model [**?**]. This general model is learned though belief propagation and EM, and generalizes SVD, linear interpolation, and Kalman filtering. The authors demonstrate that the method captures dynamics these special cases do not based on experimentation with period flow data and motion capture traces. We plan to evaluate one of the special cases (Kalman filtering) and consider scalable approaches, an area where general approaches DynaMMo currently suffer. The paper describes how to use the model for compression by intermittently storing model parameters, an approach we may consider. Feature Extraction

2.2 Feature Extraction

One feature present in systems operations data that has not often been considered when producing sensor models is configuration information that describes what jobs are running on a cluster. Although it is time-based, it consists of events and not a continuous stream. Previous work has considered how to model such aperiodic streams in the context of computational finance; for example, Lavrenko et al [?] link events (news stories) with continuous stock price data by discretizing the continuous stream and using a sliding window to associate candidate events with changes in the stream. However, efforts in this vein do not preserve the continuous nature of performance data for predictions and compression: when using event data, we plan to convert it to a stream instead.

Zhu and Shasha [?] address the problem of detecting bursts in time series when their time length are unknown. The core approach is to convert time series data to wavelets, and maintain them for a given sliding window of time in a shifted wavelet tree. This data structure allows constructing a pattern of a flexible size within that window from wavelets of different sizes. In turn, it enables analyzing the time series data at different time scales by extracting patterns of different sizes. An online algorithm for maintaining and searching the data structure gives a constant time for updating it, while a batch algorithm trades it off for constant search times. This work focuses on efficiently detecting bursts in one time series. On the other hand, we would need to extend their technique and apply it to a number of time series. We may need to examine bursts in a group of time series, not only those in individual series.

Sakurai et al. [?] introduce an effective way of finding trends in large time series data. In particular, they propose a method for selecting an appropriate window size in time that analysts would find useful. The idea is based on first using independent component analysis (ICA), and then selecting the window size that maximizes the entropy of the weight values in the mixing matrix. By using this criterion, the technique effectively extracts repetitive patterns at different time scales from various data sets, such as Ondemand TV access records and web-click traces for a search engine. This work is complementary to the burst detection work by Zhu and Shasha, as the former suggests the best window size that exhibits repetitive patterns while the latter allows examining different window sizes efficiently. Although the paper focuses on web-click records, the general technique would also be applicable to our data center logs.

Another work by Zhu and Shasha [?] presents an efficient approach to finding correlations among a large number of data streams, in an online fashion for a given sliding window size. The key idea is to apply discrete Fourier transform (DFT) to time sequences, and hash them based

on their DFT coefficients. This technique allows finding highly correlated time series for a given series, by reducing the number of candidates examined based on their grouping derived from hashing. Also, this method is highly parallelizable, which further improves its efficiency. The problem of finding correlations among a number of time series is one of our possible directions. However, the nature of data in which we are interested is different and we may accordingly need to adjust the approach; we are dealing with data center logs, whereas they focus on stock market data.

2.3   Datacenter Data

A workload prediction study [**?**] examines time-series measurement of workloads in datacenters with an eye towards identifying patterns to generate synthetic traces. To analyze patterns, they use two primary techniques: a frequency-domain representation of the workload (as computed from an FFT) and autocorrelation of the trace. Since strong periodic patterns may not appear in all workloads, the approach clusters streams by the goodness of fit of its computed model. We plan to run these analyses on our data to start in order to assess whether they are periodic, since most (76 of 139) patterns the authors study have a moderate degree of periodicity produced by "interactive and/or mixed batch and interactive work."

PAL is an anomaly localization tool to detect faulty components in distributed applications. The key idea is firstly to monitor time-series metrics of each individual component of the distributed system and discover the change points (potentially some faulty behavior happen), and then correlated different change points based on time stamp to locate the real faulty component. Their algorithm has four steps: firstly, raw time series data needs to be smoothed. Kalman filter failed to capture the critical change points so they used 5-length moving average filter. The second step is to use CUSUM to find change points, and calculate the separation level of each change point (which measures the different of mean value of data points before the change point and after it). The outlier change points (whose separation level are outliers) is the critical change points.

Another console logs mining paper [**?**] extracts specific features from unstructured logs of Hadoop system [**?**] (A distributed system for data intensive computing) and use them to detect anomalies in the system. The tool is mainly used to detect correctness problems. The author combines logs and source to extract three different signals: message types, state variables, and message identifiers. Based on these signals, they create two matrices: one matrix has state variables as the column, and time as the row, the other matrix uses message types as the column,

and message identifiers as the row. Finally, PCA is used for anomaly detection. They perform PCA on the two matrices to get a reduced set of dimensions (eigenvectors) ranked by how much variance in the data each reduced dimension explains. Anomalous rows are identified by flagging those rows whose data is largely unexplained by the first K eigenvectors, which describes 95% of the overall observed variance. After finding the anomalous rows, human need to manually interpret and further digging the root causes. In [**?**], they further applied their algorithms into some of Googles production logs. Since the data is large which makes it unrealistic to perform PCA, they used frequent mining techniques to segment the long-lasting sequence into sessions.

3 Proposed Method

Our proposed method is as follows: ...

4 Experiments

We implemented our method and compared it with the older ones. The results are very promising. Figure **??** shows the results (and, actually, how to insert postscript files):

(a)   (b)

Figure 1: A fictitious dataset (a) and its performance plot (b)

5 Conclusions

The proposed method outperforms older approaches in speed and quality.

Contents