Detecting Discontinuities in Large-Scale Systems

Haroon Malik, Ian J. Davis, Michael W. Godfrey
David R. Cheriton School of Computing
University of Waterloo, Ontario, Canada
{hmalik, ijdavis, migod}@uwaterloo.ca

Douglas Neuse, Serge Mankovskii
CA Labs, CA Technologies
{Douglas.Neuse, Serge.Mankovskii}@ca.com

Abstract—**Cloud providers and data centers rely heavily on forecasts to accurately predict future workload. This information helps them in appropriate virtualization and cost-effective provisioning of the infrastructure. The accuracy of a forecast greatly depends upon the merit of performance data fed to the underlying algorithms. One of the fundamental problems faced by analysts in preparing data for use in forecasting is the timely identification of data *discontinuities*. A discontinuity is an abrupt change in a time-series pattern of a performance counter that persists but does not recur. Analysts need to identify discontinuities in performance data so that they can a) remove the discontinuities from the data before building a forecast model and b) retrain an existing forecast model on the performance data from the point in time where a discontinuity occurred. There exist several approaches and tools to help analysts identify anomalies in performance data. However, there exists no automated approach to assist data center operators in detecting discontinuities in the first place. In this paper, we present and evaluate our proposed approach to help data center analysts and cloud providers automatically detect discontinuities. A case study on the performance data obtained from a large cloud provider and performance tests conducted using an open source benchmark system show that our proposed approach provides on average precision of 84% and recall 88%. The approach doesn't require any domain knowledge to operate.**

Keywords— Forecast; data center; anomaly; discontinuity

I. INTRODUCTION

To effectively run a data center, appropriate virtualization and cost-effective provisioning of the infrastructure, with respect to the type and size of the service requests (i.e., the workload), needs to be implemented. Overestimating the necessary infrastructure for a set of requested services in a specified period leads to waste, under-utilization, and increased costs. However, under-estimation of the future workload is also unacceptable, since it degrades the quality of the service and may lead to violations of client Service-Level Agreements (SLAs). To ensure SLAs are met, while minimizing infrastructure costs, data center operators need to know ahead of time, (i.e., short and long-term forecasts) the expected workload. The aim of the short-term forecast is to provide accurate predictions of workloads in the near future, e.g., one or two hours ahead, usually based on a week to a month of the data center's recent performance history. The data center operators use the short-term forecasting for dynamic provisioning and placement of tasks in a data center, especially for load balancing to avoid performance bottlenecks. Accurate short-term forecasting permits near-optimal provisioning, thus improving usage of the available infrastructure. Long-term forecasting of the workload is necessary for capacity planning to ensure that the cloud infrastructure supports growth and evolution of client requirements. To capture the seasonality patterns, long-term forecasting requires the use of at least a year of recent performance history from one or more data centers to predict expected workloads. The accuracy of forecasting results depends on the quality of the performance data (i.e., performance counters; such as CPU utilization, bandwidth consumption, network traffic and Disk IOPS) fed to the forecasting algorithms, i.e., missing value imputation, calculating and adjusting times stamp drifts of logged performance data across hundreds of VMs, identification and removal of outliers and anomalies and in cases, scaling and standardizing the data to remove bias among performance counters.

In a typical cloud environment, a large number of elements (i.e., VMs, routers, chillers and sensors) continuously generate large traces of performance data (terabytes (TB) in size) further complicating the data preparation step. Hence, practitioners and data scientists spend considerable time (e.g., up to 80% [1]) in preparing data for their forecast algorithms. One of the fundamental problems faced by analysts in preparing data for long-term forecast is the identification and removal of data discontinuities. To date, there does not exist any automated approach to assist data center operators in detecting discontinuities in the performance data. Data discontinuity is a special kind of anomaly that differs from behavioral and environmental anomalies, and must be addressed before making a forecast. A *behavioral anomaly* is an inconsistent behavior, when systems have been provisioned identically are receiving similar traffic (i.e., though a load balancer). An *environmental anomaly* results from lack of uniformity between the servers in a data center (usually over time). For example, even when the system is identically provisioned, drift often happens during the course of normal operations.

A discontinuity is an abrupt change in a time-series pattern that persists but does not reoccur, as shown in Fig. 1. Examples include a) a significant change in a counter's value b) a significant change in the slope (rate of change) of the counter's value, c) a significant change in a period cycle or amplitude or both. Discontinuities such as those shown in Fig. 1 do not occur instantaneously, but over a brief period called a transition period. If an analyst recognizes that a discontinuity has occurred, (s)he may want to ignore the early data and base their forecast on the measurements taken after the discontinuity. Moreover, detecting a discontinuity provide analysts a reference point to retrain their forecasting models and make necessary adjustments.

Therefore, analysts require automated techniques that can identify discontinuities among thousands of performance counters collected across hundreds of machines. Such techniques should be intelligent enough to distinguish discontinuities from anomalous data that should be ignored or removed within input data, such as irregularities in which a few individual performance counter values deviate significantly from the general pattern but do not persist; seasonal variations and recurring patterns that should be accommodated such as workload volumes that decrease each weekend but return to normal on Monday, and observed growth that should be suitably anticipated b) recurring patterns such as exponential growth of workload, i.e., where the slope of counter(s) changes smoothly — though perhaps rapidly — with time.

We identify the main contributions of this paper as:

1. We provide an overview of the entire forecasting process for a typical data center.
2. We provide an accurate and novel approach for identifying discontinuities in performance data.
3. To our knowledge, this is the largest study to date for detecting discontinuities; we use performance data from 5,000 machines over a span of seven years.
4. We empirically evaluated our proposed approach on both the data obtained from a large cloud service provider and performance experiments conducted using an open-source benchmark system. We show that our proposed approach can achieve up to 91% average precision and 95% average recall.

Organization of the paper:

The rest of the paper is organized as follow. We describe a typical forecasting process in section II. We then present our proposed approach in section III, followed by a case study setup along with case study findings in section IV. The discussion of the findings of the case study is presented in section V. We discuss limitations and threats to the validity of our approach in section VI, followed by a description of related work in section VII. Finally, we summarize our work and sketch possible future research in Section VII.

II. Steps Involved in Forcasting

A. Determine Purpose: Initially a department, team or a stockholder requests a forecast. Usually, a dedicated group or team of analysts is responsible for handling the forecast requisition. The analysts gather preliminary information from the requestor, i.e., a) forecast purpose (e.g., operations are interested to know expected workload volume on a daily to weekly basis for load balancing and dynamic placement of machines, whereas, marketing and sales are more concerned about growth in customers, planning workforce levels, scheduling and purchases) and b) a time horizon for a forecast (seconds, hours, days, months, quarters or years).

B. Technique Selection: Based on determining the forecast horizon and purpose of requestor, the analyst select an appropriate technique (e.g., moving averages with exponential smoothing for short-term forecasts and trend equations for long-term forecasts). Often, the analyst uses more than one forecasting technique to obtain independent forecasts. If selected techniques produce approximately the same precision, this would give increased confidence in the results; disagreement among forecast indicates that analysts need to revisit the technique.

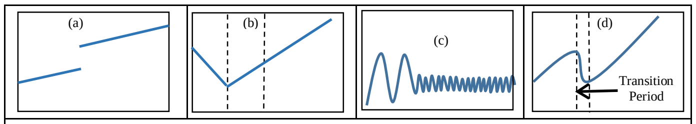


Fig. 1. Examples of discontinuities in performance counter

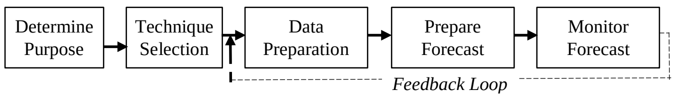


Fig. 2. Steps involved in a forecasting process

C. Data Preparation: This is the most important and expensive forecasting step for analysts. Poor forecasts can result from inadequate data preparation. In this step, analysts sanitize and preprocess the data to make it suitable for the forecasting techniques selected in the previous step. During sanitation missing, ignorable, erroneous and empty performance counter variables are treated [2-4]. Counter data is missing when a performance monitor fails to record an instance of a performance counter. A counter is empty when a resource cannot start the require service. Analysts then preprocess the data using their custom written scripts to aggregate performance counters across several subsystems of a data center to derive customer-perceived counters [5] such as transaction response time, latency, user wait time, and perceived throughput. These values capture the user interaction with their system as their transaction/request/job flows through the various subsystems in a data center. Preprocessing also involves preparing the data in the format that is required by the selected forecast techniques. Therefore, analyst preprocess (i.e., extrapolate, scale and standardize) the data accordingly.

D. Prepare Forecast: In this step, the analyst uses prepared time series training data and the selected forecast technique to create a forecast model that has minimum error rate, i.e., its predicted values are close to the actual time series value, without either underfitting or overfitting. Analyst tune the parameters of the forecast techniques several times to find the best form of the model that satisfies the requestor's forecast objective.

E. Monitor Forecast: This step is composed of two substeps: active and passive monitoring of the forecasts. In active monitoring, an analyst validates a forecast for a predetermined period of time before it is deployed in production or the model is handed over to the requester. The analyst verifies assumptions, compares the forecasted values (transaction volume, workload, or resource utilization of machines) to the actual observed values as they occur in the data center, and identify any external or internal event that affects the results of the forecast. Once the forecasting model is communicated to the requestor, a recurring monitoring checkpoint for the forecast is established (i.e., monthly, quarterly or every six months) to look for any evidence of significant variance between the actual and predicted results; identify deviation factors such as discontinuities. Any variance greater than the maximum is investigated and forecast model is either adjusted to accommodate the variance or retrained for the discontinuity.

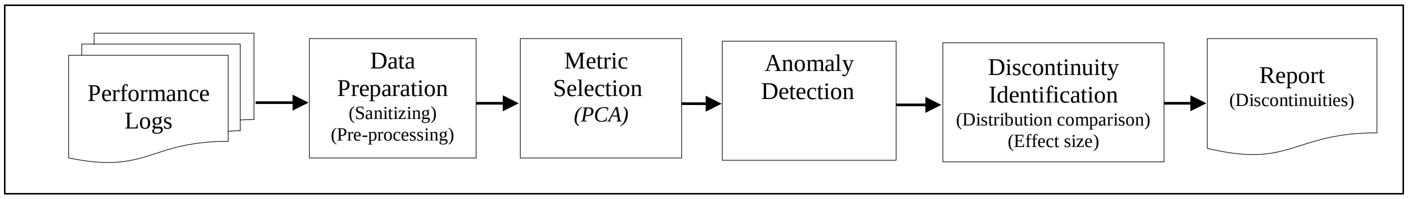


Fig. 3. The steps involved in the proposed approach

III. Proposed Approach

In this section we present our proposal to overcome the challenges discussed in the previous section. Fig. 3 shows the major steps of our proposed approach. We detail the steps as follows:

A. *Data Preparation*

The performance logs obtained from the production environment (i.e., data center) do not suffice for direct analysis by our approach. Performance logs need to be filtered for noise, e.g., missing counter data or empty counter variables. To deal with this kind of problem (incomplete data), we employed list-wise deletion. If the i^{th} observation for a counter '*T*' is missing, list-wise deletion will delete the corresponding i^{th} observation of all the counter variables. Partial empty counter variables and counter variables that have more than 2% of the missing data are automatically removed during the sanitization process. The logs also need to be prepared to make them suitable for the statistical technique employed by our approach, i.e., Principal Component Analysis (PCA). PCA is a maximum variance projection method [12]. Performance counters have different ranges of numerical values; they have different variance. PCA identifies those variables that have a large data spread (variance), ignoring variables with low variance [19]. To eliminate PCA bias towards those variables with a larger variance, we standardized the performance counters via Unit Variance scaling, i.e., by dividing the observations of each counter variable by the variable's standard deviation. Scaled performance counter data are then further mean centered to reduce the risk of collinearity. With mean-centering, the average value of each performance counter variable is calculated then subtracted from its respective counter data. Each scaled variable then has an equal (unit) variance, i.e., each variable has a mean of 0 and Standard deviation of 1.

B. *Performance Counter Selection*

The performance logs obtained from the production environment consists of thousands of performance counters. Many of the performance counters are either invariants such as *'Component Uptime'*, '*Component Last Failure'* or are configuration constants, such as *'No of DB Connections Allowed"*, '*Message Queue Length'* and *'Total Component Memory'*. These counters captures little variance and the values of such performance counters seldom change or correlate to dependent variable such as workload volume. These variables are of little help to analysts in detecting discontinuities. For example, Fig. 4 shows a few of the performance counters for one of the CPU-intensive performance test experiments conducted (explained in section IV). During the course of the performance test, a few anomalies and discontinuities are injected and performance counters across the testbed are captured in a performance log. Among them, Fig. 4 (a) is a plot of a webserver's ' *CPU utilization'* counter that does reflect all the injected anomalies (marked with circles in the figure) and discontinuities (marked with triangles). Whereas, the values of the database servers *% CPU utilization'* performance counter shown in Fig. 4 (b) shows the injected anomalies, but injected discontinuities are not clearly visible. Fig. 4 (c) is *' Disk idle time'* counter of a database server. Its values neither react to any injected anomaly nor do the values reflect the injected discontinuities. Fig. 4 (d) is *'User's thread pool'* performance counter for the load generator and is a semi-invariant, i.e., its values only change during the course of performance test, when the workload intensity is increased or decreased. The counter is able to capture the injected discontinuity, but will fail to capture other types of discontinuities arising due to the changes made to the infrastructure. Moreover, the counter fails to capture any injected anomaly. A naïve way is to apply our proposed discontinuity identification technique across all the performance counters. However, using the techniques on all the counters will also increase detection of false positive discontinuities (such as the result shown in Figure 4 (b and c) for applying the technique on *% CPU Utilization'* and *' Disk Idle Time'* counters) to analysts, thereby wasting their time in inspecting them. We use a robust and scalable statistical technique i.e., Principal Component Analysis (PCA) [22] to identify a few of the performance counter that capture the maximum variation of the collected data and have the potential to capture discontinuities in their time series counter values.

We choose PCA due to a) our previous success in using it with performance data of a large-scale system and b) its superior performance in identifying performance counters that are sensitive to minute changes in both workload and environment as compared to many other supervised and unsupervised machine learning techniques [6]. We provide an overview of the PCA based performance counter selection technique in this paper. Further details are discussed in our previous work [7]. Basically, the high level goal of using PCA in our context is the same as using clustering: selecting the least correlated subset of performance counters that can still explain the maximum variations in the data, thereby eliminating performance counters capturing little variance such as invariants and configurations related performance counters. The performance counters identified by PCA approach are potentially good candidates for detecting any occurring discontinuities. These performance counters are fed into the next step of our approach to first detect the presence of anomalies and then to identify discontinuities among them, if any exist.

C. *Anomaly Detection*

Any attempt to identify what constitutes anomalous data encounters both the difficulty of trying to categorize a very diverse set of unexpected patterns in data according to one or more common characteristics and the difficulty of choosing thresholds that realistically differentiate between normal variance in legitimate data, and unexpected potentially anomalous patterns in that same data. Borderline cases may be somewhat arbitrarily labelled as either anomalous, or not anomalous, with such arbitrary labelling potentially having a significant impact on subsequent prediction.

Some algorithms, such as regression, attempt to predict future results from only data seen to date while others (such as Fast Fourier Transform analysis) [35, 36] seek patterns within training data, so as to predict future results. When seeking to detect anomalies in recent performance data for which future performance data is currently unavailable, we are unable to distinguish between a temporary anomaly and a longer term discontinuity. However, we can track the running mean and variance within the observed data, and presume that observed values exceeding some multiple of the variance from the mean, or recent windowed data failing the t-test is anomalous.

When working with training data, we discover discontinuities by presuming that discontinuities cannot be well modelled by a low order polynomial function. Given a performance counter time series data *{v[t]}*, we approximate the series by the quadratic function $f(t) = c+bt+at^2$ that minimizes the least squared error (LSE). We presume that series containing sudden dramatic changes, anomalies, or discontinuities will not be fit as well by this approximation and so have a larger LSE.

To discover exactly where difficulties arise in fitting this model to the performance counter data, we begin by modelling the performance counter's *n* data points as *n* consecutive quadratics f_i having coefficients $\{c = v[t_i],\ a = b = 0\}$ and consequently $LSE=0$. A greedy algorithm selectively replaces pairs of consecutive quadratics modelling adjacent data by a single quadratic until our performance counter time series is modelled by a single quadratic.

At each step selection is chosen so that the increase in the total LSE is minimized. Replacements with the same increase in LSE are chosen by giving priority to those new quadratics having smaller $|a_i|$, then $|b_i|$, and then if necessary modelling shorter subsequences. At each step the two data points that cease to be at the end of a subsequence when subsequences are merged have a cost associated with them. This cost is simply the total increase in the LSE of the subsequence they formally belonged to when this subsequence is modelled by a quadratic spanning the longer now combined pair of subsequences. Inputs can be standardized (having mean $\mu=0$, and variance $\sigma^2=1$) if cost on different inputs must be comparable.

Cost reflects the poor fit when unifying consecutive subsequences at a point under a common quadratic model. Since the total LSE is related to the length of the subsequence unified, cost is also influenced by reluctance of our greedy algorithm to undertake early unification at a point. Largest costs thus suggest positions where the most egregious anomalies/discontinuities occur as shown in Fig 5 (c, b). Using dynamic programming optimal quadratic coefficients can be computed at each step in constant time. Since $\sum_{i=1}^{n-1} i = (n^2 - n)/2$ quadratics are computed, and following each computation a total LSE is then calculated on typically far fewer than n values, the algorithm runs in at worst $O(n^3)$. The biggest problem with this algorithm is detecting and coping with singularities when computing quadratic coefficients. Internally 128 bit doubles are used; decrease in LSE (which should never in theory happen) used to detect floating point under/overflow, and linear fit preferred whenever it has a smaller LSE.

D. Discontinuity Identification

This step of our approach filters out discontinuities among all anomalies identified by the previous step of our approach and is composed of the following sub-steps

1) Distribution Comparison: After the anomaly transition period has passed, the value of the performance counters returns back to its equilibrium state, i.e., stable state with respect to the workload. In the event of a discontinuity, the increase or decrease in the value of a performance counter persists after the transition period t as shown in Fig.1 (a, b). This sub-step of our approach compares the distribution of a performance counter before and after the anomaly transition period. We use Wilcoxon rank-sum test [8] to compare the two distributions. We choose this test because it is non-parametric and does not require the data to be normally distributed. We conducted Shapiro–Wilk test of normality [3] to confirm that our data obtained from both industrial and an open source system (discussed in section IV) is not normally distributed. Wilcoxon rank-sum test at the significance level of 1% (i.e., 0.01), ρ-value < 0.001 indicates that the null hypothesis (H_0) (i.e., the two distributions are same) is rejected; we can conclude the presence of a discontinuity.

2) The Effect Size for Measuring Discontinuity: When an anomaly transition period is long, i.e., spans over a few weeks (e.g., slow diffusion of a memory leak) to a month (when a recently added feature is removed or a hotfix is rolled back when a corresponding patch is ready), the value of performance counter will return to the equilibrium state reflecting the normal behavior of the system under corresponding load. However, there will be slight differences between the counter distribution before and after the long transition period either due to carry-over effect of an anomaly or due to counter extrapolation rate, such as monthly growth in workload volume and CPU consumption. In practice, analysts do not consider such a minute difference between the distribution as a disconnect, despite the difference being statistically significant.

We measure the effect sizes of the difference in the distribution of performance counter values before and after an anomaly, in order to confirm discontinuities. Unlike Wilcoxon rank-sum test, which only tell us if the difference of the mean between two populations are statically significant, effect size quantifies the difference between two populations. Research has shown that reporting only the statistical significance may lead to erroneous results [9] (i.e., if the sample size is very large, p-value can be small even if the difference is trivial). We use *Cohen's d* to quantify the effect [9]. Cohen's d measure the effect size statically, and had been used in prior engineering studies [9, 10]. Cohen's d is defined as:

$$Cohen's\ d = \frac{\bar{x}_1 - \bar{x}_2}{s} \qquad (1)$$

Where $\bar{x}_1$ and $\bar{x}_2$ are the means of two populations, and s is the pooled standard deviation [11]. The strength of the effects and the corresponding ranges of Cohen's d value are [12]:

TABLE I. BASELINE PERFORMANCE TEST

Parameter	*Value*
Test Duration	8 hours
Number of driver (load generator) threads	100
Start Request rate (load ramp-up rate)	5
Think time (time to complete and order)	30 seconds
Database size	100 GB
Percentage of new customers	20%
Average number of searches per order	5
Average number of items returned in each search	3
Average number of items per order	20

TABLE II. FAULT INJECTION IN OUR EXPERIMENTS

No	Faults	Type	Experiment
1	CPU Stress	Anomaly	1
2	Transient Memory Stress	Anomaly	2
3	Interfering Workload	Anomaly	3
4	Workload as Multiplicative Factor	Discontinuity	1
5	Change in Transaction Pattern	Discontinuity	2
6	Hardware Upgrade	Discontinuity	3

TABLE III. THE SUBJECT OF THE STUDY

No	System	Domain	Type of data
1	Industrial System	Cloud Provider	Production Data
2	Open Source	E-Commerce	Experiments with an open source benchmark application
3	Simulation	Cloud	Synthetic Data using statistical equations

$$effect\ size = \begin{cases} trivial & if\ Cohen's\ d \leq 0.2 \\ small & if\ 0.2 < Cohen's\ d \leq 0.5 \\ medium & if\ 0.5 < Cohen's\ d \leq 0.8 \\ large & if\ 0.8 < Cohen's\ d \end{cases}$$

Effect size acts as a tunable threshold to reduce false positive identification of discontinuity by our approach. Analysts (based on their domain trends and required granularity to train their forecast models) can set the effect size beyond which (despite being statistically significant), the differences between a performance counter's distribution, before and after the anomaly transition period is considered as a discontinuity.

IV. CASE STUDY

The main goal of this case study is to investigate the effectiveness of our proposed approach for identifying discontinuities in performance data.

RQ 1. How effective is our approach in identifying discontinuities in performance data?

Motivation: A methodology with lower recall won't be adopted in practice since it fails to identify many of the existing discontinuities in performance data. An approach that produces results with high recall and low precision is not useful either since it floods the performance analysts with too many false positives. An ideal approach should identify minimal and correct set of discontinuities in performance data. We evaluated the performance of our approach using precision, recall and F-measure.

A. Subject of Study and Environmental Setup

TABLE III lists the systems studied in this paper. In this section, we describe the environment setup for these systems.

The Industrial System: A data center provided us with the production performance logs of their data center spanning over terabytes (TB). The log contained a wealth of performance counters obtained from 5,500 grids hosting 279 companies over the period of 7 years. The peak number of servers running across grids in any one hour is 12,088. Maximum CPUs on a server is 32.

The Open Source System: The second system under study (SUS) is Dell DVD Store (DS2) application [13], which is an open source prototype of an online e-commerce website. It is designed for benchmarking Dell hardware. It includes basic ecommerce functionalities such as user registrations, user login, product search and purchase. DS2 consists of a back-end database component, a web application component, and a driver program (load generator). DS2 has multiple distributions to support different languages such as PHP, JSP, and ASP and databases such as MySQL, Microsoft SQL server, and Oracle. In this case study, we use the JSP distribution and a MySQL database(s). The JSP code runs in a Tomcat container. Our load consists of a mix of transactions, including user registration, product search and purchases. The configuration of our DS2 load generator for the baseline performance load in our experiments is listed in Table I, to enable the replication of our experiments.

Simulation: Practitioner of the data center provided us with an excel sheet that had synthetic data (representation of a performance log) along with manufactured discontinuities generated using statistical equations and formulas. However, they did not communicate the occurrence of manufactured discontinuities in the data to us.

B. Fault Injection

To study our approach on realistic situations, we must evaluate them in the presence of representative faults (i.e., anomalies and discontinuities). To do so, we first need to choose the category of faults, e.g., software failures, hardware failures and operator/human errors. Pertet et al. [14] performed a study on performance degradation and failure occurrences in an enterprise web service system and concluded that 80% of the performance anomalies in large software systems are due to software inconsistencies and human errors. Therefore, in this paper, we injected anomalies and discontinuities along these two categories. Table II lists the different anomalies and discontinuities for our performance test experiments. Below, we explain the rationale of choosing the anomalies and discontinuities for our experiments

1) **Anomalies**

Memory Stress: According to BlackBerry and Mozilla, the most common anomaly occurring in the field is related to Transient memory issues [30-31]. Transient memory issues (memory spikes) are large increases in memory usage over a relatively short period of time. Therefore, we choose to inject Transient memory anomalies as one of our experiment.

CPU Stress: Large enterprises report that periodic CPU saturation is one of the fundamental field problems [15]. CPU saturation causes anomalous behavior in applications, i.e., not responding fast enough and shutting down many of their features. CPU anomalies can even cause system/applications to

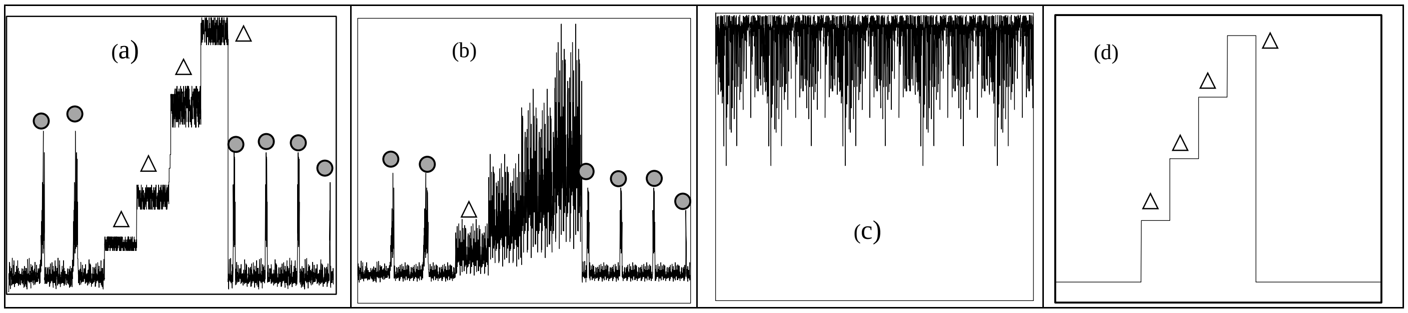


Fig. 4. Performance counters reflecting injcted anomalies and discontinuities ***Legend***:X-axis:Time; Y-axis:Resource utilization; ● Anomaly; △ Discontinuity

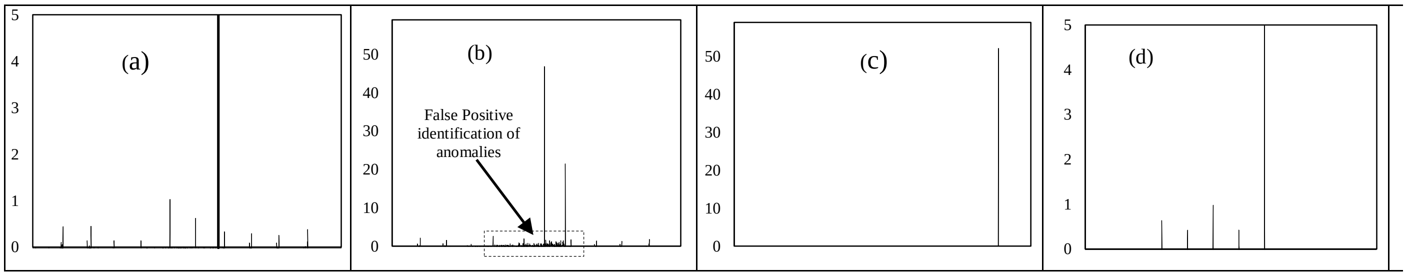


Fig. 5. Anomalies detected by our proposed approach ***Legend***: X-axis: Time; Y-axis: cost

crash or hang under heavy load. The CPU saturation can be due to an unplanned increase in the workload volume. It can also be due to software regression bug, i.e., due to an updated feature of an application in which developers forget to remove the additional executed logic as part of their debugging activity [16]. Even a small set of additional calculations added to a part of the source code which is executed frequently can produce a dramatic increase in CPU usage.

Interfering Workload: Interfering workload anomalies are the major cause of performance degradation in data centers (DC) [17]. Interfering workload anomalies results from competition for resources and occur due to various reasons; as simple as un-announced maintenance on a cluster (e.g., security scans), or a storage array that is performing a system operation such as replication and RAID construction.

2) ***Discontinuities***

The analyst of a data center indicated what they considered the most common reasons for discontinuities. We injected the three common discontinuities described below into our performance test:

Workload as Multiplicative Factor: This to represents increased business due to promotions, new products, mergers & acquisitions of other smaller companies.

Change in Transaction Pattern: A change in transaction pattern can cause discontinuities in both resource and SLA counters such as response time, throughput and latency. A transaction is composed of multiple events that execute in a sequence and are called sequence events [18]. For example, when a user buying an item from Amazon, the user needs to select the items (i.e., selection event *S1*) first before he can checkout (i.e., check out event *C1*) Moreover, he needs to put the selected items in the shopping cart (i.e., update cart event (*U1*) before checkout too. Similarly, shipping (i.e., shipping event *SH1*) cannot be performed before a successful check out to complete a transaction *T1*. Each sequence event in a transaction takes some amount of time and system resources. A new built of an application or an enterprise software deployed in a data center can either affect the future response time of the transaction (i.e., improved or deteriorate response time) or resource consumption.

Hardware and Software Upgrade: Cloud computing and data center consolidation require periodic network upgrades because they drive more data through the same amount of hardware. For example, a virtualized server holds multiple virtual machines, but still only has a single network port. This means that the bandwidth is shared between all of the VMs. Network upgrades can resolve these issues by adding more data throughput or optimizing existing infrastructure to meet current needs. Such hardware upgrades are typical causes of performance data discontinuity.

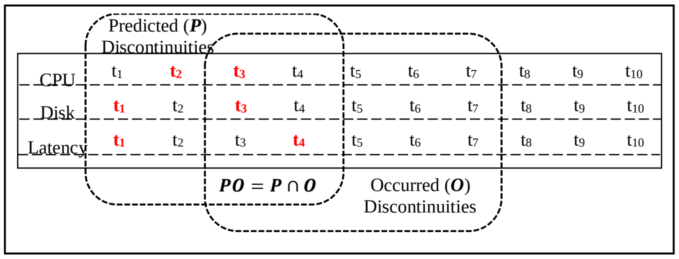
C. *Experiment Design*

We designed five experiments to answer our research question. We used the framework of Thakkar et al. [15] to automate the performance test and to ensure that the environment remains constant throughout the experiments. We used Thakkar framework due to its simplicity and its previous success in practical performance testing [15]. Except for experiment 5, which consisted of production data obtained from the industrial partner and experiment 4, which consisted of synthesizing data using mathematical equations, all other performance test experiments are repeated > 30 times (as suggested by Georges *et al.*[19]) to ensure that measure of variation is not misleading or incorrect, to overcome performance counters instability during the experiments, and to ensure consistency among our findings. The ramp-up and ramp-down (warm up and cool down) [5, 7] periods, usually spanning over 15 minutes were excluded from our analysis, as the system is usually not stable during these periods during performance tests. We used windows 'perfmon' [20] tool to collect the performance data after every 15 seconds (sampling interval) across all the eight machines. The sampling interval is set to 15 seconds to match the sampling interval of production performance data. All the performance tests are eight hours long. Each performance test has 4,242,400 observations from two hundred and twenty performance samples of counter values. We injected three anomalies and three discontinuities in all our experiments except experiment 5, which consist of production data. We now detail the settings of each experiment for faults listed in TABLE II.

Experiment 1 *(CPU Stress & Workload as Multiplicative Factor)*: For experiment 1, we injects the anomalies in DS2 application by triggering resource exhaustion. We ran a performance test with the baseline workload listed in TABLE I. Then, we slowed down the CPU of the web server using a CPU stress tool, known as winThrottle [21]. We choose winThrottle over other CPU stress tools because it is an open source tool and can use features in system hardware that directly modify the CPU clock speed, rather than using software "delay loops" or "HLT instructions" to slow down the machine. We injected discontinuities by triggering a system overload, the second most common failure trigger identified by Pretet and the most common cause of discontinuity pointed out by practitioners. This experiment keeps the workload-mix constant and increases the execution rate of our workload over a significant period of time to 8X, i.e., eight times as the baseline workload configuration.

Experiment 2 (*Memory Stress & Change in Transaction Pattern*): For experiment 2, we conducted a performance test with the same workload as the baseline load listed in TABLE II, but injected a memory bug into the webserver using a customized open-source memory stress tool called EatMem [22]. The tool allocates a random amount of available memory at recurring intervals to mimic a Transient Memory Spike. We also injected discontinuities in experiment 2 using change in transaction pattern. Accessing I/O storage devices, such as hard drives, are usually among the slowest part of a transaction. Changes to I/O operation in an execution can even cause performance regression (i.e., performance discontinuity) [16]. Adding log statements to execution is a common mistake [23]. Log statements are usually required when implementing a new feature. There is a tendency to leave the log statement behind in the source code when a change is finished. We increased the logging for the most frequently accessed source code area in Dell DVD Store, i.e., 'Item Selection' execution event thereby causing discontinuity. In experiment 2, we have to stop the load generator several times to enable increased levels of logging for the Dell DVD store application. However, the 'Perfmon' logs the performance counters for the entire duration of the experiment, i.e. eight hours.

Experiment 3 (Interfering Workload & Hardware Upgrade): This experiment aims to trigger interfering workload anomaly mostly due to procedural errors such as planning a security scan at the time when peak workload is expected or due to unconstrained activities such as RAID construction, self-cleanup activities of mail stores and storage replications. We created an interfering background workload anomaly mimicking a situation where the administrator schedules an antivirus scan that conflicts with the timing of the performance test. We scanned one of the web server machines with an antivirus every 50 minutes for 10 minutes over the course of



Predicted (*P*) Discontinuities
CPU
Disk
Latency
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
$PO = P \cap O$
Occurred (*O*) Discontinuities

Fig. 6. Illustration of our effectiveness measure

eight hours to perturb the main workload. To mimic the discontinuities arising from maintenance activities such as hardware upgrades in experiment 3, we first set an 'Affinity' [24] to use only two CPUs for MySQL process on all three database servers. Periodically, for each database server, we removed the affinity rules for the MySQL process to reflect hardware update, i.e., addition of CPUs.

Experiment 4 (*Synthetic Data*): It is hard to produce cyclic workload in a lab environment, i.e., performance counter values that respond in a continuous wavelength pattern (i.e., period formation of trough and crest) to cyclic workload stimuli. For example, Microsoft exchange server, running MMB3 workload [25], results in CPU and DISK IOPS to follow wavelength patters as shown in Fig. 2. (c). Mathematical formulas in Excel are used to generate cyclic performance countervalues (e.g., CPU utilization with respect to the transaction volume) and manufactured discontinuities (using statistical equations) were used to cause irregularities in the data.

Experiment 5 *(Production logs):* This experiment was conducted on the production data. The analyst of a data center gave us performance logs spanning over seven years without revealing the discontinuities. In particular, analysts were interested to know how our approach performs on two of the specific clients' data which they had already verified for the presence of discontinuities.

D. Measure the Effectiveness of Our Approach

To evaluate the effectiveness of our approaches, we use the following measures: Precision, Recall and F-Measure. Precision is the ratio between correctly identified discontinuities and predicted discontinuities in a performance data. Recall is defined as the ratio between the number of correctly identified discontinuities and the number of actual discontinuities present in performance data. F-measure is defined as a harmonic means of precision and recall [26]. F- Measure $= (\alpha + 1) * Precision * Recall / (\alpha * Precision + Recall)$. The value of alpha (α) ranges between 0 and infinity to give varying weights for recall and precision. For example, in this paper to indicate that recall is as important as precision, alpha has a value of 1.0. For All our experiments (1 to 3), we divided the performance test into equal time intervals from t1 to t10 as shown in Fig.6. For each performance experiment (1 to 3) corresponding anomaly is injected during interval t_1, t_2, t_8, t_9 and t_{10} and discontinuities are injected within time interval t_3, t_4, t_5, t_6 and t_7. We also logged the exact time of all the fault injections in a test. We now use Fig.6 as an example to explain how we measure the precision and recall of our proposed approach. An ideal approach should only report the intervals during which the discontinuities occurred, i.e., O= {t_3, t_4, t_5, t_6, t_7}. We applied our

TABLE IV. THE EFFECTIVENESS OF THE PROPOSED DISCONTINUITY IDENTIFICATION APPORACH

Exp Id	Cohen's d Effect Size												Counter Size	
	Trivial			**Small**			**Medium**			**Large**				
	Prec	*Recall*	*F-Mes*	*Prec*	*Recall*	*F-Mes*	*Prec*	*Recall*	*F-Mes*	*Prec*	*Recall*	*F-Mes*	*Selected*	*Total*
1	0.50	0.80	0.68	0.66	1	0.80	1.00	1.00	1.00	1.00	1.00	1.00	20	220
2	0.60	0.90	0.72	0.8	1	0.88	1.00	1.00	1.00	1.00	1.00	1.00	20	220
3	0.80	0.80	0.80	0.91	0.88	0.95	0.95	0.88	0.91	1.00	1.00	1.00	20	220
4	0.70	0.90	0.78	1.00	0.95	0.97	1.00	1.00	1.00	1.00	1.00	1.00	15	30
5	0.50	0.60	0.54	0.70	0.69	0.69	0.92	0.92	0.92	0.92	0.87	0.90	20	1256
Average	*0.62*	*0.8*	*0.704*	*0.814*	*0.904*	*0.85*	*0.97*	*0.96*	*0.96*	*0.98*	*0.97*	*0.98*	–	–

approach on the three performance counters CPU, Disk, and Latency obtained from the experiment performed and it collectively predicted (among unique time intervals shown in red) discontinuities P = {t_1, t_2, t_3, t_4}. Based on these definitions we define: Recall = $|P \cap O|/|O|$ and Precision = $|P \cap O|/|P|$. Therefore, in the above example, Recall = 2/5 = 0.4, Precision = 2/4= 0.5, and F-Measure = 0.44.

E. Case Study Results

We now report our findings. The Table IV shows, under varying effect size, the effectiveness of our proposed approach. The results are listed using the definition of our performance measure (i.e., *Precision*, *Recall* and *F-measure*) for all the case study experiments. For the first three experiments, the values reported in Table IV are the averages of thirty runs per experiment. The 'Total' counters size represents the number of performance counters harvested from the system-under-test. The 'selected' counter refers to number of performance counters selected among the pool of Top_k counters recommended by our PCA approach that have higher likelihood of revealing discontinuities (if any occurred) . The main constraint on the number of Top_k counters come from practicality. The performance analysts of our industrial partner advised us that they consider 20 performance counters as the maximum that are manageable. Any increase in the number of performance counters beyond 20 negatively affects the human capability to effectively examine and confirm the underlying discontinuities; or to understand the root-cause of an observed discontinuity so as to adjust the parameters of forecast models accordingly.

Overall, our approach has a higher average recall in comparison to its precision. For experiments 1 to 4, using large effect size, the approach performance is ideal because a) we had limited types of discontinuities to inject, and b) discontinuities variations (i.e., abrupt change (jump) in counter values) are limited as compared to what is observed in the production environment. In experiment 5 (production data), our approach performed the best with an effect size set to 'Medium'. Our approach is very sensitive to the variation in the performance data therefore; the efficiency of our approach suffers when effect size (i.e., sensitivity) is set to 'trivial'; our approach achieved the minimum precision of 0.50. We investigated the rationale behind the poor performance of the approach for 'Trivial effect size'. We found that under extreme load such as in experiment 1, where CPU anomaly and 8X workload discontinuity is injected, it took a long time for CPU counter to stabilize and return to its normal state, perturbing the equilibrium of counter's distribution till the next injected fault. The proposed technique when comparing the distribution of *'% CPU Utilization'* before and after the transition period of anomaly (i.e., injected fault), the technique picked up even this minute variation ($Cohen's\ d \leq 0.2$) due to carryover effect and marked it as a discontinuity.

Similarly, for experiment 1, we also found that under extreme CPU stress, the database server refused connection from all of the four webservers in the system under test. This is the default behaviour of *MYSQL* server under extreme stress. The webservers facing heavy workload volume (i.e., from load generators), a) started appending all the intermediate transaction to the disk on priority basis, so that the transaction are not lost and are routed to the database server as soon as the connection is established with it. This caused the values of "*Disk-IOPS*" to rise considerably higher and b) reattempted to establish connection with *MYSQL* server every 10 seconds, causing higher than normal variation in the value of the '*NIC_controller_packet sent'* counter. Moreover, due to the *MYSQL* server under stress, the transaction response time also increased. All these unexpected variations in the performance counter data are perceived as discontinuities by our proposed approach when sensitivity parameter is set too low, i.e. 'Trivial'. Our approach performed well when the effect size is set to higher levels. This is because being the carry-over effect of anomalies, and minute external variation such as linear growth in counter value or its value drift over time is filtered.

All the identified discontinuities (especially for the logs of two customers) were verified by practitioners. Our approach performed up to the satisfaction of the practitioners. For experiment 5, with effect size set to 'Small', the approach was able to identify most of the discontinuities with precision and recall of 0.70 and 0.69. With effect size set to 'Medium', the approach performed better, i.e., achieved with excellent balance of precision and recall (i.e., 0.92, 0.92). With effect size set to 'Large', the recall of the approach suffered no change in its precision.

V. LIMITATION AND THREATS TO VALIDITY

Sensitivity: We can tune the sensitivity of our approach to uncover discontinuities in performance data by adjusting effect size. Though using large effect size reduces false alarms, this may result in an analyst overlooking significant discontinuities. This is a general problem and an automated technique, cannot generally decide whether an identified discontinuity is important or is noise. Analysts have to conduct multiple experiments with different effect size to determine the optimal threshold for the performance data relevant to a project, client, or his own environment.

Distinguishability: Our proposed technique cannot distinguish between overlapping discontinuities, i.e., change is counter's behaviour due to multiple factors over same period of time. Moreover, the proposed technique only identifies the discontinuity. The analyst has to manually inspect the discontinuity and take actions accordingly

External Validity: We used one large industrial and one open source benchmark system to reduce the threat to validity. However, the proposed technique cannot be generalized to other domains such as sensory data arising from network traffic and security monitoring requiring high confidence and reliability in the recommendations. This is due to the fact that there is no guarantee that direction of maximum variance (i.e., the use of PCA) will discover good performance counters for identifying discontinuities. A large anomaly (such as in network traffic) may inadvertently pollute the normal subspace, thereby skewing the assumption that large variances always have important dynamics. Generalizing to any other systems, especially in other domains requires the replication of our approach.

Construct Validity: Since our proposed approach is evaluated, in three out of five experiments, based on injected faults, we tried to reduce the construct validity threat by being systematic with the fault injection process. Despite our careful fault injection mechanism, the type of the injected faults may not be fully representative or real faults.

Internal Validity: This study required various sets of configurations (test environment), implementations (PCA), and data analysis (data handling and statistical analysis). Therefore, to reduce the internal validity threat we used existing frameworks (e.g., Thakkar framework for automating the performance test executions) and packages (e.g., R statistics packages for PCA implementation study)

Conclusion Validity: Experiments 1-3 are executed 30 times each and the average of the results is taken for each effect size. However, the differences among the results produced by our approach might be attributable to the random nature of the experiments. We plan to extend the study with longer (i.e., 36 hours) and more runs per experiment so that statistical significant test can be meaningfully applicable.

VI. RELATED WORK

Detecting anomalies in an enterprise system is not a new problem. However, there is little work done in identifying and diagnosing anomalies in large scales systems using such performance data as, console logs, performance counter logs and executions logs. Most of the work in the literature is divided into two major dimensions, i.e., pre and post deployment anomaly detection.

A. Pre-deployment Anomaly Detection in Large-Scale System

The focus work along this dimension is to help analysts to identify and diagnose anomalies in the system early before they become critical field problems. Closest work to ours is the work done by Foo [4, 27] and Naguyen [16, 28]. Both use performance counters to automate the analysis of performance test an automatically identify performance anomalies in the system. Foo et al. [4, 27] calculate performance signatures from previous executions and use them as a baseline to compare against performance signatures of new executions. This approach is close to regression testing as it validates if anomalies are introduced into newer software versions. They, however, only do comparative analysis, which only provides a Yes/No answer on performance anomalies. In contrast, our approach can pinpoint the time duration at which the anomaly and discontinuity occurs and for how long it prevails, i.e., its transition period. Nguyen [16, 28] used a quality control technique called control charts to flag the anomalies in the performance counters using upper and lower bound limits. Their technique requires deep understanding of the domain to create control limit of performance counters. The variation of the counter values within the limit is considered as normal variation. In contrast, our approach use effect size as a tunable threshold to identify discontinuities, and does not require an analysts to have explicit knowledge about the acceptable limits of all the performance counters values. Unlike our work, Jiang [3] relies on execution logs that capture detailed information. However, such logs are vendor and application specific. This means, that different subsystems in a large-scale system (e.g. web servers, databases, and mail servers) produce a variety of execution logs, each with different levels of information and formats. Whereas, the performance counters data, provide a greater level of unification across subsystems and systems. Malik et al. [6, 29] have used principal component analysis (PCA) to generate performance signatures for each component using performance counters captured during load test. They assess the pair-wise correlations between the performance signatures of a performance test and a baseline test to identify performance anomalies and deviations. However, it's hard to find baselines in rapidly evolving large-scale systems.

B. Post-deployment Anomaly Detection in Large-Scale System

The work in this dimension aim to help analyst identify anomalies in production environment, i.e., once a system or and enterprise application is deployed. Syers et al. [30, 31] proposed an approach to identify performance anomalies and deviation in thread pool using performance counters. Their approach is limited to the detection memory related performance anomalies in enterprise systems (e.g., memory leaks, memory spikes and memory blots). Attariyan et al. [11] proposed a performance summarization approach for identifying root causes of performance anomalies based on human errors, such as misconfigurations. They used dynamic binary instrumentation to monitor an application as it executes instead of execution logs or performance counters. However, their techniques only focus on misconfigurations and do not help to find anomalies.[32-34]. Finally, there are other approaches [27, 29] that use annotated software models to detect performance anomalies [30]. These approaches, however, use software model simulations and not real production software.

VII. CONCLUSION AND FUTUREWORK

The growth in cloud environments and virtualization has increased the need for the forecasting techniques to better satisfy the scalability, elasticity, and cost-effectiveness requirements of cloud environments. The accuracy of a forecasting technique depends of the merit of input data. Analysts spend considerable preparing the data in order to conduct a forecast. We propose a technique that helps analyst automatically identify discontinuities in the performance data. Discontinuity is a change in a time-series pattern that persists (but does not reoccur) since the measurement taken before the discontinuity may be irrelevant, or nearly so. Detecting discontinuities in performance data of a data center is important to improve the forecasts. If an analyst knows a discontinuity has occurred, the analyst may want to ignore the early data and base the forecast on the measurements taken after the discontinuity. Moreover, detecting a discontinuity provide analysts a reference point to retrain their forecasting models and make necessary adjustments. We show how simple statistical techniques can be used to identify discontinuities in large performance data. A large case study on an industrial system as well as a benchmark open source system provides empirical evidence of the ability of our approaches to uncover the discontinuities in performance data. In future, we will attempt to study and categorize discontinuities with that of the corresponding workloads.

VIII. ACKNOWLEDGMENTS

We are grateful to C.A. Technologies Inc., for supporting and funding this research, and for providing access to the production data used in our case study. The findings and opinions expressed in this paper are those of the authors and do not necessarily represent or reflect those of C.A Technologies and/or its subsidiaries and affiliates. This work was funded in part by a Collaborative Research and Development grant from the National Science and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] T. Dasu and T. Johnson, Exploratory Data Mining and Data Cleaning. John Wiley & Sons, 2003.

[2] H. Malik, Z. M. Jiang, B. Adams, A. E. Hassan, P. Flora and G. Hamann, "Automatic comparison of load tests to support the performance analysis of large enterprise systems," in 14th European Conference on Software Maintenance and Reengineering, Madraid, Spain, 32-2010, pp. 222-231.

[3] Z. M. Jiang, "Automated analysis of load testing results," in Proceedings of the 19th International Symposium on Software Testing and Analysis, 42-2010, pp. 143-146.

[4] K. C. Foo, Z. M. Jiang, B. Adams, A. E. Hassan, Y. Zou and P. Flora, "Mining performance regression testing repositories for automated performance analysis," in 10th International Conference on Quality Software, 45-2010, pp. 32-41.

[5] A. Bondi, "Automating the analysis of load test results to assess the scalability and stability of a component," in CMG-CONFERENCE-, 2007, pp. 133.

[6] H. Malik, H. Hemmati and A. E. Hassan, "Automatic detection of performance deviations in the load testing of large scale systems," in 35th International Conference on Software Engineering (ICSE), 2013, pp. 1012-1021.

[7] H. Malik, Z. M. Jiang, B. Adams, A. E. Hassan, P. Flora and G. Hamann, "Automatic comparison of load tests to support the performance analysis of large enterprise systems," in Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on, 2010, pp. 222-231.

[8] F. Wilcoxon, "Individual comparisons by ranking methods," Biometrics Bulletin, pp. 80-83, 1945.

[9] V. B. Kampenes, T. Dybå, J. E. Hannay and D. I. Sjøberg, "A systematic review of effect size in software engineering experiments," Information and Software Technology, vol. 49, pp. 1073-1086, 2007.

[10] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," IEEE Trans. Software Eng., pp. 721-734, 2002.

[11] J. Hartung, G. Knapp and B. K. Sinha, Statistical Meta-Analysis with Applications. John Wiley & Sons, 2011.

[12] J. Cohen, Statistical Power Analysis for the Behavioral Sciences. Routledge Academic, 2013.

[13] D. Jaffe and T. Muirhead, The Open Source DVD Store Application, 2005.

[14] S. Pertet and P. Narasimhan, "Causes of failure in web applications," Parallel Data Laboratory, Carnegie Mellon University, CMU-PDL-05-109, .

[15] D. Thakkar, A. E. Hassan, G. Hamann and P. Flora, "A framework for measurement based performance modeling," in WOSP '08: Proceedings of the 7th International Workshop on Software and Performance, Princeton, NJ, USA, 2008, pp. 55-66.

[16] T. H. Nguyen, M. Nagappan, A. E. Hassan, M. Nasser and P. Flora, "An industrial case study of automatically identifying performance regression-causes," in Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 232-241.

[17] C. Delimitrou and C. Kozyrakis, "iBench: Quantifying interference for datacenter applications," in IEEE International Symposium on Workload Characterization (IISWC), 2013, pp. 23-33.

[18] Z. M. Jiang, A. E. Hassan, G. Hamann and P. Flora, "An automated approach for abstracting execution logs to execution events," 43, vol. 20, pp. 249-267, 43-2008.

[19] A. Georges, D. Buytaert and L. Eeckhout, "Statistically rigorous java performance evaluation," ACM SIGPLAN Notices, vol. 42, pp. 57-76, 2007.

[20] M. Knop, J. Schopf and P. Dinda, "Windows performance monitoring and data reduction using watchtower," in 11th IEEE Symposium on High-Performance Distributed Computing (HPDC11), 35-2002, .

[21] Leyda, M. and Geiss, R., "WinThrottle," 2010.

[22] E. James McCaffrey. (). . Available: http://msdn.microsoft.com/en-us/magazine/cc163613.aspx.

[23] H. W. Gunther, "Websphere application server development best practices for performance and scalability," IBM WebSphere Application Server Standard and Advanced Editions-White Paper, 2000.

[24] A. Foong, J. Fung and D. Newell, "An in-depth analysis of the impact of processor affinity on network performance," in Proceedings. 12th IEEE International Conference on Networks, 2004, pp. 244-250.

[25] S. Stanford, "MMB3 Comparative Analysis–White Paper," 2003.

[26] J. F. Limbrunner, R. M. Vogel and L. C. Brown, "Estimation of harmonic mean of a lognormal variable," J. Hydrol. Eng., vol. 5, pp. 59-66, 2000.

[27] K. C. D. Foo. Automated discovery of performance regressions in enterprise applications. 2011. Available: http://sail.cs.queensu.ca/publications/pubs/masterthesis2011_foo.pdf.

[28] T. H. Nguyen, B. Adams, Z. M. Jiang, A. E. Hassan, M. Nasser and P. Flora, "Automated verification of load tests using control charts," in 18th Asia Pacific Software Engineering Conference (APSEC), 2011, pp. 282-289.

[29] H. Malik, B. Adams and A. E. Hassan, "Pinpointing the subsystems responsible for the performance deviations in a load test," in IEEE 21st International Symposium on Software Reliability Engineering (ISSRE), 2010, San Jose, California, USA, 30-2010, pp. 201-210.

[30] M. D. Syer, B. Adams and A. E. Hassan, "Identifying performance deviations in thread pools," in Software Maintenance (ICSM), 2011 27th IEEE International Conference on, 2011, pp. 83-92.

[31] M. D. Syer, Z. M. Jiang, M. Nagappan, A. E. Hassan, M. Nasser and P. Flora, "Leveraging performance counters and execution logs to diagnose memory-related performance issues," in 29th IEEE International Conference on Software Maintenance (ICSM) , 2013, pp. 110-119.

[32] L. Cherkasova, K. Ozonat, N. Mi, J. Symons and E. Smirni, "Automated anomaly detection and performance modeling of enterprise applications," 09, vol. 27, pp. 1-32, 09-2009.

[33] G. F. Creţu-Ciocârlie, M. Budiu and M. Goldszmidt, "Hunting for problems with artemis," in Proceedings of the First USENIX Conference on Analysis of System Logs, 39-2008, pp. 2-10.

[34] R. Gunasekaran, D. A. Dillow, G. M. Shipman, D. E. Maxwell, J. J. Hill, B. H. Park and A. Geist, "Correlating log messages for system diagnostics," in Cray Users Group Conference, Edinburgh, United Kingdom, 2010, pp. 1-10.

[35] I. Davis, H. Hemmati, Ric. Holt, Michael. Godfrey, D. Neuse, Serge. Mankovskii, "An Empirical Investigation of An Adaptive Utilization Prediction Algorithm", 23rd Annual International Conference on Computer Science and Software Engineering, Canada, 2013.

[36] I. Davis, H. Hemmati, Ric. Holt, Michael. Godfrey, D. Neuse, Serge. Mankovskii, "Strom Prediction in a Cloud", 5th International Workshop on Principles of Engineering Service-Oriented Systems, USA, 2013.