# Automatic Comparison of Load Tests to Support the Performance Analysis of Large Enterprise Systems

Haroon Malik, Zhen Ming Jiang, Bram Adams, Ahmed E. Hassan
School of Computing
Queen's University
Kingston, Canada
{malik, zmjiang, bram, ahmed}@cs.queensu.ca

Parminder Flora and Gilbert Hamann
Performance Engineering
Research In Motion (RIM)
Waterloo, ON, Canada

***Abstract*— Load testing is crucial to uncover functional and performance bugs in large-scale systems. Load tests generate vast amounts of performance data, which needs to be compared and analyzed in limited time across tests. This helps performance analysts to understand the resource usage of an application and to find out if an application is meeting its performance goals. The biggest challenge for performance analysts is to identify the few important performance counters in the highly redundant performance data. In this paper, we employed a statistical technique, Principal Component Analysis *(PCA)* to reduce the large volume of performance counter data, to a smaller, more meaningful and manageable set. Furthermore, our methodology automates the process of comparing the important counters across load tests to identify performance gains/losses. A case study on load test data of a large enterprise application shows that our methodology can effectively guide performance analysts to identify and compare top performance counters across tests in limited time.**

**Keywords-Load Test; Performance Counters; PCA**

## I. INTRODUCTION

As modern software systems grow larger and more complex, periodic maintenance of these systems is required to satisfy the high business demands on system quality, availability and responsiveness. Classic approaches to system maintenance fail, when applied to large scale systems, i.e., Google, Facebook or Amazon [20]. While new approaches to maintain such large-scale software systems are popping up, load testing still remains the most integral part of testing the performance of large scale systems.

Load Testing assesses how a system performs under a given load [2]. Load is the rate at which transactions are submitted to a system [1]. Once maintenance activities (corrective, perfective and adaptive) are completed, load testing helps to uncover residual functional and performance problems. Functional problems are bugs such as deadlocks and memory leaks that slipped through the functional tests. Performance problems manifest themselves as system freezes, crashes and becoming unresponsive during the course of a load test and are related to symptoms such as high response time and low throughput under load.

During load testing, a series of load tests are conducted that may span from a few hours to many days. Often, one or more load generators are used to imitate committing thousands of concurrent transactions to an application on behalf of users. During the course of a load test, the application under test is closely monitored, resulting in a huge amount of logging data as performance counters. The performance counter log contains a plethora of usage information such as CPU utilization, disk I/O, memory consumption and network traffic. Such information is of vital interest to performance analysts as it helps them to observe system behavior under load.

The most frustrating challenge of load test faced by system analysts is the time spent and complexity involved in isolating the required information distributed across thousands of correlated performance counters. The modern hardware and advance applications used in large systems can publish large sets of counters for monitoring their performance [2], which further increases the complexity involve in analyzing load test. As a consequence, analysts are overwhelmed by thousands of performance counters obtained during load tests, making performance analysis laborious and results vague. Existing research on load testing focuses on the automatic generation of load test suites [1] and automated performance analysis of load tests based on execution logs [10][11]. Even though load testing has been extensively studied, little is known on how to do it for large scale systems with thousands of performance counters generating terabytes of log data. Our paper is the first to study such an industrial sized system and to develop methods to handle the information overload. To summarize, the paper makes the following contributions:

- ***C1.*** We apply statistical methods to reduce the dimensionality of the observed performance counter set.
- ***C2.*** We automate the ranking of counters according to their importance for load tests.
- ***C3.*** We empirically validate our proposed approach through a large case study on a real-world industrial software system.

The rest of the paper is organized as follows: Section 2 highlights the problems associated with load testing. We present our solution to these problems in section 3 along a step by step explanation of our methodology. In section 4, we present our case study. Section 5 explains the threats to our validity. Related work is presented in section 6, followed by our conclusion and future work in section 7.

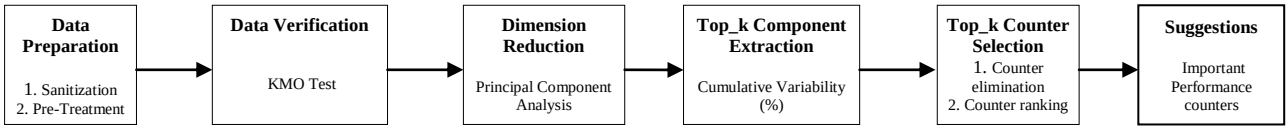


Figure 1: Steps involved in proposed methodology to identify the important performance counters in a load test.

Table 1: Sample of observations before data preparation

| | Observations | | | | | | |
|---|---|---|---|---|---|---|---|
| Var | Tot | Mis | Avail | Mini | Max | Mean | Std. Dev |
| Q | 599 | 0 | 599 | 246.18 | 1946.11 | 754.654 | 292.00 |
| R | 599 | 0 | 599 | 009.59 | 0063.46 | 023.427 | 011.14 |
| S | 0 | 0 | 0 | 000.00 | 0000.00 | 000.000 | 000.00 |
| T | 599 | 2 | 597 | 001.00 | 0117.11 | 0030.90 | 018.99 |

Table 2: Sample of observations after data preparation

| | Observations | | | | | | |
|---|---|---|---|---|---|---|---|
| Var | Tot | Mis | Avail | Mini | Max | Mean | Std. Dev |
| Q | 599 | 0 | 597 | -13.37 | 000.07 | 0.00 | 1.00 |
| R | 599 | 0 | 597 | -00.71 | 006.52 | 0.00 | 1.00 |
| T | 597 | 0 | 597 | -1.694 | 001.46 | 0.00 | 1.00 |

## II. PROBLEM DESCRIPTION

Large enterprise applications must be load tested to ensure satisfactory performance under load. Many issues in load testing remain unsolved.

### *1)* Cloud of Performance Counters.

*Size:* Load tests can last from couple of hours to several days. It generates performance logs that can be of several terabytes in size. Even logging all counters on a typical machine at 1Hz generates about 86.4 million values in a single week. A cluster of 12 machines would generate 13 TB of performance counter data per week, assuming a 64 bit representation for each counter value [15]. Analysis of such large counter logs is still a big challenge in load test.

*Redundant counter traffic:* During load testing, the large number of processing elements generates a large number of performance counter data, most of which contains unnecessary and overlapping information. Evaluation of the system performance depends upon the merit of collected counter data. Such mixes of redundant counter data act as noise, reducing the accuracy of performance evaluation.

### *2)* Time Limitations.

*Production constraints*: In a production environment, performance analysts must react in limited time to complete diagnostics on performance counter logs and to make necessary configuration changes.

*Tight schedules:* Load testing comes in to the picture after the successful completion of functional and user interface testing. It is usually the last step in an already tight and usually delayed release schedule. Hence, managers are always eager to reduce the time allocated for performance testing.

### *3)* Ad hoc checking of counter values.

*Manual analysis:* Most performance engineers manually analyze the counters. For example, they look at memory usage trend to flag memory leaks. In some cases they have custom scripts to cover a few key scenarios based on domain knowledge by comparing counters against an informal baseline [11].

*Complex tools:* Few analysis tools have been developed for performance counters [16]. These tools are either too complex or are hard to integrate with existing systems. This makes analysis laborious and extremely time-consuming [26].

Our proposed solution addresses the aforementioned load test challenges. 1) It reduces the observed number of counters by removing redundant counter traffic; 2) it automatically identifies and compares important performance counters across load tests to reduce the analysis time; 3) It can be easily integrated with other systems and requires no expertise to operate.

## III. PROPOSED SOLUTION

This section discusses our methodology in action, using a real world performance counter log consisting of 18 performance counters. We explicitly choose a small performance counter set in this instructive section as the small data size makes it easier to follow the statistical techniques in our methodology. Figure 1 show all steps involved in our methodology to obtain the top performance counters from a performance log.

### *A. Data Preparation.*

Performance log data needs to be prepared to make it suitable for the statistical techniques employed by our methodology. The effectiveness of the suggestions generated by our methodology greatly depends on data preparation. The two steps involved in data preparation are:

*1) Data sanitization:*

Performance logs need to be filtered from noise i.e., missing counter data or an empty performance counter. Counter data is missing when performance monitor fails to record an instance of a performance counter. A counter is empty when a resource cannot start the service. Table 1 shows a sample of our real world performance counters for a load test before the data preparation step is applied. Counter 'T' belongs to the missing counter data category, whereas 'S' is an empty counter. A Total of 599 observations were required for each performance counter. Monitoring tool recorded only 597 observations for performance counter 'T'. To deal with this kind of problem (incomplete data) we employed *list wise deletion*. If the $i^{th}$ observation for counter 'T' is missing, list wise deletion will delete the corresponding $i^{th}$ observation of all the counters.

Table 3: Principal Component Analysis

| *No* | *PC* | *Eigen-Value* | *Difference* | *Variability (%)* | *Cumulative Variability (%)* |
|---|---|---|---|---|---|
| 1 | PC1 | 11.431 | 8.684 | 63.506 | 63.506 |
| 2 | PC2 | 2.747 | 1.027 | 15.260 | 78.765 |
| 3 | PC3 | 1.720 | 0.794 | 9.554 | 88.319 |
| 4 | PC4 | 0.926 | 0.476 | 5.143 | 93.463 |
| 5 | PC5 | 0.449 | 0.129 | 2.497 | 95.960 |
| 6 | PC6 | 0.320 | 0.160 | 1.780 | 97.740 |
| 7 | PC7 | 0.160 | 0.023 | 0.890 | 98.630 |
| 8 | PC8 | 0.138 | 0.058 | 0.764 | 99.394 |
| 9 | PC9 | 0.080 | 0.052 | 0.442 | 99.836 |
| 10 | PC10 | 0.027 | 0.027 | 0.153 | 99.989 |
| 11 | PC11 | 0.001 | 0.000 | 0.008 | 99.997 |
| 12 | PC12 | 0.001 | 0.000 | 0.003 | 100.00 |

Table 4: Association between counters and components

| | Loadings | | | |
|---|---|---|---|---|
| *Var* | *PC1* | *PC2* | *PC3* | *PC4* |
| A | 0.399 | -0.142 | -0.075 | 0.859 |
| E | 0.933 | -0.253 | -0.068 | -0.145 |
| F | -0.154 | 0.112 | -0.888 | 0.025 |
| I | 0.000 | -0.135 | 0.912 | -0.005 |
| J | 0.623 | 0.752 | 0.053 | 0.050 |
| K | 0.624 | 0.751 | 0.066 | 0.050 |
| L | 0.864 | -0.267 | 0.221 | 0.154 |
| M | 0.972 | 0.019 | -0.046 | -0.134 |
| N | 0.974 | 0.015 | -0.043 | -0.130 |
| O | 0.889 | -0.245 | 0.022 | 0.160 |
| P | 0.944 | -0.167 | -0.065 | -0.164 |
| Q | 0.946 | -0.168 | -0.064 | -0.160 |
| R | 0.966 | -0.146 | -0.043 | -0.047 |

Empty counters such as ‘S’ and counters that have more than 2% of the data missing are removed during the sanitization process. Table 2 shows the performance counters after data sanitization.

*2) Pre-treatment:*

Pre-treatment converts the data into a format that is understood by the data reduction technique, i.e., Principal Component Analysis (PCA). PCA is a maximum variance projection method [12]. This means that PCA identifies those variables that have large data spread (variance), ignoring variables with low variance [3]. Performance counters have different ranges of numerical values; they have different variance. To eliminate PCA bias towards variables with a larger variance, we standardized the performance counters via Unit Variance scaling (UV scaling). For each performance counter, we standardized the performance counter by dividing the observations of each counter by the counter’s standard deviation. Each scaled counter then has equal (unit) variance.

Table 2 shows the counters after pre-treatment. Each counter has mean of *0* and Standard deviation of *1*. Scaled performance counter data is then further mean centered to reduce the risk of collinearity. With mean-centering, the average value of each performance counter is calculated then subtracted from its respective counter data.

*B. Data Verification.*

The second step of our methodology verifies if there exists enough association among performance counter data to proceed with the data reduction. In order to apply PCA, the KMO (Kaiser-Meyer-Olkin) measure [13] should be greater than 0.6 [4]. This measure tests the amount of variance within data that can be explained by a given measure. The KMO measure for our performance counter data is 0.789, which indicates PCA is appropriate to apply.

*C. Dimension Reduction*

We consider the elimination of redundant performance counters as a dimensionality reduction (DR) problem, where each counter corresponds to a dimension. Many different DR techniques exists, for example based on statistics clustering (factor analysis, alpha, un-weighted least-square) or machine learning (Maximum likelihood, Feature selection, cross entropy, etc.,) [32]. Among statistical techniques clustering algorithms have been widely used and perform reasonably well on datasets of low dimension, with “low” defined as less than fifteen [33]. Unfortunately, we expect to have dimensions over 1000 in our test data sets and in the field environment. Several authors have pointed out that the clustering method is not fully effective when clustering high dimensional data [33][34][35]. Maximum likelihood algorithms belonging to machine learning class are well known for dimension reduction. However, maximum likelihood procedures are limited in their ability to accurately estimate the population mean and SD when the percent of concealed data is large and sample size is small [36]. Towards this end, we used statistical technique, Principal Component Analysis (PCA), known to reduce the sheer volume of performance counters and are both; robust and scalable [12]. What PCA does is to synthesize new variables called 'Principal Components' (PC). Every PC is independent and uncorrelated with other PCs.We used custom R files and the FactoMineR package dedicated for data mining and multivariate analysis to perform the PCA analysis [27]. The result of applying PCA on our performance counter data set can be seen from Table 3. The 18 counters have been reduced into 12 Principal components (PC) thereby achieving a 33.3% reduction.

Because of the pre-processing phase the variance of each counter = 1.0. PCA groups the data of the 18 counters into components, each of which explains a particular amount of variance of the original data. This means that the total variance of our counter data can be explained as 18. The first component PC1 has eigen-value = 11.431, which means it explains more variance than a single counter, indeed 11.431 times as much, and it accounts for 63.60% of the variability of entire counter data set. The second and third components have eigen-values 2.74 and 1.720 respectively. The rest of the components explain less variance than a single counter.

*D. Top_k Components*

Many performance counters have little information value but hamper effective analysis by adding noise. In cases, few outliers may group together to form a component. These outliers may be of interest to analyst in understanding

extremes and identifying anomalies but add no value towards identifying the top_k performance counters and hence such PC needs to be discarded from analysis.

Unfortunately the methods known today, does not provide any reliable and automated techniques to identify appropriate top_k principal components. [4] [18] [13] [24]. We found it more practical to use '% Cumulative Variability' in selecting the number of top_k component. The Table 3 shows that 4 PCs account for 90% of cumulative Variability. A Cumulative variability of 90% is adequate to explain most of data with minimal loss in information [12]. Using *'% Cumulative Variability'* we achieved 66% data reduction by selecting first four PCs from total of 12 as shown in Table 3.

### *E. Top_k Counters*

Performance analysts are interested in performance counters not principal components. In this step we decompose principal components using eigen vector decomposition technique to map the PCs back to counters [6]. For each performance counter we measure its association to each top_k components. This measure of association is called as *'Loadings'*. Table 4 shows the measure of association (loadings) for few of our performance counters. The loading value ranges from ± (0 to 1). The counters 'N' and 'M' with higher loading values confirm strong association with PC1, whereas counters like 'I' and 'F' confirm weak association with PC1.

In order to remove weakly associated counters (add no value to the PCs) and to identify and rank the top_k counters our methodology performs the following two sub steps:

*1) Counter elimination:*

In this step, the counters that do not have significant association with their respective top_k dimension are removed. A Norman cut-off criterion [12] is utilized to decide on the level of importance of a variable to corresponding dimension:

$$\text{Cut off} = 5.152/[\text{SQRT}(n-2)],$$

*Where the loading value is considered 5.152 only if we have more than 100 samples and N represents number of samples.*

*2) Counter Ranking :*

In this step, the important top_k counters belonging to the top_k PC are identified and ranked. Identifying important variables has been made possible in the literature by exploiting loading values in a strict manner.

In past literature loading value of 0.7 is used as cutoff criteria to obtain important counters [5]. Hair et al. call loadings above .6 "high" and those below .4 "low" to rank important variables [22]. Raubenheimer pointed out 0.4 for the central PC and 0.25 for the other PC [8].

We believe the cut-off level to identify the top_k counters should not be fixed. It should be tunable on the basis of domain demands. If an analyst is tight on time he/she may want our methodology to suggest few top_k performance counters. In a situation where an analyst wants to conduct any fine grained analysis, analyst may require our methodology to increase the span of top_k counter in its suggestion. To server this purpose, we incorporated loading as tunable parameter in our second step. We have found while conducting our case study that loading value of 0.9 or higher works wonder in identifying important performance counters; just enough in count that can be easily managed by human for conducting analysis.

Table 5: Top_k performance counters

| *Rank* | *PC* | *Counters* | *Loadings* |
|---|---|---|---|
| 1 | F1 | N | 0.974 |
| 2 | F1 | M | 0.972 |
| 3 | F1 | R | 0.966 |
| 4 | F1 | Q | 0.946 |
| 5 | F1 | P | 0.944 |
| 6 | F1 | E | 0.933 |
| 7 | F2 | I | 0.912 |

With the loading parameter value set to 0.9, our methodology identifies 7 out of 18 important performance counters along different dimensions, thereby achieving a 61% data reduction. Table 5 shows the important performance counters ranked in the order of importance.

## IV. CASE STUDY

To find out the performance and reliability of our approach we did a case study based on the performance counters logs obtained from the load and stress tests of a large enterprise application. Our case study is built on our intuition to seek answers for our two major concerns:

Q1. How sensitive is our methodology to changes in counter data?
Q2. How much data reduction can be achieved by our methodology?

While investigating our two research concerns, we came across more research questions:

Q3. Can our methodology indentify test with varying workload intensity?
Q4. Can we identify different phases in a load test?

The Figure 2 shows our test environment. The enterprise application runs on a cluster and utilizes a database server to store its data. An external load emulator mimics user's interaction with the application, whereas, the internal load generator places load on the database by emulating large transactions.

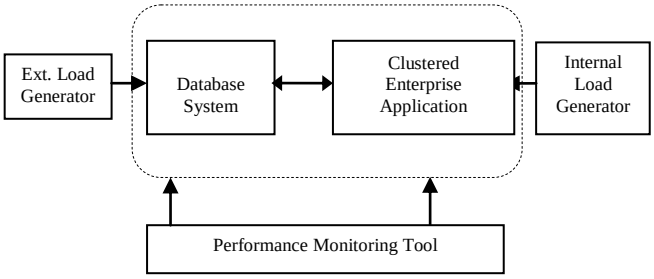


Figure 2: Component of test environment

**Q1. How resilient is our methodology to small fluctuations in a load test data?**

***Motivation:*** Each added feature in an application requires conducting various performance tests. Performance analysts compare the result of a test with similar other tests across various builds and versions of an application. This helps them to identify the performance gain/loss. A test repeated in a controlled environment may not produce 100% identical results. Statistical techniques are highly sensitive to minute fluctuation in data [24][12]. We incorporated statistical techniques in our methodology, which raise our concerns 1) is our methodology robust enough to provide consistent set of counter recommendation. 2) Statistically, how identical are the recommendations. To date, there is no previous study suggesting PCA as a stable technique to accommodate small variations in data. However, work conducted by Ahn and Vetter suggests that PCA is an appropriate technique that can deal with large volume of correlated performance counter (hardware) data, as compared to machine learning techniques [37].

***Approach:*** We conducted an experiment consisting of four runs of same test scenario with constant workload. We used the frame work of Thakkar et al. to automate the tests and to ensure the environment remains constant [29]. Each test ran for two hours. We expected these four runs of test to be similar.Domain experts provided us with 25 performance counters of their own choice from the test. We applied our methodology on the set of these 25 performance counters.

***Findings:*** Our methodology suggested 15 important counters among them. This is a 40% reduction. Domain experts agreed with the recommendation set. Among the 10 removed counters, 7 of them were found to be redundant and 3 of them were removed by our methodology as noise. More importantly, our methodology suggested the same set of performance counters for all four tests. We then plotted a line and a bar chart to compare the loadings of important performance counter across all four tests as shown in Figure 3. Visual comparison of all four tests reveals them to be analogous. We statistically evaluate the performance of our approach using spearman correlation between tests. Table 6 shows the result of the spearman correlation as a correlation matrix. It is based on the importance of variables at significance level alpha = 0.05. Tests 1 & 3 are found highly correlated. The spearman correlation coefficients in Table 6 are greater than 0.993 indicating that there is very strong correlation between the results of our approach for all four tests.

> Our methodology based on PCA is resilient to small variations in performance counter data.

**Q2. Can our methodology identify tests with varying workload intensity?**

***Motivation:*** Large systems need to be tested against odds and unforeseen of field deployment. Performance engineers conduct stress tests to measure the performance of a system under extremes i.e., application halting, malfunctioning, noticeable degradation or crashing.

Table 6: Correlation among tests

| | *Test-1* | *Test-2* | *Test-3* | *Test-4* |
|---|---|---|---|---|
| *Test-1* | 1 | 0.999 | 1.000 | 0.996 |
| *Test-2* | | 1 | 0.999 | 0.993 |
| *Test-3* | | | 1 | 0.996 |
| *Test-4* | | | | 1 |

Table 7: Work load Intensity

| *Type* | *Ext. Load* | *Int. Load* | *# of tot transactions/min* |
|---|---|---|---|
| *1-X* | 1000 | 10 (%) | 4800 |
| *2-X* | 2000 | 20 (%) | 6000 |
| *4-X* | 4000 | 40 (%) | 8400 |
| *8-X* | 8000 | 80 (%) | 13200 |

Table 8: Correlation among X Loads.

| | *1X Load* | *2X Load* | *4X Load* | *8X Load* |
|---|---|---|---|---|
| *1X Load* | 1 | | | |
| *2X Load* | 0.703 | 1 | | |
| *4X Load* | 0.570 | 0.957 | 1 | |
| *8X Load* | 0.219 | 0.462 | 0.513 | 1 |

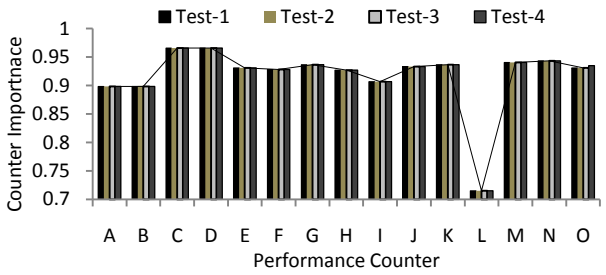


Figure 3: Consistency among tests

One way to stress test a system is by pushing more load beyond its expectation. The rationale behind such kind of performance test is that some performance bugs in system only manifest under certain workload intensity.

Our methodology has already identified tests repeated over similar scenario to be identical. Now, we are interested to find out if we can work our methodology a step further i.e., to identify test generated by same workloads mix but with varying intensity.

We illustrate what we mean by this. For example, the load of an e-commerce website would contain information such as: browsing (40%) with a min/average/max rate of 5/10/20 requests/sec, and purchasing (40%) with a min/average/max rate of 2/3/5 requests/sec. In our experiment, we keep the workload mix (browsing (40%) and purchasing (40%)) constant, but we vary the workload intensity, i.e., rate (request/sec).

***Approach:*** We conducted our second experiment based on the stress tests. For these tests the workload mix was kept constant; however we varied the load intensity to induce additional stress on the system. The Table 7 lists the intensity of our stress tests.

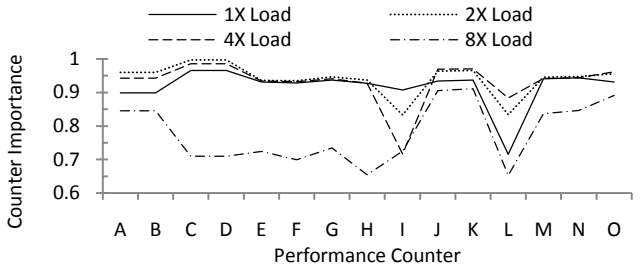


Figure 4: Stress test

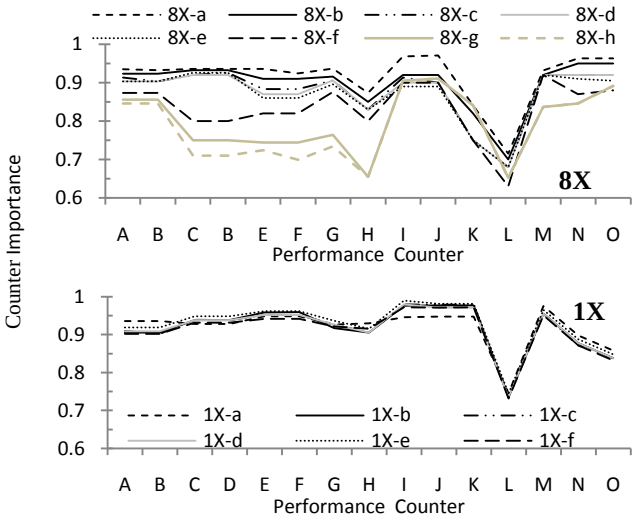


Figure 5: Evolution of performance counter importance

For each stress test 1X, 2X, 4X and 8X, we doubled the external load intensity i.e., the number of *transactions* generated by external load generator per minute. For internal load the *user activities* on the enterprise application is doubled as shown in Table 7. The '# of tot transactions/min' lists the number of total transaction processed by the enterprise application for each type of stress test. We marked the important performance counters obtained from 1X stress test as our base counters and compared their values across the other X stress tests, like a performance analyst would do to compare the performance of a new load test to the performance of an old load test.

***Findings*:** Figure 4 shows that by visualization one can easily identify that 1X, 2X and 4X stress tests are similar in nature, i.e., they share similar underlying patterns of counter importance. The 8X stress test has a high deviation from the others stress tests as seen in Figure 4 and Table 8. Under 8X stress load we noticed that the system shows abnormal behavior and produce an intolerable transaction response time. We believe a system under such stress is unable to reach a stable state resulting in a performance counter log with lot of noise.

*Our methodology can help performance analysts to identify and compare load tests with different load intensities.*

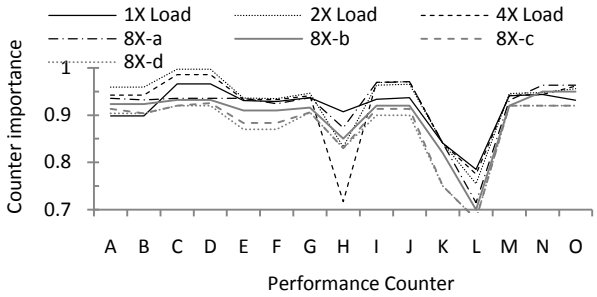


Figure 6: 8X running segments Vs stress tests

**Q3. How does the ranking of a performance counter evolve over time?**

***Motivation:*** 8X stress test had high deviation from other X stress tests in our previous experiment. One way to understand the rationale of such distinction across tests is to study the evolution of the performance counter importance over time.

***Approach**:* We compare the evolution of the important performance counters of 8X to that of 1X test (most stable test in experiment 2). The performance counter logs are divided into multiple *running segments over time to study the performance counter evolution*. Each running segment is built on top of the previous accumulative segments, i.e., the segment 1X-a consists of the performance counters collected during the first 20 minutes of the stress test. 1X-b consists of counters collected during the first 40 minutes of stress test, 1X-c for the first 60 minutes and so on.

The 8X stress test has 2 more running segments than the 1X stress test. This is because the 8X stress test took 25% more time than 1X to finish, even when the load generators were stopped at the same point of time for both tests. The reason for this is that, 8X stress have extreme load intensity than 1X. This extreme load intensity caused intense users requests queues build-up at the enterprise application. It took more time for the enterprise application to process pending users requests in the queues and route them to the database server to commit transactions.

The figure 5 shows the evolution of performance counter importance for both 1X and the 8X.

***Findings:*** The Interesting findings for this experiment are:

a. All the segments of 1-X follow the same line trajectory in Figure 5. No abrupt spike is seen for any performance counter in all segments. This confirms that the evolution of a performance counter importance is a gradual process in load test.

b. The long ramp-down phase was responsible for the divergence of 8X from other X tests. The ramp-down phase is also known as cool-down phase when pushing of the load is stopped and the load for an application slowly releases. In Figure 5, the line trajectory of 8X-f starts to noticeably depart from previous running segments. In parallel, the decrease in the importance of performance counters is observed. The segment 8X-f marks the time when the load generators were stopped.

As the ramp down or cool down period of 8X progresses, the counter importance in subsequent segments 8X-g and 8X-h gradually declines. The ramp down phase of 8X is 75% larger than other 1X test. Once the domain experts saw the visualization of the evolution of counter importance in Figure 5, they were immediately able to understanding why 8-X was way off as compared to other X-test in experiment 2.

We plotted running segments 8X- a, b, c and d (the first 100 minutes to eliminate ramp-down segments) along with the other X stress tests. The line trajectory of the 8X running segments and X stress test complimented each other to a level where they can visually be identified as similar as shown in Figure 6. Despite the system under extreme stress we were able to identify the load/stress test of similar nature (same workload mix) by filtering out the ramp down segments.

*The evolution of performance counter importance during the load test is a gradual process. Different phases of test bias the overall importance of performance counter. Tests should be compared across their corresponding phases.*

**Q4. Can we identify different phases of load test?**

***Motivation:*** From experiment 3, it was concluded that phases of a load test have a different impact on the importance of performance counters. Performance analysts are interested to know which counters are important at the three phase of the test i.e., 1) Warm-up (ramp-up) phase: During which the application is being subjected to the workload. The workload is not at its full strength, but is building up towards the designated workload intensity. 2) Steady-state phase: When the environment is well configured and the application can sustain the workload. During this phase, the performance counters are normally distributed with respect to their average data values. 3) Cool-down (ramp-down) phase: During which the load generator gradually stops injecting the workload and the resource utilizations gradually drop as the workload is winding down. Performance analysts are interested to know the important counters during warm-up phase that cause the resource saturation, whereas they are also interested to know the important counters during the cool down stage, as this helps them to understand how quickly the system can recover from stress.

***Approach:*** We took the performance counter log for our 1-X stress test and divided it into six equal segments. Each segment of the test spanned exactly 20 minutes and is not a running segment. Our intuition was to check the importance of the counters for all segments and find out if the importance values vary. We named the segments as 1X-a, b, c, d, e & f respectively as shown in figure 7.

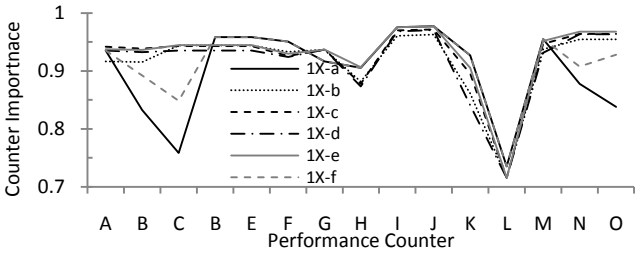


Figure 7: Equal segments of performance counter log

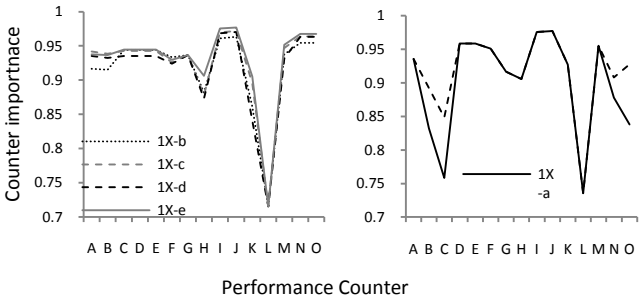


Figure 8: Segment patterns at time

We applied our methodology to rank important counters for each of the test segment. Our methodology recommended the same 15 counters across all segments; therefore, we did not have to establish a base- line segment. We compared the importance of performance counter across all six segments as shown in Figure 7

***Findings:*** At first we were surprised by the diversity of the results (line trajectories), which confirms the findings of Q3. We grouped together the segments that had similar counter importance pattern (line trajectory). Figure 8 shows that we were able to distil the two distinct patterns from all segments. We observed that the harmonized patterns shown in Figure 8 (a) belong to the segments where the system stabilized against the induced load. The two similar patterns in Figure 8 (b) belonging to segments 1X-a and 1X-f correspond to the ramp up phase, when the stress load was gradually being built up by external load generator and ramp down phase when the stress load is being released from the system. The 1-X stress load with small load intensity as compared to other X stress tests did not create any user's requests queue at application server; hence the stress load from application server was released nearly in same proportional to its built-up. Therefore, we find that both the line trajectory of 1X-a and IX-f in Figure 8 (b) are similar. We also notice from the figure that the line patterns diverge considerably at two points. In start of the trend i.e., at counter 'C' there is a 'V' shape dip and towards the end of line pattern for performance counter 'N' and 'O'. Performance counter 'C' is enterprise application's 'Disk Transfer/sec'. Which is the combination of disk reads and writes. We found out that when the stress load is being ramped up, the enterprise application starts to receive an increasing number of request from the internal load generator and the utilization of disk 'Writes' to store the

request increase. Whereas, 'Reads' from disk to process these request is not rapid as compared to 'Writes' at that ramp up phase. Thus, we see a sharp decrease in 'Disk Transfer/sec' counter. Vice-versa, for segment 1X-f, we see a sharp decrease in counter 'C' importance. On the other hand, the same effect is seen on the database server side in terms of counters 'N' and 'O' which represent the database server server's 'Total Disk Writes/sec' and '(Total) % Processor Time'.

*Our methodology can distil different phases of a load test.*

**Q5. How does our methodology performs with large set of performance counters?**

***Motivation:*** Performance of many DR techniques decreases as the number of data dimensions increase. In a large enterprise system, we expect to have performance logs consisting of hundreds of performance counters. Therefore, we want to know how well our methodology performs with a large set of performance counter logs.

***Approach:*** We extracted the performance logs of 5 load tests from a large scale enterprise system. Three load tests were marked to be of similar nature by domain experts, i.e., load tests A, B and C. Each load test generated 5 separate counter logs produced by subcomponents of the system that were geographically separated. Each test was 10 hours long. The test monitoring systems ensures that the starting, sampling and stopping of performance counters follow the test requirements. For each load test we combined the respective performance counter logs based on their sampling frequency. This resulted into 632 performance counters for each load test. We applied our methodology on the performance counter log of load test-B and it recommended 73 important counters. Our methodology showed an improved performance on large set of performance counter data of 88% counter reduction. We treated the important performance counters from load test-B as our base-line counters and compared them across other load tests.

***Findings:*** Our methodology found load tests A, B and C to be similar to each other. Figure 9 shows that the importance of performance counters of load tests A, B and C is similar.

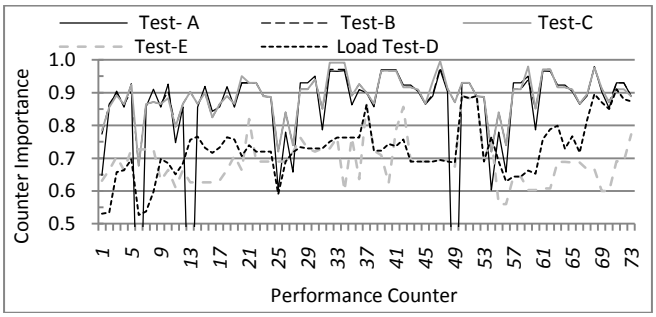


Figure 9: Important performance counters for large logs

Table 9: Correlation between load tests

| | *Test-A* | *Test-B* | *Test-C* | *Test-D* | *Test-E* |
|---|---|---|---|---|---|
| *Test-A* | 1 | 0.9409 | 0.9162 | 0.37418 | 0.1790 |
| *Test-B* | | 1 | 0.9778 | 0.45823 | 0.1359 |
| *Test-C* | | | 1 | 0.42581 | 0.1392 |
| *Test-D* | | | | 1 | 0.2305 |
| *Test-E* | | | | | 1 |

Table 9 provides statistical evidence of this. Tests A, B and C with correlation values greater than 0.91 are highly correlated with each other. However, for the load Test-A, we noticed sudden sharp spikes of decreased importance for a few counters. We investigated this issue by looking at the raw counter data. We found out that there were 5 cases where the performance monitor failed to start sampling the counters. There was also one case where the counters only sampled 15 counter instances and perhaps got stuck. Our methodology successfully removed out that problem counter as its data was 98% less that the required counter data. When comparing these problem counters (removed) in load test-A with base-line counter of load test-B, their value is substituted as 0 by our methodology. This caused those sudden spikes of decreased counter importance. The number of recommended counters can be further decreased by tweaking the tunable loading parameter in the top_k counter selection step of our methodology.

*Our methodology performs well on the large set of performance counter logs and achieves 88% of data reduction.*

## V. THREATS TO VALIDITY

In addition to addressing the challenges described in section 2, we identified the following threats to the validity or our research.

- *Generalization.* As case study subjects, we used various stress tests and load tests performance logs of a large enterprise system. We tried to generate the performance counter logs under various stress intensity to avoid potential bias in our data reduction. We were able to achieve an 88% counter reduction. However, the counter reduction rate is dependent on the correlation among the performance counters, may vary for performance counter logs per domain.
- *Robustness of results.* Our data reduction model is based on principal component analysis. The results produced were up to the satisfaction of the enterprise experts. Using more complex techniques such as to employ Naïve bays classifier and factor analysis may yield further improvement in the suggestions provided by our methodology, but are trivial to implement. Nevertheless, we plan to explore these techniques and take into account supplementary variables along with quantitative counters in future.

## VI. RELATED WORK

We discuss the two areas of related work.

*Data Reduction*

Data management and reduction have been widely studied in many areas, including medical data analysis [21], financial time series prediction [14], and biological data sampling [35]. Data reduction strategies chiefly rely on statistical techniques such as averaging, variance, covariance matrices, sampling, and principal component analysis (PCA). In the area of application performance monitoring and analysis, event throttling [23] prevents the generation of large data volumes. However, it sacrifices a consistent view of application behaviour. Dynamic clustering [19] identifies clusters of processors with similar performance metric curves and then selects one processor from each cluster to represent that cluster, thus reducing the number of processors and performance data. Statistical sampling [17] allows the analysis to focus on subsets of processors or metrics, thus reducing the data to be collected. However its usage is limited to simple cases such as estimating average load and finding free nodes. Another approach that is similar to our data reduction strategy is correlation elimination [15], which identifies a relevant statistically interesting subset of system metrics. More specifically, correlation elimination [15] diminishes the volume of performance data by grouping metrics with high correlation coefficient into clusters and only picking one metric for each cluster as a representative. Our Methodology further improves the elimination process by considering the extent of variance accounted by variable when trying to eliminate variables from principal components.

*Automated Performance monitoring and Analysis of enterprise systems*

A performance data mining framework for large-scale parallel computing tries to manage data complexity by using techniques such as clustering and dimensionality reduction [9]. This data mining framework utilizes random liner projection and PCA to reduce performance data. The framework only reduces the performance data to Principal component but doesn't achieve fine-grain analysis like us by decomposing the PC. The work of Sandeep et al. is closest to ours [25]. They employed principal feature analysis (PFA) to achieve data reduction. The main difference between their approach and ours is that they utilize machine learning to distil the large counter set into smaller sets to describe the workload. Also, their work is partially automated and requires continuous training to produce accurate results. Cohen et al. [7] develop application signatures based on the various system metrics (like CPU, memory). Jiang et al. automates the performance analysis of load test [11]. Unlike our work they relied on execution logs, which require domain knowledge to understand them.

## VII. CONCLUSION AND FUTURE WORK

In this paper we presented our methodology to automate the comparison of performance counters across tests. Our methodology uses Principal Component Analysis, a statistical technique, which achieves an 88% reduction in performance counter data set. In load test of an enterprise system a domain expert can uncover important performance counters and calculate performance gains/loss. However, in such large environments domain experts are usually busy or hard to find. Our methodology is not a substitute for domain experts however it can guide the performance analyst and domain expert when new features are added into application/system resulting into new performance counters produced during load test. A large case study on a real-world industrial software system provides empirical evidence on the ability of our methodology to uncover the discrepancy in performance among load tests.

In future work, we plan to compare our DR methodology i.e., principal component with other DR techniques such as factor analysis and Naive bayes classifiers. We also plan to strengthen our methodology to identify anomalous behavior of an application during load test.

## ACKNOWLEDGMENTS

We are grateful to Research In Motion (RIM) for providing access to the enterprise applications 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 RIM and/or its subsidiaries and affiliates. Moreover, our results do not in any way reflect the quality of RIM's software products.

## REFERENCES

[1] Avritzer. A., Larson. B., "Load testing software using deterministic state testing", In Proceedings of the ACM SIGSOFT international symposium on Software, 1993.

[2] Beizer. B., "Software System Testing and Quality Assurance"' Van Nostrand Reinhold, March 1984.

[3] Box. M. J., Box. R. M., "Computation of the Variance Ratio Distribution", The Computer Journal, pp. 277-278, 1969.

[4] Brace, N., Kemp, R., Snelgar, R., "SPSS for Psychologists: Palgrave Macmillan", 2003.

[5] Bruce Thompson., "Exploratory and Confirmatory Factor Analysis: Understanding Concepts and Applications", ISBN: 1-59147-093-5.

[6] Chatterjee, C., Roychowdhury, V.P., Ramos, J., and Zoltowski, M.D., "Self-organizing algorithms for generalized eigen-decomposition," Neural Networks IEEE Transactions, vol.8, no.6, pp.1518-1530, Nov 1997.

[7] Cohen, I., Zhang, S.,. Goldszmidt, M., Symons, J., Kelly, T., "Capturing, indexing, clustering, and retrieving system history", In Proceedings of the twentieth ACM symposium on Operating systems principles, 2005.

[8] Hair, J.F., Jr.l Anderson, R. E.; Tatham, R. L.; and Black, W. C. (1998). “Multivariate data analysis with readings”, 5th ed.. Englewood Cliffs, NJ: Prentice-Hall.

[9] Huck, K.A.; Malony, A.D., "PerfExplorer: A Performance Data Mining Framework For Large-Scale Parallel Computing," *Supercomputing, 2005. Proceedings of the* ACM/IEEE SC 2005 Conference, vol., no., pp. 41-41, 2005.

[10] Jiang, Z. M., Hassan, A. E., “Automatic identification of load testing problems”. In Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM), 2008.

[11] Jiang, Z. M., Hassan, A. E., Hamann, G., Flora, P., Automated Performance Analysis of Load Tests. In Proceedings of the 25th IEEE International Conference on Software Maintenance (ICSM) 2009, Edmonton, Canada, September 20-26, 2009.

[12] Jolliffe IT., “Principal Component Analysis”, Second Edition. New York, Springer-Verlag; (Springer Series in Statistics), 2002.

[13] Kaiser, H. F., "An Index of Factorial Simplicity," Psychometrika, vol. 39, pp. 31-36, 1974.

[14] Lendasse, A., Lee, J., Bodt, E.d., Wertz, V. and Verleyen, M., “Input Data Reduction for the Prediction of Financial Time series”, 2000.

[15] M.W. Knop, J.M. Schopf and P.A. Dinda, “Windows performance monitoring and data reduction using watch tower”, Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN), 2002.

[16] May, J. M., “Software for multiplexing hardware performance counters in multithreaded programs”. Proceedings of 2001 International Parallel and Distributed Processing Symposium, April 2001.

[17] Mendes, C.L. and Reed, D.A., “Monitoring Larger Systems Via Statistical Sampling”, Proceedings of the LACSI Symposium, Santa Fe, 2002.

[18] Montanelli, R.G. Jr., and Humphreys, L. G., “Latent roots of random data correlation matrices with squared multiple correlations on the diagonal: A monte carlo study”, Psychometrika , pp. 341–348, 1976.

[19] Nickolayev, O.Y., Roth, P.C. and Reed, D.A., “Real-Time Statistical Clustering For Event Trace Reduction”, The International Journal of Supercomputer Applications and High Performance Computing, pp 144-159, 1997.

[20] Parets, J., and Torres, J.C., "Software Maintenance versus Software Evolution: An Approach to Software Systems Evolution", Workshop on Engineering of Computer Based Systems (ECBS'96), pp. 134, 1996.

[21] Qu, Y., Adam, B.l., Thornquist, M., Potter, J.D., Thompson, M.L., Yasui, Y., Davis, J., Schellhammer, P., Cazares, L., Clements, M., Jr., G.L.W. and Feng, Z., “Data Reduction Using a Discrete Wavelet Transform in Discriminant Analysis of Very High Dimensionality Data”, Biometrics, 2003.

[22] Raubenheimer, J. E., “An item selection procedure to maximize scale reliability and validity”. South African Journal of Industrial Psychology, 30 (4), 59-64, 2004.

[23] Reed, D.A., Aydt, R.A., Noe, R.J., Roth, P.C., Shields, K.A., Schwartz, B.W. and Travera, L.F., “Scalable Performance Analysis: The pablo Performance Analysis Environment”. Proceedings of the Scalable Parallel Libraries Conference, 1993.

[24] Ringberg, H., Soule, A., Rexford, J., Diot, C., “Sensitivity of PCA for traffic anomaly detection”, In ACM SIGMETRICS,San Diego, CA, USA, 2007.

[25] Sandeep, S. Ratna., Swapna, M, Thirumale Niranjan., Sai Susarla., Siddhartha Nandi., “CLUEBOX: A Performance Log Analyzer for Automated Troubleshooting”’ WASL, 2008.

[26] Schwartz. Jeffry. A., “Utilizing performance monitor counters to effectively guide windows and SQL server tuning efforts”’, pp. 933-944, 2006.

[27] Sébastien Lê., Julie J., François Husson., “FactoMineR: An R Package for Multivariate Analysis”, Journal of Statistical Software, pp.1-18, 2008.

[28] Sprunt, B., "The basics of performance-monitoring hardware," Micro, IEEE , vol.22, no.4, pp. 64-71, Jul 2002.

[29] Thakkar, D., Hassan, A.E Hamann, G., Flora, P., “A framework for measurement based performance modeling”. In WOSP ’08: Proceedings of the 7th international workshop on Software and performance, pages 55–66, New York, NY, USA, 2008.

[30] The R Foundation for Statistical Computing., “R Project for Statistical Computing”’ http://www.r-project.org, 2007.

[31] Yoccoz, N.G., J. D. Nichols and Boulinier, T., “Monitoring of Biological Diversity in space and time, Trends in Ecology and Evolution”, 2001.

[32] Rencher, A. C. Methods of Multivariate Analysis. Wiley, New York, 1995.

[33] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is “nearest neighbor” meaningful? In Lecture Notes in Computer Science, volume 1540, pages 217–235, 1999.

[34] A. Hinneburg, C. Aggarwal, and D. Keim. What is the nearest neighbor in high dimensional spaces? In The VLDB Journal, pages 506–515, 2000.

[35] B. Milenova and M. Campos. O-cluster: Scalable clustering of large high dimensional data sets. Oracle Corporation, 2002.

[36] Limitations of Maximum Likelihood Estimation Procedures When a Majority of the Observations Are Below the Limit of Detection Ram B. Jain, Richard Y. Wang Analytical Chemistry 2008 80 (12), 4767-4772

[37] Ahn. D., and J. Vetter., ‘Scalable analysis techniques for microprocessor performance counter metrics’. In Proceedings of Supercomputing, 2002.