A Methodology to Support Load Test Analysis

Haroon Malik
School of Computing
Queen's University
Kingston, ON, Canada
1-(613)-533-6802
http://sailhome.cs.queensu.ca/~haroon
malik@cs.queensu.ca

ABSTRACT
Performance analysts rely heavily on load testing to measure the performance of their applications under a given load. During the load test, analyst strictly monitor and record thousands of performance counters to measure the run time system properties such as CPU utilization, Disk I/O, memory consumption, network traffic etc. The most frustrating problem faced by analysts is the time spent and complexity involved in analysing these huge counter logs and finding relevant information distributed across thousands of counters. We present our methodology to help analysts by automatically identifying important performance counters for load test and comparing them across tests to find performance gain/loss. Further, our methodology help analysts to understand the root cause of a load test failure by finding previously solved problems in test repositories. 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 thereby archiving 88% counter data reduction.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: Measurement Techniques

General Terms
Performance, Automation, Counters.

Keywords
Load Test, Principal Component Analysis, Performance Counters.

1. INTRODUCTION
Measuring effective performance in a distributed and ultra large scale system (ULSS) i.e., Google, Face book, Amazon or eBay is a challenging research problem due to the exponential growth in scale and complexity of system's resources and applications. Where classic performance measuring techniques cannot cope with these distributed ULSSs, load testing still remains the most integral part of testing and measuring the system's performance [1].

ICSE'10, May 2–8, 2010, Cape Town, South Africa.

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 simulate 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. The performance counter log captures run time system properties such as CPU utilization, disk I/O, queues, network traffic etc. Such information is of vital interest to performance analysts as it helps them to observe the system's behavior under load by comparing against documented behavior of an application/system or with an expected behavior.

In practice, for ULSS, it is impossible for an analyst to skim through the huge volume of performance counters to find the required information. Instead, analyst employ few key performance counters known to him from past practice, performance gurus and domain trends as 'rules of thumb' [3]. In a ULSS, there is no single person with complete knowledge of end to end geographically distributed system activities [2]. An analyst with good knowledge of the mail server will quickly uncover important mail counters from a performance counter log however; he may overlook some of the important database counters. Applying same 'rules of thumb' on load tests can mislead the performance issues [2]. Existing research on load testing focuses on the automatic generation of load test suit [5][6]. Even though load testing has been extensively studied, little is known on how to effectively analyze load test in ULSS with thousand of performance counters generating terabyte of log data. The goal of our research work is:

1. Support analysts to effectively manage load test analysis in limited time.
2. Help analysts to understand the root cause of a load test failure by finding previously solved problems in test repository, thereby reducing the need of expertise-intensive activities.

The paper first present a motivating scenario (Section2), followed by research problems (Section 3). Paper then present our research hypothesis and sketch our methodology (Section 4). Next, case-study is presented (Section 5) followed by related work and conclusions and contributions in (Section 6 and Section7).

2. Motivating Scenario
It was not a surprise for John to see pile of testing documents being shoved-up on the in-tray of his desk; what a start of the week. While John hastily gazes at the documents to understand

the new testing job, his boss appears on the door and says: "we want to go live with the BigMail project this Friday. We have a new automatic load test analysis system to help you. I am sure you can do it John." John has been load testing the features of the BigMail in past. But this time there is a twist. The lab (mail servers supporting 30,000 users) is physically at another location due to all testing resources being fully occupied at such busy time. John has been assigned the remote lab from 5 pm to 5 am. John gets busy like a bee setting up the test environment (installing some domain specific application, configuring load testing tools (load generators, performance monitors, emulators etc.,). John is done setting up the environment by 4:30. He again verifies all the steps with his cheek-list to ensure everything is setup correctly. John knows that a slight negligence can cost him a whole day's effort. He starts the first test at 5:00pm and then leaves to home. Next day, John finds the load test analysis report in his mailbox generated by an automated load test support tool. John is surprised to see a neatly organized report with the list of important load test counters, a visualization that compares the important counters with the base load test and a correlation table that provides statistical analysis between the two tests. The report also includes few 'test id' and their links to test repositories. John is frustrated to see intolerable response time of BigMail. The visualization clearly shows high deviation of the mail server 'A's counter 'disk Read\sec' from base line. John clicks on a link in the report to find similar load tests. After going through few of the suggested similar tests, john finds a test with analogous problem reported by the tester at the remote lab location. The tester indicated that heavy disk utilization was noticed for the server 'A' resulting from scheduled maintenance activity that runs every sunday night. No doubt, John and the remote lab location have 24 hours difference. John is happy; the automated system helped him to identify the problem. He did not have to spend time skimming through large volume of performance counter data, piecing together the cause of test failure.

3. PROBLEM STATEMENT

We formulate our problem statement by highlighting the current challenges and threats faced by performance analysts during load test analysis of ULSS.

3.1 Redundant Counter Traffic

Load test can last from several hours to days. It generates performance logs that can be of several terabytes in size [8]. There is a lot of redundant information in the huge amount of collected log data. This noise reduces the accuracy of performance evaluation. Analyzing large volume of redundant counter data makes performance analysis laborious and results vague.

3.2 Time Limitations

Load testing mostly comes 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, a manager is always eager to reduce the time allocated for load testing. This means that analysts have to analyze the load test performance data quickly and in limited time [9].

3.3 Interfering Work load

The most common threat; when a performance analyst or a tester is unaware of the various background activities such as disk scrubs, antivirus, applications updates, scheduled maintenance and replication activities. Interfering workload threat is hard to avoid in many cases [3][4].

3.4 Under Provisioning

Analyst need to interact with systems and applications of greater sophistication, size and integration quickly and in limited time to perform load test. Lot of tools exists for automatic generation of load tests. However, there exists no tool for configuring the application under test; most of it being manual and error prone [11]. It is not unusual for analysts to under provision or mis-configures application/system under test. For example, the number of concurrent connection allowed for database may be incorrect. This will prevent several users to login, thus reducing the actual load scheduled for the load test [1]

All the above threats manifest themselves in two ways; 1) slowing down of application/system and 2) unexplained behavior of application/system under load. Analysts have to piece together a variety of hidden symptoms to come up with diagnostics.

4. Research Hypothesis

Performance analysts face lots of challenges in the analysis of performance counter logs. Automated techniques/tools are needed to help them to analyze performance logs in limited time and to understand the cause of load test failure.

The biggest challenge for performance analysts is to identify the few important performance counters in a highly redundant performance counter data and to find the root cause of a load test failure. Research on automated load test analysis—starting with tools to analyze and interpret performance data—has not kept pace with the demands for practical solutions in the field. The dissertation focuses on helping analysts to effectively analyze load test and understand the cause of performance problem by the use of historical load test data stored in the test repositories. We present our work into three parts. First, we show that our methodology helps analyst in analyzing the large volume of performance counters data. Second, we show that our methodology finds related historical load tests in the test repository to help analysts zero-on the cause of the performance problem. Lastly, we validated our first part based on the study of performance counter logs of a commercial enterprise system.

4.1 Supporting Load Test Analysis

The first part of our methodology consists of two steps: data reduction and identifying top performance counter variables.

4.1.1 Data reduction

We used the robust and scalable statistical technique, Principal Component Analysis (PCA) to reduce the sheer volume of performance counters [12]. What PCA does is to synthesize new variables called 'Principal Components' (PC). Every PC is independent and uncorrelated with other PCs. PCA is maximum variation projection method, which means that PCA identifies those variables that have the large data spread (variance), ignoring variables with low variance. To overcome this, we standardize the performance counters so that variance of each performance counter variable =1.0. Table1 shows the PCA for a real world performance counter log consisting of 18 counter variables. PCA groups the data of the 18 counter variables into principal components, each of which explains particular amount of variance of the original data. This means 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 variable, infact 11.431 times as much and account for 63.60% of the variability of entire performance 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 variable. Deciding on the number of PCs that capture sufficient variability of data for analysis is challenging problem. Unfortunately, methods known today do not provide any automated and reliable technique to identify top_k principal components. To further trim the performance counter data we use '% Cumulative Variability' in selecting the number of top_k components. Using '% Cumulative Variability' of 90% is adequate to explain most of the data with minimal loss in information [12]. Thus we can extract first four PCs from table 1.

Table1. Principal Component Analysis (PCA)

PC	*Eigen-Value*	*Variability (%)*	*Cumulative Variability %)*
PC1	11.43	63.506	63.506
PC2	2.47	15.260	78.765
PC3	1.720	9.554	88.319
PC4	0.926	5.143	93.463
⋮	⋮	⋮	⋮
PC12	0.001	0.003	100.00

4.1.2 Identifying Top_k Performance Counter Variables

Performance analysts are interested in performance counters not principal components. Table 1 show that we have four principal components that can explain over *90%* of the original counter data variability. We now decompose principal components using the eigenvectors technique to map the PCs back to the performance counter variables [13] . Each performance counter is given a weight in accordance to its association with a component. Larger the weight of a performance counter, the more it contributes to a PC. We applied the threshold to decide on the important performance counter variables and discard the rest.

Table 2. Top_k performance counters

Rank	PC	Var	Weight	% Imp
1	PC1	N	0.974	5.636
2	PC1	M	0.972	5.613
3	PC1	R	0.966	5.544
4	PC1	Q	0.946	5.317
5	PC1	P	0.944	5.294
6	PC1	E	0.933	5.172
7	PC2	I	0.912	4.942

Table 2 shows seven performance counter variables among the performance log consisting of 18 performance counters ranked according to their importance. Our methodology achieved 61% data reduction.

4.2 Understanding the Root-Cause

We are currently working on the second part of our methodology. This section sketches our approach which is still an on-going work. When a load test fails, performance analyst want to know the reason for its failure. If the failure occurred due to inevitable threats such as interfering workload, the test must be repeated. If the test failed for other reasons such as memory-leak, overloaded queues or high CPU utilization, the test results are sent to developers and other experts to rectify the problem. Troubleshooting is an expert-intensive and time sensitive activity. The longer it takes to identify and solve a problem higher the cost.

Our methodology will use historical information stored in test repositories to guide performance analysts in diagnosing and repairing failed load tests. Our intuition is that history can help in understanding the anomalous behavior of a load test by finding the load test results of previous, similar load tests. These may help analysts to even zero-on the root cause by finding previously solved load test cases.

Our methodology will train itself on the load test repository and will make use of performance counter logs and test reports to build a signature for load tests. The signature will act as a finger print to identify a load test. With the help of signature, an analyst can retrieve similar or near to similar load tests from the test repository. Locating a previous test report, of which the root cause has been found, may provide hints of a repair action. We now provide the mechanism to craft load test signature.

Load test is characterized by multiple factors: Workload, environment (hardware resources, software version etc., status (pass/fail), high-level metrics (response time, throughput, latency etc.,).Workload has two dimensions: a characteristics and intensity. The *characteristics* of a workload refer to attributes like read/write ratio and arrival distribution. The *intensity* of a workload refers to attributes such as the arrival rate of the operations and throughput of data read. The test report contains all load test factors. Our methodology will take into account all of the available artifacts in load test reports, including the important performance counter variables identified by PCA to construct a load test signature. All the artifacts are considered the keys of the load test signature. A load test signature is a unique representation of similar load tests. We will use either random forests or principal Component regression (PCR), on all the artifacts of the load test to obtain the unique key values for the load test signature. We plan to use these two techniques because random forests create explainable models, which are useful to understand the behavior of an application/system. On the other hand Principal component regression works well with the PCA technique used by us in 4.1.

5. CASE STUDY

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 test A, B and C. Each performance counter log contained 632 performance counter variables. We applied our methodology on the performance counter log of load test-B and it recommended 73 important counter variables. We treated the important performance counters from the load test-B as our base-line counters and compared them across all other load tests.

Table 3. 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

Findings—our methodology found that load tests A, B and C had similar counter importance for all the 73 important performance counters as shown in figure 1. The load tests D and E were found completely different from all other load tests.

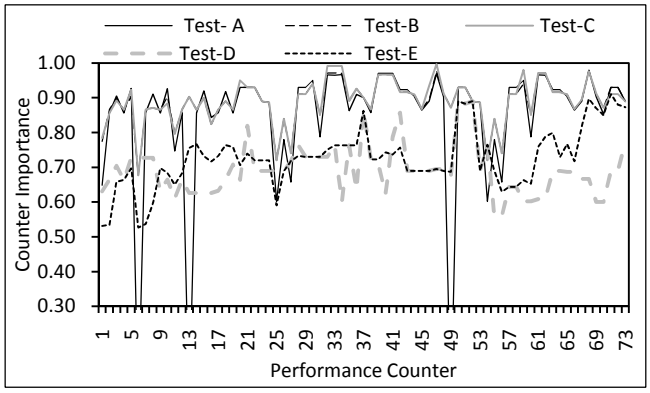

Figure 1: Important performance counters for large logs

Table 3 provides statistical evidence of it. However, for the load Test-A, we noticed sudden sharp spikes of decreased importance for some of the performance counters. We investigated this issue by looking at each raw counter data. We found out that there were 5 cases where the performance monitor failed to start sampling the counter variable. There was also one case where the performance monitor only sampled 15 counter instances of a counter variable and perhaps due to a non-responsive thread. Our methodology successfully removed that problem counter. While comparing these problem counters (removed) in load test-A with base-line counter of load test-B, their value is substituted as 0, causing those sudden spikes of decreased counter importance.

6. Related Work

Most of the work done in search and retrieval to find similar instances based on signature is in the domain of antivirus. Sandeep et al. work is closest to ours [4]. They employed principal feature analysis (PFA) to achieve data reduction and then used clustering algorithm to find the optimal subset of performance counters. Their work is partially automated and requires continuous training to produce accurate results. We have reduced and ranked the performance counter data by using only one technique, PCA. Cohen et al. developed application signatures based on the various system metrics (like CPU, memory) and utilizing complex clustering mechanism, however clustering mechanism do not scale [2]. Several authors have pointed out that clustering method is not fully effective when clustering high dimensional data [15]. Where as we expect performance counter logs consisting of thousands of performance counters.

7. CONCLUSIONS AND CONTRIBUTIONS

So far, we have conducted a case study on the load testing logs of a commercial system. We want our methodology to be representative of an open-source system as well. In particular, we are currently performing case studies on 1) The Dell DVD store enterprise application developed by Dell [11] and 2) Rice University Bidding System[1] (RUBis), which is a three-tier web-application that is believed to be similar in architecture and design to real-commerce applications. We have presented our methodology to support load test analysis and help analyst understand the root cause of load test failure. We list our major contributions as:

(a) Reducing the dimensionality of the observed performance counter set by the use of a robust statistical method PCA [12].
(b) Automate the ranking of performance counter variables according to their importance for a load test. This reduces the time required by an expert to zero-on the root-cause.
(c) Automate the process of comparing load test and generating test report consisting of simple visualization and straightforward correlation tables.
(d) Showing how an effective workload signature profile can be crafted in a nontrivial way to efficiently retrieve similar or near to similar load tests from the test repository.

[1] http://rubis.ow2.org/

8. REFERENCES

[1] 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.

[2] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In Proc. 6th USENIX OSDI, San Francisco, CA, Dec. 2004.

[3] Cohen, I., Zhang, S., Goldszmidt, M., Symons, J., Kelly, T., and Fox, A. 2005. Capturing, indexing, clustering, and retrieving system history. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles* (Brighton, United Kingdom, October 23 - 26, 2005). SOSP '05. ACM, New York, NY, 105-118. DOI= http://doi.acm.org/10.1145/1095810.1095821

[4] Sandeep, S. Ratna., Swapna, M, Thirumale Niranjan., Sai Susarla., Siddhartha Nandi., "CLUEBOX: A Performance Log Analyzer for Automated Troubleshooting"' WASL, 2008.

[5] V. Garousi, L. C. Briand. Y. Labiche., "Traffic-aware stress testing of distributed systems based on uml models", In Proceedings of the 28th international conference on Software engineering, 2006.

[6] Bayan, M. S., Cangussu, d J. W., "Automatic stress and load testing for embedded systems". In 30th Annual International Computer Software and Applications Conference, 2006.

[7] 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.

[8] 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.

[9] Schwartz. Jeffry. A., "Utilizing performance monitor counters to effectively guide windows and SQL server tuning efforts"', pp. 933-944, 2006.

[10] Research Centre Juelich GmbH., "The performance counter library: A common interface to access hardware performance counters on microprocessors". http://www.fzjuelich.de/zam/PCL/.

[11] 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.

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

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

[14] J.O.Kephart and D.M.Chess. "The vision of autonomic computing. In computer", 2003.

[15] 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.