Managing Performance Testing with Release Certification and Data Correlation

Tuli Nivas
Performance Lab
Sabre Holdings Inc.
Southlake, TX 76092, USA
Tuli.Nivas@sabre.com

Christoph Csallner
Computer Science and Engineering Department
University of Texas at Arlington
Arlington, TX 76019, USA
csallner@uta.edu

ABSTRACT

Performance testing is a key element of industrial software development. While basic performance testing concepts are well understood, it is less clear how to manage performance tests in practice. I.e., we have encountered the following two problems. (1) While testing textbooks prescribe writing tests against performance goals, we find that it is impractical to gather from business analysts performance goals that are detailed enough for finding subtle performance bugs. (2) Once performance tests are conducted, we were asked questions such as the following, which we found hard to answer. How can you be confident that the executed tests assess the performance of the software truthfully?

To enable practitioners to address these problems, this paper introduces two additional performance testing process components, which we call release certification and test data correlation. Our key idea to address problem (1) is to run two different versions of the same subject application side-by-side in the same test environment. This allows us to use the performance profile of the previous version as the detailed performance specification of the version under test. Our key idea to address the questions of (2) is to correlate the performance measurements of the test and production environments. We also report on our experience of applying this implementation on several releases of a commercial airline sales application.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Life cycle, software quality assurance*; D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Management, Measurement, Performance, Reliability, Verification

Keywords

Performance testing, release certification, data correlation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FSE'11, September 5-9, 2011, Szeged, Hungary.
Copyright 2011 ACM X-X-XXXX-XXX-X/XX/XX…$10.00.

1. INTRODUCTION

For a commercial computing environment that processes transactions 24*7, 365 days a year, software and hardware stability and reliability are crucial. Any disruption of service could have catastrophic effects on the company in terms of revenue earned, brand credibility and customer loyalty. Bottlenecks or other issues that could degrade the performance of a deployed application should thus ideally be found and resolved in the test environment.

While an application is being used on production servers, developers often already work on subsequent versions. Before deploying such a new version to production servers, the new version has to be tested, to minimize the risk that it will cause service disruption. The conventional wisdom on performance testing is to specify performance goals such as response times and to test the application against these goals. Following is a typically textbook description of this process: *"The objective of performance testing is to validate the software 'speed' against the business need for 'speed' as documented in the software requirements. Software 'speed' is generally defined as some combination of response time and workload during peak load times." [6, page 129]*

However, in practice, we have found it hard to impossible to gather a performance specification that is detailed enough for finding subtle performance bugs. Performance goals are typically maintained in service level agreements (SLAs), which are created by business units. SLAs contain some valuable high-level information, such as end-to-end response times. But missing in SLAs are fine-grained performance goals that are expressed in terms of low-level technical performance metrics, because business units that formulate SLAs are not familiar with such technical metrics. Examples of such fine grained metrics include CPU time, garbage collection time, virtual memory cache misses and hits, virtual memory usage, and system exceptions.

The following example from our experience of testing an airlines sales application, shown in Figure 1, highlights the importance of having such fine-grained performance goals. The CPU usage behavior for the two releases being compared is similar, but there is a significant increase in the virtual and residual memory of the new release. This behavior can cause the system to crash, which means we would like to catch and fix this problem during performance testing.

Although fine-grained technical performance goals are important, we have found that results obtained from measuring performance metrics in the test environment cannot be used directly to make predictions about the eventual performance in the production environment.

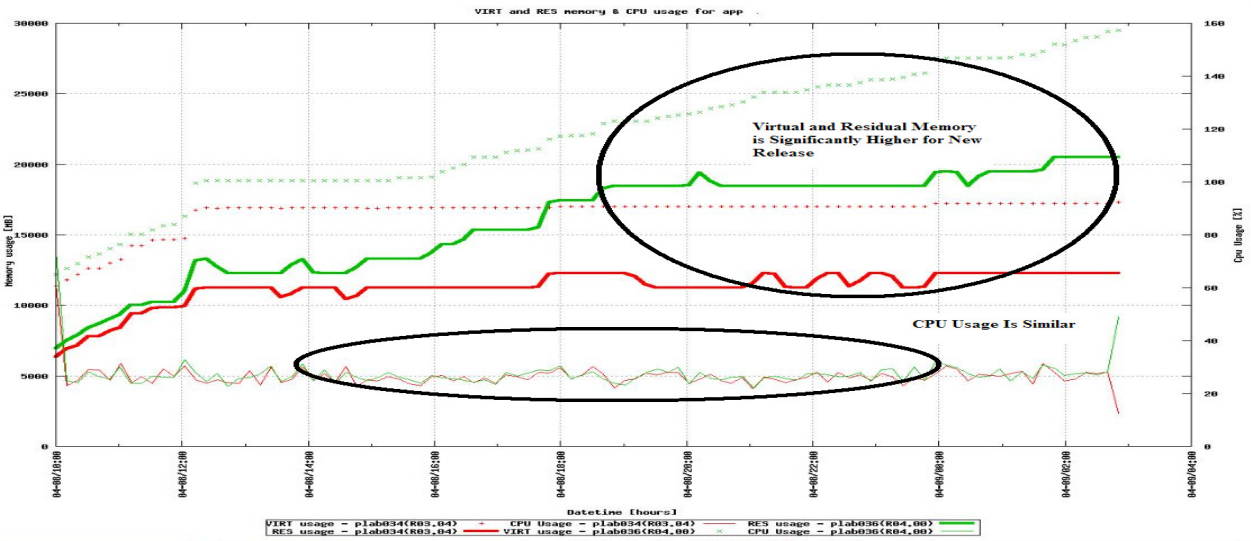

Figure 1 Performance Bug in Airline Sales Application Found with Fine Grained Performance Goals

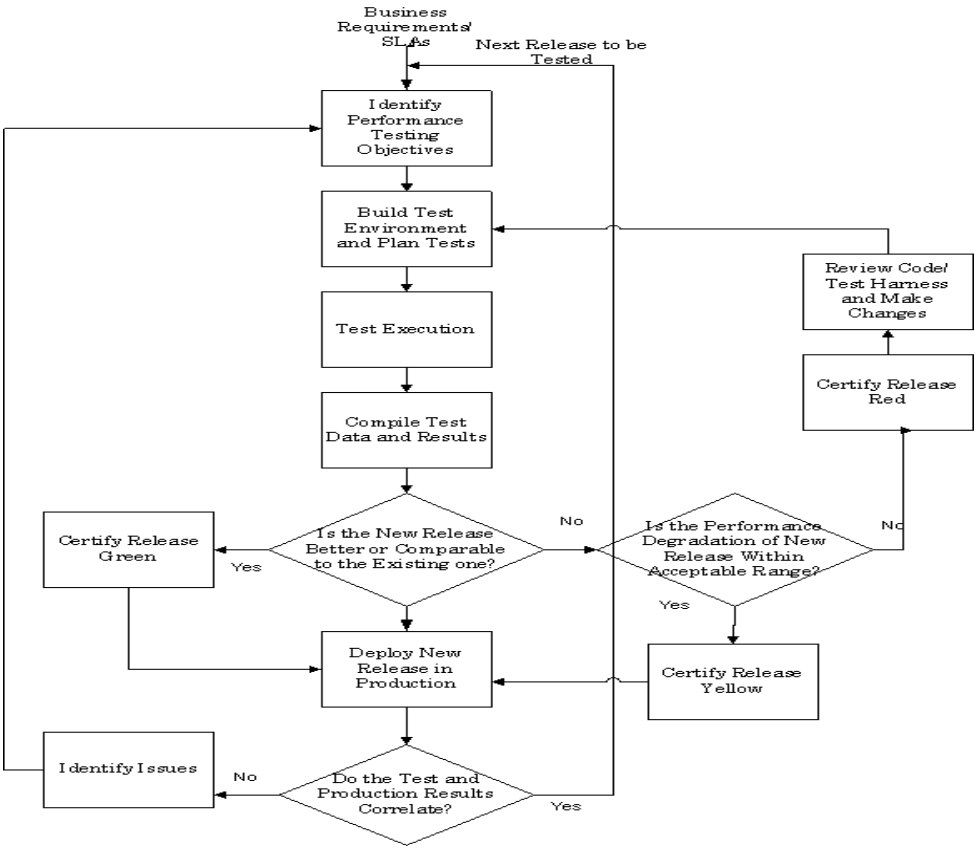

Figure 2: Performance Testing Workflow Extended with Release Certification and Data Correlation Components

The reason is that the behaviors of the test and production environments typically diverge at some point. To address this problem, we are experimenting with a technique for test data correlation. This has allowed us to pinpoint problems in the test environment and in the test cases used to emulate production usage of the software under test.

2. SOLUTION OVERVIEW

We propose to take a traditional performance testing workflow and extend it, by adding two components, release certification and test data correlation. Figure 2 shows an example resulting workflow, which we have been using over the last year for performance testing of several large-scale applications, including an airline sales application. The workflow is complete and can be

repeated for new releases if the last branch of the flow chart ends in a "Yes". If it ends in a "No", the offending issues need to be identified and resolved before the next new release can be certified and deployed in production. This will ensure that the test environment and workflow are as close to production as possible.
The process is general enough to be utilized for any end to end enterprise system. It is not dependent on any particular hardware or software and can be followed to get reliable performance test numbers before an application is deployed in production.

3. RELEASE CERTIFICATION

Release certification is the process of gathering all application and system metrics needed to review the performance of that specific application and setting targets for them. All metrics for two releases of the same application are considered and then the performance targets are classified as red, green or yellow. We picked the colors in analogy to traffic lights. Green is a go, with yellow you move cautiously and red means stop. For us these categories encode the range of performance targets from unacceptable to acceptable. The numbers for each category are determined together by the developer and performance testers, taking into account the SLAs and the performance measurements of previous releases.
We follow this process even if we add a new feature or traffic profile. In production, the new release will have to process both the existing and new traffic profiles. So when the workload analysis is done, we need to establish the percentage of the new traffic as part of the total and then use that as the input to the new release.
Establishing a baseline is a related approach [7]. As in release certification, a broad set of key performance indicators can be used, such as response time, processor capacity, memory usage, disk capacity, and network bandwidth. The key difference is that in release certification we test the existing and new releases at the same time, each time a performance test is being run. We found that establishing a baseline and later using it as a reference for a new release is often not practical, as requirements change constantly. For example, in a web-facing application, the amount and mix of traffic an application is expected to handle can change significantly in a short time frame. Such changes can quickly, within a few months or even weeks, render a once valuable baseline obsolete. On the other hand, we maintain baseline results and compare current results with these baselines. This remains useful when traffic profiles remain relatively similar.
To implement release certification, we need a test environment in which both the existing and the new application release can be tested side by side on the same traffic.

3.1 Example: Certification for Airline A

To illustrate the certification process, consider our example of an online airline shopping website. Users on the website are able to search for flights between a source and destination city, get schedules and availability data. They are also able to specify how many solutions they want returned for their particular request.
Table 1 is part of the certification criteria table for this sample shopping application. The first column specifies the metric being measured and the next three columns are the criteria for the new release, some expressed as absolute values and some as a percentage of the measurement of the previous release. In order to properly compare memory utilization and CPU usage, performance tests are run for at least 48 hours and therefore these metrics are compared in terms of usage per day.

Setting up a test environment with two identical data flow paths for the two releases enables us to test the existing and new releases side by side. For example, if we have a SLA for elapsed time (the time taken by a piece of code to execute a particular transaction) set as maximum 4 seconds, anything less than 4 seconds is good. Historical data indicated that even if the piece of code is taking about 5 seconds to process transactions it does not affect the application, so that is a warning but still an acceptable number. However, anything over 5 seconds is unacceptable and the application will not be deployed in production to prevent performance-related problems.

Metric	Green	Yellow	Red
Client resp. time [s]	< 10	10–15	>15
Add. CPU time	< 5%	5–10%	>10%
Add. trans. time	< 5%	5–10%	>10%
Trans. time [ms]	< 4	4-5	>5
CPU utilization	< 60%	60–70%	>70%
RAM utilization	< 20%	20–30%	>30%

Table 1: Release certification criteria for a flight shopping application for airline A.

4. TEST DATA CORRELATION

The decision to deploy a particular release in production depends on the results from performance tests, which therefore should be reliable, giving stakeholders the confidence to trust the numbers. Even though performance engineers and application teams try to create a test environment that mirrors production, it usually does have certain differences. I.e., the test environment is typically a scaled down version of the production environment. For example, the storage area network used for databases in test might be slower than the one being used in production or the network configurations might be different. For this reason it is common practice to set a 5% plus or minus acceptable range between the test and production numbers. In this step, we compare results from the test environment with actual performance numbers seen in production. This enables improving the test harness, test cases, data collection procedures, scripts, and workload analysis. Correlation needs to be done for each metric that is collected during tests.
The process of test data correlation is to take the 95% percentile value for each measured performance metric during a successful test and then compare that with the 95% percentile value of the same metric in production after the new release has been deployed. During testing, scenarios of varying load along with peak traffic are simulated during a 48 hour interval, so resource utilization numbers measured during tests might be higher than those observed in production. This along with incorporating the differences associated with the test environment as compared to production is the reason why a plus minus 5% range between the numbers is considered acceptable. The duration over which the numbers are collected in production also needs to be closely monitored. Performance tests are run for short durations as compared to the life of an application when deployed in production. The traffic profile used during testing which might contain new types of transactions might become live only after a few days of application deployment. For this reason the metrics in production should be collected after a few days of the application going live. This will give us a basis to compare the performance of the same release under the same traffic profile in the two environments.

4.1 Example: Correlation for Airline A

Table 2 shows the correlation of the time taken by a piece of software to execute a particular transaction for the sample flight shopping application between test and production. There is a large difference between the values recorded during test (pLab) and production (Prod). This indicates that changes have to be made either to the environment setup, data collection technique, test scenarios, or even the workload being used. Since the test results are not comparable to production numbers, the tests being run are not reliable. After changes are made to better the system from release 2009.06.00 onwards correlation data looks more favorable, but improvements are still needed. This exercise ensures that proper tests are being run using the right traffic profiles. As can be seen from table 2, the correlation exercise also results in a red or a green release signifying how close the test results are to production.

Release	pLab Obs.	Prod Avg.	% diff from prod
2008.10.06	1.32	0.61	116.4
2009.00.00	0.66	0.64	3.1
2009.01.01	0.81	0.45	80.0
2009.02.02	0.51	0.39	30.8
2009.03.01	0.93	0.47	97.9
2009.04.02	0.85	0.42	102.4
2009.05.00	0.53	0.41	29.3
2009.06.00	0.5	0.54	-7.4
2009.07.02	0.59	0.5	18.0
2009.08.00	0.56	0.55	1.8
2009.09.01	0.6	0.58	3.4
2009.10.01	0.66	0.57	15.8
2009.11.02	0.55	0.58	-5.2
2009.12.01	0.8	0.56	42.9

Table 2: Correlating Production and Test for Airline A

5. RELATED WORK

A lot of work has been done in the field of performance testing and measurement [8], [2], [3]. But the majority of the research is devoted to web testing and load drivers that can drive traffic to web applications. [4] discusses which metrics should be collected during testing to measure performance, which are limited to client side response times and errors. It also explains how Little's Law can be used to determine if test results are close to numbers calculated by that law. How well a particular piece of code will perform is dependent on more than just how much throughput it can sustain or how fast it processes transactions. [7] describes testing processes and automation taking web applications as examples. [9] describes the importance of workload analysis and gathering of test requirements and explains how test cases and traffic profiles can be created. [8] explores performance testing in distributed computing environments and how performance test cases can be derived from system architecture design. In addition to the above, we should also confirm that test cases, data collection, and workload used during testing are emulating real world scenarios. Our certification and correlation phases, added to the general performance testing guideline, will help in determining all factors – system and application that affect the performance of a particular piece of code. These two steps will not only be able to provide the necessary performance evaluation process but also help in validating that the right set of activities are being followed during testing.

6. CONCLUSIONS

In performance testing, following the regular conventional steps of identifying test objectives, setting up a test environment, analyzing the workload and running tests for a particular release of the application is often not sufficient in the practical world. Revenue, credibility, and customer loyalty are all at stake for a company when applications are deployed in production. So in order to guarantee that a new release going live will not hamper the processing of incoming traffic, we have found it useful o add to the traditional performance testing activities two new steps, certification and correlation. Both certification and correlation ensure that the releases going into production are thoroughly and properly tested and compared to the existing release in terms of performance.

7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1017305. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] A. Avritzer and E.J. Weyuker, 1995. The automatic generation of load test suites and the assessment of resulting software, IEEE Transactions on Software Engineering (TSE), (21)9: 705-716

[2] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, 2005. Capturing, indexing, clustering, and retrieving system history, In Proc. 20th ACM Symposium on Operating Systems Principles (SOSP), p105-118

[3] Z.M. Jiang, A.E. Hassan, G, Hamann, and P. Flora, 2008. An automated approach for abstracting execution logs to execution events, Journal of Software Maintenance and Evolution: Research and Practice, (20)4:249-267

[4] R. Mansharamani, A. Khanapurkar, B. Mathew, and R. Subramanyan, 2010. Performance testing: Far from steady state, In Proc. 34th Annual IEEE Computer Software and Applications Conference Workshops (COMPSACW), p341-346

[5] G.D. Everett and R. McLeod Jr., 2007. Software Testing: Testing Across the Entire Software Development Life Cycle, Wiley-IEEE Computer Society Press

[6] J.D. Meier, C. Farre, P. Bansode, S. Barber, and D. Rea, 2007. Performance Testing Guidance for Web Applications, Microsoft Press

[7] T. Riley and A. Goucher, 2009. Beautiful Testing, O'Reilly Publications

[8] G. Denaro, A. Polini, and W. Emmerich, 2004. Early performance testing of distributed software applications, In Proc. 4th ACM International Workshop on Software and Performance (WOSP), p94-103

[9] E.J. Weyuker and F.I. Vokolos, 2000. Experience with performance testing of software systems: Issues, an approach, and case study, In IEEE Transactions on Software Engineering (TSE), (26)12:1147-1156