Load Testing Large-Scale Software Systems

Zhen Ming (Jack) Jiang
Software Construction, AnaLysis and Evaluation (SCALE) Lab
Department of Electrical Engineering and Computer Science
York University Toronto, ON, Canada
Email: zmjiang@cse.yorku.ca

***Abstract*—Large-scale software systems (e.g., Amazon and Dropbox) must be load tested to ensure that they can service thousands or millions of concurrent requests every day. In this technical briefing, we will describe the state of research and practices in the area of load testing. We will focus on the techniques used in the three phases of a load test: (1) designing a load test, (2) executing a load test, and (3) analyzing the results of a load test. This technical briefing is targeted at load testing practitioners and software engineering researchers interested in testing and analyzing the behavior of large-scale software systems.**

I. Introduction

Large-scale software systems ranging from e-commerce websites (e.g., Amazon) to telecommunication infrastructures (e.g., BlackBerry) must support concurrent access to thousands or millions of users. Failure to scale would cause catastrophic problems and unfavorable media coverage (e.g., the launch of HealthCare.gov [8] and the IPO of Facebook [5]). To ensure the quality of these systems, load testing is a required testing procedure in addition to conventional functional testing procedures like unit testing and integration testing.

Load testing, in general, refers to the practice of assessing the system behavior under *load*. A load is the rate of the incoming requests to the system [4]. A typical load test uses one or more load drivers that simultaneously send hundreds or thousands of requests to the system under test (SUT). During the course of a load test, the SUT is monitored and large volumes of system behaviour data (counters and execution logs) are recorded.

Load testing is a challenging task, as it requires a great understanding of the SUT [11]. Practitioners face many challenges such as tooling (choosing and implementing the testing tools), environments (software and hardware setup) and time (limited time to design, test and analyze). Due to its critical importance, industry has invested large amount of resources into building tools and infrastructures for load testing (e.g., Apache JMeter [1] and LoadRunner [2]). There are also an increasing number of research works done recently in the area of load testing (e.g., [3], [6], [7], [10], [12]). Load testing will become more important, as more services (e.g., Microsoft Office 365 and Salesforce.com) are being offered in the cloud to millions or even billions of users.

This technical briefing will cover the state of research and practices in load testing large-scale software systems. It is targeted at load testing practitioners as well as software engineering researchers interested in testing and analyzing the behavior of large-scale software systems under load. In the next section, we will present the outline of this technical briefing.

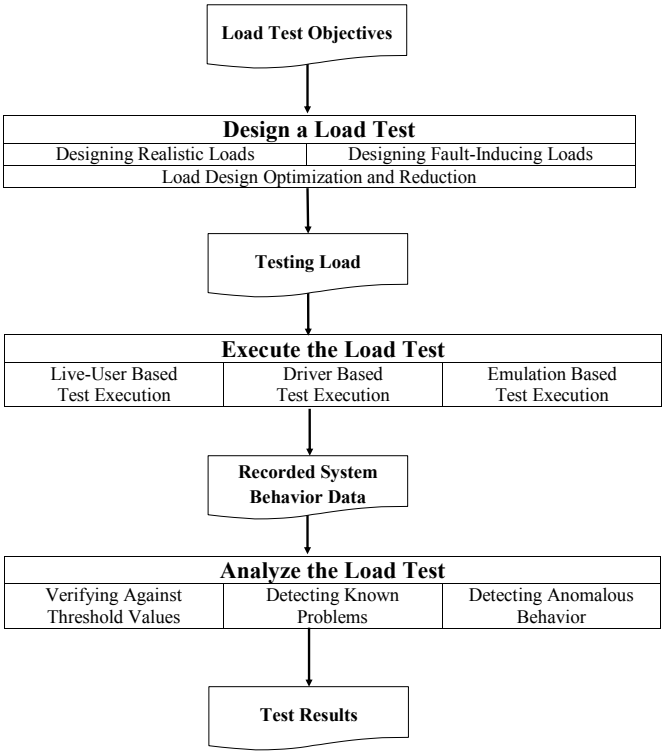

Fig. 1: Overview of the Load Testing Process

II. Outline of the Technical Briefing

This technical briefing will consist of five parts. Part One will provide the background information about load testing. Part Two, Three and Four will cover the techniques used in the three phases of a load test as shown in Figure 1. Part Five will present the challenges and opportunities in conducting research on load testing.

Part One: Background

In this part, we will cover the background information about load testing. We will start with the motivation of load testing. Then we will describe the context and objectives of load testing by comparing to other types of software testing (e.g., functional testing, performance testing and stress

testing) and software performance engineering activities (e.g., benchmarking and performance tuning).

Part Two: Designing a Load Test

There are two general approaches to design a load test:

1) *Designing realistic testing loads* aims to design a load test, which closely resembles the expected usage in the field.
2) *Designing fault-inducing testing loads* aims to design a load test, which is likely to reveal load-related problems.

In this part, we will explain various load design techniques derived from the above two approaches. In addition, we will also cover load design optimization and reduction techniques.

Part Three: Executing a Load Test

There are three general approaches to execute a load test:

1) *Live-user based test executions* hire human testers to manually generate testing loads;
2) *Driver based test executions* use load drivers to automatically generate testing loads; and
3) *Emulation based test executions* execute the load tests on specialized platforms (e.g., CHESS [9]).

In this part, we will describe the test execution techniques derived from the aforementioned three approaches.

Part Four: Analyzing a Load Test

During the course of a load test, there are two types of system behavior data being recorded: metrics and execution logs. Metrics can be functional related (e.g., number of pass and failed requests) or performance related (e.g., CPU and memory utilizations). Execution logs, which are generated by debug statements embedded in the source code, record software activities (e.g., item purchased) and runtime errors (e.g., database connection timeout). There are three general approaches to analyze the generated system behavior data:

1) *Threshold based techniques* verify the system behavior data against some known threshold values (e.g., SLA or the response time values from previous releases).
2) *Pattern based techniques* check the system behavior data against known problems (e.g., analyzing memory usage data for memory leaks).
3) *Anomaly detection based techniques* automatically learn the "normal/expected" behavior from the past tests and flag suspicious system behavior from the current test.

In this part, we will describe various load test analysis techniques derived from the above three approaches.

Part Five: Future Directions

In this part, we will present the challenges and opportunities in the area of load testing large-scale software systems.

III. About the Presenter

Zhen Ming (Jack) Jiang (http://www.cse.yorku.ca/~zmjiang/) is an Assistant Professor at the Department of Electrical Engineering and Computer Science, York University, Canada. Prior to joining York, he worked at BlackBerry's Performance Engineering Team for over half a decade. His research interests lie within Software Engineering and Computer Systems, with special interests in software performance engineering, mining software repositories, source code analysis, software architectural recovery, software visualizations and debugging and monitoring of distributed systems. Some of the tools resulted from his research are already adopted and used in practice on a daily basis to monitor and debug the health of several large-scale commercial software systems. He is the co-founder and co-organizer of the annually held International Workshop on Large-Scale Testing (LT), formally called International Workshop on Load Testing Large-Scale Software Systems. He is the recipient of several best paper awards including ICSE 2013, WCRE 2011 and MSR 2009 (challenge track). He received his PhD from the School of Computing at the Queen's University. He received his MMath and BMath degrees in Computer Science from the University of Waterloo.

References

[1] Apache JMeter. http://jakarta.apache.org/jmeter/, visited 2014-10-08.

[2] HP LoadRunner software. http://www8.hp.com/ca/en/software-solutions/loadrunner-load-testing/, visited 2014-10-08.

[3] C. Barna, M. Litoiu, and H. Ghanbari. Autonomic load-testing framework. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*, 2011.

[4] B. Beizer. *Software System Testing and Quality Assurance*. Van Nostrand Reinhold, March 1984.

[5] David Benoit. Nasdaq's Blow-by-Blow on What Happened to Facebook. http://blogs.wsj.com/deals/2012/05/21/nasdaqs-blow-by-blow-on-what-happened-to-facebook/, visited 2015-02-09.

[6] M. Grechanik, C. Fu, and Q. Xie. Automatically finding performance problems with feedback-directed learning software testing. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012.

[7] H. Malik, H. Hemmati, and A. E. Hassan. Automatic detection of performance deviations in the load testing of large scale systems. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE)*, 2013.

[8] Michael D. Shear and Robert Pear. Obama Admits Web Site Flaws on Health Law. http://www.nytimes.com/2013/10/22/us/politics/obama-pushes-health-law-but-concedes-web-site-problems.html, visited 2014-10-08.

[9] M. Musuvathi, S. Qadeer, T. Ball, G. Basler, P. A. Nainar, and I. Neamtiu. Finding and reproducing heisenbugs in concurrent programs. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008.

[10] M. D. Syer, Z. M. Jiang, M. Nagappan, A. E. Hassan, M. Nasser, and P. Flora. Continuous validation of load test suites. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2014.

[11] W. Visser. Who really cares if the program crashes? In *Proceedings of the 16th International SPIN Workshop on Model Checking Software*, Berlin, Heidelberg, 2009. Springer-Verlag.

[12] P. Zhang, S. Elbaum, and M. B. Dwyer. Compositional load test generation for software pipelines. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA)*, 2012.