Automated Analysis of Load Testing Results

Zhen Ming Jiang
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
zmjiang@cs.queensu.ca

ABSTRACT

Many software systems must be load tested to ensure that they can scale up while maintaining functional and performance requirements. Current industrial practices for checking the results of a load test remain ad hoc, involving high-level checks. Few research efforts are devoted to the automated analysis of load testing results, mainly due to the limited access to large scale systems for use as case studies. Automated and systematic load testing analysis is going to be much needed, as many services have been offered online to an increasing number of users. This dissertation proposes automated approaches to detect functional and performance problems in a load test by mining the recorded load testing data (execution logs and performance metrics). Case studies show that our approaches scale well to large enterprise systems and output high precision results that help analysts detect load testing problems.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: [Testing and Debugging]

General Terms

Performance, Reliability, Verification

1. INTRODUCTION

Many systems ranging from e-commerce websites to telecommunication infrastructures must support concurrent access by hundreds or thousands of users. Studies show that many problems reported in the field are not related to feature bugs, but to systems not scaling to field workloads [5, 20]. The inability to scale causes catastrophic failures and unfavorable media coverage. For example, some web service providers did not fully load test or plan their resources carefully for heavy traffic before the launch of their new product (e.g. MobileMe [4]) or new release (e.g. Firefox website [1]).

Unlike many functional testing techniques (e.g. unit testing or integration testing), which focus on testing a system based on a small number of users, load testing studies the behavior of a system by simulating hundreds or thousands of users performing tasks at the same time. A typical load test uses one or more load generators that simultaneously send requests to the system under test. A load test can last from several hours to a few days, during which execution logs along with performance metrics are collected. Execution logs are generated by debug statements that developers insert into the source code to record the run time behavior of the application under test. Performance metrics can be collected periodically by resource monitoring tools like PerfMon [2]. Performance metrics record the resource usage information such as CPU utilization, memory, disk I/O and network traffic.

The primary goal of a load test is to verify whether a system's functionality and performance can scale to a large number of users. This verification is difficult, due to the following challenges:

1. **No Documented System Behavior:** Correct and up-to-date documentation of the behavior of an application rarely exists [15].

2. **Monitoring Overhead:** Techniques that monitor or profile an application have a high overhead on an application and are not suitable for a load test.

3. **Time Pressure:** Load testing is usually the last step in an already delayed release schedule. The time allocated to analyze a test is even more limited, as running a load test usually takes days.

4. **Large Volume of Data:** A load test records performance metrics and logs that are usually hundreds of megabytes or even larger. This data must be analyzed thoroughly to uncover any problems in the load test, yet, in-depth manual analysis is not possible.

Motivated by the importance and challenges of the load testing analysis, this dissertation proposes automated approaches to detect functional and performance problems in a load test by analyzing the recorded execution logs and performance metrics.

Organization of the Paper

Section 2 describes the related load testing research. Section 3 outlines the current industrial practices on load testing analysis and their limitations. Section 4 presents our research hypothesis. Section 5 outlines our research plan. Section 6 lists the expected contributions in this dissertation. Section 7 concludes this paper.

2. STATE OF LOAD TESTING RESEARCH

There are various approaches for automatic generation of load test suites (e.g. [7, 11]), and for monitoring and diagnosing production systems (e.g. [8, 9]). There is very few work on systematic approaches to automatically analyze the load testing data for functional verification and performance evaluation. Yet, load testing analysis is an important problem, as studies show that many field problems are not related to feature bugs but to load problems [20]. It would be preferable that functional and performance issues can be caught early before the system is released to the field.

We believe that the current limited research on load testing analysis is mainly due to two reasons: First, there is limited access to large scale multi-user systems and load testing infrastructure, as many of such systems are developed in-house by commercial companies. Second, scalability problems are not a big concern for most prototype systems developed by researchers. However, as more and more services are offered in the cloud for thousands and millions of users, load testing analysis research will become indispensable.

3. STATE OF INDUSTRIAL PRACTICES

The primary goal of a load test is to verify functional correctness and to evaluate performance criteria for a system under load. Current industrial practices for load testing analysis remain ad hoc, involving high-level manual checks.

Verifying Functional Correctness

Load test engineers first check whether the application has crashed, restarted or hung during the load test. Then, load test engineers perform a more in-depth analysis by grepping through the log files for specific keywords like "failure" or "error". Load test engineers analyze the context of the matched log lines to determine whether these lines indicate problems or not.

There are two limitations in the current practice for checking functional correctness: First, not all log lines containing terms like "error" or "failure" are worth investigating. A log such as "Failure to locate item in the cache" is likely not a bug. Second, not all errors are indicated in the log file using the terms "error" or "failure". For example, even though the log line "Internal queue is full" does not contain the words "error" or "failure", it might be worthwhile investigating, as newly arriving items are possibly being dropped.

Evaluating Performance Criteria

Load test engineers first use domain knowledge to check the average response time of a few key scenarios. Then, load test engineers examine performance metrics for specific patterns (e.g. memory leaks). Finally, they compare these performance data with previous releases to assess whether there is a significant increase in the utilization of system resources.

We believe that current performance analysis practice is not efficient, since it takes hours of manual analysis. Current practice is neither sufficient for the following two reasons: First, checking the average response time does not provide a complete picture of the end user experience, as it is not clear how the response time evolves over time or how response time varies according to load. Second, merely reporting symptoms like "system is slowing down" or "higher resource utilization" does not provide enough context for developers to reproduce and diagnose the issues.

4. RESEARCH HYPOTHESIS

Two artifacts are recorded during a load test: execution logs and performance metrics. Execution logs record software activities (e.g. "User authentication successful") and errors (e.g. "Fail to retrieve customer profile"). Execution logs are widely available both in the testing and field environment for large enterprise systems, as they are used to support remote issue resolution and to cope with recent legal acts such as the "Sarbanes-Oxley Act of 2002" [3]. Performance metrics record the system's resource usage like CPU, memory, and disk I/O. Performance metrics can be collected by resource monitoring tools like PerfMon [2] with very little overhead. The information from execution logs and performance metrics complement each other, as over the course of a load test, execution logs record the system behavior and performance metrics keep track of the system resource utilization.

In this dissertation, we plan to propose automated approaches to verify functional correctness and to evaluate performance criteria of a system under load by mining the large set of execution logs and performance metrics data from load tests. Our underlying research hypothesis is as follows:

Historical load test repositories, which are a valuable, readily available and rarely explored resource, can form the basis of effective, automated load test analysis.

Our research hypothesis will be validated in the following three steps:

1. **Automated Abstraction of Load Testing Data:** We abstract the execution logs and performance metrics to facilitate the automated analysis in the next two steps.
2. **Automated Verification of Functional Correctness:** We verify a system's functional correctness by inferring functional models from the abstracted execution logs.
3. **Automated Evaluation of Performance Criteria:** We evaluate a system's performance criteria by building performance models from the abstracted execution logs and performance metrics.

We will explain the current progress and future plans of these three steps in the next section.

5. OUR APPROACH

As illustrated in Figure 1, our approach consists of three steps: We first abstract the log lines to execution events and performance metrics into performance categories. Then, we derive functional models by mining the sequences of execution events to uncover functional problems. We derive performance models by analyzing the test data from the test repositories to uncover performance problems. For each step, we explain our motivation, compare our proposed work against existing research, and describe our current progress and future plans. Some of our research results [12, 13, 14] are already adopted by industry.

(1) Automated Abstraction of Load Testing Data

Execution logs are hard to parse and analyze automatically, as they are free-form and have an infinite set of vocabulary. Each log line is a mixture of dynamic and static information. Log lines generated by the same output statements

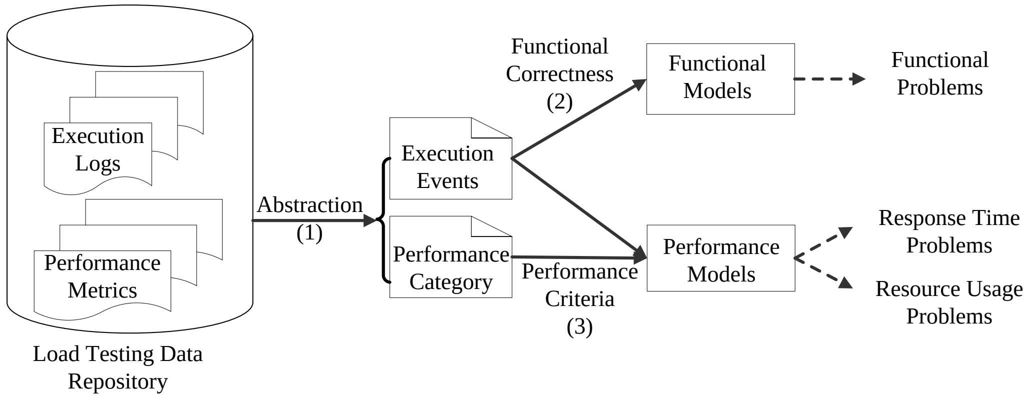

Figure 1: Our approach for automated load testing analysis

will have identical static information and the same structure of dynamic information. For example, the log line "Update shopping cart, item=*100*" should be automatically classified with all the other log lines of the abstracted form "Update shopping cart, item=*$v*". We call these abstracted forms *execution events*.

Existing log abstraction techniques, which work for generic log formats, either cannot scale to large log files [16] or cannot uniquely map one log line to one execution event [19]. Our approach is influenced by source code clone detection techniques to recognize the static and dynamic parts of the log lines [12]. Case studies show that our approach can handle large log file sizes with satisfying results ($>80\%$ in precision and recall).

As the system handles concurrent client requests, log lines from different scenarios are intermixed with each other in the execution logs. We have used various abstracted representations for scenarios formed by execution events (e.g. pairs [13] and sequences [14]), which are used in automated functional and performance analysis.

It is difficult for humans to interpret raw performance metrics, as it is not clear how to categorize these raw metric values into performance categories (e.g. high, medium and low). Furthermore, some data mining algorithms (e.g. Navie Bayes Classifier) only take discrete values as input. We are currently exploring generic approaches to classify performance metrics into discrete performance categories using techniques like control charts [18] to facilitate our future work in performance analysis.

(2) Automated Verification of Functional Correctness

We can verify a system's functional correctness by analyzing the execution logs. As a load test repeatedly executes a set of scenarios and is conducted after the functional testing is complete, the execution of the same scenario should generate identical event sequences. Any variations of these sequences might indicate potential problems.

We have proposed an approach [13] that derives the pair-wise temporal relations out of execution logs. Unlike many temporal specification mining approaches (e.g. [6, 10, 21]), which profile and analyze the applications on the method-invocation level, our analysis examines the execution logs to avoid huge performance overhead during a load test. Compared to method-invocation level data, execution logs are hard to abstract and group into scenario sequences. Our approach will flag the following case as problematic: the "lock open" event is followed by the "lock acquire" event 99% of the time, whereas the remaining 1% of the time "lock open" is followed by some other events. Case studies show that our approach detects various types of problems in the load testing environment, load generators, and the application under test, and scales well to large enterprise systems, flagging less than 1% of the log lines as troublesome.

We are currently extending our pair-wise anomaly detection approach to more generic representations like [10]. In addition, mining execution sequences of a whole load test at once might miss certain functional problems. For example, if the disk on the database server fills up halfway during a load test, the application under test will report errors for all the incoming requests that arrive afterwards. To resolve this problem, we plan to segment the log files into various chunks and compare temporal properties across chunks.

(3) Automated Evaluation of Performance Criteria

We can evaluate a system's performance criteria by comparing the data from different load tests. As similar loads are applied on load tests, performance data should be comparable across tests and informal performance baselines can be derived. We plan to evaluate a system's performance in two aspects: the end-user experience (response time) and the resource usage efficiency (e.g. CPU, memory, and network).

Response time throughout a test is not constant, as a typical workload usually consists of periods simulating peak usage and periods simulating off-hours usage. Therefore, we need to evaluate the end-user experience by examining the entire response time distribution instead of merely comparing the average response time. If the current run has scenarios that follow a different response time distribution than the baseline, this run is probably troublesome and worth investigating. In [14], we have presented an approach that automatically flags scenarios with response time problems by comparing the durations of execution sequences from the current test against previous test(s). We recover execution sequences from logs and calculate the duration of these sequences using the time stamps associated with each log line. Then, we compare the duration of various sequences using statistical techniques and visualize the problems in a report. Our approach not only reports scenarios with performance problems but also pin-points the performance bottlenecks within these scenarios. Case studies show that our approach produces few false alarms (with a precision of 77%) and scales well to large industrial systems. Unlike [9], our approach detects performance problems without specifications like SLOs. In contrast to [8], our approach provides more context for developers to reproduce and diagnose problems.

We are currently working on building performance models to evaluate the resource usage efficiency. We correlate general performance data (from metrics and response time) with the executed load (from execution logs). We expect to detect performance problems like abnormal performance spikes (e.g. unexpected resource contentions) and performance degradation (e.g. memory leaks).

6. EXPECTED THESIS CONTRIBUTIONS

The following are our expected thesis contributions:

1. **Techniques for abstracting load testing data:** Our log and metric abstraction techniques can be used for other types of research like mining customer profiles [17] or proposing new approaches for detecting load testing problems.
2. **A general methodology to analyze the behavior of a system during a load test:** We will propose methods to infer functional and performance models from large volume of load testing data. These methods can be used to systematically detect and diagnose functional and performance problems. Some of our research results [12, 13, 14] are already adopted by industry.

7. CONCLUSIONS

Large software systems must be load tested to ensure they can support a large number of concurrent users. Analyzing a load test is challenging, because it is hard to build models from the large set of load testing data. Current industrial practice consists mainly of high-level manual checks. Such practice is not efficient, nor is it sufficient. In this dissertation, we propose automated approaches to detect functional and performance problems by analyzing the recorded load testing data.

Acknowledgments

The author thanks Dr. Ahmed E. Hassan and Dr. Bram Adams for their comments on the draft. The author appreciates the generosity of the Performance Engineering team at Research In Motion (RIM). Working with the team as an embedded researcher, the author has gained an appreciation of the current practice of load testing and the daily challenges facing load test engineers.

8. REFERENCES

[1] Firefox download stunt sets record for quickest meltdown. `http://tinyurl.com/5ehfvq`.

[2] PerfMon Sample. `http://tinyurl.com/yzf8d32`.

[3] Sarbanes-Oxley Act of 2002. `http://www.soxlaw.com/`.

[4] Steve Jobs on MobileMe. `http://tinyurl.com/6dnng9`.

[5] Applied Performance Management Survey, Oct 2007.

[6] B. Anton, M. Leonardo, and P. Fabrizio. Ava: Automated interpretation of dynamically detected anomalies. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, 2009.

[7] A. Avritzer and E. J. Weyuker. The automatic generation of load test suites and the assessment of the resulting software. *IEEE Trans. Softw. Eng.*, 21(9), 1995.

[8] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni. Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change. In *IEEE International Conference on Dependable Systems and Networks*, 2008.

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

[10] M. Gabel and Z. Su. Javert: fully automatic mining of general temporal properties from dynamic traces. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008.

[11] V. Garousi, L. C. Briand, and Y. Labiche. Traffic-aware stress testing of distributed systems based on uml models. In *Proceedings of the 28th International Conference on Software Engineering*, 2006.

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

[13] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora. Automatic identification of load testing problems. In *Proceedings of the 24th IEEE International Conference on Software Maintenance*, 2008.

[14] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora. Automated performance analysis of load tests. In *Proceedings of the 25th IEEE International Conference on Software Maintenance (ICSM)*, 2009.

[15] D. L. Parnas. Software aging. In *Proceedings of the 16th International Conference on Software Engineering*, 1994.

[16] J. Stearley. Towards informatic analysis of syslogs. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, 2004.

[17] D. Thakkar, Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora. Retrieving relevant reports from a customer engagement repository. In *Proceedings of the 24th IEEE International Conference on Software Maintenance*, 2008.

[18] I. A. Trubin and L. Merritt. Mainframe global and workload level statistical exception detection system, based on masf. In *2004 CMG Conference*, 2004.

[19] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations and Management*, 2003.

[20] E. J. Weyuker and F. I. Vokolos. Experience with performance testing of software systems: Issues, an approach, and case study. *IEEE Trans. Softw. Eng.*, 26(12), 2000.

[21] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: mining temporal api rules from imperfect traces. In *Proceedings of the 28th International Conference on Software Engineering*, 2006.