Profile-based, Load-Independent Anomaly Detection and Analysis in Performance Regression Testing of Software Systems

Shadi Ghaith, Miao Wang, Philip Perry and John Murphy
School of Computer Science and Informatics
University College Dublin, Ireland
Email: shadi.ghaith@ucdconnect.ie

***Abstract*—Performance evaluation through regression testing is an important step in the software production process. It aims to make sure that the performance of new releases do not regress under a field-like load. The main outputs of regression tests are the metrics that represent the response time of various transactions as well as the resource utilization (CPU, disk I/O and Network). In this paper, we propose to use a concept known as Transaction Profile, which can provide a detailed representation for the transaction in a load independent manner, to detect anomalies through performance test runs. The approach uses data readily available in performance regression tests and a queueing network model of the system under test to infer the Transactions Profiles. Our initial results show that the Transactions Profiles calculated from load regression test data uncover the performance impact of any update to the software. Therefore we conclude that using Transactions Profiles is an effective approach to allow testing teams to easily assure each new software release does not suffer performance regression.**

***Keywords*-application change, transactions, performance models, regression testing**

I. Introduction

Regression testing is an important step in the development process of software systems. It aims to make sure a new version of software performs better, or at least no worse, than previous versions of the same software [1]. Generally Performance Teams need to run regression tests on each release to make sure performance has not regressed. Such runs apply a workload similar to expected field-like load for a period of time up to a few hours [2]. A huge amount of data containing hundreds of performance counters may be generated by such runs [3].

Each transaction triggers a request that propagates through the application layers to fulfil the user requirement by returning the appropriate response. Among various performance counters, the total time spent on all system resources, known as response time, is one of the key measurements that the performance regression testing measures and analyses. Unfortunately, regression testing is usually conducted at a late stage of the development lifecycle [1] leaving little time to analyse performance counters to detect performance problems. Investigating such measurements manually is time consuming and may not give detailed information about the cause of the reduced performance to help the engineers to fix the responsible bugs.

Another key factor in performance regression testing is the workload applied to the system [4]. Various transactions applied concurrently by various users, will compete for the available system resources resulting in queues thus increasing the response time. At the same time, test teams may need to execute various runs with different workloads to emulate various field-like scenarios. Hence, an increase of the response time can be due to workload variations rather than being caused by a bug [4]. Therefore, a load-independent method to detect and analyse bugs in the system that have caused response time regression would be a useful tool for the development process.

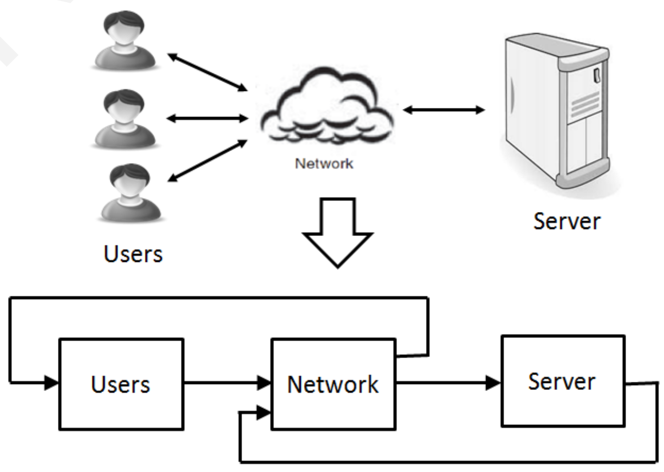

Figure 1. A Computer System And Its Queue Model.

On the other hand, a computer system can be represented as a queueing network [5]. It comprises of nodes and connections between these nodes. Each node corresponds to a certain hardware (or even software) resource, like CPU, and a queue. Figure 1 shows an example of a computer system represented as a queueing network. Transactions are represented as requests which are initiated by the Users node and pass through the connections to visit various nodes in the network. The response time of such transactions is therefore the sum of time spent in all nodes of the system excluding

Users node.

In this paper we propose to use a queueing network representation of the computer system to analyse the results of performance regression runs by utilizing the concept of Transactions Profiles (TP), which can be considered as a load independent representation of transaction response time. The main benefit of a TP is to provide an insight on the performance characteristics of the transaction in order to detect software anomalies and can only change if the application actually changes. The TP is load-independent and can be calculated from the regression test data for a given load using the queueing network representation of the system.

II. Background Information

We are applying concepts from the capacity management and computer modelling fields to the world of anomaly detection in load testing. In this section we introduce some of the main concepts from those fields.

A. Transaction Profile

Enterprise applications consist of transactions that visit various system resources (like CPU, Disk I/O) to fulfil the users request. Figure 2 depicts the distribution of the time required to process certain transaction between client PC and Server and the network.

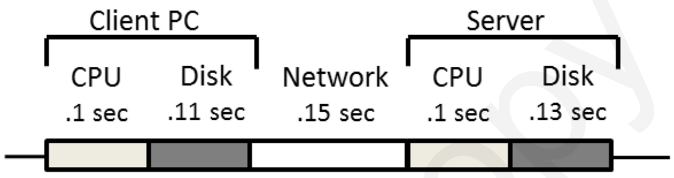

Figure 2. Time Spent By a Certain Transaction On All System Resources.

The total amount of time required to serve the transaction on each resource is called service demand. It equals the number of visits to the resource multiplied by the service time requirement per visit.

The complete series of service demands for one transaction is known as the Transaction Profile (TP). The TP represents the lower bound value of the response time for that transaction, or in other words it is the response time when the transaction is the only one in the system (no queueing). Accordingly, the TP is considered to be load-independent representation of the transaction. It can be visually represented by a horizontal bar that shows each component of the service demand as shown in Figure 3. In this example, a TP detail is broken down into CPU and I/O utilizations on the client and server machines. Although a software system may comprise thousands of transactions, the performance analysis usually focussed on a small number of key transactions that are critical to system performance.

It worths mentioning that the TP does not represent a real working scenario as no enterprise system is expected to work in a single user fashion; instead it is just a hypothetical concept that is used with the queueing network representation of the system to predict performance under varying loads as will be explained shortly.

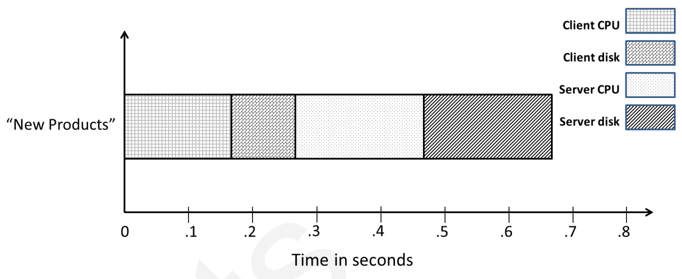

Figure 3. TP for the "New Products" Transaction.

B. Modelling Computer Systems as Queueing Networks

A queueing network can be used to represent computer systems [5]. Each node corresponds to a certain hardware (or even software) resource. Using the same example previously discussed in Figure 1 each node consists of one or many processing units (like CPUs) and a Queue. If a processing unit is available it processes the request directly otherwise the request has to wait in the queue. The time to process a request in a node equals the sum of the time spent in the queue and the time required by the processing unit. The queue length may vary as workload varies; hence queue time is the main factor that determines the overall time required to serve the request in each node and so the overall transaction response time. Modelling computer systems by a queueing network is widely used in performance analysis.

C. TP Usage in Capacity Management

One of the main applications of TP is in the capacity management of software systems. The goal is to predict the response times of the various transactions and resources utilization under load and decide if more hardware or software resources are required. To do so it is required to perform the process outlined in Figure 4 which is composed of the following tasks [5]:

1) Generate the queueing network representation of the computer system.
2) Determine the TP: It can be obtained by direct measurment using load generator software. Alternatively it may be calculated by measuring the normal load response times and resource utilization and calculate the service demands using techniques, which have been described in [6] [7].

3) For various workloads, solve the model to produce the corresponding response time and resource utilization. This can be done with a variety of commercial tools or open source tools such as the Java Modelling Tool (JMT) [8].

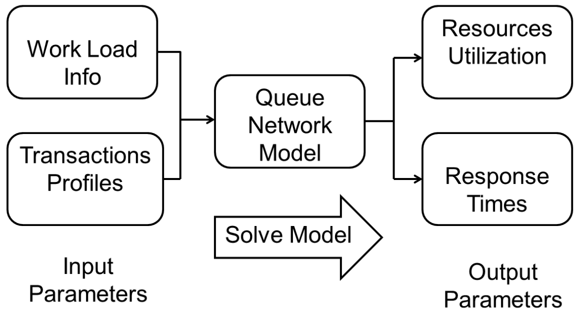

Figure 4. Process to Predict System Capacity Using TP.

III. Novel TP Approach

Based on the observation that the TP may reveal some important performance characteristics of the transaction, we propose to use the TP as a load-independent indicator of performance characteristics of transactions within a software system. As explained above, the TP is independent of the load applied to the system and it changes only when the application is modified in a way that affects the performance. Such changes may be caused by normal on-going development tasks like functional bug fixes as well as by adding new functionalities or modifying existing functionality. The central premise here is:-

if we infer the TP for two releases from corresponding regression testing data, then a visual (or automated) comparison between the TP of the two releases will highlight anomalous behaviour caused by software changes.

As shown in Figure 7 the comparison (visual or automated) between the TP of two successive releases may uncover anomalies when the new release has a longer TP time than the previous one. This behaviour can even be analysed further by looking at the components making up the time of the TP as shown in Figure 8.

We could measure the TP in a similar way to what is used in the capacity management as explained above. In this case a dedicated deployment for the TP measurement needs to be provided. Such a deployment is usually simpler than the normal load testing topologies.

Yet we propose calculating the TP from the data readily available by the normal load test runs. As shown in Figure 4, it is possible to solve the model knowing the TP and workload to obtain the transactions response time and resource utilizations. However in our case, the input parameters are the resource utilization and response time, which were the outputs in Figure 4. As shown in Figure 5, by reverse solving the model the corresponding TP can be calculated based on the new set of input parameters for a known workload.

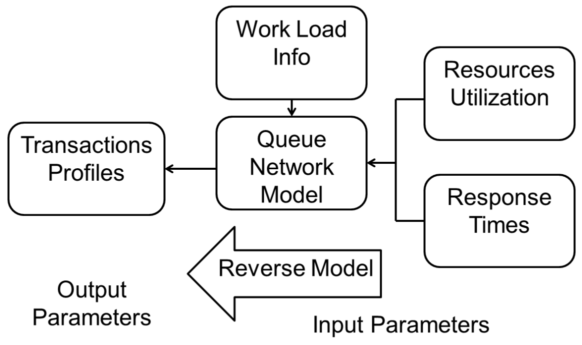

Figure 5. Calculating TP from Regression Test Data.

This concept is not new to the literature and was covered by some papers already like [6] [7]. Those papers assume that the queueing network model can be simplified to a single node. Casale et al [6] have described how to do a linear regression between resource utilization and workload (multiple runs with various workloads are required) and use this relation to calculate the TP. While Kraft et al [7] have followed a similar approach but uses the response time instead of the resource utilization.

However, these two approaches would only be suitable to calculate TP in simple deployments due to the single node assumption. For a more complicated deployments we propose to use the open source tool JMT and use a search based approach to find the TP.

As shown in Figure 6, we propose to solve the model with an initial TP input (for a known workload), the initial value can be the TP from the previous run or it can be measured approximately (by single user test). Else a random value can be used but this may increase the time required to converge the search. Then compare the result with the measured response time and resources utilization to adjust the input based on the difference. We do so until we get the value of TP that produces the correct results (measured resources utilization and response time).

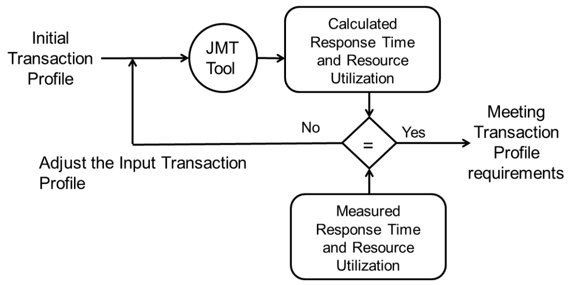
Initial Transaction Profile
JMT Tool
Calculated Response Time and Resource Utilization
No
=
Yes
Meeting Transaction Profile requirements
Adjust the Input Transaction Profile
Measured Response Time and Resource Utilization

Figure 6. Evaluate TP from Regression Test Data Using JMT.

Our approach requires the testing team to build and maintain a queueing network representation of the test environment. Such model represents the various hardware (and even software) resources and their number and specifications. Nevertheless, changes to such environments are not as frequent as other factors like the frequency of the test runs or the workload changes.

IV. Initial Experiments

To prove the use of TP to detect anomalies in software systems we measured the TP for a sample web application for a selected transaction "New Products". Our measurements were conducted using a load generator and various performance measurement tools like Pefmon and TCPDUMP. We have measured the TP for two releases of the software. The first one, release 05, is considered as a baseline TP. While the second one, release 06, contains an anomaly known to cause an extra processing in the Server CPU which increases the response time of the "New Products" transaction. The goal is to prove that the measured TP reflects this anomaly both by an increase of the entire TP time and by showing an increase of the utilization of the Server CPU.

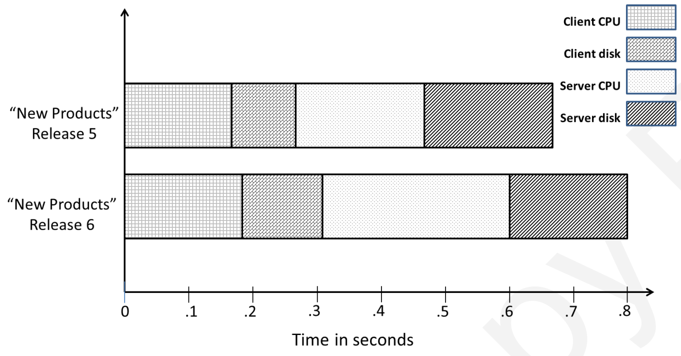

Figure 7. TP for "New Products".

The TP for "New Products" is shown in Figure 7 and we can see that the TP time has regressed in release 06 compared to release 05 from around 0.67 seconds to 0.8 seconds which is around 19.4% increase. Figure 8 can be used to further investigate the anomaly. It shows that the Server CPU component of the TP is the major contributor to the above increase of TP time as it undergoes an increase of 38%. While the client CPU, Disk and the Server Disk undergo changes of 5%, 9% and 0% respectively, those can be ignored given the low contribution to the entire TP time increase. In this scenario, the performance team needs to raise a software bug to report a performance regression of the "New Products" transaction which should include the information about the increase of Server CPU usage. In other cases where the new TP has a similar or a shorter time than the previous release, then no performance regression needs to be reported.

From this, it is clear that the TP helps in identifying the transactions with anomalous behaviour by test team and

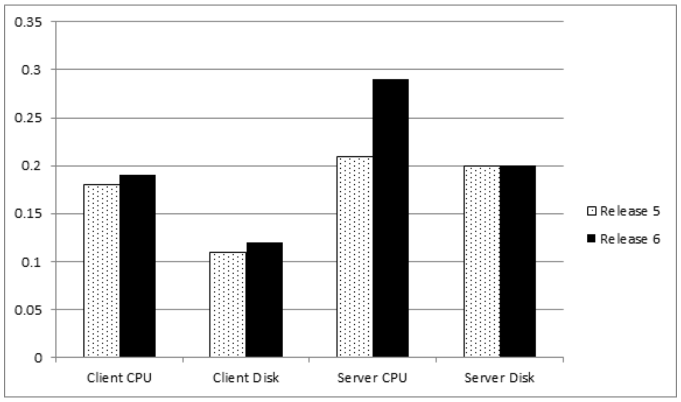

Figure 8. Components of TP for "New Products" Transaction.

also it gives useful information for analysts and developers about the component which service demand has significantly changed causing the TP to regress. This TP is independent from the load being applied to the system.

For the proof of concept above we measured the TP, however as explained above in the final solution we plan to calculate the TP from available load test data as shown in Figure 6. The initial implementation for this part employs the First Ascent Hill Climbing [9] search technique. The difference between the transaction response time and resources utilization, which are measured and calculated forms the fitness function that controls the input TP until the output parameters matches the measured ones.

Initial tests are showing good results for few nodes in the queueing models, but it is caught by a local optimum for bigger models. We plan to use different search techniques like Simulated Annealing [9] to avoid this.

Another experiment was conducted to make sure TP is really load-independent as per its definition. In this case we did two different runs on the same software release and same hardware with two different loads. Then we inferred the TP from both runs and we verified it is, as expected, the same in both cases. We omit the details of this experiment due to space limitation.

V. Related Work

Performance testing of software systems based on the resource utilisation and response time as a transaction is executed is a standard performance monitoring technique, such as the system used in [10].

Industries currently lack a complete solution for the processing of the results of regression tests [1]. Surprising, the manual approach has been widely used to analyze the quality of software systems. However, a manual solution is usually error-prone and time consuming especially for a timely release schedule making such a solution not viable.

Statistical techniques are among the first techniques used to detect anomalies in various fields including regression testing data. Particularly, Statistical Control charts were examined and some researches [4] [2] have been published about it. These papers show that Control charts used for Process Control [3] of industrial systems can be applied to detect anomalous behaviour of regression tests data. Another statistical technique is to find a correlation between performance counters and provide a "confidence" value defined as the level of how probable that a counter value will follow a certain related counter(s) value. A confidence close to 1 denotes a high probability and close to zero if it is not almost related. So if a confidence is changed between counters in the current release compared to previous release an anomaly is raised [11]. Statistical techniques rely on pre-set thresholds to determine the level at which anomalies are declared. Determining the appropriate threshold is load dependent, making the technique less suitable to regression testing where the goal is to find anomalies that are caused by application change rather than workload change [4].

Furthermore, Mi et al [12] has described an interesting approach to analyze the regression test data using application signatures. This approach relies on calculating the service demand of a transaction on the CPU of the application server and assumes this service demand will not change unless changes have been made to the program that caused a performance regression. This effort uses intrusive probes on the J2EE container to calculate service demands. Also it is not extendable to multi-tiers applications as well as to resources other than CPU utilization like disk I/O and Network.

Calculating TP from the output of load runs, which include the response time and resources utilization, has been discussed in many papers. Among those, Casale et al [6] have described how to do a linear regression between resource utilization and workload (multiple runs with various workloads are required) and use this relation to calculate the TP. While Kraft et al [7] have followed a similar approach but uses the response time instead of the resource utilization. However, these two approaches would only be suitable to calculate TP in simple deployments due to the single node assumption, and they require multiple runs with multiple loads.

VI. Conclusion

Our results show that TP is capable of being used as a good load-independent representation of performance of software systems. The TP approach can be used in performance regression testing to detect and analyse anomalies. Because this approach has been designed as an automated and load independent solution, the amount of time and efforts can be significantly reduced especially when testing with different loads is required. Clearly, this approach will significantly improve the test process by reducing testing time for multiple workloads and by improving the information fed back to developers about performance regressions.

Our future work will prove the validity of the approach using real multi-tier applications with different kinds of hardware resources. We also plan to study the effect of software resources like RAM and threads on the model to get a more accurate TP.

Acknowledgment

This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

References

[1] Jiang, Z. M. 2010. Automated analysis of load testing results. In *Proceedings of the 19th international symposium on Software testing and analysis*. ACM, New York, NY, USA.

[2] Nguyen, T. H. D. 2012. Using control charts for detecting and understanding performance regressions in large software. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE Computer Society, Washington, DC, USA.

[3] Nguyen, T. H., Adams, B., Jiang, Z. M., Hassan, A. E., Nasser, M., and Flora, P. 2012. Automated detection of performance regressions using statistical process control techniques. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*. New York, NY, USA.

[4] Bereznay, F. M. 2006. Did something change? using statistical techniques to interpret service and resource metrics. In *32nd International Computer Measurement Group Conference, December 3-6, 2006, Reno, Nevada, USA, Proceedings*. Computer Measurement Group.

[5] Grinshpan, L. 2012. *Solving Enterprise Applications Performance Puzzles*. John Wiley and Sons, Inc., Hoboken, New Jersey.

[6] Casale, G., Cremonesi, P., and Turrin, R. 2008. Robust workload estimation in queueing network performance models. In *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*. IEEE Computer Society, Washington, DC, USA.

[7] Kraft, S., Pacheco-Sanchez, S., Casale, G., and Dawson, S. 2009. Estimating service resource consumption from response time measurements. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium.

[8] Bertoli, M., Casale, G., and Serazzi, G. 2009. Jmt: performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev. 36,* 4, 10–15.

[9] Ghaith, S. and Ó Cinnéide, M. 2012. Improving software security using search-based refactoring.

[10] Parsons, T., Mos, A., and Murphy, J. 2006. Non-intrusive end to end run-time pathtracing for j2ee systems. In *IEE Proceedings - Software, 153 (4) 149-161*.

[11] Foo, K. C., Jiang, Z. M., Adams, B., Hassan, A. E., Zou, Y., and Flora, P. 2010. Mining performance regression testing repositories for automated performance analysis. In *Proceedings of the 2010 10th International Conference on Quality Software*. IEEE Computer Society, Washington, DC, USA.

[12] Mi, N., Cherkasova, L., Ozonat, K. M., Symons, J., and Smirni, E. 2008. Analysis of application performance and its change via representative application signatures. In *NOMS* (2008-12-10). IEEE.