SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/stvr.1573

Anomaly detection in performance regression testing by transaction profile estimation

Shadi Ghaith[1,*,†], Miao Wang[1], Philip Perry[1], Zhen Ming Jiang[2], Pat O'Sullivan[3] and John Murphy[1]

[1]*Lero, Performance Engineering Lab, School of Computer Science and Informatics, University College Dublin, Dublin, Ireland*
[2]*Department of Electrical Engineering and Computer Science, York University, Toronto, ON, Canada*
[3]*Systems and Performance Engineering, IBM Dublin, Dublin, Ireland*

SUMMARY

As part of the process to test a new release of an application, the performance testing team need to confirm that the existing functionalities do not perform worse than those in the previous release, a problem known as performance regression anomaly. Most existing approaches to analyse performance regression testing data vary according to the applied workload, which usually leads to the need for an extra performance testing run. To ease such lengthy tasks, we propose a new workload-independent, automated technique to detect anomalies in performance regression testing data using the concept known as transaction profile (TP). The TP is inferred from the performance regression testing data along with the queueing network model of the testing system. Based on a case study conducted against two web applications, one open source and one industrial, we have been able to automatically generate the 'TP run report' and verify that it can be used to uncover performance regression anomalies caused by software updates. In particular, the report helped us to isolate the real anomaly issues from those caused by workload changes with an average F1 measure of 85% for the open source application and 90% for the industrial application. Such results support our proposal to use the TP as a more efficient technique in identifying performance regression anomalies than the state of the art industry and research techniques. Copyright © 2015 John Wiley & Sons, Ltd.

Received 27 November 2013; Revised 11 November 2014; Accepted 14 February 2015

KEY WORDS: software update; performance models; performance regression testing

1. INTRODUCTION

When an enterprise application is tested during development, it is important to not only test its ability to perform desired functions through functional tests but also to assess how well it performs those functions through performance testing. As part of this performance testing process, testing teams need to perform regression testing on each new release or milestone build of the software to make sure that the existing functionalities do not perform worse than the previous version [1], such a problem is known as performance regression anomaly [2].

A performance testing run involves exposing the application to a field-like workload using load generators such as HP LoadRunner [3] and JMeter [4] (open source load generator) for an extended period [5] to simulate normal users interacting with the system. In most cases, testing teams need to collect a huge amount of performance counters to analyse and compare against previous base-line release counters to identify performance regression anomalies.

When interacting with the enterprise application, users initiate software transactions to invoke various application functions such as the system login and browsing system catalogues. When such

*Correspondence to: Shadi Ghaith, School of Computer Science and Informatics, University College Dublin, Ireland.
†E-mail: shadi.ghaith@ucdconnect.ie

a transaction request is triggered, it propagates through the system tiers and is served by each system resource such as CPU, disk input/output (I/O) and Network to fulfil the user demand, and a response is sent back to the user once it is complete [6].

The performance counters that are gathered during the software tests are mainly under the following two categories [7]:

1. Transaction response time (TRT). The total time to process the request using the various system resources [8]. The TRT, along with transactions types and transactions rates, can be collected by most load generators, such as HP LoadRunner and JMeter.
2. Resource utilization (RU) of the computer system resources such as CPU, disk I/O and Network [9]. It is produced by monitoring tools (such as Perfmon (the system monitoring tool on Windows platform), NMON [10] and TCPDump [11]) on the various servers.

These counters are then investigated for anomalous behaviours such as an increase in the TRT for a particular transaction, unless it has been redesigned and equivalent functionality is discontinued, or an increase of the RU, such as CPU, for one of the servers [12]. For example, if the TRT of a certain transaction is increased from 0.25 to 0.28 s, a performance regression anomaly is then flagged, and the run is considered as a failure, otherwise the run is considered as a pass. Figure 1 shows a high level diagram for the software performance regression testing process [12].

Based on the process illustrated in the Figure 1, the following challenges need to be overcome:

1. Workload changes: Runs on new releases usually need to be carried out with a different workload to account for changing field requirements [13]. The workload changes can happen by either changing the number of users accessing the system, or by using a new transaction mix, which is defined as the fractions of the users issuing each transaction type. Thus, changes to the performance counters can be caused by either a change of the workload applied during the testing run or an anomalous change in the new software release. Traditionally, because of the workload dependency, an extra run on the new release with a similar workload to the previous release workload is conducted, which is time and resource consuming.
2. Manual process: The testing engineer needs to investigate the various performance counters mentioned earlier to look for anomalies in the TRT or RU. However, in a typical project life cycle performance regression testing is conducted only at later stages, where it is usually delayed and leaving very little time to properly run regression tests and manually analyse the result [1]. Furthermore, as a huge amount of performance counters are collected, manually carrying out the data analysis is not viable.

From the preceding text, it can be inferred that the need for a workload-independent, automated solution to analyse performance regression testing data is appealing [1, 13, 14]. Accordingly, in [15, 16] we introduced an approach to reduce the lengthy process to detect performance regression anomalies in software systems caused by a software update and isolate them from those caused by load variations. We do so by applying concepts from queueing networks domain as used for capacity management process of computer systems. This workload-independent, automated approach aims

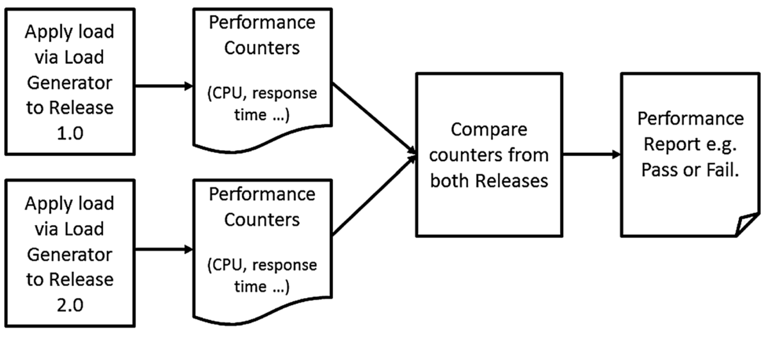

Apply load via Load Generator to Release 1.0
Performance Counters (CPU, response time …)
Apply load via Load Generator to Release 2.0
Performance Counters (CPU, response time …)
Compare counters from both Releases
Performance Report e.g. Pass or Fail.

Figure 1. Performance regression testing process [12].

to detect the majority of anomalies and therefore reduce the testing time, the time taken for each release cycle, the costs and the time to market for each release.

We achieve this by introducing a new metric, other than TRT, which is independent of the applied workload. This metric is called transaction profile (TP) [17]. We propose to use the output of performance testing process, mainly the TRT and RU counters, and the queueing network model (QNM) [18] of the testing system to calculate the TP. Conceptually, if the testing system is viewed as a QNM, then the TP is the input and the TRT and RU are the outputs. Given the TRT and RU, the QNM can be reverse-solved to obtain the TP. Because the TP is independent of the workload applied to the system, it is only sensitive to variations caused by software updates.

The main contributions of this paper are as follows:

1. Introduce a technique to allow applying the TP approach to deployments with complex configurations, such as clusters of application and database servers.
2. Compare the TP-based technique with the main existing industrial approach (i.e. manual process using TRTs) and the state of the art research approach based on control charts [5].
3. A comprehensive case study on a large-scale industrial application provided by our business partner on their testing environment, with a realistic test run size and duration.

The remainder of the paper is structured as follows. First, we discuss the current performance testing process in Section 2, then we introduce our TP approach in Section 3, followed by a description of the QNM of computer systems and solving them in Section 4. Section 5 introduces proposed approaches to obtain the TP. A case study is presented in Section 6 followed by a discussion of the current limitations and future work in Section 7. Finally, related work and conclusions are covered in Sections 8 and 9, respectively.

2. PERFORMANCE REGRESSION TESTING PROCESS

In conventional software building process, the performance testing step is usually performed towards the end of the release cycle. Performance testing is concerned with the responsiveness (speed) of various transactions when performed concurrently by multiple users. Performance testing does not involve any functional verification including those caused by users concurrently accessing the system, which is covered in previous phases of the testing process.

The TRTs of various transactions are measured and compared with targeted values, typically specified in the service level agreement (SLA). Additionally, it is important to ensure that transactions from the previous version have not regressed [12, 19].

There is no separate performance regression testing process set; instead, it is usually performed as part of the general performance testing process, which involves at least two runs [12, 20]

1. Primary performance testing run (PPTR) (usually referred to as just performance testing run): This run is performed with new expected field-like workload using the entire transactions set (new and previous). The TRTs are measured and compared with targeted values as per the SLA).
2. Performance regression testing run (PRTR): The TRTs of the PPTR cannot be compared with TRTs from previous release because of different workloads (number of users and the transaction mix) [9]. Hence, the PRTR is performed on the new release but with a workload similar to that used in the previous release [19]. This run is compared with previous release run as depicted in Figure 1.

Each of these two runs takes several hours to execute in addition to various tasks required before and after each of them (e.g. reset the database, clean up and collect logs of various measuring tools), which is added to the time required to analyse both runs data to identify anomalies. Table I shows data obtained from our industrial collaborator about the time taken for a typical performance test. Depending on the development process, each run is repeated many times ranging from 10 to 100 times every release.

Following the analysis step, the information about any anomalous behaviours is fed back to the development team to fix. In addition to the two major challenges mentioned in Section 1, namely

Table I. List of activities required to conduct a performance test run and the time required to do each one of them (data provided by our industrial collaborator).

Activity	Typical time	Comment
SUT restore*	38–40 min	
SUT start servers	21 min	
SUT start and run of performance workload	1 h 44 min 2 h 24 min 3 h 52 min (most common) 4 h 20 min	A number of schedules with different timings are used.
SUT collect logs	12–20 min	This includes zipping the entire performance tester workspace
SUT parse result	6 min	To produce reports
Initial review of reports	A few minutes	To see if the run is good
Any more in-depth analysis	Variable time	

*SUT, system under test (i.e. all the machines making up a test system).

the time constraints and the workload changes, narrowing a regressed transaction in a large complex system down to a specific component is an extra considerable challenge.

Moreover, performance testing process in an agile software production process is performed multiple times at the end of each iteration or group of iterations. The time window available to conduct such performance runs is even tighter than the conventional process putting extra challenges on the performance testing process. New functionalities are introduced in each iteration in the agile process and so the transactions mix changes, which lead to different workload levels in each iteration making the regression testing process even more time and resources consuming. Hence, improving the process described earlier will be even more substantial for the agile process. Additionally, trends evolving for various performance counters across iterations are important for the decision makers participating in the software-building process including requirements engineering, system architects, developers and product management.

In this paper, we modify the performance testing process by eliminating the need for the PRTR and instead utilize the PPTR conducted at the new workload levels. We achieve this by introducing a new variant of the TRT called the TP, which will not change across runs with different workloads. The TP is obtained by analysing the readily available data of the PPTR (with new workload requirements). The goal is to improve the performance testing step by significantly reducing (up to 50%) the time and resources required to perform performance testing and keeping the process as simple as possible. Similarly, our modified process provides useful information about the actual resources contributing to the performance regression anomaly.

3. NOVEL TRANSACTION PROFILE APPROACH

3.1. What is the transaction profile?

The systems under investigation here are composed of end users connected through a network to a software system deployed on one or more servers, which can be represented by a QNM as the one shown in Figure 2.

Each user initiates one transaction at a time that visits various resources, such as CPU and disk I/O, to fulfil the user's request [17, 21]. The distribution of the time required to process a certain transaction, the TRT, is shown in Figure 3. The TRT varies with workload as it includes the time spent waiting in queues for access to share resource. The queue lengths depend on the number of users accessing the system and the mix of transactions being processed at a given time. The mix of transactions varies according to different usage scenarios. Hence, we look for an alternative metric to the TRT that does not depend on workload (i.e. not sensitive to queueing effect).

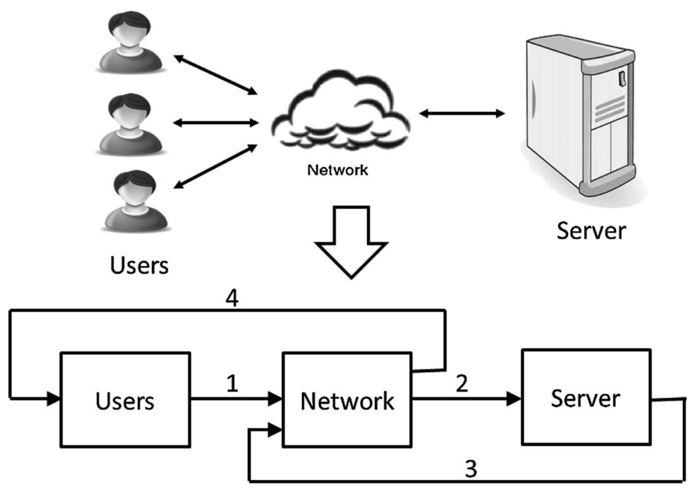


Figure 2. Queueing network model of a computer system.

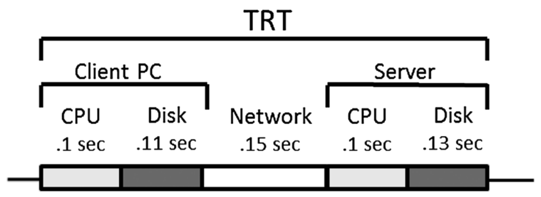


Figure 3. Time spent by a certain transaction on various system resources. TRT, transaction response time.

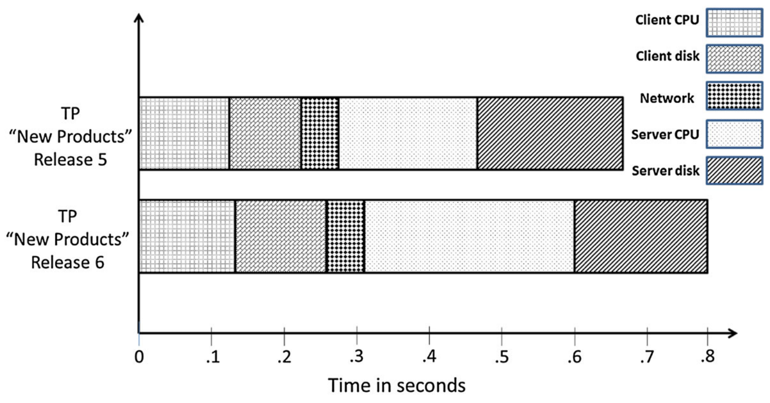


Figure 4. Comparison of transaction profile (TP) for 'New Products' transaction between two releases.

The total amount of time required to serve the transaction on each resource, excluding the time on the queues, is called the service demand (SD) [22]. It equals the number of visits to the resource multiplied by the service time per visit [23].

The complete series of SDs over all resources for one transaction is known as the TP [17]. The TP represents the lower bound value of the TRT for that transaction, or in other words, it is the TRT when the transaction is the only one in the system (no queueing). Given that the TP excludes the time spent in the queues of various resources, it is considered as a workload-independent representation of the transaction. To easily visualize each component, TP is represented by a horizontal bar as shown in Figure 4 for an example transaction called 'New Products' in an online shopping application.

The components building the TP (which are the service times per visit, the numbers of visits (and so the SDs) and which resources are visited by each transaction) are all characteristics of the software. They only change when the software is updated in a manner that affects its performance. Such changes include things such as requiring more visits to the disk I/O or extra CPU processing as well as requiring service from new resources such as the introduction of a database call to retrieve an extra piece of data to fulfil a new functional requirement.

Although an enterprise application may consist of hundreds of transactions, usually only a few of them are key transactions that are used repeatedly by most users. The set of all TPs for a single application is known as the application profile [17].

It is worth mentioning that the TP does not represent a real working scenario, as it is not realistic to have a single user accessing the system; instead, it is just a hypothetical concept that is used with the QNM of the system to predict performance under different workload levels as will be explained in Section 4.

3.2. Applying the transaction profile approach to performance regression testing

Given that the TP reveals some important performance characteristics about the transaction, we propose to use it as a workload-independent indicator of transaction performance in software systems, mainly in performance regression testing. The TP can change only if the software is updated in a way that affects its performance (and not by applying a new workload). Such changes may be caused by normal development tasks such as functional bug fixes as well as by adding new functionalities or modifying existing ones.

The central premise here is as follows:

An automated (or visual) comparison between the TP of the two releases will highlight anomalous behaviour caused by software update as opposed to those caused by applying a new workload.

As shown in Figure 4, a comparison between the TP of two software releases uncovers any performance anomaly caused by a software update rather than higher workloads. If the new TP is longer than the old TP, as the case in Figure 4, then this transaction has regressed in the new release. This regression can be analysed further to uncover the resource whose SD contributes most to the increase in the TP, which is server CPU in this example.

This approach addresses the two major challenges of regression testing explained in Section 1, namely the workload-dependency and the manual approach.

The key novelty is to calculate the TP from normal regression testing data along with the QNM of the system as will be explained in Section 5.

3.3. Transaction profile run report

After the running of new performance tests, the performance engineer now needs to search for performance regressions in the captured performance data. One key question the engineer needs to answer is that ‘whether the new software update has caused any degradation of the TRT of various transactions?’ To answer such a question, the engineer needs to distinguish this degradation from those caused by the workload changes between current and previous runs.

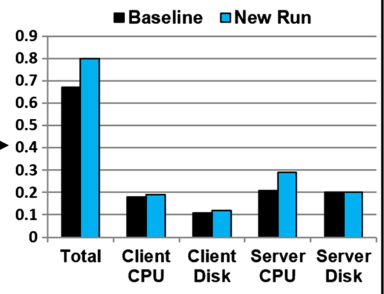
The TP run report tool presented in this paper can be used to do this. By running it against normal performance data for both releases (including TRTs, RUs and the workload information), it produces a report as shown in Figure 5 that consists of two parts (i) TP report summary and (ii) detailed TP graph.

3.3.1. Transaction profile report summary. The summary page shown in Figure 5(a) shows the transactions (named ‘New Products’, ‘Home’ and ‘Best Sellers’) that are flagged because their current TP increased from previous TP by a percentage that exceeds a preset threshold, which we will determine in the case study section. Such a report gives a first idea about the performance characteristics of each transaction.

3.3.2. Detailed transaction profile graph. After obtaining the information about the TP deviation for the transactions exceeding the threshold, the engineer can investigate the detailed TP graph shown in Figure 5(b). This graph shows the details of the TP deviation both the total TP time and the individual components making up the TP such as server and client CPU. In this example, the

Transaction Name	*TP Deviation (%)*
New Product	19.4
Best Seller	7.2
Home	4.3

(a) TP Report Summary.

Transaction Name	*TP Deviation (%)*
New Product	
Best Seller	7.2
Home	4.3

(b) Detailed TP Graph.

Figure 5. Transaction profile (TP) run report.

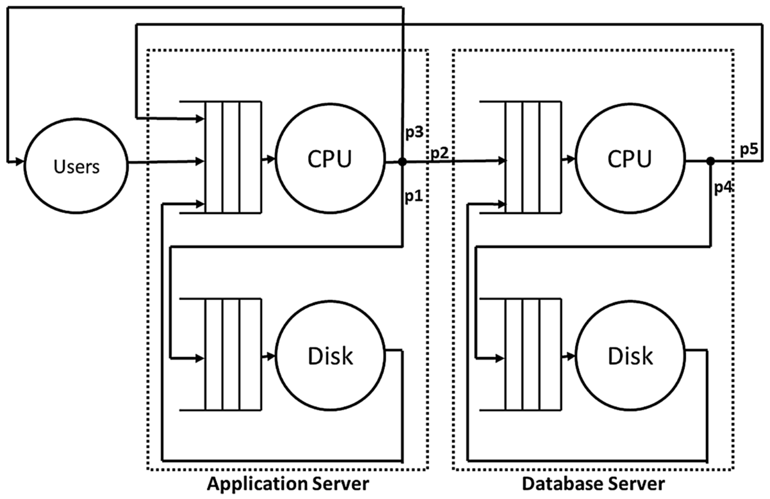


Figure 6. Computer system and its queueing network model.

SD on the server CPU has the main contribution to the increase in the TP of the ‘New Products’ transaction. Such a change to the SD is likely caused by a change in the transaction code.

The test engineer will now have enough information about each regression and can open an anomaly report including the information from the TP run report.

4. QUEUEING NETWORK MODEL

4.1. Overview

Similar to many other systems, software systems can be represented as a queuing network [17, 23–25] as shown in Figure 6. Each node represents one system resource such as CPU, disk I/O and Network. Each one of these nodes is composed of a processing unit and a queue. The processing unit serves each request if it is available, otherwise the request waits in the queue [23].

The users node in Figure 6 models end users performing various requests, known as transactions with a certain arrival rate. When the request leaves the users node, it goes to the application server CPU. During processing, the application server CPU may need to visit the application server disk I/O to perform certain actions. After the disk I/O finishes serving the request, it returns to the application server CPU, which resumes serving the request and might visit the disk I/O again or request a service from the database server. Similar to the application server, the database server CPU invokes its disk I/O and returns to application server CPU upon completion. The application server CPU resumes serving the request and may again visit the application server disk I/O, the database server CPU or return back to the users node. Then, a response is sent back to the users node, it waits in the users node for a period called the think time (TT), which is the time the user spends analysing the response before issuing the next request.

In queueing networks terminology, nodes are referred to as stations and the request types as classes, while each request is known as a customer [24]. The time to serve a customer in each node is known as service time. The customer may visit each node multiple times, known as visits, and the total time spent by a customer in each node, during all visits, is known as the SD [24] (as was defined in Section 3.1).

The model in Figure 6 models the CPU and the disk I/O and ignores the network. This is a valid approximation given the fast network connections between the various servers. This is verified as good results are obtained by applying this approximation, which is also adapted widely in the capacity management field. For example, this assumption is made in the following capacity management work [26] and in this book [17].

4.2. Solving queueing network model to predict system performance

Solving a queueing network requires knowledge of two aspects [27]

1. Workload characterization, which includes [17] the following:
   (a) A list of all transaction types (classes).
   (b) The number of users issuing each transaction.
   (c) TT spent in the users nodes.
2. Service times and number of visits for each transaction (class) at each resource (station) or the SDs (total service time over all visits).

Solving the network will produce information about the performance of the system, most importantly the TRT for the various classes and the RU of stations [27]. This process is depicted in Figure 7. The QNM can be solved via tools similar to the Java modelling tool (JMT) [28]. The input workload is varied, and the corresponding TRT and RU are found and analysed. TRT and RU are the main two parameters of the system performance as they affect the end user of the system. The TRTs need to meet the information in the SLA, while RUs provide information on the resource (or server) that is going to be a bottleneck of the system, this helps in planning and upgrading the system capacity.

Capacity management projects are usually very time constrained [29], so the Baskett–Chandy–Muntz–Palacios (BCMP) approximation is often used to reduce the QNM solving time [27, 30]. This allows a model such as the one shown in Figure 6 to be solved using SDs (total service time

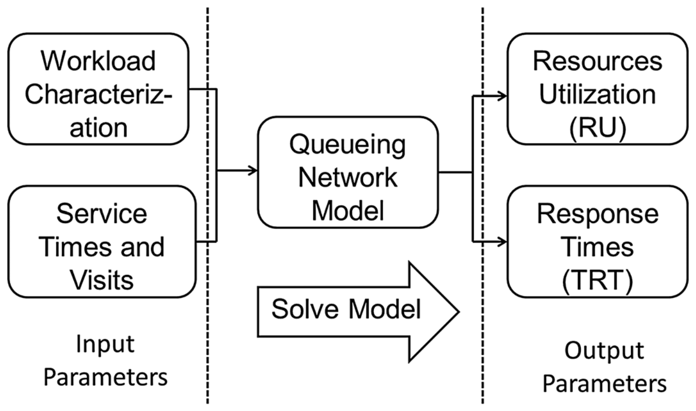


Figure 7. Solving the queueing network model to predict system performance.

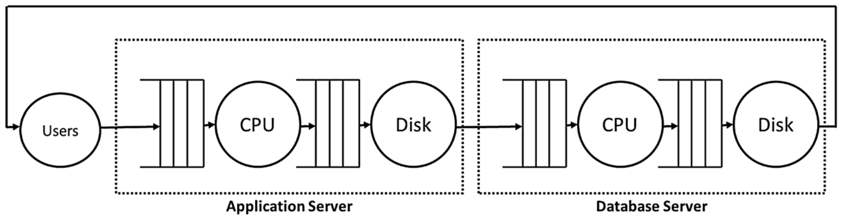


Figure 8. Representation of computer system using service demands instead of service times and the number of visits to resources.

over all visits) instead of service times and number of visits. Consequently, the QNM in Figure 6 can be simplified as shown in Figure 8 allowing for fast, efficient analytical solving of the QNM [31]. This enchantment of the solver speed is even more critical to our approach to be presented in Section 5.2.1 in the succeeding text as it involves solving the QNM multiple times.

5. APPROACHES TO OBTAIN TRANSACTION PROFILE

In this section, we present our proposed way to obtain the TP from performance regression testing data along with the QNM of the testing system. First we explain the limitations by directly measuring the TP using a single user test.

5.1. *Direct measurement*

The TP can be measured by applying each transaction type individually by a load generation tool, such as JMeter, and measuring the RU on the various system resources, a process known as single user testing run. The SD on each resource can then be calculated from the RU and throughput, by the service demand law [26] as follows:

$$D = \left(\frac{RU}{X}\right) \tag{1}$$

where D is the SD on certain resource, RU is the utilization on that resource and X is the transaction throughput.

This approach has the following serious issues preventing it from being used for performance regression testing (and even in capacity management):

1. Some behaviours of enterprise systems, such as caching, are difficult to model with the current state of the QNM art. Hence, inferring the TPs from multiuser runs data is an approximation that cures this problem [32, 33].
2. High inaccuracy in measuring the SDs due to their small values [32, 34].
3. It is unreliable to measure SDs on virtualized systems [32].

Additionally, as explained in Section 2 we aim to speed up, and reduce resources to perform, the performance testing process by eliminating the PRTR and utilize the data from the PPTR. Introducing a new type of runs to measure the SDs, even if it is shorter, seems to defeat this purpose, and even worse as it requires new tools and expertise.

5.2. *Inferring transaction profile from performance data*

5.2.1. Presented approach. In Section 4.2, we showed the process used to predict application performance in the capacity management field. The process is depicted in Figure 7 where the SDs (and so the TPs) and the workload characterization along with the QNM form the input to the solver (such as JMT) in the process to obtain the TRTs and RUs.

In regression testing, the TRTs and RUs are available from performance testing data, in addition to the workload characterization. The QNM of the testing system can be obtained easily. Therefore, we present to reverse-solve the model where the output parameters in the normal process form the input for the reverse process. The output now are the SDs (that are used to form the TPs). This process is depicted in Figure 9.

Novel approach to reverse-solve queueing network model. Looking for an analytical solution to infer SDs (TPs) from TRTs and RUs is not possible. The equations for closed multiclass QNM are recursive and cannot be inverted [35]. Yet, estimating SDs from RUs and TRTs has been previously explored in the capacity management domain [32, 34, 36] (more details about these techniques, including their limitations that prevent us from using them in regression testing, are presented in Section 8).

Hence, we present a search-based approach [37] to reverse-solve the QNM using JMT, starting with an initial TP value, as shown in Figure 10. The initial value can be the TP from the previous run or measured approximately by a single user test. Alternatively, a random value can be used, but this may increase the time required to converge the search process. The other input, shown in Figure 7, is the workload characterization, which is known and will be fixed across the process so it is not shown in Figure 10.

A relatively standard search-based process is applied, where the local search technique [37] loops through all initial SDs. For each one, it tests the adjacent points by incrementing (decrementing) that SD with selected steps. The QNM is solved for each of these points, and the outcome is evaluated using an EVal Function (which measures the distance between the current and targeted TRTs and

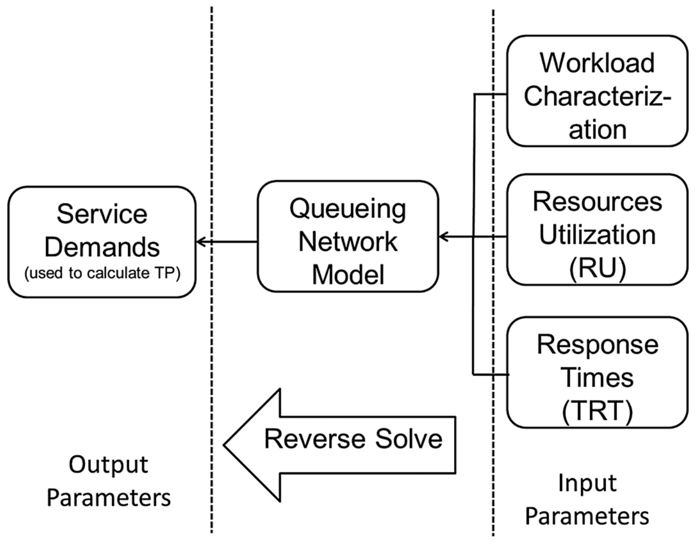

Workload Characteriz-ation
Service Demands (used to calculate TP)
Queueing Network Model
Resources Utilization (RU)
Response Times (TRT)
Reverse Solve
Output Parameters
Input Parameters

Figure 9. Presented approach to infer transaction profile from performance data (transaction response time and resource utilization).

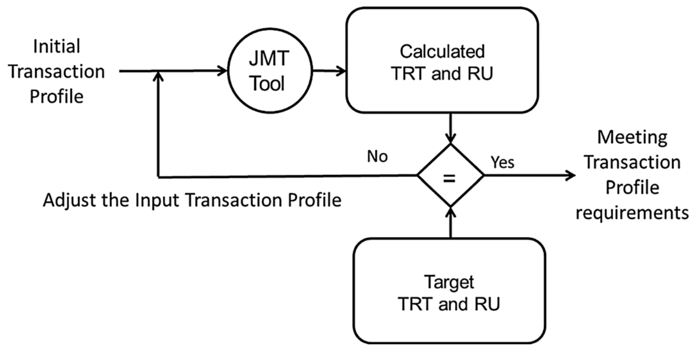

Initial Transaction Profile
JMT Tool
Calculated TRT and RU
No
=
Yes
Meeting Transaction Profile requirements
Adjust the Input Transaction Profile
Target TRT and RU

Figure 10. Applying a search-based technique to reverse-solve the queueing network model. TRT, transaction response time; RU, resource utilization; JMT, Java modelling tool.

RUs), and if the best of these neighbour points is better than the current point, the search will move to that point. The same process is followed for all points in the loop. If the EVal of the proposed solution generated at the end of the loop is lower than an epsilon value, then the entire search process concludes with a suitable solution. Otherwise, if the proposed set of TPs yields a result that is better than the previous one, but not within the target epsilon, then this is set as a new set of starting TPs, and the loop is restarted. If the proposed solution is not better than the current one (local maximum), then the local search iteration concludes, and a new one is triggered with a new set of starting TPs.

The full details of this approach have been described in a generic way to infer TP from TRTs and RUs (for capacity management purposes) in our recent work [33].

This search-based process involves solving the QNM multiple times before converging to the required TP. Hence, the performance of the QNM solving process is a key concern for this approach. For simple deployments, such as the one shown in Figure 6, an efficient analytical solution is affordable. Nevertheless, more complex deployments, mainly those with server clusters, are not readily solvable by such efficient techniques. In the following subsection, we propose a novel approach to allow for efficient solution for server clusters.

Novel approach to model clusters in queueing network model. Clusters of servers are commonly used at each tier of most industrial applications. For example, the application may contain clusters of web servers, application servers and database servers. This is performed to achieve a better performance by utilizing a load balancer to distribute workload between cluster members. In such cases, the TP contains the SDs from any of the cluster members given that in this paper we assume all cluster members are identical.

To explain our approach, we present the system shown in Figure 11. A load balancer software, not shown in the figure, distributes the requests between the two application servers in a cluster. The workload characterization for this system, Figure 11, is shown in Table II. We assume that the workload consists of 20 users distributed equally between the two nodes. It contains two transactions, *login* and *search*, with SDs of 0.05 and 0.06 s, respectively.

In order to simplify the QNM to one as shown in Figure 12, to be solved analytically, multiple transaction groups are introduced; each one corresponds to a member of the application server cluster. Each group contains the same set of the given transactions (i.e. for the search and login transactions we have search1 and login1 in the transaction group1 that corresponds to AppServer1 cluster member and so on). The new workload characterization is shown in Table III.

The total number of users is divided between the various transaction groups, with the SD set to the given value on the corresponding cluster member and zero on the remaining cluster members. For example, search1 has SD of 0.06 on AppServer1 and zero on AppServer2. For the

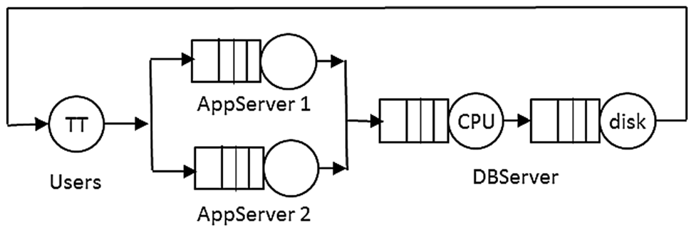


Figure 11. Queueing network model of a computer system with a clustered application server. TT, think time; App, application; DB, database.

Table II. Service demands for system shown in Figure 11.

Transaction	Number of users	Users TT (s)	AppServer1 (s)	AppServer2 (s)	DB server CPU (s)	DB server disk (s)
Login	20	60	0.05	0.05	0.08	0.09
Search	20	60	0.06	0.06	0.08	0.07

TT, think time; App, application; DB, database.

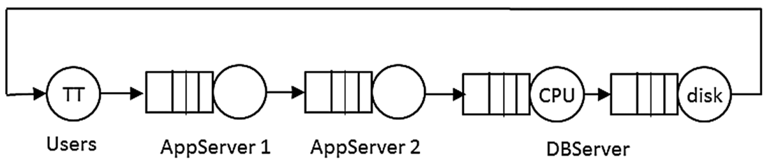


Figure 12. Simplification proposed for Figure 11. TT, think time; App, application; DB, database.

Table III. Proposed service demands for system shown in Figure 12.

Transaction	Number of users	Users TT (s)	AppServer1 (s)	AppServer2 (s)	DB server CPU (s)	DB server disk (s)
Login1	10	60	0.05	0.0	0.08	0.09
Search1	10	60	0.06	0.0	0.08	0.07
Login2	10	60	0.0	0.05	0.08	0.09
Search2	10	60	0.0	0.06	0.08	0.07

TT, think time; App, application; DB, database.

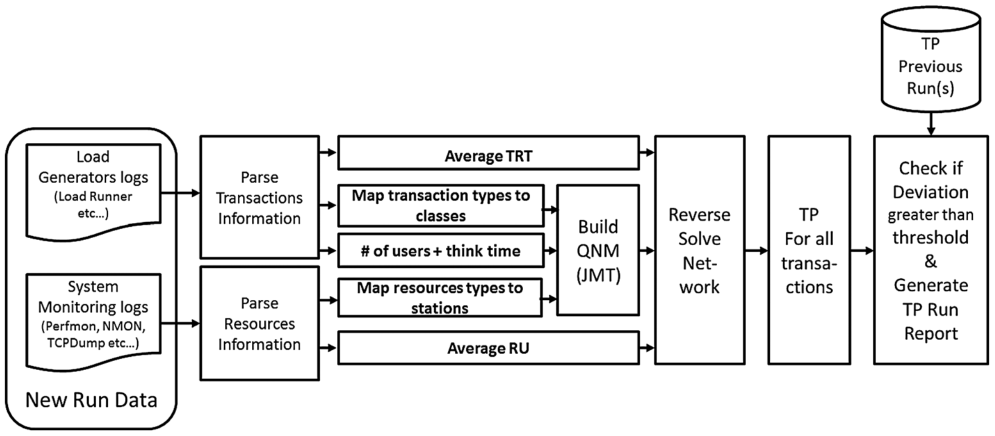

TP Previous Run(s)
Load Generators logs (Load Runner etc...)
System Monitoring logs (Perfmon, NMON, TCPDump etc...)
New Run Data
Parse Transactions Information
Parse Resources Information
Average TRT
Map transaction types to classes
# of users + think time
Map resources types to stations
Average RU
Build QNM (JMT)
Reverse Solve Net-work
TP For all transa-ctions
Check if Deviation greater than threshold & Generate TP Run Report

Figure 13. Outline of the presented approach. TP, transaction profile; TRT, transaction response time; RU, resource utilization; QNM, queueing network model; JMT, Java modelling tool.

remaining stations (users, database server CPU and disk I/O), the SDs remain the same within all transaction groups.

The suggested QNM, shown in Figure 12, accompanied with the workload characterization shown in Table III provides similar results to the QNM shown in Figure 11 when both networks are solved via the JMT (first one analytically and the second one by simulation). The TRT is the average from the various transactions in each group (so TRT of search transaction is the average TRT for search1 and search2). These results are obtained given that all nodes are identical, which is a common assumption in many computer systems. We will ease this condition in our future work.

5.2.2. System overview. Figure 13 shows a high level outline of the presented approach. The data of the new performance run forms the input to the process, it includes:

1. Log files from load generation tools, such as JMeter, which are parsed to obtain the following information:
   (a) Transaction types: The name of the transactions applied during the testing run.
   (b) Number of users issuing each transaction.
   (c) Transaction TT: Time required by the user to review the response and to make a new request.
   (d) TRT for all transactions. The average TRT over the entire run period.

2. Log files from the system monitoring tools (such as Perfmon). One file is available for each server in the system that contains the RU for each resource (mainly CPU and disk I/O). Those are parsed to obtain the following:
    (a) The name of each resource (station) such as database server CPU, DBS and disk I/O.
    (b) The RU for each of the aforementioned resources. The average RU over the entire run period.

After parsing the input log files, the QNM is built (in JMT) where the transaction types and resource types are mapped to classes and stations, respectively. This step removes the overhead of manually building the QNM, which is made possible given the BCMP approximation, which simplifies such QNMs as explained earlier. But even for more complicated systems, where BCMP may not be fully applicable, building and validating the QNM will only need to happen once at the beginning of the first testing cycle and may just need to be updated in the following releases. It is also worth mentioning that QNMs of typical deployment topologies are proposed by capacity management researchers and are applicable to all systems deployed with that topology. This even applies to the more complicated QNMs such as those which are software contention aware for three tier systems such as the one introduced in our recent work [29].

Then, as shown in Figure 9, all of the QNM (built in JMT), the TRT and the RU are used to reverse-solve the model as discussed in Section 5.2.1 to obtain the SDs (used to obtain the TP) for all transactions. The first ever run stops at this point and the generated TPs from the *TP previous run(s)* for the next run.

Then for all subsequent runs, this set of TPs is compared with the *TP previous run(s)* to produce the TP run report shown in Figure 5. It marks the transactions whose TPs are higher than the corresponding TP in the previous runs by the threshold.

All the parts of Figure 13 are automated and require no new procedure from the testing team beyond running the tool and analyzing the TP run report. Our implementation supports JMeter and Perfmon.

6. CASE STUDY

6.1. *Case study design*

We have made some assumptions to support the technique presented in this paper. In this section, we investigate whether, and the extent to which, these assumptions are valid by testing them on two enterprise applications: one open source and one industrial.

In this case study, we plan to verify whether, and the extent to which, the TP

1. is stable under different workload levels? This question is discussed in Section 6.4.1;
2. is stable under application changes that should not affect its performance? We will also determine the thresholds to be used to detect the TP deviations in the last step of Figure 13. This question is discussed in Section 6.4.2;
3. can detect a single application change that affects its performance under the same workload. This simulates the case where the performance testing team need to validate that a proposed functional fix will not cause performance regressions [9]. This is discussed in Section 6.4.3;
4. can detect multiple application changes that affect its performance under the same workload. This question is discussed in Section 6.4.4;
5. can detect multiple application changes that affect its performance, while the workload is set to a different level in the new run. This is a merge of Sections 6.4.1 and 6.4.4 and reflects a more real scenario that the performance team usually face when they need to find performance regressions in a new release with a growing workload level. This question is discussed in Section 6.4.5;
6. can be used with more complex systems, mainly those with clustered servers (under different workload levels)? This is similar to Section 6.4.5 but performed on deployments composed of a three server cluster. This question is discussed in Section 6.4.6.

In addition, we will analyse the data obtained from various kinds of tests using the existing statistical process control (SPC) charts approach presented in Section 6.3.2. We will compare the efficiency of the TP and SPC charts-based solutions and show how both approaches are affected when the workload level is varied between runs.

Section 6.2 introduces the two enterprise applications used in the case study. Then in Section 6.3, we present the evaluation approach. Section 6.4 shows the results of the conducted experiments. Finally, in Section 6.5 we discuss the results.

6.2. *Description of applications*

In our case study, we used the following two enterprise applications:

1. JPetStore 6 application [38], which is an E-Commerce web application that offers various types of pets online. It is a reimplementation of the Sun's java enterprise edition (JEE) reference implementation PetStore. We evaluate our approach by using the set of transactions shown in Table IV.
2. An industrial web 2.0 application provided by our industrial collaborator. It is a social networking application offering various kinds of services such as contacts, groups and discussions. We evaluate our approach by using the set of transactions shown in Table V. This application is designed to serve a large number of users and transactions, both on premises and in the cloud. In this situation, a regressed transaction type can impact on many transaction instances and their users.

Given the small size of the JPetStore application, most of its transaction types were utilized when designing the performance runs scenario. While, in the industrial application that includes many transaction types, we selected a subset of these transaction types from a list of transaction types classified by the industrial collaborator as key transaction types. The selected transaction types cover various categories such as browsing catalogues, searching for items and creating and viewing of items. Each of these categories has different performance characteristics (e.g. the *search for items* transaction type is mainly processed by the application server using a file-based search engine, while the *browsing catalogues* transaction type depends mainly on database queries). Such common characteristics are clearly reflected in the TPs from the same category.

Table IV. JPetStore transactions set.

Transaction name	Description
Sign up	Registers in the application, this requires providing (among other information) the user name and password
Sign in	Signs in to the application using a user name and password and redirects to the home page (a list of all pets' categories, such as dogs and birds) The application is populated with a large number of users and categories
Browse products in a category	Shows all products from a certain category, such as Amazon Parrot and Finch in the birds category The application is populated with a large number of products
Browse items in a product	Shows all of the advertised items in a product The application is populated with a large number of Items
View an item	Shows the selected item info
Add an item to cart	Adds the item to cart
Update item quantity	Updating the quantity of an item in the cart
Checkout items	Checks out all items in the cart Large number of items is added to cart during the run
Open home page	A list of all pets' categories, such as dogs and birds The application is populated with a large number of categories
Search for products	Search for products by product name The application is populated with a large number of products.

Table V. Industrial application transactions set.

Transaction name	Description
Search contacts by name	Performs a search based on someone's name
Search contacts by keyword	Performs a search based on keywords
View contact	Views person contact
Update contact status	Updates your status message
Browse my contacts	Manages your network (contacts, people you are following and are following you)
Browse my groups	Views your groups
Browse public groups	Views browse public groups in same organization to find groups that are related to your knowledge
View group	Shows the description of the group and a summary of the most recently content added to the group
Create a group	Starting a group about your area of interest
Search my groups	Performs a search within your groups
Search public groups	Performs a search within public groups
Followed groups	Accesses the groups you are following
Browse public discussions	Views browse public discussions in same organization to find discussions that are related to your questions
Browse owner discussions	Keeps track of discussions you own
Create discussion	Creates a new discussion to start a discussion
Create topic	Creates a regular topic (question)
View discussion	Opens a discussion and browse its topics
View topic	Opens a topic and view its replies
Follow discussion	You are notified when a discussion is updated
Browse following discussions	Keeps track of the discussions (topics) you are following
The following four transactions represent features added in the new release	
Update contact	Updates user information
Update group	Updates group information
Update discussion	Updates discussion
Update topic	Updates topic

Table VI. System deployment topology for both applications.

Machine	Software Component
Application server	IBM WebSphere application server network deployment 7.0 + JPetStore 6/industrial enterprise application
Database server	IBM DB2 9.7 server
Client machine	JMeter (load generator)

The two enterprise applications earlier (JPetStore 6 and industrial) were each deployed on clusters of three machines as shown in Table VI. The system is a typical three tier, one where the client machine runs the load generator program JMeter. The business tier contains the application server that is an IBM WebSphere application server version 7.0 on top of which the JPetStore 6 or industrial enterprise application is deployed and configured. The back-end tier contains the IBM DB2 database server.

All servers are Microsoft Windows 7 Professional 64 bits. Each has 4 GB of RAM with a 2.66 GHz single processor, except for the database server, which has dual processors. Each server is equipped with a 7200 rpm disk.

To avoid known issues with JMeter when used with large a number of users, we heavily loaded the two systems by reducing the TT to a low value of 1.5 s (typical values are of the order of 30 s). This means that the effective workload applied in our tests is more than 20 times higher than would be expected for the actual applied number of users. For example, when we use 750 users with the industrial application, this is equivalent to the scenario of 15 000 actual users that are using the system.

Table VII. Common performance defects.

Performance defect (bad practice)	Description
No index on key columns that are frequently queried	It is important to place an index to avoid complete table scan for frequently used queries.
Enable logging	Logging statements should only be enabled while debugging problems to collect information for developers. Enabling them will cause performance degradation.
Not limiting the database query	Database queries should not return all rows in the tables; instead, they should limit the number of rows returned to what will be displayed in the user interface.

To demonstrate our approach to detect performance regressions caused by application changes, we injected bad practice defects that are often introduced during the application development process. Table VII describes these defects, which are similar to those used by Nguyen *et al.* in their SPC charts approach [5], which we will be using for comparison. In addition, we consulted a number of performance experts who confirmed that these kinds of defects are among the most common performance defects repeatedly introduced by developers in new releases. Finally, the effect of these defects is spread across the various system servers (application and database) and resources (CPU and disk I/O); this allows exploring the coverage of the TP technique across various system components.

For the JPetStore application, we selected four performance defects from Table VII and introduced each defect into one of the transactions shown in Table IV. We removed the index from database tables *categories* and *products*; this is expected to inject performance defects to the *home* and *list products* transactions. We also enabled logging of the *list items* transaction; this is expected to inject a performance defect in it. We finally removed the limit on the structured query language (SQL) query to make it return all elements in the *carts* table; this is expected to introduce a performance defect in the *checkout* transaction.

Similarly, for the industrial application, we selected 10 performance defects from Table VII and introduced each defect into one of the transactions shown in Table V. We removed the index from database tables used by the *browse my contacts*, *browse my groups*, *browse public discussions* and *browse following discussions* transactions. We also individually enabled logging of the *search contacts by keyword*, *create a group* and *follow a discussion* transactions. Furthermore, we also removed the limit on the SQL query to make it return all elements in the tables used by *view contact*, *browse public groups* and *view a discussion* transactions.

These defects were either injected individually (one per run) or collectively (all in a single run) based on the test type required.

6.3. *Case study evaluation*

We selected one version of each application to be the baseline version with no performance anomalies. Then to verify the various research questions, we generated new versions of each application in which we inject either what is considered as *performance safe* changes or *bad practice* defects. The manual approach used by the industrial collaborator was then used to assess each application and identify which transaction(s), if any, contained a defect. This provided the reference score for each experiment. Then, the TP approach and the SPC charts approach are applied, and their results are evaluated.

We define some quality metrics in the next subsection before providing an overview of the SPC charts approach in the following subsection.

6.3.1. Evaluation metrics. In our tests, we define the precision as follows:

$$Precision = \left(\frac{N}{P}\right) * 100\% \tag{2}$$

N is the number of tests with a defect that has been detected and P is the total number of predicted defects including false positives.

While we define recall as follows:

$$Recall = (\frac{N}{T}) * 100\% \tag{3}$$

T is the total number of tests with a defect whether detected or not.

The precision is the fraction of detected defects that are relevant, while the recall is the fraction of relevant defects that are detected. While F1 is a measure that combines precision and recall where both are evenly weighted.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{4}$$

6.3.2. Statistical process control charts approach. We will compare the results of the TP approach with widely used statistical techniques, which we will explain in this section. Further research work on the area is presented in Section 8.

Statistical techniques were among the first techniques proposed by researchers to detect performance regression anomalies, such as in [5, 13]. They suggest applying SPC charts, which are typically used to monitor the quality of manufacturing processes, to the analysis of performance regression testing data. For each performance counter, TRT or RU, data from a previous set of tests are used as a baseline to draw the upper control limits (UCL) and lower control limits (LCL) and the centre line (CL). Then the new test data is compared with these limits, and a violation ratio is calculated to represent the number of points that exceeds those limits. If the violation ratio is higher than a certain predetermined threshold, an anomaly is declared.

A typical choice for the CL is the median value of all sample points in a counter, such as the TRT, during the test period. The UCL and LCL can be determined in two ways, first using CL plus three standard deviations for UCL and minus three standard deviations for the LCL. The second way is to use the fifth percentile (the counter value below which 5% of samples lie) for the LCL and the 95th percentile (the counter value below which 95% of samples lie) for the UCL. The violation ratio is calculated by testing the new data against these control limits, were the number of sample points that exceed the UCL, or less than the LCL, is divided by the total number of the sample points within the entire testing period for that counter.

The thresholds are also calculated from the baseline test(s). So if five previous datasets are available, then four of them will be used to generate the control limits, as described earlier, and the fifth run is tested against those limits, and the corresponding violation ratio is calculated. The same will be repeated for each of the five runs (the remaining data in each case are used to generate the control limits). The violation ratio to be used with the new test data is then nominated as the maximum violation ratio from all five values found here. If the fifth and 95th percentiles are used to calculate the LCL and UCL, respectively, then the minimum value of the threshold is 10%.

6.4. Experiments

In the JPestStore application, the transaction mix is composed of 10% for signing up and signing in, 25% browsing, 25% searching, 20% viewing (home page and item) and 20% for buying (add to cart, update quantity and checkout). While in the industrial application, for each of the three components (contacts, groups and discussions), the transaction mix is composed of 25% browsing, 25% searching, 20% viewing, 20% creating and 10% for the remaining transactions (the last four transactions in Table V are not applied except in the last two experiments as we will explain later). Each test run lasts for 1 h in the case of the JPetStore application and 3 h for the industrial application.

We performed two runs with 300 and 450 users on the JPetStore and industrial applications, respectively, and set them as baselines. These runs were performed on the standard version of each application. The TPs of each dataset are found using the process shown in Figure 13, by ignoring the last step as no *TP previous run* is available for it. Then TP run reports of all other runs in the succeeding text are generated using the process shown in Figure 13 where the TPs of the baseline run are set as the *TP previous run*. The thresholds are set to zero (in order to record all deviations) in Sections 6.4.1 and 6.4.2 and set to the values found in Section 6.4.2 for the following experiments.

6.4.1. Transaction profile stability under different workload levels. As presented in Section 3, the TP is expected to be less sensitive to the workload (number of users) applied to the application. This is unlike the TRT that varies substantially with the applied workload because of queueing. In this experiment, we show that the TP is more stable with respect to the applied workload, that is, it varies much less than TRT under the same workload changes.

In this experiment, we execute performance tests with multiple workloads, on both applications (JPetStore and industrial). The TRT and RU of the various transactions and resources, respectively, are recorded. The tests are performed with 100, 200, 300, 400 and 500 users for the JPetStore application and with 150, 300, 450, 600 and 750 users for the industrial application. These numbers of users cover the entire range of normal operation for the specified hardware characteristics of these deployments, the TTs and the large amount of populated data. Increasing the workload over these limits, 500 and 750 for the JPetStore and industrial applications, respectively, will result to saturation of the servers RUs, so the TRTs will increase to unacceptable values.

Results. Table VIII shows the TP variations with the change of the workload applied to the JPetStore application. The middle workload, 300 users, is the reference workload, and the deviation of other workloads TPs is calculated from it. It can be noticed that the TP is stable under changing workload levels. Increasing the workload from 300 to 500 (or decreasing it to 100) has a small impact on the TP, which is less than 6% in both directions. In this case, 500 users is the highest number of users based on the capacity of the deployed system so the maximum TP deviation of the system is around 6%.

On the contrary, Table IX shows much larger variations in the TRT as workload changes. It can be noticed that the TRT reaches an increase of around 23% when the workload is increased from 300 to 500 and a decrease of around 24% when the workload is reduced from 300 to 100.

Similarly, Table X supports the TP stability under various workload levels for the industrial application. It can be seen that increasing the workload from 450 to 750 (or decreasing it to 150) has a small impact on the TP, which is less than 6.5%, slightly higher than the JPetStore application, in both directions. For this application, 750 users is the highest number of users based on the capacity of the deployed system so the maximum TP deviation of the system is around 6.5%.

Also from Table XI, it can be noticed that the TRT reaches an increase of around 32% when the workload is increased from 450 to 750 and a decrease of around 29% when the workload is reduced from 450 to 150.

From the aforementioned results on the two applications, it is clear that TP is much less susceptible to workload level variation when compared with the TRT. This supports our expectation that TP is close to a workload-independent representation of the transaction performance characteristics. The small deviations recorded in the TP are explained in details in Section 6.5.

Table VIII. Transaction profile stability between runs each with a different workload for the JPetStore application.

Transaction name	100 users (%)	200 users (%)	300 users (%)	400 users (%)	500 users (%)
Sign up	−4.8	−2.3	0.0	1.1	5.1
Sign in	−2.5	−0.4	0.0	2.9	4.6
List products	−5.0	−1.3	0.0	2.1	4.9
List items	−3.3	−2.4	0.0	2.5	4.9
Open item	−5.7	−2.0	0.0	1.3	3.3
Add to cart	−2.9	−1.5	0.0	3.5	4.6
Update item quantity	−3.9	−2.4	0.0	2.9	3.9
Checkout	−2.7	−0.9	0.0	4.1	5.8
Home	−4.2	−2.4	0.0	1.0	3.8
Search for products	−5.4	−2.6	0.0	1.4	3.8

Table IX. Transaction response time changes between runs each with a different workload for the JPetStore application.

Transaction name	100 users (%)	200 users (%)	300 users (%)	400 users (%)	500 users (%)
Sign up	−18.9	−11.1	0.0	9.9	22.2
Sign in	−13.9	−7.0	0.0	11.1	19.8
List products	−20.2	−9.6	0.0	10.4	19.1
List items	−17.4	−12.7	0.0	10.5	17.7
Open item	−22.6	−16.8	0.0	9.8	15.6
Add to cart	−12.7	−8.2	0.0	13.4	19.6
Update item quantity	−14.7	−10.3	0.0	10.8	16.0
Checkout	−11.2	−6.3	0.0	15.0	23.8
Home	−15.9	−9.9	0.0	9.2	17.7
Search for products	−20.6	−12.2	0.0	8.9	15.9

Table X. Transaction profile stability between runs each with a different workload for the industrial application.

Transaction name	150 users (%)	300 users (%)	450 users (%)	600 users (%)	750 users (%)
Search contacts by name	−4.9	−2.6	0.0	4.4	5.4
Search contacts by keyword	−5.9	−3.8	0.0	1.5	2.7
View contact	−4.0	−3.3	0.0	0.5	3.9
Update contact status	−2.8	−2.1	0.0	1.6	6.3
Browse my contacts	−3.9	−2.5	0.0	1.8	5.1
Browse my groups	−3.1	−2.0	0.0	2.8	5.9
Browse public groups	−4.2	−2.1	0.0	1.7	6.4
View group	−5.4	−3.2	0.0	2.1	3.8
Create a group	−5.1	−1.1	0.0	0.6	3.8
Browse my groups	−3.4	−1.9	0.0	1.2	4.7
Search public groups	−3.7	−2.0	0.0	2.5	4.9
Followed groups	−3.8	−0.9	0.0	2.5	5.2
Browse public discussions	−5.4	−2.2	0.0	1.0	4.3
Browse owner discussions	−5.1	−2.5	0.0	1.9	5.0
Create discussion	−3.4	−0.9	0.0	3.4	5.1
Create topic	−3.7	−3.1	0.0	1.0	3.7
View discussion	−4.1	−1.9	0.0	2.5	4.5
View topic	−4.7	−1.9	0.0	2.4	5.8
Follow a discussion	−2.9	−1.5	0.0	1.8	5.1
Browse following discussions	−4.0	−2.6	0.0	2.8	4.6

6.4.2. *Transaction profile stability with performance-safe application changes.*

The other hypothesis we made in Section 3 is that the TP provides a good representation of the transaction performance characteristics. We claimed that the TP changes only if the application changes in a way that affects its performance. In this section, we investigate whether, and the extent to which, the TP is immune to application changes that are not supposed to affect its performance, that is, performance-safe changes. In addition, we will find the appropriate threshold for each transaction to be used in the last step in Figure 13 when we attempt to detect the actual performance defects in the coming subsections.

We performed eight test runs on various versions of each application that are known to have modifications that should not affect its performance. These changes include modifications of the data presentation in the user interface, translations and changes to logging messages. Four out of the eight tests were conducted with the reference workload (300 and 450 for the JPetStore and industrial applications, respectively), and the other four were made with other selected load levels. Then we used the process depicted in Figure 13 to calculate the TP deviations for all transactions in each test. The threshold is set to zero to make sure all deviations are recorded.

The appropriate thresholds, for each transaction, to be used when we detect actual performance regression anomalies using the TP approach in the coming experiments are nominated as the

Table XI. Transaction response time changes between runs each with a different workload for the industrial application.

Transaction name	150 users (%)	300 users (%)	450 users (%)	600 users (%)	750 users (%)
Search contacts by name	−22.6	−11.2	0.0	17.4	24.1
Search contacts by keyword	−25.8	−13.3	0.0	11.1	17.7
View contact	−19.7	−11.4	0.0	10.1	21.0
Update contact status	−17.3	−8.5	0.0	9.6	29.2
Browse my contacts	−18.1	−10.1	0.0	12.6	25.2
Browse my groups	−17.6	−12.3	0.0	15.3	27.7
Browse public groups	−22.9	−13.8	0.0	12.2	23.1
View group	−27.4	−18.0	0.0	15.9	29.3
Create a group	−23.4	−11.2	0.0	8.9	18.2
Browse my groups	−19.7	−9.9	0.0	12.4	23.7
Search public groups	−16.5	−11.1	0.0	14.7	22.7
Followed groups	−17.7	−8.7	0.0	15.5	30.2
Browse public discussions	−28.9	−13.2	0.0	12.1	24.3
Browse owner discussions	−25.2	−15.9	0.0	14.2	25.2
Create discussion	−17.2	−6.8	0.0	18.4	31.4
Create topic	−18.2	−11.2	0.0	10.6	19.0
View discussion	−22.9	−14.1	0.0	13.9	23.1
View topic	−23.1	−12.9	0.0	16.5	27.8
Follow a discussion	−18.6	−9.5	0.0	14.8	25.8
Browse following discussions	−22.9	−12.0	0.0	13.4	22.2

Table XII. TP deviation when applying performance safe changes to the JPetStore application (used to determine threshold).

Transaction name	Versions with performance safe updates (TP deviation (%) from baseline build)								Threshold	
									TP	SPC
Sign up	1.4	0.9	2.0	0.3	0.2	0.9	4.0	0.1	3.3	11.2
Sign in	0.2	2.0	0.7	1.2	3.8	2.0	4.8	0.9	4.4	13.3
List products	1.0	0.5	1.3	2.1	3.0	2.8	4.4	0.7	3.9	13.8
List items	2.2	2.3	0.8	1.1	1.2	1.5	3.7	0.3	3.2	8.7
Open item	2.6	0.5	1.6	1.5	2.2	1.8	4.3	0.2	3.7	11.9
Add to cart	2.6	0.5	0.4	1.5	1.9	0.7	3.8	0.6	3.4	14.1
Update item quantity	0.8	2.7	1.1	0.0	1.3	3.8	3.6	0.3	3.7	11.1
Checkout	0.5	2.2	3.2	3.1	3.1	0.7	5.5	1.5	4.7	22.1
Home	1.5	3.6	4.2	0.0	1.2	1.4	3.8	1.1	4.1	17.2
Search for products	0.5	2.0	2.7	1.2	0.7	1.1	4.6	1.8	3.9	13.4

TP, transaction profile; SPC, statistical process control.

95th percentile. This means that our system accepts 5% as false positives rate when detecting TP deviations. This percentile value is chosen based on the common practice in the industry.

In addition, these runs are used to calculate the thresholds to be used with the SPC charts technique presented in Section 6.3.2. We also used the fifth and 95th percentiles to find the LCL and UCL values, respectively, based on the case study in the paper [5], which aligns with our choice for the TP thresholds.

Results. The TP deviations found from the workload runs on the JPetStore application versions with performance-safe changes are shown in Table XII. Each column corresponds to a test performed on a version of JPetStore with a performance-safe change, the TP deviation percentage is calculated against the baseline dataset (no defects injected) TPs. We see that the TP deviates with values up to 5.5%. All these variations are false positives and are caused by the inaccuracy of the technique that is caused by factors such as software contention, network delays SDs and the BCMP approximation.

We used these deviations to calculate the threshold for each transaction that separates the actual defects from those caused by the inaccuracies of the system. To do so, we calculate the 95th

Table XIII. TP deviation when applying performance safe changes to the industrial application (used to determine threshold).

Transaction name	Versions with performance safe updates (TP Deviation (%) from baseline build)								Threshold	
									TP	SPC
Search contacts by name	2.1	4.1	2.9	4.1	0.5	3.4	3.3	2.6	4.1	17.1
Search contacts by keyword	1.0	3.3	3.3	1.8	1.4	2.3	2.7	1.5	3.3	8.9
View contact	1.9	3.8	2.5	2.5	2.3	1.2	3.5	1.3	3.7	11.4
Update contact status	0.6	3.9	3.0	2.4	1.2	0.2	3.5	0.1	3.8	13.6
Browse my contacts	2.9	4.3	3.4	3.8	3.0	3.2	3.0	4.7	4.6	23.1
Browse my groups	2.0	4.4	3.9	3.2	1.1	0.8	3.6	2.7	4.2	17.3
Browse public groups	3.9	4.5	3.0	2.9	4.1	1.3	2.7	3.0	4.4	18.7
View group	1.5	3.7	2.6	1.6	2.1	1.7	2.6	2.1	3.3	14.5
Create a group	3.7	2.8	2.5	1.8	0.4	4.0	2.6	4.3	4.2	21.1
Browse my groups	3.5	3.3	2.8	2.0	0.5	0.5	2.8	1.7	3.4	17.0
Search public groups	2.0	3.0	3.4	2.7	0.5	1.9	3.2	1.9	3.3	14.9
Followed groups	0.3	4.3	4.0	2.7	0.3	3.6	3.9	1.9	4.2	22.3
Browse public discussions	0.6	3.3	1.7	1.5	1.5	0.0	2.4	3.5	3.4	16.1
Browse owner discussions	0.2	3.4	2.6	3.2	2.6	2.3	3.4	1.3	3.4	16.9
Create discussion	2.0	4.0	3.9	3.3	3.3	4.5	3.4	2.7	4.3	20.9
Create topic	1.4	3.5	2.2	2.3	1.3	1.5	2.2	0.4	3.1	9.7
View discussion	2.7	2.9	2.5	2.4	0.6	0.9	3.8	4.7	4.4	19.9
View topic	2.8	4.2	2.9	2.1	1.6	4.7	3.4	2.1	4.5	24.8
Follow a discussion	0.9	4.7	2.9	3.5	1.4	2.9	3.5	1.4	4.3	22.5
Browse following discussions	3.7	4.3	4.2	2.7	0.4	4.3	3.8	3.9	4.3	23.3

TP, transaction profile; SPC, statistical process control.

percentile from all deviations in each row of Table XII. These thresholds (along with the corresponding thresholds from the SPC charts-based technique) are shown in the last two columns of the same table and will be used to detect actual anomalies in the next sections.

Similarly, the TP deviations found from the workload runs on the industrial application versions with performance-safe changes are shown in Table XIII. Each column corresponds to a run performed on a version of the industrial application with a performance-safe change, the TP deviation percentage is calculated in the same manner as the JPetStore application. We see that the TP deviates with values up to 4.7%. All these variations are false positives and caused by the inaccuracy described in the JPetStore section earlier. The thresholds for both the TP and SPC charts techniques are also calculated in a similar fashion to the JPetStore application. These thresholds are shown in the last two columns of the same table and will be used to detect actual anomalies in the next sections.

From the results, on both applications, we conclude that the TPs is insensitive to performance-safe application changes.

6.4.3. Using transaction profile to detect a single performance defect. In this part, we conducted tests on both web 2.0 applications (JPetStore and industrial) to verify whether, and the extent to which, the TP run report is effective in detecting defects caused by modifications to the transactions code that negatively affect their performance.

We performed multiple test runs, on both applications, to compare each of them with the baseline dataset. In each run, we injected a performance defect in one transaction, then we generated the TP run report and observed that the defect was detected by the report because of an increase in the corresponding transaction TP. Moreover, we investigated the detailed TP graph to see if the changes in the SDs on the various hardware resources can be justified by the nature of the defect injected. Then we removed the injected defect and introduced another one, in a different transaction, and performed a new run. All runs, including the baseline run, were performed with the reference workload, 300 users for the JPetStore and 450 for the industrial Application. We will ease this condition in Sections 6.4.5 and 6.4.6 in the succeeding text.

In total, we created four versions of the JPetStore application each with one of the defects described earlier and performed four test runs. All runs were conducted with 300 users, and the thresholds, which have been determined in Section 6.4.2, are used when generating the TP run report.

Similarly, we created 10 versions of the industrial application each with one of the defects described earlier and consequently performed 10 test runs. All runs were conducted with 450 users, and the thresholds, which have been determined in Section 6.4.2, are used when generating the TP run report.

All data generated from each test run, in addition to the baseline run, are analysed using both the TP-based approach, presented in this paper, and the SPC charts-based approach by Nguyen *et al.* [5] described in Section 6.3.2.

Results. After running the tests on the JPetStore application and analysing the corresponding TP run reports, we found that three of the four injected defects were detected, and the other one was not detected and is therefore a false negative. This gives a recall of 75%, and because all defects detected were genuine ones, the precision is 100%. No false positives were detected in any of the four runs. The details of the data for those runs are shown in Table XIV.

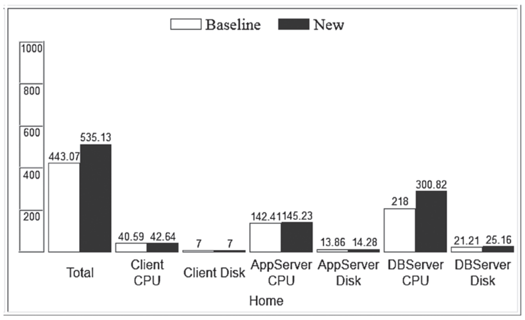
In the first two cases of Table XIV, both defects were detected. We obtained a clear increase in the corresponding TPs, which reached to 20.8% in the case of home transaction and around 15.6% in the list products transaction. Figure 14(a) shows the detailed TP graph for the home transaction. Investigating this detailed TP graph, it is clear that the increase on the database server resources has a significant impact on the TP. Unexpectedly, most of the increase were in the CPU resource and a small percentage caused by the disk I/O. This could be a result of the large RAM on the database server machine, which means that most of the database are already loaded into memory.

In the next test, where logging was enabled, the defect was not detected by the TP run report for the list items transaction. Although it is generally expected that enabling logging will have an

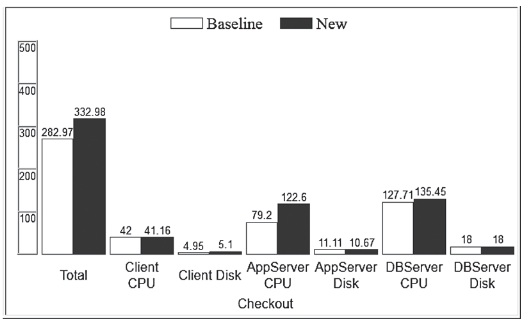
Table XIV. Defects injected ‘individually’ (one per run) in various transactions in the JPetStore application (at same workload level) and the resulting TP deviation and SPC violation ratio.

Run	Transaction	Injected anomaly	TP Deviation (from baseline)	SPC violation ratio
1	Home	No index on corresponding	20.8%	33.9%
2	List products	table’s key columns	15.6%	Not caught
3	List items	Enabling transaction logging	Not caught	14.4%
4	Checkout	Not limiting the database query	17.7%	33.2%

TP, transaction profile; SPC, statistical process control.



(a) the “Home” Transaction with Index Removed.



(b) the “Checkout” Transaction with No Limit on the SQL Query.

Figure 14. Detailed transaction profile graph of JPetStore (single defect) for (a) the ‘Home’ transaction with index removed and (b) the ‘checkout’ transaction with no limit on the SQL query.

impact on performance, the limited number of logging statements in this application does not have a major impact on the transaction performance on such powerful machines.

In the last transaction, the defect was detected by the TP run report. The increase was 17.7%. The detailed TP graph for the checkout transaction is shown in Figure 14(b), which shows that the application server CPU demand has increased by 55%. This information can be passed back to the development team to help them find the cause of the regression.

Analysing the same data using the SPC charts technique, we found that three of the four injected defects were detected, which is the same as the TP approach.

Similarly, after running the tests on the industrial application and analysing the corresponding TP run reports, we found that 9 of the 10 injected defects were detected (the other one represents a false negative), which gives a recall of 90% and precision of 100%. No false positives were detected in any of the 10 runs. The details of the data are shown in Table XV.

In the first four cases of Table XV, all four defects were detected. In each case, there is a clear increase in the corresponding TPs, which reached to 39.5% in the case of browse my contacts transaction and 15%, 19.3% and 36.2% in the browse my groups, browse public discussions and browse following discussions transactions, respectively. Figure 15 shows the detailed TP graph for the browse my contacts transaction. Investigating this detailed TP graph, it is clear that the increase on the database server resources is the main cause of the change in TP. The change is distributed between the CPU and disk I/O resources. This is unlike the JPetStore application where most of the change was due to CPU. The industrial application has much more data in the database, and it is not

Table XV. Defects injected ‘individually’ (one per run) in various transactions in the industrial application (at same workload level) and the resulting TP deviation and SPC violation ratio.

Run	Transaction	Injected anomaly	TP Deviation (from baseline)	SPC violation ratio
1	Browse my contacts	No index on corresponding table’s key columns	39.5%	34.1%
2	Browse my groups		15.0%	22.1%
3	Browse public discussions		19.3%	18.2%
4	Browse following discussions		36.2%	25.5%
5	Search contacts by keyword	Enabling logging on transaction code	Not caught	14.4%
6	Create a group		16.6%	Not caught
7	Follow a discussion		28.1%	26.7%
8	View contact	Not limiting the database query	24.8%	13.4%
9	Browse public groups		11.8%	Not caught
10	View discussion		25.1%	29.4%

TP, transaction profile; SPC, statistical process control.

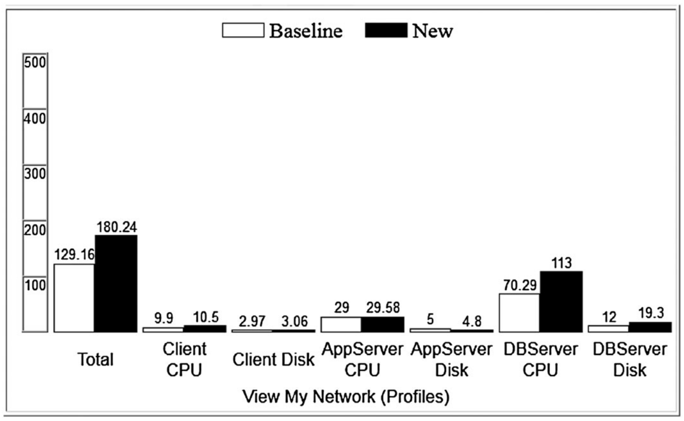


Figure 15. Detailed transaction profile graph for the ‘browse my contacts’ transaction of industrial application with index removed (single defect).

expected to have the entire database loaded in the memory, and consequently, the visits to the disk is expected to increase as well as the CPU usage.

In the following three transactions, where logging was enabled, two defects were detected with a deviation reached to 28.1% for the follow a discussion transaction and 16.6% for the create a group transaction. The application server CPU has the largest impact followed by the disk I/O on the same server. The TP run report failed to detect the defect in the search contacts by keyword transaction. Looking at the corresponding detailed TP graph, we can see that a reduction of the database server CPU demands compensated for the increase of the application server CPU caused by enabling the logging.

In the last three runs, all three injected defects were detected by the TP run report with a TP deviation of 24.8%, 11.8% and 25.1% for the view contact, browse public groups and browse public discussions transactions, respectively. Again, the application server CPU demand has been substantially increased by over 35% in all of them.

Analysing the same data using the SPC charts technique, we found that 8 of the 10 injected defects were detected (the other two represent false negatives), which gives a recall of 80% and a precision of 100%. These are slightly lower than the corresponding TP-based approach values.

The results from both applications show that the TP run report is capable of detecting performance regression anomalies caused by individual software updates. This scenario is a common one when the performance testing team need to evaluate if a proposed functional fix will negatively impact transactions' performance [9].

6.4.4. Detecting multiple defects under the same workload level. In this part, we conducted tests on both web 2.0 applications (JPetStore and industrial) to verify whether, and the extent to which, the TP run report is effective in detecting multiple defects caused by multiple modifications to the transactions code that negatively affect their performance.

We performed a single run on each application to compare with the baseline run. In each run, we injected multiple performance defects in multiple transactions, one per transaction, then we generated the TP run report as shown in Figure 13 and ensured that the defects were detected by the report because of an increase in the corresponding transactions TP. Both runs were executed with the same workload as the baseline runs (300 users for the JPetStore and 450 for the industrial application).

The same defects were introduced to the same transactions as described in Section 6.4.3, but in this case, they were introduced together (i.e. one version of each application with all defects was built and deployed). One test run is performed on each application (JPetStore and industrial) with 4 and 10 defects, respectively. The thresholds, which have been determined in Section 6.4.2, are used when generating the TP run report.

Results. After running this test on the JPetStore application and analysing the corresponding TP run report, we found that five defects were detected. Four of them correspond to the four defects we injected and one false positive, which gives a precision of 80%. All injected defects were detected, and this gives a recall of 100%. While, using the SPC charts approach, we found that four defects

Table XVI. Defects injected 'collectively' (all in a single run) in various transactions in the JPetStore application (at same workload level) and the resulting TP deviation and SPC violation ratio.

Transaction	Injected anomaly	TP deviation (from baseline) (%)	SPC violation ratio
Home	No index on corresponding table's key columns	24.4	35.1%
List products		20.8	14.1%
List items	Enabling transaction logging	8.1	Not caught
Checkout	Not limiting the database query	18.8	31.3%
Open item	No bug introduced	11.9*	—
Update item quantity		—	16.1%*

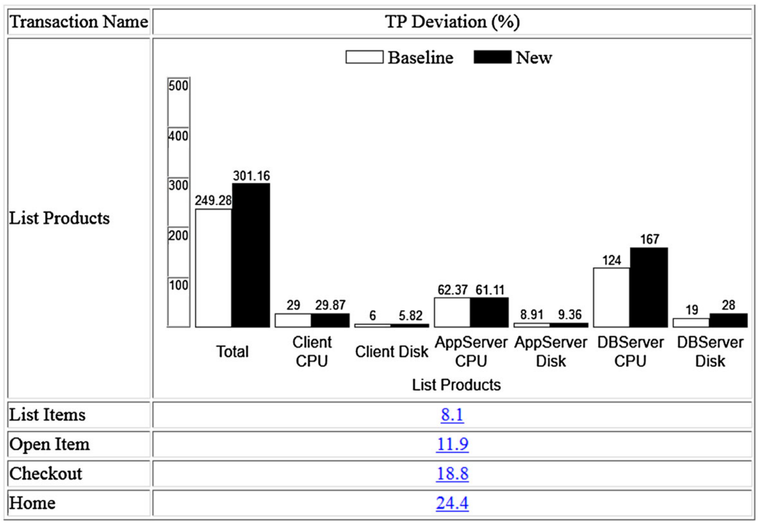
TP, transaction profile; SPC, statistical process control.
*False positive.

were detected. Because three of them correspond to injected defects, there is one false positive and one false negative, which give a precision and a recall of 75%.

The details of this run are shown in Table XVI (only showing transactions with TP deviation (or SPC violation ratio) exceeding the threshold). The entire TP run report is shown in Figure 16.

Similarly, for the industrial application, we found that 11 defects were detected. Of these, nine of them correspond to defects that we injected with one false negative and two false positives, which give a precision of 82% and a recall of 90%. While, using the SPC charts approach, we found that 12 defects were detected. Of these nine correspond to injected defects, with three false positives and one false negative, which give a precision of 75% and a recall of 90%.

The details of this run are shown in Table XVII (only showing transactions with TP deviation (or SPC violation ratio) exceeding the threshold). The entire TP run report is seen in Figure 17, which includes the detailed histogram for browse public discussions, which shows a 35% increase in database server disk I/O access time and a 31% increase in activity in the database server CPU to process this data.



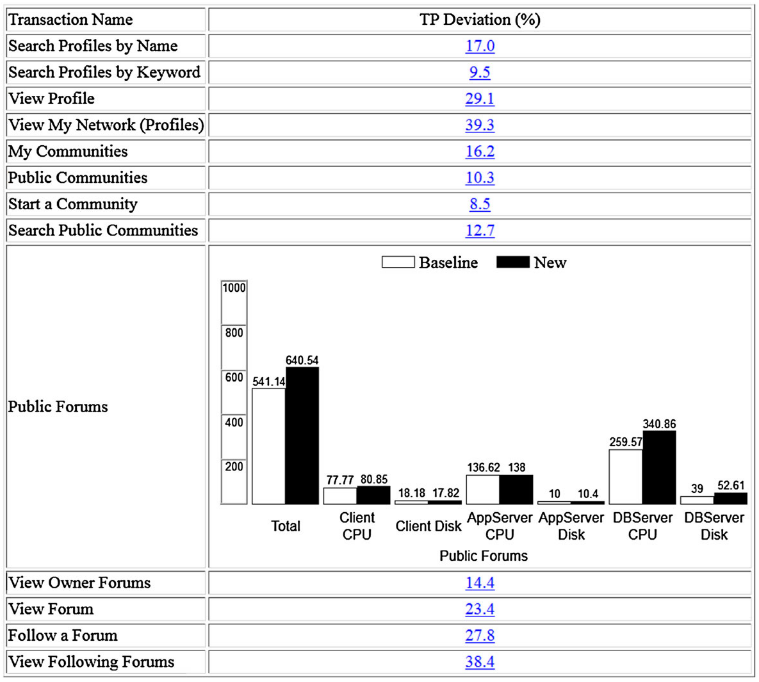
Transaction Name	TP Deviation (%)
List Products	
List Items	8.1
Open Item	11.9
Checkout	18.8
Home	24.4

Figure 16. Transaction profile (TP) run report for JPetStore application with multiple defects. App, application; DB, database.

Table XVII. Defects injected ‘collectively’ (all in a single run) in various transactions in the industrial application (at same workload level) and the resulting TP deviation and SPC violation ratio.

Transaction	Injected anomaly	TP deviation (from baseline)	SPC violation ratio
Browse my contacts	No index on corresponding table’s key columns	39.3%	34.8%
Browse my groups		13.4%	19.7%
Browse public discussions		16.2%	22.2%
Browse following discussions		38.0%	27.0%
Search contacts by keyword	Enabling logging on transaction code	Not caught	14.9%
Create a group		8.4%	Not caught
Follow a discussion		32.4%	27.3%
View contact	Not limiting the database query	25.8%	14.8%
Browse public groups		13.4%	20.1%
View discussion		24.8%	27.7%
Search public groups	No bug introduced	12.3%*	—
Browse owner discussions		17.9%*	19.8%*
Followed groups		—	29.9%*
View topic		—	26.6%*

TP, transaction profile; SPC, statistical process control.
*False positive.



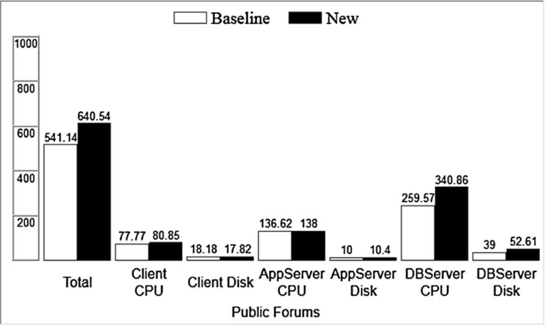
Transaction Name	TP Deviation (%)
Search Profiles by Name	17.0
Search Profiles by Keyword	9.5
View Profile	29.1
View My Network (Profiles)	39.3
My Communities	16.2
Public Communities	10.3
Start a Community	8.5
Search Public Communities	12.7
Public Forums	
View Owner Forums	14.4
View Forum	23.4
Follow a Forum	27.8
View Following Forums	38.4

Figure 17. Transaction profile (TP) run report for industrial application with multiple defects. App, application; DB, database.

The results from both applications show that the technique is also capable of detecting multiple simultaneous defects with a high precision and a reasonable recall.

6.4.5. Detecting multiple defects under a new workload level. In this part, we extend the previous tests in Section 6.4.4 to study the effect of performing the baseline and the test runs with a different workload. We combine tests from both sections (6.4.1 and 6.4.4) to find the overall effect of changing the workload level and at the same time changing the application in a manner that affects its performance. This scenario emulates a real performance regression test as the performance testing team is likely to need to perform a run on a new version of the application, that contains multiple changes, with an increased workload to cater for new field-like requirements.

We performed a single run on each application to compare with the baseline run. In these runs, we injected multiple performance defects in multiple transactions, one per transaction. The details of these test runs, including the injected defects, are the same as the ones described in Section 6.4.4, except for the following changes:

For the JPetStore application, the workload was modified by increasing both the number of users to 500, and the transaction mix was adjusted to be 15% for signing up and signing in, 15% browsing, 20% searching, 25% viewing (home page and item) and 25% for buying (add to cart, update quantity and checkout). No new transaction types were added to the mix.

While, for the industrial application, the workload was modified by increasing both the number of users to 750, and the transaction mix, for each of the three components (contacts, groups and discussions), was adjusted to be 20% browsing, 20% searching, 15% viewing, 15% creating, 20% updating (new functionality, last four transactions in Table V, added in this release) and 10% for the remaining transactions.

Results. After running this test on the JPetStore application and analysing the corresponding TP run reports, we found that five defects were detected. Because four of them correspond to the four defects, we injected plus one false positive, the precision is 80%, and the recall is 100%. While, using the SPC charts approach, we found that nine defects were detected. Again, all four of the injected defects were detected to give a recall of 100%, but there were five false positive, which gives a precision of 44%, which is explained in the discussion section in the succeeding text. The details of this run are shown in Table XVIII (only showing transactions with TP deviation (or SPC violation ratio) exceeding the threshold).

Similarly, for the industrial application, we found that 12 defects were detected using the new technique. This included all 10 defects that had been injected plus two false positives, which give a precision of 83% and a recall of 100%. While, using the SPC charts approach, we found that 18 defects were detected. Again, all 10 of the injected defects were detected, but eight false positives were also flagged, which gives a precision of 56% and a recall of 100%. The details of this run are shown in Table XIX (only showing transactions with TP deviation exceeding the threshold). Although the new transactions introduced in the new release (i.e. the last four transactions in Table V) are not tested for regressions in this new release as they do not have a previous baseline value, their existence is significant to the performance data analysis looking for regressions in old transactions as they have demands on various resources, which compete with other transactions, this

Table XVIII. Defects injected 'collectively' (all in a single run) in various transactions in the JPetStore application (at a new workload) and the resulting TP deviation and SPC violation ratio.

Transaction	Injected anomaly	TP deviation (from baseline) (%)	SPC violation ratio (%)
Home	No index on corresponding table's key columns	22.3	54.2
List products		23.8	28.1
List items	Enabling transaction logging	9.1	34.2
Checkout	Not limiting the database query	20.8	66.8
Open item	No bug introduced	12.4*	33.1*
Sign up		—	19.4*
Sign in		—	44.4*
Update item quantity		—	28.1*
Search for products		—	29.9*

TP, transaction profile; SPC, statistical process control.
*False positive.

Table XIX. Defects injected 'collectively' (all in a single run) in various transactions in the industrial application (at a new workload level) and the resulting TP deviation and SPC violation ratio.

Transaction	Injected anomaly	TP deviation (from baseline) (%)	SPC violation ratio (%)
Browse my contacts	No index on corresponding table's key columns	35.0	66.7
Browse my groups		12.0	45.2
Browse public discussions		20.4	33.4
Browse following discussions		42.3	67.7
Search contacts by keyword	Enabling logging on transaction code	6.3	55.4
Create a group		16.3	54.5
Follow a discussion		25.7	39.2
View contact	Not limiting the database query	27.3	32.1
Browse public groups		8.5	29.2
View discussion		27.1	55.5
Search contacts by name	No bug introduced	15.5*	43.3*
Browse owner discussions		19.2*	44.0*
Update contact status		—	21.1*
Followed groups		—	49.1*
View topic		—	43.6*
Create discussion		—	27.7*
View group		—	24.0*
Browse my groups		—	32.1*

TP, transaction profile; SPC, statistical process control.
*False positive.

Table XX. Defects injected 'collectively' (all in a single run) in various transactions in the JPetStore application (at a new workload level) with three application server cluster and the resulting TP deviation and SPC violation ratio.

Transaction	Injected anomaly	TP deviation (from baseline)	SPC violation ratio
Home	No index on corresponding table's key columns	22.5%	49.2%
List products		22.3%	25.5%
List items	Enabling transaction logging	Not caught	Not caught
Checkout	Not limiting the database query	12.9%	56.9%
Search for products	No bug introduced	—	30.1%*
Add to cart		—	19.2%*
Sign in		—	36.3%*
Update item quantity		—	27.9%*
Open item		—	29.0%*

TP, transaction profile; SPC, statistical process control.
*False positive.

results in longer queues and TRTs. Nevertheless, such new transactions are usually compared with projected baseline values (often from an SLAs) as part of the analysis of the PPTR (Section 2).

The results from both applications show that the technique is also capable of detecting multiple simultaneous defects with a high precision and a moderate recall under a higher workload in the new run.

6.4.6. Detecting multiple defects in an application server cluster under a new workload level. In this section, we do similar experiments to those in Section 6.4.5 earlier. The only difference is that we modified the deployment topology, shown in Table VI, so that it contains a three application server cluster. All cluster members have similar hardware specifications and contain the same software, which means the SD is the same across all the cluster members. No modifications were made to the client and database server machines. All other details of the runs are same as in Section 6.4.5 earlier.

The goal here is to investigate whether, and the extent to which, the technique presented in this paper can be used with more complex environments, mainly those with clustered servers. The approximation mentioned in Section 5.2.1 allows for efficient analytical solution of such QNMs leading to a solution time comparable with the normal deployments without clustering.

Results. After running this test on the JPetStore application and analysing the corresponding TP run reports, we found that four defects were detected. This included three of the four defects we injected plus one false positive and one false negative, which give a precision and recall of 75%. While, the SPC charts precision is 38%, and the recall is 75%. The details of this run are shown in Table XX (only showing transactions with TP deviation (or SPC violation ratio) exceeding the threshold).

Similarly, for the industrial application, we found that 13 defects were detected. In this case, 10 of them correspond to the 10 defects we injected, which give a precision of 77% and a recall of 100%. While, the SPC charts precision is 53%, and the recall is 100% The details of this run are shown in Table XXI (only showing transactions with TP deviation (or SPC violation ratio) exceeding the threshold).

It is interesting to note that the time to reverse-solve these test cases was less than 15% more than the previous tests (details in the discussion section in the succeeding text). These results, from both applications, suggest that the technique is still suitable for the more complicated deployments.

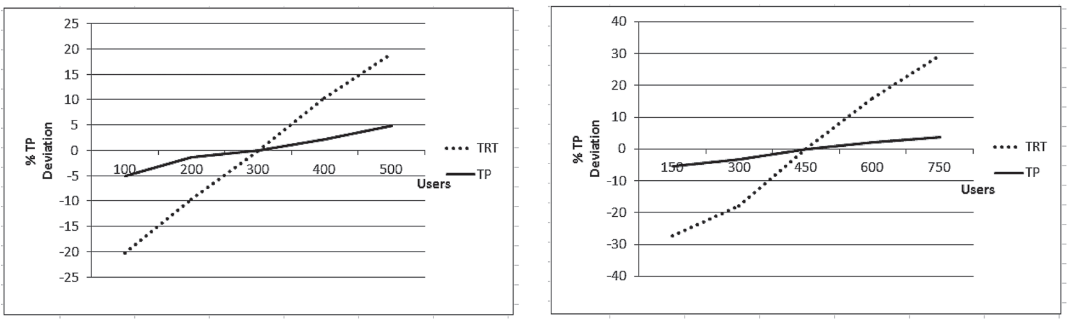
6.5. Case study discussion

The results of the first experiment, Section 6.4.1, show that TP is much more stable than TRT for both applications. This supports our expectations that the TP can be used as a workload-independent measure of application performance. Having said that, Tables VIII and X show that the TP slightly increases with increasing the workload. This is also evident in Figure 18(a) and (b).

Table XXI. Defects injected 'collectively' (all in a single run) in various transactions in the industrial application (at a new workload level) with three application server cluster and the resulting TP deviation and SPC violation ratio.

Transaction	Injected Anomaly	TP deviation (from baseline) (%)	SPC violation ratio (%)
Browse my contacts	No index on corresponding table's key columns	39.3	57.1
Browse my groups		16.2	44.4
Browse public discussions		18.4	35.7
Browse following discussions		38.4	60.0
Search contacts by keyword	Enabling logging on transaction code	9.5	51.1
Create a group		8.5	49.8
Follow a discussion		27.8	41.1
View contact	Not limiting the database query	29.1	32.1
Browse public groups		10.3	25.5
View discussion		23.4	43.3
Search contacts by name	No bug introduced	17.0*	42.9*
Search public groups		12.7*	23.9*
Browse owner discussions		14.4*	37.6*
Update contact status		—	20.3*
Followed groups		—	44.9*
View topic		—	33.8*
Create discussion		—	28.9*
View group		—	24.9*
Browse my groups		—	34.5*

TP, transaction profile; SPC, statistical process control.
*False positive.



(a) "List Products" of JPetStore Application. (b) "View Group" of Industrial Application.

Figure 18. Transaction profile (TP) and transaction response time (TRT) changes with workload for the transaction.

We noticed that measuring the TP with a single user test, as shown in Section 5.1, provides a slightly lower value compared with when it is calculated using performance testing results. This difference increases slightly as the workload used by the performance tests increases.

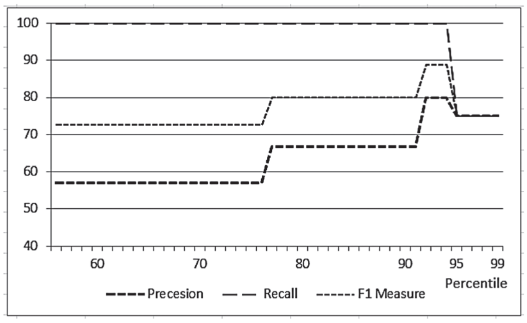
When this single user TP is used in the capacity management process depicted in Figure 7, the resulting TRT is approximately 15% lower than the value measured in workload testing. At the same time, the other parameters such as RU and throughput show less than 5% difference. These findings are supported by the prior work by Kounev *et al.* [26]. This can be explained mainly by the software contention effect, such as the number of threads and database connections, which is not modelled by our hardware resources QNM so that the predicted TRTs is lower than that which is measured. This difference in TRT increases as the workload increases because of the increased competition for software resources, while the effect of the software contention has a smaller impact on the RU and throughput.

It was also noticed that using the measured TRT to calculate the TP as shown in Figures 9 and 10 yields a slightly higher TP compared with the TP measured by a single user test. This explains the behaviour of the slight increase of TP with workload shown in Tables VIII and X. To further study this phenomenon, we plan to include software contention in the QNM as presented in our recent work [29], which is expected to reduce the slope of Figure 18(a) and (b) and then investigate its impact on the precision and recall of the TP-based approach.

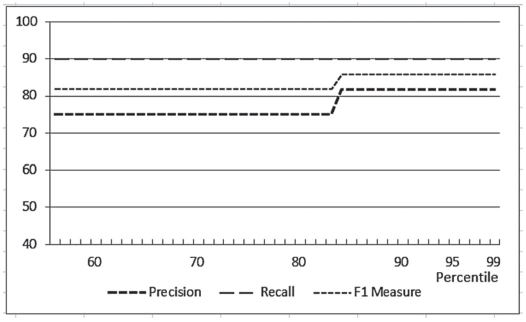
The threshold for each transaction was chosen to be the 95th percentile from the past runs for which that transaction was deemed to be good, and they are refreshed after each new run (the entire process can be fully automated). This percentile value was chosen based on common practice in the industry and also to be similar to the SPC charts case study [5]. In Figure 19(a) and (b), we show the variability of the precision, recall and F1 measure with respect to a change of the percentile value from 50 to 99 for two selected runs, one for each application. These results show that JPetStore seems particularly sensitive to the threshold and perhaps a value of 93% would yield better results. However, the choice of 95th percentile seems to be satisfactory. A more thorough threshold sensitivity analysis across a number of different applications and transactions would be required to comprehensively characterize the impact of the threshold.

The results from the last four experiments of the case study are shown in Table XXII. We can note two other interesting phenomena.

First, an obvious drawback of the SPC charts technique appears to be that thresholds and control limits depend on the workload applied to the system during the baseline run(s). Hence, they cannot be used if the new test run is conducted with a higher workload to account for new field-like requirements. For example, in the industrial application, the baseline run is conducted with 500 users, the control limits and thresholds calculated from this workload, 500 users, are not usable with the new



(a) JPetStore Application Run in Section6.4.6.



(b) Industrial Application Run in Section6.4.4

Figure 19. Precision, recall and F1 measure variations against percentile value used to calculate thresholds.

Table XXII. Summary of results on JPetStore and industrial applications.

		JPetStore application			Industrial application		
Test description		Precision (%)	Recall (%)	F1 measure (%)	Precision (%)	Recall (%)	F1 measure (%)
Single defect	TP	100	75	86	100	90	95
	SPC	100	75	86	100	80	89
Multiple defects with the same workload	TP	80	100	89	82	90	86
	SPC	75	75	75	75	90	82
Multiple defects with a new workload level	TP	80	100	89	83	100	91
	SPC	44	100	61	56	100	72
Multiple defects in cluster with a new workload level	TP	75	75	75	77	100	87
	SPC	38	75	50	53	100	69

TP, transaction profile; SPC, statistical process control.

run, which is conducted with 750 users as the violation ratio is always high not because of a performance regression anomaly but due to the higher workload in the new run. Consequently, we can see that the precision of the SPC charts approach in the last two experiments (with new workload) goes down drastically for both the JPetStore and industrial applications. This result explains why an extra performance testing run with exactly the same workload as previous release (baseline) run is needed as was discussed in Section 2. On the contrary, the precision of the TP approach maintains its value even with the new higher workload. Moreover, the precision results of the TP approach are slightly better than the SPC charts approach under the same workload.

Second, introducing one defect each time gives better precision and a lower recall compared with the multidefect runs. This can be understood by the fact that such applications have overlapping functionalities, for example, the group page has a discussion widget installed. This means that when we remove the index from the discussion database table, it not only yields a higher TP for the discussion transaction but also impacts the group transactions. So introducing multiple defects at the same time is likely to have an accumulated impact on some transactions, which increases the possibility of false positives that reduces the precision but will increase the recall. This kind of false positives therefore may reflect genuine regressions for that transaction caused by functionalities other than the core transaction functionality. The TRT is expected to suffer the same effect as can be seen from the SPC chart technique results. It should also be noted that such issues are expected to be pronounced in the traditional development process were the performance testing step is performed towards the end of the project life cycle. On the contrary, the agile process is expected to suffer less because of the smaller number of changes introduced in each iteration. This finding, as well as others concerning the application of the TP technique in the agile process, deserves further investigation.

Finally, the proposed reverse solution of the QNMs for each of the experiments earlier took an average of around 6 min for the JPetStore application and 14 min for the industrial application. This represents a fundamental saving when compared with the time needed for a performance run dedicated to find performance regression anomalies as discussed in Section 2. Such a performance run lasts around 1 h for the JPetStore application and 3 h for the industrial application for the tests performed in this case study. The overheads of such a performance run, such as restoring the database, cleaning/collecting logs and reserving the test environment for the duration of the run in addition to the time and expertise knowledge required to analyse the results, are added to these run times resulting of a potentially considerable time saving when the TP approach is used.

7. CURRENT LIMITATIONS AND FUTURE WORK

7.1. *Software contention*

The presented technique works well under normal workload levels, while when the system is pushed beyond normal or peak workload conditions, we noticed that a considerable change in the TP was found even when no performance sensitive changes were made to the application code. Our investigation shows that this is due to software resources playing a main role in the QNM at such a high workload. In our recent work [29], we proposed a way to model software contention, which we plan to use in our future work to study its effect on the inferred TP. This includes studying the effect of the number of threads and the database connections pool on the model and the calculated TP.

7.2. *Testing system evolution*

It is possible for the testing system to evolve between the previous, baseline, run and the new run. This is such as adding a new node to the application server cluster or upgrading the database server. In such cases, the individual SDs (making up the TP) will need to be corrected according to the specification of the new hardware. For example, if the database server CPU SD is 100 ms on a 2 GHz CPU, then upgrading the database server hardware to 4 GHz CPU will mean that the SD is reduced to 50 ms. In this way, the TP can be corrected for each new hardware upgrade. Some capacity management tools, which use the TP as well as implement this and have a database of all the servers' specifications, which is used to correct the TP for various hardware options.

7.3. *Performance of reverse solver*

Another area for future enhancement is the reverse solver shown in Figure 10. As the size of the system increases, both classes (transactions) and stations (resource), the speed of this solver will become a crucial factor. So the need will be to make it more efficient and quicker to converge to a solution. We here refer to our recent related work [33], which discusses this issue in more detail.

7.4. *Change of populated data*

The amount of data populated in the database can vary in the new run compared with the previous run (baseline). This is usually happening to account for growing field requirements. This is less frequent than the change of the workload applied to the system, but it could happen and should be taken into account. Unlike workload changes, the data changes can have an effect on the TP particularly on the database server SDs for both CPU and I/O. The change to the other SDs, mainly the application server CPU, should be minimal as the various queries ought to be limiting the number of data returned to the amount the user interface can handle (as we explained in the case study).

The effect of the change of populated data on TP is an interesting one that deserves more research. Initially, we suggest two solutions that may be explored more in future work

1. Provide a once-off option to ignore the database server SDs from the TP comparison when a change of populated data happens.
2. Correct the database server SDs by a factor determined with the help of the database population experts.

8. RELATED WORK

The main approach currently applied in industry to analyse performance regression testing data is the manual approach. The testing engineer will investigate the performance counters, both RU and TRT by hand. He or she will look for an uptrend in RU, such as CPU utilization, or in TRT and uses his or her judgment to reason about them. A run with workload similar to previous run is needed.

Jiang *et al.* [1] suggested an automated technique to analyse performance regression testing data using the readily available application execution log files. Accordingly, this technique does not require special instrumentation of the system under test. The approach relies on extracting log events by separating the static and dynamic fields of log lines entries. Then event sequences are established by combining the events with the same session ID. The occurrences of those event sequences in the log files are then collected, and the time required to run each event sequence is restored from the time stamps of the log entries. This is carried out for both previous and new runs, and TRT of each sequence is compared statistically, and a deviation report is generated. This approach is limited by the granularity of the log files and also suffers from the same issues as the statistical techniques.

Malik *et al.* [39] suggested to reduce the number of counters generated by performance regression testing to a manageable number that can be analysed by performance experts and is known as application signature. The number of 20 was suggested by industrial experts as the amount of counters that performance testing teams will be able to analyse out of hundreds of counters generated in a complex environment. This signature is generated by keeping only one from each group of correlated counters, such as the various CPU utilization counters (e.g. CPU processor user time and CPU total user time) generated by system monitoring tools. The signature generation task can be seen as dimension reduction problem. Four techniques were suggested to find the signature, one supervised and three unsupervised. Then the deviation is found in the signature by applying a control chart approach such as the ones mentioned in [5, 13] to highlight anomalous behaviour. The dimension reduction techniques include some simple techniques such as random sampling and clustering techniques and two more complicated approaches mainly the principal components analysis and the wrapper. The approach was tested against an open source and an industrial application and provided a good precision and recall. This approach uses the SPC charts and so will have the same problems of the statistical technique mentioned earlier, mainly the workload sensitivity and the normal distribution. Also, experts' intervention is required in various points in the signature generation process.

Inferring the SDs (and so the TP) from the TRT and RU has been studied before. This is carried out as measuring these counters is much easier than the SDs. Casale *et al.* [34] proposed to solve a linear regression between the RU and SD. While Kraft *et al.* [32] proposed a similar approach to calculate SDs based on TRT instead of RU. Liu *et al.* [36, 36] proposed a formulation for estimating the parameters of queueing networks using a quadratic programming framework. They require the end-to-end parameters TRT, RU and throughput from a set of possibly incompatible runs. The main limitation of these approaches is that they assume that the system can be approximated with a single server model, while in regression testing, a simple system such as the one shown in Figure 6 is composed of multiple nodes (application server CPU and disk I/O and database server CPU and disk I/O). We need to explore multiple tier systems and to cater for multiple resources, such as CPU and disk I/O. Additionally, all these approaches require multiple runs with different workloads, which are time consuming to do in regression testing.

Using models to predict the performance of software systems under various workload levels (known as capacity planning) has been covered by [17, 40]. Almeida *et al.* [40] describe the details of modelling of computer systems by a QNM. They provide in detail the theoretical background behind the queueing models of various computer system resources in addition to use case studies for various systems. Grinshpan [17] mentioned the concept of TP and its use along with the QNM of the computer system and the workload characterization to predict the performance of the system under various workload levels. The TP in this case is to be measured by a single user test (using load generators) to obtain the various SDs of each transaction on each resource. Then the queueing model is solved via a various set of tools both commercial and open source. The JMT is used in the examples provided to solve the model and to obtain TRT and RU of the various transactions and resources, respectively. In the capacity management approach, it is assumed that a single user test is carried out to obtain TP of the various transactions. This is unlike performance regression testing where the addition of single user test is considered an overload to the testing team and consequently is not a viable approach to obtain the TP. Apart from that, the modelling techniques used in the capacity planning field should work well in the proposed regression testing approach.

9. CONCLUSIONS

In this paper, we presented an approach to analyse performance testing data to detect performance regression anomalies caused by software updates. This approach can replace the lengthy process currently used to discover such anomalies, which is usually conducted late in the project lifecycle. We showed how the TP is less prone to changes in the level of the workload applied to the system than the TRT that is used by state of the art techniques, which eliminates the need for a specific PRTR as part of the performance testing process. Furthermore, we showed that the TP can be used as an automated way to discover such performance regression anomalies. Additionally, we proposed a novel way to infer the TP from already available performance data, mainly the RU, TRTs and the QNM of the testing system.

Through a variety of case studies, we showed the effectiveness of this approach, via the 'TP Run report', on two web 2.0 applications: one open source and one industrial. The results showed that it could detect anomalies with an average F1 measure of around 85% for the open source application and 90% for the industrial application. In addition, we showed that the technique is able to handle complex systems, particularly those with server clusters.

We expect that this workload-independent, automated approach will help reduce the time and effort required to detect performance regression anomalies caused by application changes. This improves the testing process and provides better details about the nature of the performance problem to the application developers and designers.

ACKNOWLEDGEMENT

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 ZM. Automated analysis of load testing results. In *Proceedings of the 19th International Symposium on Software Testing and Analysis*. ACM: New York, NY, USA, 2010; 143–146.
2. Kim J, Porter A, Rothermel G. An empirical study of regression test application frequency. *Software Testing, Verification and Reliability* 2005; **15**(4):257–279. DOI:10.1002/stvr.326.
3. Hewlett-Packard. Loadrunner, 2014. Available from: http://www8.hp.com/ie/en/software-solutions/software.html?compURI=1175451#.UQ-2QWfuo-A [last accessed 28 February 2015].
4. Apache. Jmeter, 2014. Available from: http://jmeter.apache.org/ [last accessed 28 February 2015].
5. Nguyen THD. 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, 2012; 491–494.
6. Li J, Zhu H, He J. Specifying and verifying web transactions. *Formal Techniques for Networked and Distributed Systems–FORTE 2008* 2008; **5048**:149–168.
7. Adrion WR, Branstad MA, Cherniavsky JC. Validation, verification, and testing of computer software. *ACM Computing Surveys* 1982; **14**(2):159–192. DOI:10.1145/356876.356879.
8. Jain R. *The Art of Computer Systems Performance Analysis*, Vol. 182. John Wiley & Sons: New York, 1991.
9. Cherkasova L, Ozonat KM, Mi N, Symons J, Smirni E. Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change. *DSN,* Anchorage, AK, 2008; 452–461.
10. Griffiths N. nmon performance: a free tool to analyze AIX and Linux performance, 2003.
11. Jacobson V, Leres C, McCanne S. Tcpdump, 1989. Available via anonymous ftp to ftp.ee.lbl.gov [last accessed 21 September 2013].
12. Nguyen THD, Adams B, Jiang ZM, Hassan A, Nasser M, Flora P. Automated detection of performance regressions using statistical process control techniques. *Proceedings of the Third Joint WOSP/SIPEW International Conference on Performance Engineering,* New York, NY, USA, 2012; 299–310.
13. Bereznay FM. Did something change? using statistical techniques to interpret service and resource metrics. In *Proceedings 32nd International Computer Measurement Group Conference, December 3-6, 2006*. Computer Measurement Group: Reno, Nevada, USA, 2006; 229–242.
14. Gao T, Ge Y, Wu G, Ni J. A reactivity-based framework of automated performance testing for web applications. *2010 Ninth International Symposium on Distributed Computing and Applications to Business Engineering and Science (DCABES),* Hong Kong, 2010; 593–597.
15. Ghaith S, Wang M, Perry P, Murphy J. Profile-based, load-independent anomaly detection and analysis in performance regression testing of software systems. *17th European Conference on Software Maintenance and Reengineering (CSMR'13),* Genova, Italy, 2013; 379–383.
16. Ghaith S, Wang M, Perry P, Murphy J. Automatic, load-independent detection of performance regressions by transaction profiles. In *Proceedings of the 2013 International Workshop on Joining Academia and Industry Contributions to Testing Automation*, JAMAICA 2013. ACM: New York, NY, USA, 2013; 59–64. DOI:10.1145/2489280.2489286.
17. Grinshpan L. *Solving Enterprise Applications Performance Puzzles*. John Wiley and Sons, Inc.: Hoboken, New Jersey, 2012.
18. Alalfi MH, Cordy JR, Dean TR. Modelling methods for web application verification and testing: state of the art. *Software Testing, Verification and Reliability* 2009; **19**(4):265–296. DOI:10.1002/stvr.401.
19. Huang P, Ma X, Shen D, Zhou Y. Performance regression testing target prioritization via performance risk analysis. *Proceedings of the 36th International Conference on Software Engineering*, ACM: Hyderabad, India, 2014; 60–71.
20. Foo KC, Jiang ZM, Adams B, Hassan A, Zou Y, Flora P. 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, 2010; 32–41.
21. Urgaonkar B, Pacifici G, Shenoy P, Spreitzer M, Tantawi A. An analytical model for multi-tier internet services and its applications. *SIGMETRICS Perform. Eval. Rev.* 2005; **33**(1):291–302. DOI:10.1145/1071690.1064252.
22. Menascé D, Almeida V. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall: PTR Upper Saddle River, 2002.
23. Walrand J. *An Introduction to Queueing Networks*, Vol. 165. Prentice Hall Englewood Cliffs: NJ, 1988.
24. Franks G, Al-Omari T, Woodside M, Das O, Derisavi S. Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering* 2009; **35**(2):148–161.
25. Lazowska E, Zahorjan J, Graham G, Sevcik K. *Quantitative System Performance*. Prentice-Hall: Englewood Cliffs, 1984.
26. Kounev S, Buchmann AP. Performance modeling and evaluation of large-scale J2EE applications. *International CMG Conference,* Dallas, TX, 2003; 273–283.
27. Baldwin R, Iv ND, Midkiff SF, Kobza J. Queueing network analysis: concepts, terminology, and methods. *Journal of Systems and Software* 2003; **66**(2):99–117.

28. Bertoli M, Casale G, Serazzi G. Java modelling tools: an open source suite for queueing network modelling and workload analysis. In *Proceedings of Qest 2006 Conference*. IEEE Press: Washington, DC, USA, Sep 2006; 119–120.
29. Ghaith S, Wang M, Perry P, Murphy L. Software contention aware queueing network model of three-tier web systems. *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering,* ACM, Dublin, Ireland, 2014; 273–276.
30. Baskett F, Chandy KM, Muntz R, Palacios F. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM* 1975; **22**(2):248–260.
31. Serazzi G. Performance evaluation modelling with JMT: learning by examples. *Technical Report 366*, Politecnico di Milano - DEI, Milano, 2008.
32. Kraft S, Pacheco-Sanchez S, Casale G, Dawson S. 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, 2009; 48.
33. Ghaith S, Wang M, Perry P, Murphy J. Transaction profile estimation of queueing network models for IT systems using a search-based technique. *6th International Symposium, SSBSE,* Fortaleza, Brazil, 2014; 234–239.
34. Casale G, Cremonesi P, Turrin R. 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, 2008; 183–187.
35. Lazowska ED, Sevcik KC. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc.: Englewood Cliffs, 1984.
36. Liu Z, Wynter L, Cathy HX, Zhang F. Parameter inference of queueing models for IT systems using end-to-end measurements. *Performance Evaluation* 2006; **63**(1):36–60.
37. Clarke J, Dolado JJ, Harman M, Hierons R, Jones B, Lumkin M, Mitchell B, Mancoridis S, Rees K, Roper M, Shepperd M. Reformulating software engineering as a search problem. *Software IEE Proceedings* 2003; **150**(3):161–175.
38. MyBatis. JPetStore 6, 2014. Available from: http://mybatis.github.io/spring/ [last accessed 28 February 2015].
39. Malik H, Hemmati H, Hassan A. Automatic detection of performance deviations in the load testing of large scale systems. *35th International Conference on Software Engineering, Software Engineering in Practice,* San Francisco, CA, USA, 2013; 1012–1021.
40. Menasce D, Dowdy L, Almeida V. *Performance by Design: Computer Capacity Planning by Example*. Prentice Hall PTR: Upper Saddle River, NJ, USA, 2004.

ACRONYMS

TP	Transaction profile. 1–7, 9–14, 16–33
TRT	Transaction response time. 2–6, 8–13, 17, 18, 28–33
RU	Resource utilization. 2, 3, 8–13, 17, 18, 29, 32, 33
TT	Think time. 8, 11, 12, 15, 18
SD	Service demand. 5–13, 20, 21, 28, 31–33
PPTR	Primary performance testing run. 3, 4, 9, 28
PRTR	Performance regression testing run. 3, 4, 9, 33
SLA	Service level agreement. 3, 8, 28
QNM	Queueing network model. 3, 4, 6, 8–13, 28–31, 33
BCMP	Baskett–Chand–Muntz–Palacios. 8, 13, 20
JMT	Java modelling tool. 8–13, 33, 35
SPC	Statistical process control. 14, 16, 17, 20–32

UCL	Upper control limits. 17, 20
CL	Centre line. 17
LCL	Lower control limits. 17, 20
DBS	Database server. 13
JEE	Java enterprise edition. 14
SQL	Structured query language. 16, 22