An automation approach to apply idle-time analysis efficiently in a performance testing scenario

Andres-Omar Portillo-Dominguez[1], Miao Wang[1], Philip Perry[1], John Murphy[1], Nick Mitchell[2], and Peter F. Sweeney[2]

[1] Lero - The Irish Software Engineering Research Centre, Performance Engineering Laboratory, UCD School of Computer Science and Informatics, University College Dublin, Ireland
`andres.portillo-dominguez@ucdconnect.ie`,
`{philip.perry,miao.wang,j.murphy}@ucd.ie`,
[2] IBM T.J. Watson Research Center, Yorktown Heights, New York, USA
`{nickm,pfs}@us.ibm.com`

Abstract. [[PENDING: The abstract should summarize the contents of the paper and should contain at least 70 and at most 150 words. It should be written using the *abstract* environment]].

Keywords: [[PENDING: performance testing, automation, performance analysis, system monitoring]]

¡¡ Creating fine tuned and stable systems is very important and requires use of a list of testing tools that analyze various resources (like GC logs, heapdumps, native memory, etc). Due to the nature of those tools, this kind of analysis can only be performed by a small group of expert users that have high technical skills. In this paper we present an approach for expert tool development in the field of performance testing. The result of this approach is the creation of GcLite tool, an expert tool for analyzing garbage collection logs. A case study was carried out in a real industry environment showing the benefits to a number of testing teams. The benefit of the tool is that it allows a wider range of testers to carry out expert analysis.

Our approach relies on a lightweight error detection mechanism based on the idea of replaying test executions against the model. We further show how the error detection capabilities can be integrated into a convincing argument for tool qualification, going through the necessary verification activities step-by-step. We highlight the key steps for the RT-Tester Model-Based Test Generator, which is used in test campaigns in the automotive, railway and avionic domains. The approach avoids having to qualify several complex components present in model-based testing tools, such as code generators for test procedures and constraint solving algorithms for test data elaboration.

Performance is a pervasive quality of software systems; everything affects it, from the software itself to all underlying layers, such as operating system, middleware, hardware, communication networks, etc. Software Performance Engineering encompasses efforts to describe and improve performance, with two distinct approaches: an early-cycle predictive modelbased approach, and a late-cycle measurement-based approach. Current progress and future trends within these two approaches are described, with a tendency (and a need) for them to converge, in order to cover the entire development cycle.

Major Garbage Collection is a common cause of performance degradation in Java applications, usually affecting throughput and response time. If systems know when they will occur, self-adaptive techniques can be applied to mitigate the performance costs. Moreover, the ability to predict these events can enable systems to take other actions based on this knowledge: A system might decide to adapt its workflow schedule or invoke a Major Garbage Collection at a time when CPU load is low. This work aims to support the decision processes of self-adaptive systems by presenting an algorithm to forecast when a Major Garbage Collection will occur. This work used a 3-phase experiment which involved implementing a prototype and testing the algorithm against a set of standard Java benchmarks. The current results have shown the feasibility of the proposed logic by achieving a reasonable level of accuracy in most scenarios. The results also showed some interesting trends that can help to tune the algorithm to specific applications and conditions. ¿¿

1 Introduction

Nowadays quality is a key element of any software development process due to its direct impact in the successful adoption of any software product as well as the total cost of ownership. For example, a study performed in USA in 2008 by Capers Jones [?] reported that achieving high quality in the software usually generates cost savings of around 40%: While average software costs per function point (FP) are close to $1,000 USD in development and maintenance, these costs are reduced to only $700 USD and $500 USD, respectively, when the software quality is above the average. This shows that investment in quality is not only the correct strategy to follow but also a necessity.

Performance will always be a key non-functional requirement, as it plays a major role in the successful adoption of any software product. This is even more important on enterprise-level applications, where issues might not only affect the users experience, but also have a considerable financial impact

Software performance (considered here as concerned with capacity and timeliness) is a pervasive quality difficult to understand, because it is affected by every aspect of the design, code, and execution environment. By conventional wisdom performance is a serious problem in a significant fraction of projects. It causes delays, cost overruns, failures on deployment, and even abandonment of projects, but such failures are seldom documented. A recent survey of information technology executives [15] found that half of them had encountered performance problems with at least 20% of the applications they deployed.

[15] Compuware, Applied Performance Management Survey, Oct. 2006.

Currently, there is a real need for enterprise applications to meet their performance requirements such that client requests are satisfied in a timely manner. Delays in response time can lead to lost revenue for example in e-commerce applications. Thus, being able to create and fine tune a stable system is very important. For enterprise Java applications garbage collection logs, thread dumps, thread usage statistics, heap dumps, CPU utilization, JVM memory usage, JDBC pool status and server response time are only some of the resources a tester can use in order to understand how the application is performing. Unfortunately, often a deep technical system understanding (of the intervals of JVM, the application server, the J2EE technology, and in some cases the application source code) is required in order to fully understand the information contained in the aforementioned logs. System and performance testers may not necessarily have this technical background and identifying problems or fine tuning systems can be difficult in these circumstances.

The latest trends in the information technology world, like Service Oriented Architecture (SOA) and Cloud Computing, have augmented the complexity of distributed applications and brought new challenges. For example, applications in the Cloud are commonly installed in huge server farms composed by hundreds or even thousands of nodes, while previously they used to be restricted to relatively small clusters of servers. Similarly SOA has broken barriers between previously unconnected technologies and allowed their integration. Increased complexity of the applications and a steady growth in the volume of transactions have exposed new areas of potential failure points in applications further complicating testing, monitoring and performance tuning.

Like other software engineering activities, SPE is constrained by tight project schedules, poorly defined requirements, and over-optimism about meeting them. Nonetheless adequate performance is essential for product success, making SPE a foundation discipline in software practice.

A determining factor for performance is that resources have a limited capacity, so they can potentially halt/delay the execution of competing users by denying permission to proceed. Quantifying such effects is an important task of SPE.

Performance testing Hard to detect and identify the root cause of performance related issues in a multi-node environment: When doing performance testing, the detection of performance related issues and the identification of their root causes are challenging and time consuming tasks, especially in highly distributed environments, usually requiring a lot of different tools plus highly related to expertise. Problem: Limitation in testing process / Reduce Costs/Time + System Complexity

Performance engineering is gaining attention, as companies discover to their detriment that the performance of their applications is often below expectations. In the past, these problems were not found until very late in the development of a product as performance validation, if any, was one of the last activities done before releasing the software. With agile processes, the problem is unchanged if not worse [11]. Thus early warning of performance problems is still the challenge for SPE.

[11] S. Barber, Tester PI: Performance Investigator,Better Software, March 2006, pp 20 25.

A common issue with current performance tools is that the produced output requires a user with expert knowledge in order to understand and analyze it (e.g. heap dump analysis where a heap dump may contain information on millions of instances of objects). In addition, these deep technical skills required to use such tools are usually held by a small number of testers within a given organisation. This can lead to issues whereby particular analysis activities can only be carried out by a small number of experts which can reduce the overall productivity of large testing teams where expertise is not evenly distributed. Moreover, current tools are sometimes difficult to understand because of the (i) large volume of the data presented, (ii) the bad layout of the data which often fails to highlight what data is important and (iii) the lack of additional context data which can be used to understand the data in the context of a particular situation (e.g. context information might indicate that an application is under heavy load). From working with a large number of industry test teams we have found that there is a real need to develop tools that can produce information which is consumable by both domain and non-domain experts, allowing for a wider number of testers to carry out complex system analysis.

Lessons from the current work. There are many weaknesses in current performance processes. They require heavy effort, which limits what can be attempted. Measurements lack standards; those that record the application and execution context (e.g. the class of user) require source-code access and instrumentation and interfere with system operation. There is a semantic gap between performance concerns and functional concerns, which prevents many developers from addressing performance at all. For the same reason many developers do not trust or understand performance models, even if such models are available. Performance modeling is effective but it is often costly; models are approximate, they leave out detail that may be important, and are difficult to validate.

The survey of information technology executives [15], which found that half of them had had performance problems with at least 20% of the applications they deployed, commented that many problems seem to come from lack of coverage in performance testing, and from depending on customers to do performance testing in the field.

Significant process issues are unresolved. Detail in measurement and modeling must be managed and adapted. Excessive detail is expensive and creates information overload, while insufficient detail may miss the key factor in a problem. Information has to be thrown away. Models and measurements are discarded even though they possibly hold information of long-term value.

Source: Future Performance Engineering

Today, the state of industrial performance measurement and testing techniques is captured in a series of articles by Scott Barber [8][9] including the problems of planning, execution, instrumentation and interpretation.

[8] S. Barber, Creating Effective Load Models for Performance Testing with Incomplete Empirical Data, in Proc. 6th IEEE Int. Workshop on Web Site Evolution, 2004, pp. 51-59. [9] S. Barber, Beyond performance testing, parts 1-14, IBM DeveloperWorks, Rational Technical Library, 2004, www-128.ibm.com/developerworks/rational/library/41

The tools used by performance analysts range from load generators, for supplying the workload to a system under test, to monitors, for gathering data as the system executes.

Developers and testers use instrumentation tools to help them find problems with systems. However, users depend on experience to use the results, and this experience needs to be codified and incorporated into tools. Better methods and tools for interpreting the results and diagnosing performance problems are a future goal.

Technical developments -¿ Visualization and diagnostics

Understanding the source of performance limitations is a search process, which depends on patterns and relationships in performance data, often revealed by visualizations. Promising areas for the future include better visualizations, deep catalogues of performancerelated patterns of behaviour and structure, and algorithms for automated search and diagnosis. Present visualization approaches use generic data exploration views such as Kiviat graphs (e.g. in Paradyn [47]), traffic intensity patterns overlaid on physical structure maps [47], CPU loading overlaid on scenarios, and breakdowns of delay [66]. Innovative views are possible. For example, in [60] all kinds of resources (not just the CPU) are monitored, with tools to group resources and focus the view. The challenge for the future is to visualize the causal interaction of behaviour and resources, rather than to focus on just one or the other. [[[47] Merson, P. and Hissam, S. Predictability by construction Posters of OOPSLA 2005, pp 134-135, San Diego, ACM Press, Oct. 2005. [60] J.A. Rolia, L. Cherkasova, R. Friedrich, Performance Engineering for EA Systems in Next Generation Data Centres, Proc. 6th Int. Workshop on Software and Performance, Buenos Aires, Feb. 2007. [66] C.U. Smith, L. G. Williams, Performance Solutions, Addison-Wesley, 2002]]

Technical developments -¿ Bottleneck identification

a search for a saturated resource which limits the system, is a frequent operation. In [21], Franks et al describe a search strategy over a model, guided by its structure and results, and detects under-provisioned resource pools and over-long holding times. It combines properties of resources and behaviour, for a nested use of resources. It scales to high complexity, is capable of partial automation, and could be adapted to interpret measured data. A multistep performance enhancement study using these principles is described in [83]. Another search strategy purely over data ([9], part 7) focuses on reproducing and simplifying the conditions in which the problem is observed. The actual diagnosis of the cause (e.g. a memory leak) depends on designer expertise. A bottleneck search strategy combining the data and the model could detect more kinds of problems (e.g. both memory leaks and resource problems) and could provide automated search assistance. Patterns (or anti-patterns) related to bottlenecks have been described by Smith and Williams [66] and others (e.g., excessive dynamic allocation, one-lane bridge). For the future, more patterns and more kinds of patterns (on measurements, on scenarios or traces) will be important. Patterns that combine design, model and measurement will be more powerful than those based on a single source. [[[9] S. Barber, Beyond performance testing, parts 1-14, IBM DeveloperWorks, Rational Technical Library, 2004, www-128.ibm.com/developerworks/rational/library/4169.html

[21] G. Franks, D.C. Petriu, M. Woodside, J. Xu, P. Tregunno, Layered Bottlenecks and Their Mitigation, Proc 3rd Int. Conf. on Quantitative Evaluation of Systems, Riverside, CA, Sept. 2006.

[83] J. Xu, M. Woodside, and D.C. Petriu, Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time, in Proc. 13th Int. Conf. Modeling Techniques and Tools for Computer Performance Evaluation, Urbana, USA, Sept. 2003]]

Scalability analysis and improvement is largely a matter of identifying bottlenecks that emerge as scale is increased, and evolving the architecture. Future scalability tools will employ advanced bottleneck analysis but will depend more heavily on models, since they deal with potential systems.

Technical developments -¿ More efficient testing

Efficient testing covers the operational profile and the resources of interest with minimum effort needed to give sufficient accuracy. Accuracy is an issue because statistical results may require long runs, and it can be affected by other factors in the test design such as the load intensity and the patterns of request classes used. For example, systems under heavy load show high variance in their measurements, which contributes to inaccurate statistical results. It may be more fruitful to identify the heavily-used resources at moderate loads and (for purposes apart from stress testing) use the results to extrapolate to heavy loads using a performance model.

More effort is also required in performance testing tools. The lack of standards for tool interoperation increases the effort to gather and interpret data, and reduces data-availability for new platforms. Lightweight and automated instrumentation are old goals that will continue to demand attention. Load drivers are at present well-developed in commercial tools, but more open tool development could speed progress.

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

WAIT (idle-time analysis) has proven successful in simplifying the detection of performance related issues and their root causes in Java environments.

The main strengths of WAIT makes it an attractive candidate to the performance testing domain, as it would have minimal impact in the testing results: - WAIT uses a very light-weight approach which does not require any instrumentation/changes to the monitored environment. - It also has very low overhead.

Hard to use WAIT in these circumstances: Despite its strengths, WAIT has certain scalability limitations that prevent its effective usage in the performance testing domain: - Manual data collection and uploading per monitored process. - Lack of synchronization with load testing tools. - Lack of capability to perform periodic data refresh during the test execution to have incremental results. Even though the above limitations might be manageable in small testing environments or test runs, they prevent WAIT to be effectively use in bigger testing environments (precisely the scenario where its performance analysis capabilities would be more valuable). The identified adoption barriers are the result of recurrent discussions with our industrial partner, which practically make WAIT non-scalable, in its current form, in highly distributed environments.

Benefits: Less Time (setup, monitor, analysis, go/no-go decision) -¿ Less Learning curve / Less Error-Prone, Share knowledge / experience (remain), Real-time, Keep as light-weight as possible (minimum add-ons + low overhead) Qualifiers: Testing Domain, Distributed env, Using Available info

- (automation) Although the approach is usable by the companys current employees, it was considered complex to apply and error prone. Doing each step by hand requires too much attention and if any single mistake is committed the whole process is compromised. Thus, the automation of the approach was considered key to its adoption.

. Second, scalability problems are not a big concern for most prototype systems developed by researchers. However, as more and more ser- vices are offered in the cloud for thousands and millions of users, load testing analysis research will become indispens- able.

...Further automation of data collection and better methods for deducing causality look promising. More powerful and general approaches to problem diagnosis are necessary and possible.... [20] E. J. Weyuker and F. I. Vokolos. Experience with performance testing of software systems: Issues, an approach, and case study. IEEE Trans. Softw. Eng., 26(12), 2000.

- According to Myers [1], software testing is a process, or a series of processes, designed to make sure that computer code does what it was designed to do and that it does not do anything unintended. Software testing has become essential to the companies to ensure the product quality regardless of whether the development methodology. Software testing automation is one of the main approaches that have been applied to decrease testing costs and time while test automation requires automated test execution and results verification [2]. [[1. Myers, G.J.: The Art of Software Testing. Ed. John Wiley & Sons, Inc., Hoboken, New Jersey. (2004) 2. Shahamiri, S.R.: Kadir, W.M.N.W.; Mohd-Hashim, S.Z.: A Comparative Study on Automated Software Test Oracle Methods. In: ICSEA '09. Fourth International Conference, pp. 140 145, (2009).]]

In more detail/ additionally supported by the needs of our industrial partner: o Manual data collection. o Uploading too. o Benchmark / Overhead. o Aggregated results. o Real time / Incremental. Important to rephrase it as estimated tester time benefit? (at least a high level estimate do not forget the time dimension!)

To ensure the research work is helpful for solving real-life problems for software industries, we have carried out regular frequent meetings with IBM System Verification Test (SVT) team managers and discussed how our current research work can improve their testing experiences by avoiding many manual tasks. We assessed the quality of our solution based on their feedback in relation to high system throughput, memory efficiency, and accuracy. Furthermore, we are also occasionally involved in IBM cross team collaborations to share our knowledge and discuss how our research algorithms can be used to solve different problems faced by other teams. In order to improve industrial testing experiences, we have made the decision to design and implement a lightweight solution with specific rules for functional and system testing teams to ease their troubleshooting tasks.

To ensure the research work is helpful for solving real-life problems for software industries, we have carried out regular frequent meetings with IBM System Verification Test (SVT) team managers and discussed how our current research work can improve their testing experiences by avoiding many manual tasks. We assessed the quality of our solution based on their feedback in relation to high system throughput, memory efficiency, and accuracy. Furthermore, we are also occasionally involved in IBM cross team collaborations to share our knowledge and discuss how our research algorithms can be used to solve different problems faced by other teams. In order to improve industrial testing experiences, we have made the decision to design and implement a lightweight solution with specific rules for functional and system testing teams to ease their troubleshooting tasks.

This paper proposes We introduce a lightweight framework for replaying concrete test executions with the aim of identifying erroneous test case executions. Our proposed solution is to develop an expert tool that can be used by a wider pool of testers where traditionally this work was done only by a small number of experts. Our approach was to develop a tool that can produce information consumable by both domain experts and non experts. This approach provides a high level view for non experts allowing them to identify issues and a more detailed fine grained view for experts allowing for detailed root cause analysis of the issues. We followed a two step approach for developing such

Highlights of methodology. This work was achieved through a 3-phase experiment: The first two phases focused on defining the preferable execution parameters for the proposed algorithm and the selected Java benchmarks, while the third phase focused on the validation of the algorithm.

The results of this work have provided evidence regarding the feasibility of the proposed approach + summary of key results. We have validated the tool through a case study that was carried out across 6 different test teams in IBM, showing how such a tool allowed for a larger number of testers to carry out garbage collection analysis and ultimately identify and raise a large number of bugs as part of the garbage collection analysis.

The main contributions of this paper are: 1. Approach/Architecture + implementation to automate the data gathering and execution of WAIT in sync with a performance test loader (or make WAIT scalable / usable in a performance testing domain). 2. Generic enough solution so that it can be adaptable to other performance analysis tools and/or performance test loaders. 3. Keeping the main strengths of WAIT: low overhead + avoiding intrusion (under the scope of the paper). 4. Any qualitative insights? (i.e. javacore cost of the class loader section)

The contribution of this paper is an approach to make the principle of parameterized unit testing available to black-box GUI testing. The approach is based on the new notion of parameterized GUI tests. We have implemented the approach in a new tool. In order to evaluate whether parameterized GUI tests have the potential to achieve high code coverage, we apply the tool to four open source GUI applications. The results are encouraging.

The paper is structured as follows. In the next section we explain the approach for creating an expert tool. Section 3 details the motivation for creating a garbage collection analysis tool and how the approach was realised through the implementation of GcLite. In this section we also give design and implementation details. In section 4 we discuss how we have validated our approach through a case study carried out within IBM across 6 different test teams and in section 5 we give our conclusion.

The rest of this paper is organized as follows: Section II discusses the background. Section III shows the related work. Section IV describes the proposed approach. Section V describes the experimental setup, while Section VI explores the experimental results. Finally Section VII presents the conclusions and future work.

2 Background

Performance Testing Process

In this paper, when we speak of software performance testing, we will mean all of the activities involved in the evaluation of how the system can be expected to perform in the field. This is considered from a user's perspective and is typically assessed in terms of throughput, stimulusresponse time, or some combination of the two.

He uses different tools: - Heap dumps: WAIT? - Thread dumps: WAIT (locks, contentions) around every 5 - 10 mins / ISA (performance analysis kit based on Eclipse). - Overall development expertise (i.e. architecture, layers, frameworks, tools). - Code profiling: tprof to profile in real time (recording every 1-2 mins to know the CPU usage per transactions); jprofiling (single transaction costs, such as CPU usage). - Database: Tunning of SQL queries (or expert systems that encapsulate the identification of the most common issues). - GC: gclite (expert system) or jmap.

Key message/conclusion: The identification of the root causes of performance issues, relies very heavy on the expert knowledge of the user + requires a lot of different tools. There is no possible to indicate an average time per issue identification, but overall is very challenging/time-consuming.

WAIT and/or javacores? (check my bookmarks)

WAIT: Key Attributes WAIT is a tool for performance triage Gives high-level, whole-system, summary of performance inhibitors

WAIT is zero install Leverages built-in data collectors Reports results in a browser WAIT is non-disruptive No special flags, no restart Use in any customer or development location WAIT is low-overhead Uses only infrequent samples of an already-running application

WAIT is simple to use Novices to experts: Start at high level and drill down WAIT uses centralized knowledge base Allows rules and knowledge base to grow over time, and be fixed quickly when found to be inadequate.

[Collection-¿Upload-¿Report]

- Distributed Environments? - Evaluate Cloud Distributed environment as possible focus/background info?

3 Related Work

[PENDING to review my other papers?]

Automated Performance Testing (support tools). Performance Analysis Tools/Techniques (from WAIT papers). WAIT

¡¡ (Intro) Related Techniques: In the software development process, performance testing is commonly conducted to determine an application's performance as experienced by the user. In this context, the focus is generally not on a single application session but on the application in its entirety, i.e., one does not test a single, isolated re- quest of a single user but rather the performance when many users interact with the application simultaneously. Software testers commonly use techniques such as load testing, stress testing, etc., to perform these kinds of tests [3]. There is a variety of commercial products and free open-source tools available for the task. For example, IBM's Rational Performance Tester [4] and HPs LoadRunner [5] both do scalability testing by generating a real work load on the application. Open-source tools such as Apache's JMeter [6] and Grinder [7] provide a similar functionality. Motivated by the large cost for commercial performance testing tools, Chen et al. created Yet Another Performance Testing Framework [8]. It enables users to create custom test programs which de

ne the business operations to be per- formed during the test. Chen's framework then executes these tasks concurrently. In [9], Zhang et al. present a cloud-based approach to performance testing of web services. Their system provides a frontend in which users can specify test cases which are then dispatched to Amazon EC2 [10] cloud instances for execution. Similar to all the previous tools, their system is testing the performance under concurrent user access to the system. [[3. Molyneaux, I.: The Art of Application Performance Testing. Volume 1. O'Reilly Media (2009) 4. IBM: Rational Performance Tester. http://www-01.ibm.com/software/ awdtools/tester/performance/ 5. Hewlett Packard: HP LoadRunner. http://www8.hp.com/us/en/software/ software-product.html?compURI=tcm:245-935779 6. Apache Software Foundation: Apache JMeter. http://jmeter.apache.org/ 7. Aston, P.: The Grinder, a Java Load Testing Framework. http://grinder.sourceforge.net/ 8. Chen, S., Moreland, D., Nepal, S., Zic, J.: Yet Another Performance Testing Framework. In: Australian Conference on Software Engineering (ASWEC). (2008) 170 {179 9. Zhang, L., Chen, Y., Tang, F., Ao, X.: Design and Implementation of Cloud-based Performance Testing System for Web Services. In: Conference on Communications and Networking in China (CHINACOM).

(2011) 875 {880 10. Amazon Web Services LLC: Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/ (2012)]]

Related Work: The idea of replaying a test execution in a simulator is, of course, not new. The overall approach is frequently referred to as the capture and replay paradigm, and has long been studied in different contexts such as testing of concurrent programs [2]. ... To our best knowledge, our approach is the first to combine replay with model-based methods for error detection within a test-case generator. Our contribution is not a theoretical one, but comes from an industrial perspective.

Related Work: Unlike these approaches, our work aims at proposing a generic and platform independent test system based on the TTCN-3 standard to execute runtime tests. The proposed test system supports different test isolation mechanisms in order to support testing different kinds of components: test sensitive, test aware or even non testable components. Such test system has an important impact on reducing the risk of interference between test behaviors and business behaviors as well as avoiding overheads and burdens.

SPE, The commonest approach is purely measurement-based; it applies testing, diagnosis and tuning late in the development cycle, when the system under development can be run and measured (see, e.g. [2][4][8][9]).

[2] M. Arlitt, D. Krishnamurthy, J. Rolia, Characterizing the Scalability of a Large Web-based Shopping System", ACM Trans. on Internet Technology, v 1, 2001, pp. 44-69.

[4] A. Avritzer, J. Kondek, D. Liu, and E. J. Weyuker, "Software performance testing based on workload characterization," in Proc. WOSP2002, Rome, , pp. 17-24.

[8] S. Barber, Creating Effective Load Models for Performance Testing with Incomplete Empirical Data, in Proc. 6th IEEE Int. Workshop on Web Site Evolution, 2004, pp. 51-59. [9] S. Barber, Beyond performance testing, parts 1-14, IBM DeveloperWorks, Rational Technical Library, 2004, www-128.ibm.com/developerworks/rational/library/416

Performance testing on part or all of system, under normal loads and stress loads [8]. The use of test data to solve problems is the subject of [9]. This activity is discussed in Section 3 below.

[8] S. Barber, Creating Effective Load Models for Performance Testing with Incomplete Empirical Data, in Proc. 6th IEEE Int. Workshop on Web Site Evolution, 2004, pp. 51-59. [9] S. Barber, Beyond performance testing, parts 1-14, IBM DeveloperWorks, Rational Technical Library, 2004, www-128.ibm.com/developerworks/rational/library/416

Performance Testing Tools? (HP Load Runner, Jmeter, RPT).

[Pending to see where better to fit these points:] Benefit: Tools usability, productivity increase (less time identifying issues and their root cause), any benefit in the consolidation (other than reducing from N*M reports to 1), i.e. trends in time and/or nodes (probably good input from performance team): * How does the process currently work? (steps and time) i.e. 2 hours debugging, 2-4 RCA + fixing OO * Issues in different nodes sound like environmental ones? * Issues in different times (kind of trending) might involve the application? They refer to end to end testing the most common issue are deadlocks.

¿¿

4 Problem Definition

PROBLEM DEFINITION: WAIT Challenges in Performance Testing environment XXX (Rephrase Industrial partner case)

Problem Statement: Performance of commercial enterprise application is off by a lot e.g. order of magnitude

Where is the problem? Database? File System? Network? App Server? Deadlock? Insufficient load? Livelock? Memory leak? Garbage Collection?

Who should identify and fix the problems? Developers Testers Field Operators

Current focus is not distributed environments, but mainly individual Java processes: Problem with consumability in distributed environments: Time-consuming/complex data gathering (one data source per monitored process). Time-consuming/Complex report analysis (one report per monitored process). Plus required synchronization with the performance testing execution.

SME expertise required to interpret the reports: Problem with consumability in parts of Dev, SVT, Perf and Support (few engineers outside of Watson are thread dump experts)

Adoption Barriers:

Data collection scripts should be kicked off and stopped manually. Uploading results to central server is not fully automated. Aggregating results from large distributed systems (e.g. 100+ servers) is not supported today. No real time information available to assess health of a workload run. Benchmark is needed to know the requirements/trade-offs of using the tool.

5 Proposed Approach and Implementation

¡¡ Approach and Implementation: In this section we present details of our approach and its implementation. As outlined in Section 1, there exist a bunch of issues in order to make the approach of parameterized GUI tests applicable to real world applications. Our approach depicted in Figure 7 consists of the following consecutive steps: (1) Event Flow Construction; (2) Symbolic Widget Injection; (3) Symbolic Event Injection; (4) Event Handler Elevation; (5) Generation of Parameterized GUI Tests, (6) Symbolic Execution, and (7) Replayer. + [Figure]

In this section we present our approach for creating an expert tool in the field of performance testing. From a high level view as part of the approach for creating such a tool we need to (1) document expert knowledge (usually in the form of rules describing known issues) and then (2) apply the expert knowledge to flush out or identify issues that exist in the real system. Identifying issues in a real system using documented expert knowledge follows a four step approach previously outlined in the literature [9][10]. Next we describe how we applied this approach in the context of garbage collection analysis. The four main steps are (a) to monitor the system, (b) analyze the data collected, (c) detect issues within the analyzed data and (d) present any issues or potential issues identified to the end user.

Our approach relies on a lightweight error detection mechanism based on the idea of replaying test executions against the model. We further show how the error detection capabilities can be integrated into a convincing argument for tool qualification, going through the necessary verification activities step-by-step. We highlight the key steps for the RT-Tester Model-Based Test Generator, which is used in test campaigns in the automotive, railway and avionic domains. The approach avoids having to qualify several complex components present in model-based testing tools, such as code generators for test procedures and constraint solving algorithms for test data elaboration. ¿¿

1. SCOPE LIMITS o Performance Testing, o Web Applications

4. ARCHITECTURE OVERVIEW/DESIGN XXXOverview (similar visualization that I have presented before but more polished-). 5. ARCHITECTURE DETAILS XXX Explaining how the opportunity areas are addressed.

The GcLite tool is an expert tool for analyzing garbage collection behavior. It is a Java application with an extensible design. Figure 1 shows a high level view of GcLites design and how it enhances the design of current GC tools [12][13][14].

[PENDING: DIAGRAM TO SHOW THE APPROACH]

1. PROTOTYPE (CURRENT) IMPLEMENTATION Describe RPT WAIT + WebAgent; explaining the reason behind each main technical decision.

¡¡ The Run-Time Correlation Engine [4] (RTCE) has been developed in conjunction with our industrial partner IBM, and serves us as a platform base for the symptom matching implementation presented in this paper. RTCE has been deployed by several test teams across the world for monitoring enterprise applications. RTCE can signifi- cantly save administrators time spent on system analysis [20]. For example, one particular testing team reports one full log analysis in 1 hour which is estimated to take up to 23 working hours without automated support of RTCE. ¿¿

ANY LIMITATIONS? RPT, but architecture generic enough (i.e. Web Agent)

¡¡ - (automation) A prototype tool is designed and developed based on the concepts presented here. ANTLR parser generator is used to create syntax grammar for complete CICS based COBOL language specifications. Similarly other parsers are also developed using Java. Using the prototype tool developed, case studies have been conducted on an industrial application. The application is legacy library management system running in one of our clients library. The number of test cases generated for each of the three modules is listed in Fig 3.a. A sample test scenarios generated is shown in Fig 3.b.

- (automation) Figure 2 shows the architecture of the tool we have implemented as a plug-in for Eclipse [8], on top of EMF [11] and the UML2 plug-in. The functionality of this tool includes . . .

Prototype: To realize trace verification, we have developed a passive testing tool [2], which aims to automate the process of trace verification. A description of this tool is given in Fig.3. ¿¿

6 Validation

This section describes the evaluation of the proposed approach. First the methodology, metrics and the main threats to validity are presented. Then, the results of the experiments are presented.

6.1 Methodology

¡¡ Experimental Setup: Application Set. During the experimentation, three real world applications were used: - A - B ¿¿

4. METHODOLOGY

¡¡ We define the following two research questions: Q1 ... Q2 ... Results of the Experiments: Figure 12 shows the results of the experiments. We answer Q1 with Yes ... ¿¿

¡¡ In this section we evaluate our approach. We compare how our approach performs, (a) when the computation of input data is replaced by the use of random values, and (b) when the Event Flow Graph is not considered for event sequence generation. ¿¿

Phase 1: o Objective: Validate WAIT Overhead vs. WAIT-RPT one. o Combinations (3): WAIT Presence (3): Test without WAIT, Test with WAIT (manual test), Test with WAIT-RPT Web-Applications (1): Portal (single node) Test configurations*: Workloads (1): To be defined (It should be big!) - 2,000 Duration: 1 hour WAIT-RPT configuration: HD Threshold: Disabled (so that only the time threshold controls the process). Time Threshold: 10 minutes (Constant, so that 6 intervals occur). Sampling frequency: 8 minutes.* —-¿ 30 secs? * For each identified combination, there will be 3 runs (9 hours of total execution). WebAppServer restarted before every run. NOTE: These configurations will be based on expert judgment of the SVT team (reflect common practice in the industry?) Phase 2: o Objective: Validate Architecture (qualitative, quantitative) o Combinations: WAIT Presence (2): Test run without WAIT, Test run with WAIT-RPT Integration Web Applications (1): Either Lotus Connection or JPetStore. Test configurations*: Workloads (1): To be defined (Tentatively 1, which should be big!) 2,000 per node (4,000 assuming 2 nodes) Duration: 24 hour (SVT uses 1 hr, 24 hour, 5 days, discarding 1 hr due to relevance and 5 days to do time constraints). WAIT-RPT configuration*: HD Threshold: 100 MB (pending to confirm). Time Threshold: 10 minutes (pending to confirm). Sampling frequency: 8 minutes.

For each identified combination, there will be 3 runs (6 days of total execution). WebAppServer restarted before every run. NOTE: These configurations will be based on expert judgment of the SVT team (reflect common practice in the industry?) It might also help to show that the reliability of the solution (I might take a look to other fancy NFR-related words)

[PENDING: To create figure showing the environment - single PID, then multi-PID-]

Phase 3: Use case detection benefits of 1 report vs. many! For the sake of progress, probably the last piece of the experiment! Injecting defects, showing how they are better/more easily identified. PetStore. Two dimensions: - Time: Early detection or trends (stress early detection). o Nodes: Environmental issues.

Benefits of WAIT: - Injecting bugs in JPetStore (Ideas from WAIT + SVT team). - Benefit of single WAIT report, possible two dimensions: - Time: Early detection (avoid wasting resources if serious issues identified). - Nodes: Environmental issues (very likely cause of having an issue in some nodes and other not). - Meeting to know the fixing side of the performance testing/analysis to get better insights about which use case would be preferrable. - TEST METRICS: TESTER PRODUCTIVITY? (single case, but give an idea of the benefit) http://www.softwaretestinggenius.com/articalDetails?qry=967

¡¡ The teams used performance and load testing tools like LoadRunner [16] and IBM Rational Performance Tester [15]. They executed reliability runs (the ability of a system or component to perform its required functions under stated conditions for a specified period of time [24], usually 5 or 7 days) and performance runs (short period runs in order to measure something specific, like the response rate [25]). As The main aim of both performance and reliability testing was to ensure that the applications had good performance (transaction and page response rates) even under heavy load, made reasonable usage of memory and didnt have any memory leaks [31]. The performance of a software system has been described as an indicator of how well the system meets its requirements for timeliness [32]. Smith and Williams [32] describe timeliness as being measured in either response time or throughput, where response time is defined as the time required to respond to a request and throughput is defined as the number of requests that can be processed in some specific time interval. For

15: IBM. Rational Performance Tester. http://www-01.ibm.com/software/awdtools/tester/performance 24: Wikipedia. Reliability engineering. 25: B.Subraya. Integrated Approach to Web Performance Testing: A Practitioner's Guide. IRM Press 2006

By the above case study we tried to see if our approach in the creation of an expert tool in the field of garbage collection was successful. The measures of success were the ability to identify bugs and problems that affect the applications as well as additional test coverage by identifying new types of issues (e.g. finalizers, system.gc, etc). Moreover, the tool should allow a wider range of testers to perform the expert analysis, offering at the same time significant time savings for the expert users.

The results from using the tool were very encouraging as many bugs were identified as well as the time saved by using GcLite in some cases was quite high. In the table 2, we can see a summary of our case study. We can notice that the time saved per application is around 1.6 hours. GcLites automated collection and analysis of the GC logs, the ease of use, the layout of the output and the use of recommendations are responsible for the time savings. We also notice that an average of 6.5 defects was identified using the tool.

Another important advantage of using the GcLite tool was the ability to open precise bugs. Testers used the contextual information from the tool in order to investigate deeper the issues and provide detailed description in the SPRs that they were opening. Furthermore, less experienced testers used this information in order to gain a deeper understanding of several re-occurring issues.

In conclusion, the test case showed that the use of an expert tool that uses the 4 step approach can have several advantages. The time saving from using GcLite automated collection of GC logs and analysis instead of another tool in some cases was quite high. This was also enhanced by the tools ease of use and the outputs ease to navigate. Moreover, the amount and variety of bugs that were identified shows that GcLite can analyze and discover issues within a GC log by using the recommendationEngine. ¿¿

3. KEY METRICS - Qualitative vs. Quantitative results! - Qualitative: Possible verbiage about how the detected adoption barriers are addressed. - Previously you would have ended with: - X reports, assuming you monitored X nodes and generated all the reports after the test execution finished. - X*Y reports, assuming you monitored X nodes and generated the reports every 1/Y time. - Now, in both cases you would have ended with a single report. - Sanity checks: Not lost zip files, hard disk threshold respected. - Quantitative: Overhead costs: (possible min, max, avg) - Performance Testing: Response Time, Throughput - Node Resource Usage: RAM, CPU

Z. THREADS TO VALIDITY:

Big variance in 24-hour runs it seems to different behaviours: Week and Weekend! (more runs). Environmental issues

¡¡ - Of course, the results observed cannot be generalised to other applications different from the two case studies considered here. The presented case studies are just meant to illustrate the differences that may arise. Wider experiments need to be executed to get more general conclusions.

Threats to Validity: The main threat to the validity of these results is the fact that only three test subjects were used during the experimentation. Despite these subjects being real world applications being diverse in both the size of the application (lines of code) and size of the test suite, the limited number of subjects implies that not all types of systems are tested. This means that a system with characteristics that are completly different might present different results. Another threat is that the number of injected bugs is not enough to lead to accurate results, as these bugs might simply be lucky bugs that intercept a collar variable. Naturally, there are also threats that are based on the implementation of the invariants, the instrumentation or the pattern detector algorithms themselves. The reduce these threats, additional testing was made prior to the experimentation to guarantee the quality of the experimental results in this regard.

Threats to Validity: Beyond the selection bias due to the limited availability of open source C# applications, we report one threat to external validity: We evaluated four C# open source applications which incorporate the Windows Forms toolkit for building the GUI. Alternative programming languages and GUI toolkits, e.g., Java Swing, follow different paradigms of building graphical user interfaces. For example, it might be not possible to obtain event handlers during the construction of the EFG. Thus, the construction of the EFG, the generation of parameterized GUI tests, and the symbolic execution must be adapted to the corresponding environment. In principle, there is no reason to believe that our approach is not applicable to other environments.

¿¿

Statistic references: Enable Add-on in Excel: http://click4biology.info/c4b/1/2007.htm Finding t critical value in excel: http://math.stackexchange.com/questions/10992/finding-critical-value-using-t-distribution-in-excel how to use t-test in excel (example 2): http://blog.excelmasterseries.com/2010/08/how-to-use-t-test-in-excel-to-find-out.html Other T-test reference: http://www.ruf.rice.edu/ bioslabs/tools/stats/ttest.html Excel t-test function: http://www.excelfunctions.net/Excel-Ttest-Function.html T-test Excel Help: http://office.microsoft.com/en-001/excel-help/ttest-HP005209325.aspx Good example to learn T-test: http://www.aspfree.com/c/a/braindump/comparing-data-sets-using-statistical-analysis-in-excel/ —

Experimental Environment

We installed the benchmark in the system environment depicted in Figure 6. The benchmark application is deployed in an Oracle WebLogic Server (WLS) 10.3.3 cluster of up to eight physical nodes. Each WLS instance runs on a 2-core Intel CPU with OpenSuse 11.1. As a database server (DBS), we used Oracle Database 11g, running on a Dell PowerEdge R904 with four 6-core AMD CPUs, 128 GB of main memory, and 8x146 GB SAS disk drives. The benchmark driver master, multiple driver agents, the supplier emulator and the DNS load balancer were running on separate virtualized blade servers. The machines are connected by a 1 GBit LAN, the DBS is connected with 4 x 1 GBit ports.

7 Experimental Results

1. [PHASE 1] o Compare response time & throughput with and without WAIT/WAIT-RPT. o Compare resource consumption per node with and without WAIT/WAIT-RPT. + Testing/Process metrics. 2. [PHASE 2] o Compare response time & throughput with and without WAIT-RPT. o Compare resource consumption per node with and without WAIT-RPT. o Validate that there was not information lost between the collection and uploading.

3.[PHASE 3] X,Y to an actual number (use case) more qualitative, but showing how it is best 1 than many! Injecting 4 bugs, then run it using the same config than Phase 2 (2 in both nodes, 2 only on a specific one). Then two iterations will be run: 1st to catch up as many issues as possible. The 2nd one to see results after fixing issues (possible an intermediate run might be needed, as some issues might mask other ones) After each run, the detected bugs will be fixed.

¡¡ - (automation) This section presents a case study adapted from [8] and initially discussed in [3]. The SUT is a burglar alarm system whose goal is to monitor sensors to detect the presence of intruders in a building. Consider a test case, presented in [3], that covers the occurrence of an interruption (transfer to the backup power) followed by the detection of an intruder finishing with a call to the police. The objective of the case study presented in this paper is to show how to use the developed API in test case automation (from step 2 to step 4 in Fig. 1). For this, a version of the burglar alarm system was developed to run on the FreeRTOS environment based on industrial PC (x86) port.

- This section summarizes a simplistic case study whose purpose is to illustrate through a practical example the application of the principles described in Section 3. It is worthy mentioning that rates have been included as constants declared using reference values which are not shown. The actual values for these constants should be changed as more detailed reliability data is available and has no influence on the scope of this paper.

Empirical Results: In this section the experimental setup is presented, along with the workflow of the experiments themselves. After that the experimental results are discussed.

¿¿

8 Conclusions and Future Work

The objective of this work was to XXXX. To achieve this goal the paper presented a novel XXXX approach and its validation, which was composed of a prototype and a study case. The results have proved that the proposed approach addresses the adoption barriers for WAIT in a distributed performance testing environment from a qualitative perspective (desired behavior + minimum impact in the environment) and a quantitative (solution remains light weight and minimum overhead).

Furthermore, the approach can be easily extended to support other XXX tools?.

Future work will focus on three main venues:

Assess how better to address the identified major overhead costs of using WAIT in a distributed performance testing environment (i.e. more than 50% of the javacore information is not useful but still propagated from nodes to WAIT server. If removed it, the network overhead would go down by that percentage).

Similarly, evaluate how best to exploit the information that can now be obtained from a testing environment (i.e. test workload, response time, throughput and transactions) to improve the diagnosis quality and quantity of problems that WAIT can identify.

More immediate is to evaluate the benefits of the proposed architecture/approach in a bigger scenario, most likely through a study case of its adoption in our industrial partner SVT/IBM. / Besides, we will concentrate on further evidence of the practicability of our approach by applying it to additional case studies and industrial systems.

¡¡ Testing and tuning enterprise applications can be challenging due to their complexity. It requires use of a list of testing tools from expert users that have high technical skills. In this paper we presented an approach for expert tool development in the field of performance testing.

The approach has been tested in a real industry environment and the results are encouraging for the tool and for the approach that we followed. The test case showed that the use of an expert tool that uses the 4 step approach can have several benefits to the user like time savings, opening precise bugs and supporting them with contextual information and educating less experienced testers.

- (automation) This paper presents an ongoing work that addresses a solution to automation of test case execution for RTES at the software and system integration level. The solution provides an API to define PCOs such as the observation of values returned by functions, received messages, and timing associated with system responses. A case study is also presented to show the applicability of the work. Though it still needs to be extensively validated, the solution addresses the issues on test execution raised in the introduction. It can deal with different development and execution platforms once the SUT is instrumented to run on its target platform and logs are generated to be evaluated at any environment. Moreover, the log generation mechanism allows testing applications with hardware limitations.

Conclusion and Future Work: In this paper we have proposed a novel approach to the generation of GUI tests, implemented in a new tool called Gazoo. Gazoo selects event sequences from the EFG of an application and generates a set of Parameterized GUI tests. Then, Gazoo applies Pex in order to instantiate parameterized GUI tests. Finally, Gazoo replays instantiated GUI tests on the application. In the terminology of the black-box/white-box dichotomy, Gazoo starts with a black-box approach (using the EFG in order to select executable test sequences), then moves on to a white-box approach (in order to generate parameterized GUI tests and instantiate them using Pex), and finally goes back to the black-box approach (using a replayer in order to execute the (instantiated) GUI tests on the application). As shown in the paper, we needed to overcome a number of non-trivial technical hurdles in order to establish the appropriate interface between the black-box approach and the white-box approach. The scope of this paper was to show that our approach can achieve high code coverage. Usually one expects that high code coverage translates to high bug detection rate. For future work, we need to evaluate that this holds true in our setting. This evaluation requires its own series of experiments where one applies statistical methods to fault-seeded versions of AUTs, following, e.g., [11, 18]. Our work opens an interesting perspective for future research because the general scheme behind our approach goes well beyond a specific tool, here Gazoo. We need to explore different alternatives such as, e.g., [2, 16] and, e.g., [12, 19] for going back and forth between the black-box approach and the white-box approach in the sense described above. ¿¿

Example cite [1] blabla bla.

Acknowledgments

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. Abdelkader Lahmadi, Laurent Andrey, Festor, O.: Ambient Networks. **3775** (2005)