Introducing Performance Engineering by means of Tools and Practical Exercises

Alexander Ufimtsev, Trevor Parsons, Lucian M. Patcas,
John Murphy and Liam Murphy

Performance Engineering Laboratory,
School of Computer Science and Informatics, University College Dublin, Ireland
{alexu,trevor.parsons,lucian.patcas,j.murphy,liam.murphy}@ucd.ie

Abstract

Many software engineers complete their education without an introduction to the most basic performance engineering concepts. IT specialists need to be educated with a basic degree of performance engineering knowledge, so they are aware of why and how certain design and development decisions can lead to poor performance of the resulting software systems. To help address this need, the School of Computer Science and Informatics at University College Dublin offered a final year undergraduate/first year postgraduate module on "Performance of Computer Systems" in Autumn 2005. In this paper we document how performance engineering was introduced to the students through practical exercises, and how these exercises relate to industry problems.

1 Introduction

Software languages and frameworks have developed significantly since their early days. They have become more abstract and developers no longer need to concern themselves with lower level issues such as resource usage. An example of this can be seen in modern languages (Java, C#) that provide garbage collection facilities freeing developers from the task of having to manage memory which had typically been complex and time consuming exercise. This is even more obvious with enterprise level software frameworks (J2EE, .NET), whereby the framework can be expected to handle complex underlying issues such as security, persistence, performance and concurrency to name but a few. Freeing developers from having to worry about what is happening under the hood allows them to concentrate more of their efforts on developing the functionality of a system. However, a downside of this trend is that developers become less familiar with the mechanics of the underlying system and, as a result, can make decisions that have an adverse effect on the system. This is especially evident in the area of performance of systems, where projects often fail to deliver in time functional systems which meet their performance requirements, fact that leads to major project delays and subsequently higher development and maintenance costs [5]. Two major factors contribute to this fact. First, many developers do not have an understanding for the basic concepts of performance engineering. This is hardly surprising, since many software engineers complete their education without an introduction to the most basic performance engineering concepts. As a result, it is common that they are unaware of the performance implications of many decisions taken during the design and implementation of a system. Second, since many developers do not have an understanding for performance engineering, often they believe that system performance can be addressed at the end of development cycle after a system has been designed, implemented, functionally tested and deployed. This common misconception often leads developers to take the opinion that performance is a matter of ”production tuning” that merely involves tweaking different system parameters at the end of the development cycle. It is more often the case however that performance issues have a deeper structural nature. Consequently, it is common that a major rework of the system is required to meet performance goals leading to expensive delays at the end of the project.

To address the above problems, developers need to be educated with a basic degree of performance engineering knowledge so that they are aware of why and how certain design and development decisions can lead to poor performance of the resulting software systems. To help address this need, the School of Computer Science and Informatics at University College Dublin offered a final year undergraduate/first year postgraduate module on ”Performance of Computer Systems” in Autumn 2005. The course introduced a basic theoretical framework of Performance Engineering, while the practical work for this course consisted of exercises to allow students to develop an understanding of the performance aspects of industry scale software systems.

In this report we detail certain aspects of this course. We focus on the performance tools used, paying particular attention to two relatively new performance tools, AdaptiveCells and PredictorV. We document how these tools were used to educate the students on performance engineering through practical exercises, and how the exercises relate to industry problems. Section 2 presents the performance engineering concepts that were introduced to students. Section 3 describes the main features of the tools used for the practical work, and motivates their choice. Section 4 describes the practical assignments and steps that the students followed to conduct a complete analysis of a sample J2EE application. Sections 5 and 6 present our observations and conclusions upon the course aspects presented in this report.

2 Performance Engineering

The performance engineering concepts we present in this section relate to the performance metrics, as well as the software performance engineering (SPE) methodology that the students followed for their practical work.

We introduced to students the most common performance metrics used to characterise the performance behaviour of a system: *throughput* (the number of requests the system can handle in a given time period); *response time* (the temporal delay between the moment the system receives a request and the moment it fulfils that request); *latency* (the temporal delay between the receiving of an event and the system reaction to that event); *capacity* (the maximum load the system can take while meeting its throughput, response time, and latency requirements). In addition to these performance metrics, the performance behaviour of a system can as well be characterised by *resource utilisation* (*e.g.* CPU or memory consumption).

The methodology that the students followed for the performance evaluations they carried out was inspired from the SPE guideline presented in [2]. That is, *performance requirements*, which define the values expected from a functioning system, must be considered from early stages in the software development cycle and tracked through the entire process. Therefore, the students carried out performance testing (to address the performance problem at the testing and maintenance stages) and performance modelling (design stage) of sample J2EE internet applications.

Usually in the testing stage, *functional testing* is performed before performance tests in order to catch the errors in the application to be tested that are not caused by performance issues. Thus, functional testing tries to eliminate non-performance problems before any performance testing is conducted.

Performance testing deals with verifying whether the system meets its performance requirements under various workloads. The *workload* of a system denotes how many users the system can handle at any given time. In order to allow the performance analysis of the system, various workloads are generated and the system performance behaviour under these various workloads is recorded in the *load testing* phase. Usually, load testing involves generation of simulated requests to the system using load generation tools (see Section 3.1). Performance testing can also involve finding the highest workload the system can take while still fulfilling its requirements, that is *stress testing* the system. *Performance optimisations* can be tried in order to correct some of the problems discovered during performance testing. Load and stress testing treat the system as a "black box" to whose internal structure test engineers do not have any insights into.

Performance modelling is a complementary method to building prototypes or simulating the system behaviour. During the system design, this method provides answers to questions like "How does the system scale up?", "What is the capacity that allows the prescribed quality of service for expected workloads?", "Which components of the system are bottlenecks?", or "Which components are more sensitive to variations?". Performance modelling makes use of discrete event simulation (e.g. workload generation tools) to generate usage scenarios of the system, and *profiling* to capture the interactions between system components corresponding to those usage scenarios and to record the performance metrics. The data obtained from profiling constitutes the performance model of the system. The analysis of this model helps identifying bottleneck or sensitive components. This analysis also allows for *capacity planning*, which aims to ensure that sufficient capacity is available so that the system can cope with increasing demand. Performance modelling can help developers understand how the components they are working on contribute to the performance of the whole system. Performance modelling is a "white box" approach, developers and system designers having an insight into the internal structure of the system.

3 Tools

Students conducted performance testing and modelling of sample J2EE applications that were generated with the AdaptiveCell[1] tool. They carried out the performance testing using the JMeter[2] load generation tool, and the performance modelling using PredictorV[3]. This section introduces these tools and motivates their choice.

3.1 JMeter

JMeter is a tool for workload generation and performance measurements. It can be used to simulate a heavy concurrent load on a J2EE application and to analyse the overall performance under various load types. It also allows for a graphical analysis of the performance metrics (*e.g.* throughput, response time) measured.

3.2 AdaptiveCells

AdaptiveCells is a novel tool that allows for the development of complex artificial J2EE testbed applications without requiring a single line of code. These applications can be used for a number of different purposes, such as performance testing, middleware infrastructure testing and even for the creation of working J2EE applications. The initial learning curve for writing J2EE applications is often prohibitively high and means that even developing simple test cases can be a major effort. AdaptiveCells solves this problem by allowing for the creation of working (complex) J2EE applications without having to write the code. The testbed applications generated with AdaptiveCells have a fully controllable behaviour at runtime. By selecting the appropriate configurations, testers and developers can replicate how resources such as CPU and memory are consumed by the different parts of the application. In fact, AdaptiveCells goes further by allowing for the emulation of performance bugs (*e.g.* memory leaks) which often occur in real systems. The applications generated can also be configured

[1]http://adaptivecellsj.sourceforge.net
[2]http://jakarta.apache.org/jmeter
[3]http://www.crovan.com

to throw exceptions at certain points. These characteristics of AdaptiveCells represent real advantages not only in learning environments such as the one discussed in this report, but in the development of real-world component-based software systems as well. For example, applications generated with AdaptiveCells can be used to compare the performance of real application servers, or, in the area of middleware infrastructure, they can be used for testing problem determination tools [3] or monitoring tools such as COMPAS [1].

3.3 PredictorV

PredictorV is a modelling and simulation tool that aims to help IT architects and developers solve performance problems in enterprise J2EE based systems by predicting their performance early in the design cycle. PredictorV is an Eclipse-based product available in form of both a plug-in and standalone application [4]. The tool offers a framework for performance monitoring, modelling and prediction of component-based systems. The aim of the framework is to capture the information from a system and automatically generate a performance model. Then the performance model is used to determine quickly where the performance problems are in the system. One advantage of this approach is that it reduces the time and skill required to build a model, because information to build the model itself is captured directly from the system. Another advantage is that as much of the process as possible is automated, so it reduces the risk of human errors being added into the model. The tool comprises of three modules: a module to monitor the system, a module to model the transaction flow on that system, and a module to predict the performance of the system.

Monitoring module The monitoring module tries to collect enough information from a system under test so that a predictive model of the system can be built. In order to so, it needs structure information (what the system does) and resource information (how much resource the system uses) for each business transaction that occurs in the usage scenarios employed for testing. The best method for collecting this information in a Java-based system is to use profiling. The profiler makes use of the Java Virtual Machine Profiler Interface (JVMPI) to collect information relating to the current state of the JVM, such as memory and CPU information.

Modelling module The modelling module is responsible for displaying the call graph for each business transaction that has been collected using the monitoring module. UML Event Sequence Diagrams notation is used to display this call graph. The sequence diagrams in PredictorV show the ordered sequence of events that happen inside the application as it services a user request, and are annotated with profile data showing the CPU and memory that are consumed.

Prediction module The prediction module takes the UML models and detects performance bottlenecks (the *InSight* analysis level), identifies ways of correcting these bottlenecks (*MoreSight*), assesses capacity planning under different hardware configurations (*ForeSight*), and evaluates performance for different usage scenarios (*ClearSight*).

4 Practical Exercises

This section describes the tasks that the students had to accomplish for their practical assignments, as well as the steps they followed in order to conduct a complete performance analysis of a sample J2EE application.

The practical assignments were intended to imitate real-life development environments. For this purpose, the students formed groups and each group played the role of a Quality Assurance (QA) team in a software development company. The tasks[4] of each team were to conduct performance testing and modelling of a sample J2EE application. The sample applications comprised of seven components and ten possible configurations (different interaction scenarios between components), and were generated with AdaptiveCells (see Section 3.2). We introduced randomly some per-

[4]The assignments can be found online at http://floating1.ucd.ie/comp4015

formance problems likely to occur in real applications, such as memory leaks or excessive resource usage [6], into the applications tested by students. Each QA team submitted in the end a report that contained two parts: an executive summary of their work to the managerial staff, and a detailed informative report to the developers. To accomplish their tasks, students had to follow the methodology described in Section 2.

Functional testing Students had first to perform functional testing of their application to eliminate the configurations which were obviously not working properly. They had to identify whether the cause of the problem was indeed a performance related problem (using garbage collection and debug information) or a problem of a different nature.

Performance testing Once the functional tests were performed, students used the JMeter load generation tool (see Section 3.1) to simulate a workload on the application. The complexity of this exercise lay in the requirement that the students construct a realistic workload on the application. Students had to decide upon common usage scenarios of the application at hand. It was explained to the students that testing systems with an unrealistic workload would produce unreliable results, which is not acceptable in real-life situations. After generating a realistic workload, students tested their system by gradually increasing the number of clients. They collected both the throughput and response time, and performed an analysis of the variation of these metrics against the number client requests. Students had to notice the trend in the system behaviour under different workloads (load testing) and the number of clients threshold for which the throughput started to decrease and response time started to increase (stress testing).

Performance optimisations The next step included various optimisation techniques that would lead to improved application performance. J2EE platform offers variety of levels where performance can be tweaked, including Java Virtual Machine (JVM) layer, Application Server (AS) layer, Operating System (OS) layer, hardware, and application itself. However, only JVM and AS layers were considered in the scope of the course. After a few initial hints students were asked to use available resources, including the Internet, to find and experiment with performance of Sun Microsystem JVM and JBoss AS[5]. They were also asked to explain why certain parameter would result in better, or worse application performance.

Application profiling Until this step, students were using a "blackbox" approach to the application they were testing. They did not have any insights into how the application was working internally, and only used the external functional interfaces to extract behavioural and performance information. Successful modelling, on other hand, requires a deep inside knowledge of the application to be modelled. Therefore, students extracted information about the inner logic of the application using the profiler of PredictorV (see Section 3.3). Once extracted, these scenarios could be easily visualised in PredictorV in the form of sequence diagrams. Profiled data also contained information about resource utilisation.

Performance modelling Students used the profiled usage scenarios, augmented with resource utilisation, together with the data they extracted in the load and stress testing steps to build a model of their application in PredictorV. The model consisted of specified hardware topology (network devices, application server, and database server) and all the necessary data (variations of performance metrics with workload, resource utilisation), in order to create an environment that mimics a real system. Students used this model for various types of performance simulations and predictions: to determine system performance on alternative hardware configurations; to provide hardware estimates for doubling and tripling the throughput (capacity planning); to analyse the design of the application in order to identify performance antipatterns [2], which are common design mistakes.

[5]http://www.jboss.com/products/jbossas

5 Observations

The course feedback received from the students was analysed at the end of the course through an anonymous system. The feedback was mainly positive with the one exception being the amount of time and effort required to test their application correctly. Most of the students had to conduct the performance tests a few times, mainly because many parameters had to be taken into account to run the test correctly (*e.g.* performing load and stress testing, measurements, profiling, optimisations). The student felt that while the "help from lab demonstrators" was good to very good the "computer facilities" were only fair. This would be mainly due to the number of applications that they were required to master in the course. Most students agreed that they put a "lot of effort" into this course, and that it was "interesting" and a "clear link between the lectures and the practicals".

All the reports that the students produced were read by three staff members and about 25% of the students were subjected to an interview to ensure the quality of the material presented. The students scored well in this course, with it being the second highest scoring course over about ten courses that were held for that cohort of students. For example over 80% of the students would have got a second class honours grade one or higher in this course.

6 Conclusions

In this paper we present our experience in introducing performance engineering concepts to university students through practical exercises. These exercises can easily be integrated as part of performance engineering solution in a real industry environment. We give details on a number of tools that we used for the practical part of the course. In particular we focus on the test bed application, AdaptiveCells, and modelling and simulation tool, PredictorV. However one of the key lessons from this exercise was the need for intensive laboratory support, which would limit the number of applications and the scale of the problems that could be addressed in a single course.

References

[1] Mos A. *A Framework for Adaptive Monitoring and Performance Management of Component-Based Enterprise Applications.* PhD thesis, Dublin City University, Ireland, 2004.

[2] Smith C.U. and Williams L. G. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Addison Wesley, 2002.

[3] Chen M. et al. Pinpoint: Problem determination in large, dynamic, internet services. In *Proc. Int. Conf. on Dependable Systems and Networks (IPDS Track)*, 2002.

[4] Murphy J. Whitepaper: Peeling the layers of the "performance onion". http://crovan.com/download/crovan_whitepaper.pdf.

[5] Musich P. Survey questions Java App reliability. http://www.eweek.com/article2/0,1895,1388603,00.asp.

[6] Haines S. Solving common Java EE performance problems. http://www.javaworld.com/javaworld/jw-06-2006/jw-0619-tuning.html.