

A simulation model of a multi-server EJB system

David Mc Guinness,
Performance Engineering Laboratory
Department of Computer Science
University College Dublin
Belfield, Dublin 4 Ireland
david.mcguinness@ucd.ie

Liam Murphy
Performance Engineering Laboratory
Department of Computer Science
University College Dublin
Belfield, Dublin 4 Ireland
liam.murphy@ucd.ie

ABSTRACT

Despite the fact that EJB (Enterprise Java Beans) is a widely used technology, research in the area of performance modelling of EJB application servers is quite sparse. This paper will describe how Workbench™, an advanced simulation modelling tool, can be used to build a scalable model of a multi-server EJB system that allows users to input variables that describe interactions and their constituent methods, as well as system parameters. The model will output the average time for each given user interaction and allow users to seek system improvements by changing the system parameters and workloads.

1 Introduction

The Java 2 Platform Enterprise Edition has demonstrated its use as an important standard for developing component-based multi-tier enterprise applications. The J2EE platform simplifies enterprise applications by basing them on standardized, modular components and providing a complete set of services to those components by handling many details of application behavior automatically, without complex programming [1]. A set of APIs is defined in J2EE to enable quick and efficient application building as well as a run-time infrastructure to host them. A key component of J2EE, Enterprise JavaBeans (EJB), has established itself as one of the leading architectures for developing and deploying secure, scalable and reliable applications. As the use of such systems in Web-based, e-commerce environments expands, the need to model the performance of these systems becomes ever more important in facilitating optimal architectural design choices and system parameterisation. The goal of this paper is to describe a model that has been developed to simulate an EJB system in terms of its input parameters and to see what effect these inputs have on the response times for given interactions.

The modelling environment used is Hyperformix Workbench™ 4.0. Workbench™ is a general Discrete Event (DE) simulator with an extensive set of high-level simulation operations as well as a rich set of statistical and queuing functions. As Workbench™ is particularly well suited to specifying and evaluating

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

A-MOST'05, May15–16, 2005, St. Louis, Missouri, USA.
Copyright 2005 ACM 1-59593-115-5/00/0004...\$5.00.

complex systems involving a high degree of concurrent processing, it is ideally suited to modelling an EJB system [2].

This paper will be split into the following sections. Section 2 provides some background on EJB. Section 3 describes the Workbench™ simulation software used, discusses the parameters that are input to the model and outlines how the model runs and what its outputs are. Section 4 describes the experiments that were run and Section 5 draws some conclusions and identifies some future research directions.

2 EJB Environment

EJB defines a model for the deployment of reusable EJB components that can be assembled into a secure, scalable and reliable application that can run on any EJB-compliant application server.

The goal of this technology is to provide users with the following features:

- *Platform Independence* - Regardless of the underlying operating system, protocol or enterprise middleware services the EJB application will operate in a consistent manner.
- *Component Portability* - Components written for one EJB application can be reused in any other EJB product regardless of the vendor.
- *Standard, Agreed-Upon Technology* - That EJB is a common and standard technology means it is easier to find pre-trained staff, benefit from best-use principles, sell software and call upon external support.
- *Improved Development Efficiency* - The EJB architecture provides developers with a host of services (transaction, security, automated persistence, etc.) that enable the developer to quickly and efficiently create enterprise applications by focussing on application-specific business logic [11].

2.1 EJB Architecture

The EJB architecture is designed to facilitate distributed, component-based computing. EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology. A key aim of this technology is to leverage Java's "Write once, run anywhere" model.

2.2 EJB Components

The EJB Component Model comprises three different bean types alongside the EJB container that are used to carry out business

interactions. These three bean types are Session Beans, Entity Beans and Message-Driven Beans (MD Beans). Session beans can be further sub-divided into stateless and stateful, whereas Entity Beans can be characterised by their method of persistence: Container-Managed Persistence (CMP) or Bean-Managed Persistence (BMP). Session Beans typically represent business processes and are relatively short-lived entities. Stateful Session Beans contain conversational state on behalf of an individual client. They are intended to represent business processes that span multiple method requests (e.g. a shopping cart). Stateless session beans on the other hand represent single request conversations and as such do not retain state. Stateless session beans can easily be pooled and reused by several different clients. An example of the use of a stateless Session Bean could be a Credit Card Validator (that receives the credit card details as an input parameter) but doesn't need to hold any state beyond the method execution.

It has been shown [5] that stateless session beans with BMP can run very efficiently (comparable to servlet-only implementations). Hence our model is designed to represent only stateless session beans at this time.

3 Model Design

3.1 Workbench™

A previous single-server version of the model was created using Ptolemy II [10]. This model was ported to Workbench™ and then further developed into the multi-server version described in this paper. One of the most drastic differences between the models was the execution time. The model created in Workbench™ runs several orders of magnitude faster despite being more complex. This is due to the fact that Workbench™ provides highly efficient high-level actors, whereas in Ptolemy such high-level actors need to be created from multiple lower-level actors.

Workbench™ is an industrial strength modeling and simulation package that enables modeling of large, complex systems and returns timely, useful results that can be used to influence system design. It is a general Discrete Event (DE) simulator with an extensive set of high-level simulation operations as well as a rich set of statistical and queuing functions. As it is based on embedded C code it executes very quickly and supports very large designs.

3.2 Parameterization

There are many parameters that affect the performance of an EJB server and where possible these parameters have been included in the model. The model also uses several *background* parameters that are also mentioned to aid understanding of how the model works.

3.2.1 User-modifiable parameters

The parameters detailed in Table 1 are input parameters to the model which can be modified by the user before running the model. All of these parameters are set at runtime and remain constant throughout the model execution. Each server in the model has its own set of user-modifiable parameters. The 3 interactions are differentiated by the interaction parameters mentioned below, e.g. interaction 1 runs only one method, whereas the others run two and Interaction 3 requires the same bean for both its methods.

USER-MODIFIABLE PARAMETERS
Per server
<i>numberOfThreadsAvailable</i>
<i>memorySize</i>
<i>numberOfBeansDeployed</i> (array) – each bean's deployment is detailed here
<i>numberOfCPUs</i>
<i>CPU Speed</i> (scalar)
<i>I/O Speed</i> (scalar)
For the entire system
<i>Bandwidth</i> (scalar)
<i>numberOfUsersPerInteraction</i> (array)
<i>endTime</i> (when the model should stop executing)
Interaction parameters
Interaction Parameters (array of arrays) – each interaction is represented by a single array. This array itself contains the following parameters for each method in the interaction:
<ul style="list-style-type: none"> ➤ <i>CPU Time Required</i>, ➤ <i>memory Usage</i>, ➤ <i>I/O Time Required</i>, ➤ <i>bean Type Required</i>, ➤ <i>localOrRemoteCall</i> (0 for local / 1 for remote / -1 denotes end of interaction)

Table 1: User-modifiable Parameters

3.2.2 Model Parameters

Additional runtime parameters used by the model but not modifiable by the user are shown in Table 2. These parameters are all modified by the system as it executes and represent the state of the system (except *warmUpPeriod*, *Timestamp* and *interactionIdentifier* whose values do not change). Event Token Parameters represent the state of a currently-executing interaction. As was the case with the user-modifiable parameters, each server has its own set of model parameters.

MODEL PARAMETERS
Per server
<i>numberOfThreadsInUse</i>
<i>memoryInUse</i>
<i>beanCounts</i> (array)
For the entire system
<i>warmUpPeriod</i>
Event Token Parameters
<i>Event Token Parameters</i> – each Event Token contains 4 parameters:
<ul style="list-style-type: none"> ➤ <i>Timestamp</i> ➤ <i>interactionIdentifier</i> ➤ <i>indexForInteractionArray</i> ➤ <i>totalMemoryInUseForInteraction</i> ➤ <i>currentServer</i>

Table 2: Model Parameters

3.3 Model Design

Some previous work exists that looks at the performance of EJB Systems (see [4], [5], [6]). There are also some analytical models of an EJB Server (see [7], [8]). However there is little or no work in the area of simulation of EJB systems, particularly ones that may be used by those with little simulation experience. Due to the accuracy that can be achieved with simulation models in comparison with analytical models, as well as the advantages in flexibility of simulations compared to real working systems, a simulation model of an EJB system is overdue. Simulation models have many advantages over analytical ones such as the ability to model the effects of priority scheduling as well as transient and bursty effects and the ability to compute complex statistics such as the 95th percentile.

In Web-based EJB systems it is the Web-server or more specifically, proxy client components created by the web-server that call the EJB server. As our area of interest is purely the EJB server it was decided that it would be modelled as a closed system. Many simplifications of a real system have been made in order to ensure that the model does not become too cumbersome to use or too expensive in terms of execution time. There is a fine balance between the amount of detail necessary to make a model meaningful and a level of detail that is very CPU-intensive and hence creates restrictively long runs [9]. The simplifications for this model include the facts that the CPU is visited only once per method, that all CPUs on the machine have equal power, that all methods have equal priority, and that security has not been modelled. Elements of the model that are seen to have only a small impact on output values may be simplified or ignored.

submodel shown here represents a reference to that submodel. So when a user of a given transaction enters the *controller* submodel at (a), the user soon exits to the *allocateThread* submodel where they will wait (if necessary) until a thread becomes available. When the user enters the model they will have the parameter *currentServer* already set to whichever server they are directed to. This parameter is used each time they need to access resources, e.g. if the user's *currentServer* parameter is set to 0, then when they look for a thread they look at *numberOfThreadAvailable[0]*, i.e. the number of threads available on server 0. Then they will return to the controller model and move into the *allocateMemory* submodel, where they will similarly look for memory from the appropriate server. Then the user looks for the appropriate bean on the appropriate server, e.g. *beans[0][1]* is where they would look for bean 0 on server 1. The user now has acquired an execution thread and a bean and can now carry out its necessary activities so it goes to the *CPU* submodel. The CPU model operates a form of semaphore system. The *numberOfCPUs* parameter for the given server determines how many tokens are available on that server. When a user arrives he takes a token (after waiting for one to become available if necessary) and waits at a delay node for the current *CPUTimeRequired* specified by the interaction it is executing. Then it releases the token to the next user in the CPU queue for that particular server. From the CPU submodel the user either moves to the IO submodel and queues for I/O resources, or bypasses it, if there is no I/O associated with the method it is currently executing, i.e.:

IOTimeRequired [indexOfInteractionArray -1]==0

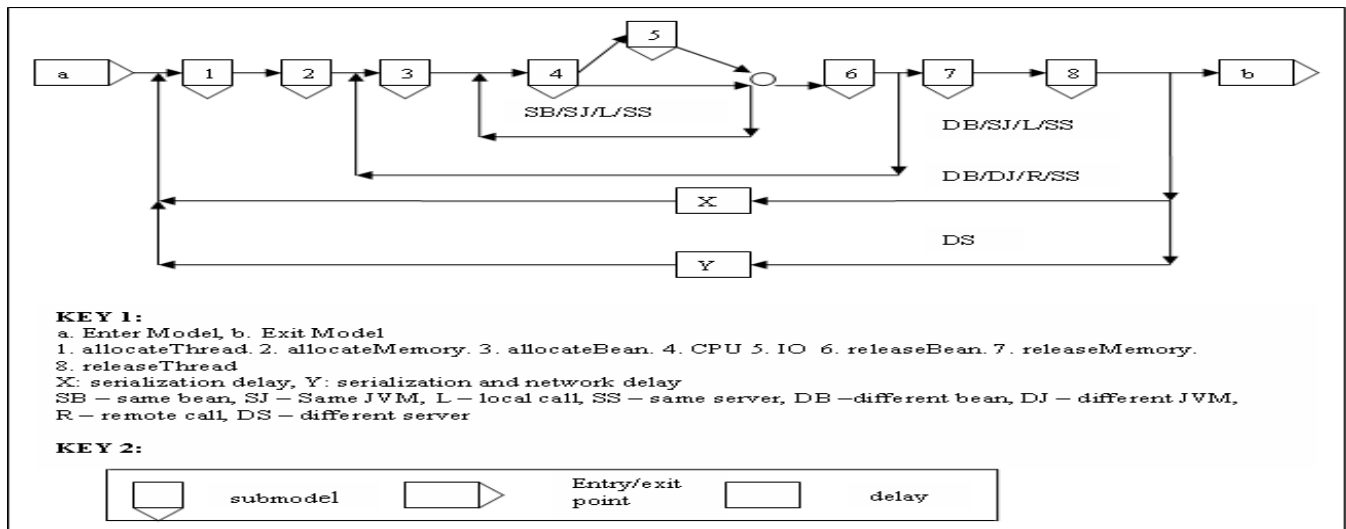


Figure 1: EJB Model (Flow of control) – the controller submodel

The users for each transaction are created at start time and then released uniformly into the system during the warm-up period. The release rate is calculated as follows:

$$\text{Release rate} = \text{warmUpPeriod} / \text{numberOfUsersForInteraction}$$

By the end of the warm-up period all users will have entered the system and will remain in the system until measurements have stopped, i.e. it is a closed system. The *controller* submodel represents the flow of users through the system (see Figure 1). Each

At this point the method has ended. However where the user moves on to next depends on numerous factors. If the user wants to execute the next method on the same server, via a local call to the same bean:

```
(number_of_methods > indexOfInteractionArray)
    &&
    (currentServer ==
    nextServer_for_bean[beanNeeded
    [indexOfInteractionArray]] % numberOfServers)
    &&
```

```
(localOrRemoteCall[indexForInteractionArray -1]==0)
    &&
(beanNeeded[indexForInteractionArray]==
beanNeeded[indexForInteractionArray -1])
```

This means there is no need to relinquish the thread, the memory it has accumulated or even the bean it is using to work with (path SB/SJ/L/SS in Figure 3). The next option is the same except that a new bean is needed:

```
(beanNeeded[indexForInteractionArray]! =
beanNeeded[indexForInteractionArray -1])
```

This means that the user will proceed along DB/SJ/L/SS in Figure 1 having already passed through the *releaseBean* submodel. If however the call to the next bean is a remote call (even if the user still wants to use the same server) or if the user wants to use a new Java Virtual Machine (JVM) on the same server, it needs to release both its accumulated memory and the execution thread it holds and hence it moves to the *releaseMemory* and *releaseThread* submodels.

At this point the user has released all resources on the server it is operating on and is ready to move on. Here the distinction is made between a remote call to a bean on the same server or a call to a bean on a different JVM on the same server on the one hand (DB/DJ/R/SS) which incurs a serialisation delay, and the situation where the user actually needs to go to a different server on the other hand, which incurs the serialisation delay as well as a network delay.

As we are dealing with stateless session beans only the serialisation delay is set to zero as this operation does not take place for stateless session beans.

The next server is determined on a round robin basis in terms of bean requests. For example if we have 2 servers and the last request for bean A was directed to server 0, then the next request for bean A will be directed to server 1. This does not take into account where the user is currently executing. This model assumes that the bean deployment is equal on each of the servers but can be easily modified to a weighted round-robin if the bean deployment varies across servers.

If the interaction is completed, i.e. the last method has been completed the user leaves the *controller* submodel at (b) and goes back into the *generateWorkload* submodel and starts over again. It is here that the throughput figures for the relevant interaction are updated and the response time is measured. The user will continue this cycle until the specified runtime is reached.

3.4 Model Outputs

The model collects various statistics about the system. As previously mentioned, only those statistics that refer to events that were started after the warm-up period will be included so as not to include details of events that execute before the system is fully populated (as these would execute interactions faster than the steady-state interactions, as well as cause resource utilisation statistics for the steady-state to be undervalued).

Outputs include:

- *averageInteractionExecutionTime* (end-to-end) for each interaction
- *throughputPerInteraction*
- *averageUtilisationPerCPU*
- *I/OUtilisation*
- *maximumMemoryInUse*

4 Experiments

The goal of these four sets of experiments is to look at the effect of varying one parameter and its effect on the system outputs. Specifically we are interested in response times and throughput of each of the interactions, and the resource utilisation of the system. Unless stated otherwise all experiments involve a 3- server cluster.

The first set of experiments looks at the effect of changing the CPU scalar. An increase in the CPU scalar results initially in a fall in response times and an increase in throughput. The effects on resource utilisation are that CPU utilisation falls as the CPU scalar increases as it requires less time to process the same code, thus putting more pressure on the I/O resources thus increasing their utilisation.

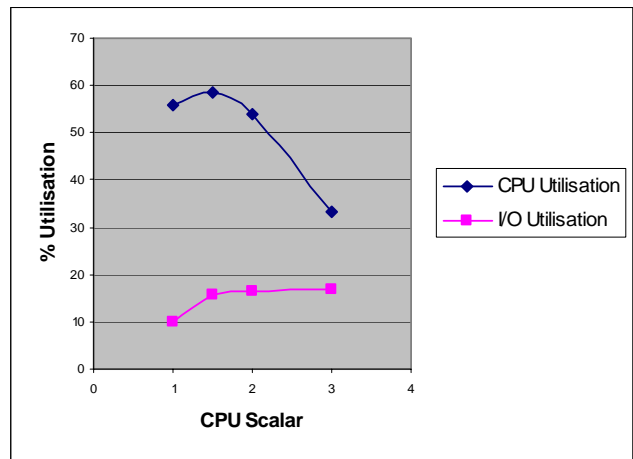


Figure 2a: Resource Usage with changing CPU speed

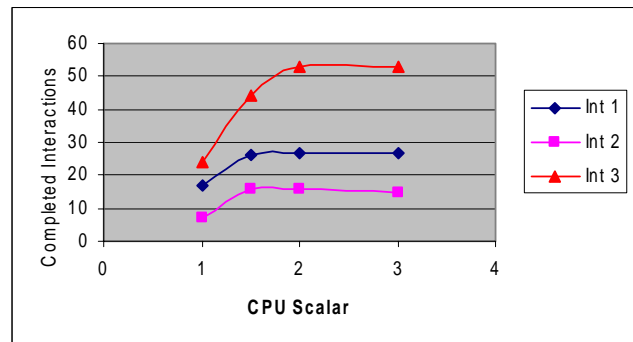


Figure 2b: Effects on Throughput from changing CPU speeds

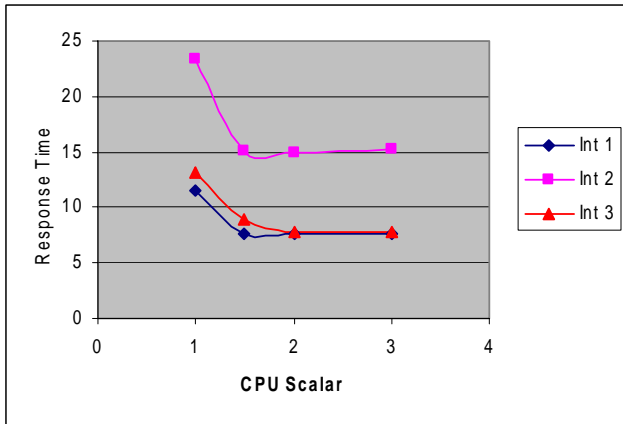


Figure 2c: Effects on Response Time from changing CPU speeds

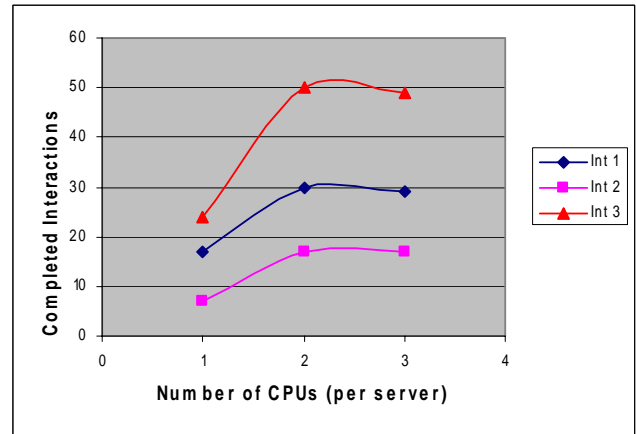


Figure 3b: Effect on Throughput from changing number of CPUs

However the returns from increasing CPU power diminish at a certain point (somewhere between a CPU scalar of 1.5 and 2.0). Up until this point the gains in throughput and response times were more or less linear but after it the gains are practically zero.

The second set of experiments looks at the effects of changing the number of CPUs on interaction throughput and response times as well as resource utilisation. The effect on resource utilisation almost mirrors that of changing the CPU scalar (ignoring the 1.5 value for the CPU scalar as we cannot have 1.5 CPUs!). The shape and positioning of the response time curves and the throughput curves also very closely match those of the first experiment. This suggests that the advantages of increasing CPU power compared to increasing their number are more or less equivalent.

As we increase from one CPU to two per machine we have clear gains in response times and throughput for each of the interactions but adding a third makes little difference to anything other than CPU utilisation suggesting that adding a third CPU to each machine would be a waste of resources in this case.

The third set of experiments looks at varying the number of servers. All servers are given the same set of hardware resources (memory, number of CPUs, power of CPUs, etc.) and software resources (number of beans, number of threads etc.) for this experiment. This experiment shows that increasing the number of

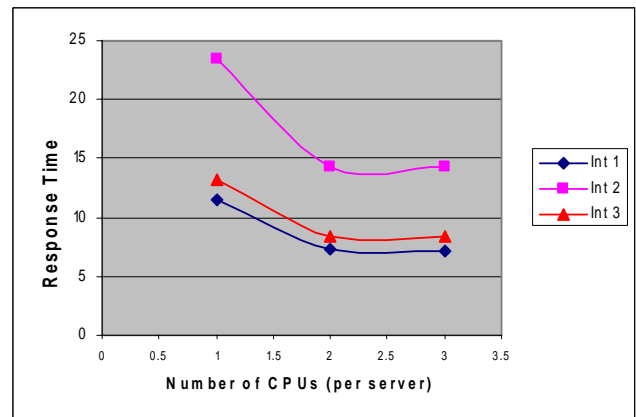


Figure 3c: Effect on Response Time from changing number of CPUs

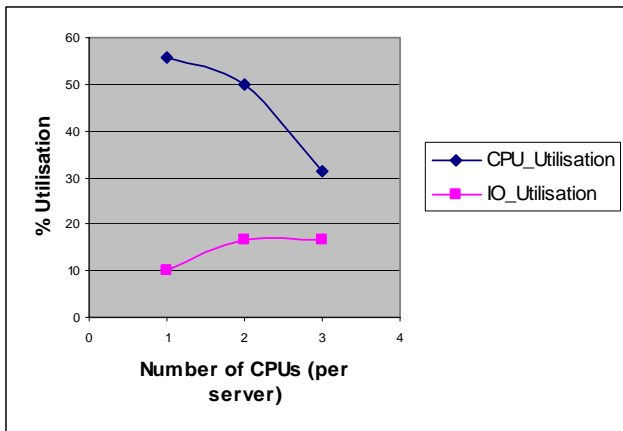


Figure 3a: Resource Utilisation with changing number of CPUs

servers has the effect of decreasing the response times for each of the interactions (though not uniformly) and increases the throughput (though again not uniformly). With a 4-fold increase in the number of servers response time decreased in the best instance to about a third of its original value whereas for both of the other interactions the improvement was less than 50%. What is interesting is that both of these changes are linear and that at 4 servers show no sign of diminishing returns. The improvements for throughput from a 4-fold increase in the number of servers are also linear but the actual increase in the number of completed transactions ranges from 2 to 9 times the original throughput.

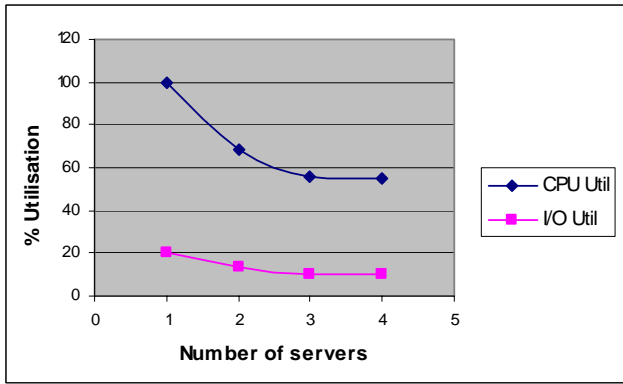


Figure 4a: Resource Utilisation with changing number of servers

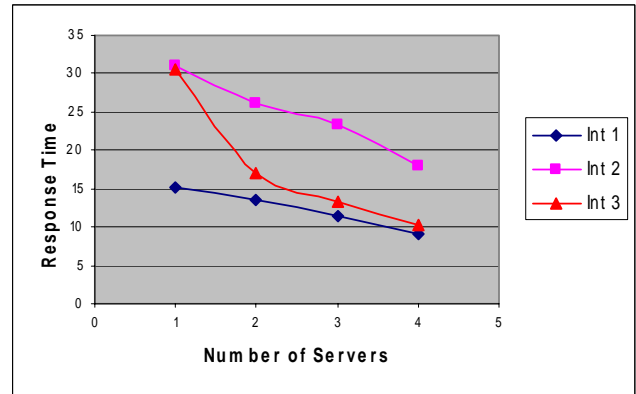


Figure 4c: Effect on Response Time from changing number of servers

The final experiment looks at bean deployment and its effect on response times, throughput and resource utilisation. As bean requests are routed according to a round robin routing policy among the different servers it makes sense to have an equal number of any given bean on each of the servers. For simplicity's sake we have also deployed the same number of each bean type on each server. This enables us to draw more meaningful relationships between the number of beans deployed and the various system metrics we are interested in. Again as with the changes in system hardware (number of CPUs, CPU speed, I/O speed) we see an increase in throughput and a decrease in response times up until a certain point and after that point the extra resources are mostly redundant in this case.

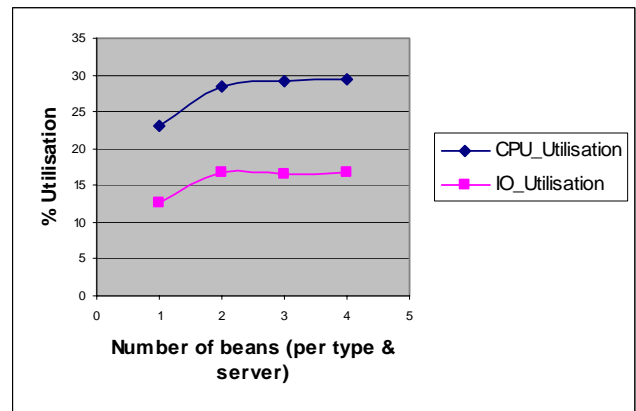


Figure 5a: Resource Usage with changing bean deployment

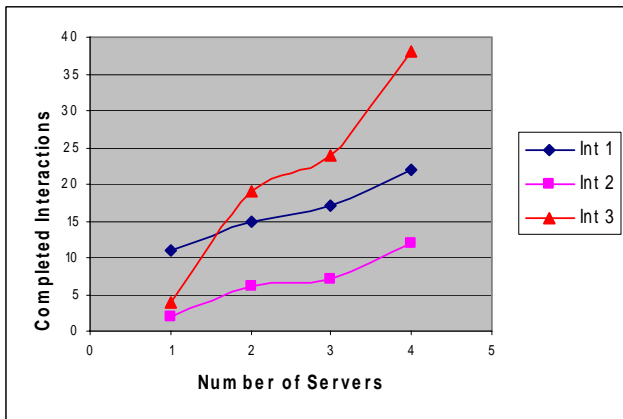


Figure 4b: Effect on throughput from changing number of servers

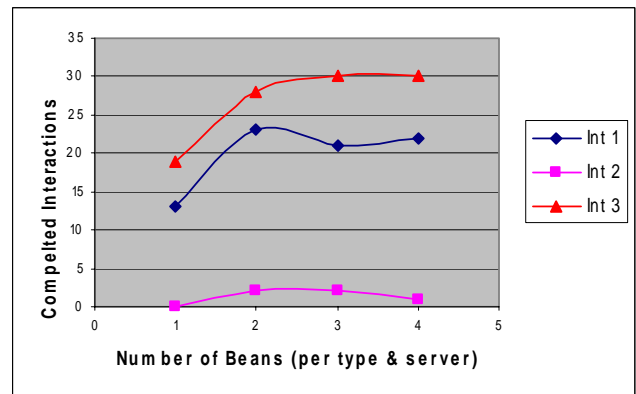


Figure 5b: Effects on Throughput of Bean Deployment

In this experiment after a third bean of each type was added to each server the gains made were negligible. This suggests that the extra beans are for the most part unused. There may however be differences between the different bean types and further experiments that modify the bean deployment for each type individually would be needed to test this.

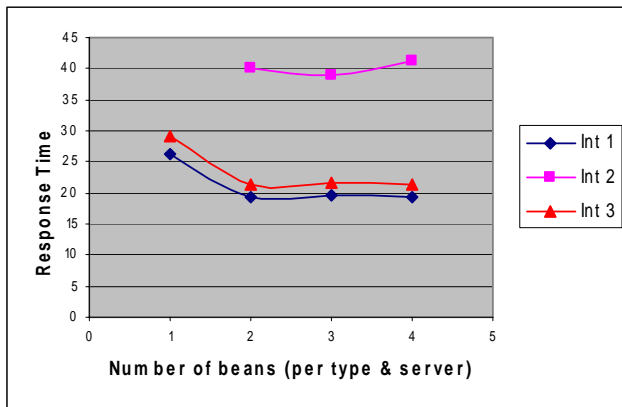


Figure 5c: Effects on Response Time from changing bean deployment

5 Conclusions

For EJB Systems like many other large systems it is very difficult to estimate performance given the profusion of factors that influence its performance. For this reason a simulation model that represents the most important elements in the system can aid in making better architectural choices as well as help reveal trade-offs and difficulties in trying to guarantee performance in Enterprise-level EJB systems. With multiple interactions, beans, CPUs, users and so on, it is very difficult to generalise about the effects of many of the changes. In a multi-server environment it becomes less clear where exactly the bottleneck is. Much depends on the bottleneck and if the bottleneck of the system is not fully understood the effects of hardware or software changes can cause some unexpected results, even in a simulation environment where parameters such as *numberOfUsers* can be held constant.

Not only can this model be used to predict performance indicators for EJB systems, it can also be used to build intuition in users of how changes to a real EJB server would be likely to affect individual interaction (response) times and throughput and helps the user to ascertain which change's effects can more easily be generalised in terms of performance, and which need more careful analysis.

There are many simplifications in the model to prevent it being prohibitively expensive in terms of running time but it now needs to be verified that the model can still perform accurate tests in terms of real running EJB systems and to see if the simplifications made were reasonable ones or if certain levels of detail that are included in the model are redundant. Future work would require a more complete set of experiments modifying more of the system resources such as the number of threads, I/O speeds, memory, etc. In addition it is necessary that the model be verified against a distributed, enterprise-level EJB system.

6 Acknowledgements

The support of the Informatics Research Initiative of Enterprise Ireland is gratefully acknowledged

7 References

- [1] <http://java.sun.com/products/ejb/>
- [2] <http://www.hyperformix.com/Default.asp?Page=74>
- [3] <http://ptolemy.eecs.berkeley.edu/ptolemyII/ptII3.0/ptII3.0.2/doc/design/ptIIIdesign2-software.pdf> (Actor Package)
- [4] S. Ran, P. Brebner, I Gorton, "The Rigorous Evaluation of Enterprise Java Bean Technology", a long paper in the 15th International Conference on Information Networking (ICOIN-15), 2000.
- [5] Ian Gorton, Anna Liu, "Evaluating the Performance of EJB Components". IEEE Internet Computing 7(3): 18-23 (2003).
- [6] Emmanuel Cecchet, Julie Marguerite, Willy Zwaenepoel, "Performance and Scalability of EJB Applications". (OOPSLA'02).
- [7] C. M. Lladó, J. Lüthi, P. G. Harrison, "Studying Sensitivities of an EJB Performance Model" (MASCOTS'02).
- [8] Samuel Kounev, Alejandro Buchmann, "Performance Modeling and Evaluation of Large-Scale J2EE Applications" (CMG'03).
- [9] D. J. Lilja – "Measuring Computer Performance" 2000, ISBN: 0-521-64105-5.
- [10] D. McGuinness, L. Murphy and A. Lee, "Issues in developing a simulation model of an EJB system", Computer Measurement Group 2004 International Conference (CMG 2004), Las Vegas, Nevada, Dec. 5-10, 2004.
- [11] Ed Roman "Mastering EJB 2", 2002, ISBN 0-471-41711-4