Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

# A Scalable Application Server On Beowulf Clusters

A thesis presented in partial fulfilment of the requirement for the degree of

**Master of Information Science**

At Albany, Auckland, Massey University
New Zealand

Supervised by: **Dr. Chris Messom**

**Michael Zhiyong Zhou**

2005

# Abstract

Application performance and scalability of a large distributed multi-tiered application is a core requirement for most of today's critical business applications.

I have investigated the scalability of a J2EE application server using the standard ECperf benchmark application in the Massey Beowulf Clusters namely the Sisters and the Helix. My testing environment consists of Open Source software: The integrated JBoss-Tomcat as the application server and the web server, along with PostgreSQL as the database. My testing programs were run on the clustered application server, which provide replication of the Enterprise Java Bean (EJB) objects.

I have completed various centralized and distributed tests using the JBoss Cluster. I concluded that clustering of the application server and web server will effectively increase the performance of the application running on them given sufficient system resources. The application performance will scale to a point where a bottleneck has occurred in the testing system, the bottleneck could be any resources included in the testing environment: the hardware, software, network and the application that is running.

Performance tuning for a large-scale J2EE application is a complicated issue, which is related to the resources available. However, by carefully identifying the performance bottleneck in the system with hardware, software, network, operating system and application configuration, I can improve the performance of the J2EE applications running in a Beowulf Cluster. The software bottleneck can be solved by changing the default settings, on the other hand, hardware bottlenecks are harder unless more investment are made to purchase higher speed and capacity hardware.

# Acknowledgement

My greatest appreciation goes to my supervisor: Dr Chris Messom, who has provided endless support both technically and morally. His broad and insightful understanding of the technology involved, his willingness to research emerging technology, his concise yet plain explanation to technically challenging concepts has made my learning process a pleasant experience.

Thanks to Dr Martin Johnson, Mr Andre Barczak and other people who have contributed greatly developing the Massey Beowulf computers: the Sisters and the Helix (the supercomputer) and made it available to us. Of course, thanks to the Allan Wilson Centre for Molecular Ecology and Evolution (AWC) for funding the development of the supercomputer.

Thanks to support from the system administrator James and the manager Lorri.

Also thanks to Massey University for the award of a Vice-Chancellor's Masterate Scholarship to support my study financially.

I must also thank the people working in the Open Source communities that have provided high quality open source software for my study, especially to the people working on Linux operating system, JBoss Group, Apache Tomcat Project and PostgreSQL database.

Finally, thanks to my family for their understanding and support. They are my source of energy and pleasure.

# Table of Contents

**ABSTRACT** .......... **II**

**ACKNOWLEDGEMENT** .......... **III**

**TABLE OF CONTENTS** .......... **IV**

**TABLE OF TABLES** .......... **VII**

**TABLE OF FIGURES** .......... **IX**

**TABLE OF ABBREVIATIONS** .......... **XI**

**CHAPTER 1: INTRODUCTION** .......... **1**

1.1 INTRODUCTION .......... 1
1.2 MOTIVATION OF SCALABILITY STUDY FOR DISTRIBUTED APPLICATIONS .......... 1
1.3 TODAY'S TECHNOLOGY SUPPORT FOR SCALABLE APPLICATION .......... 2
1.4 SIGNIFICANCE OF MY STUDY .......... 3
1.5 OVERALL STRUCTURE OF THE THESIS .......... 4
1.6 SUMMARY .......... 6

**CHAPTER 2: BACKGROUND KNOWLEDGE** .......... **7**

2.1 INTRODUCTION .......... 7
2.2 COMPUTING TECHNOLOGY FOR A DISTRIBUTED SYSTEM .......... 7
2.3 COMPARING J2EE WITH THE COMPETING TECHNOLOGIES .......... 9
*2.3.1 CORBA* .......... *9*
*2.3.2 J2EE* .......... *11*
*2.3.3 .NET* .......... *12*
*2.3.4 Comparing J2EE with CORBA* .......... *14*
*2.3.5 Comparing J2EE with .NET* .......... *16*
2.4 CURRENT STUDY OF THE J2EE APPLICATION SERVER PERFORMANCE .......... 18
2.5 MY RESEARCH APPROACH .......... 22
2.6 SUMMARY .......... 23

**CHAPTER 3: HARDWARE FOR THE TEST** .......... **25**

3.1 INTRODUCTION .......... 25
3.2 INTRODUCTION OF SUPERCOMPUTERS AND BEOWULF CLUSTERS .......... 25
3.3 MASSEY BEOWULF CLUSTER .......... 27
3.4 SUMMARY .......... 29

**CHAPTER 4: SOFTWARE FOR THE TEST** .......... **30**

4.1 INTRODUCTION .......... 30
4.2 WHY CHOOSE OPEN SOURCE .......... 30
4.3 OVERVIEW OF OPEN SOURCE SOFTWARE FOR THE J2EE APPLICATION SEVER .......... 31
4.4 JBOSS APPLICATION SERVER .......... 33
*4.4.1 JBoss structure based on JMX standard* .......... *33*
*4.4.2 JBoss Clustering & Naming service* .......... *34*
4.5 CHOOSING THE DATABASE .......... 36
4.6 CHOOSING ECPERF AS THE TESTING TOOL KITS .......... 38
*4.6.1 Why choose ECperf* .......... *39*

4.7 SUMMARY ........ 42

**CHAPTER 5: TEST DESIGN ........ 43**

5.1 INTRODUCTION ........ 43
5.2 TWO TYPES OF THE TESTING ARCHITECTURE ........ 43
*5.2.1 Centralized workload architecture ........ 43*
*5.2.2 Distributed workload architecture ........ 44*
5.3 TESTING PROGRAMS AND RELATED CONFIGURATION ........ 45
5.4 TEST DESIGN FOR SISTERS ........ 46
*5.4.1 Type of planned test ........ 46*
*5.4.2 Two hardware architectures for testing the sisters ........ 47*
*5.4.3 Test design for JVM test ........ 49*
*5.4.4 Test design for Clustering of Session Beans ........ 52*
*5.4.5 Test design for Cluster of all* EJB ........ *53*
*5.4.6 Test design for two databases ........ 54*
*5.4.7 Test with Two partitions and two databases ........ 56*
5.5 TEST DESIGN FOR HELIX ........ 58
*5.5.1 Type of planned test ........ 58*
*5.5.2 Test design for JVM test ........ 59*
*5.5.3 Test design for using the default DB pooling in JBoss ........ 60*
*5.5.4 Test design using optimised database pooling in JBoss cluster ........ 61*
5.6 CONCLUSION ........ 61

**CHAPTER 6: TEST ON SISTERS ........ 63**

6.1 INTRODUCTION ........ 63
6.2 TEST WITH DIFFERENT JVM HEAP SIZE VALUE ........ 63
6.3 TEST WITH CLUSTERING OF ONLY SESSION BEANS ........ 68
*6.3.1 Preliminary tests using the default connections in PostgreSQL ........ 68*
*6.3.2 Test by using optimised connections in PostgreSQL ........ 70*
6.4 TEST WITH CLUSTERING OF ALL EJB ........ 74
6.5 TEST WITH FIRST-AVAILABLE LOAD POLICY FOR CLUSTERING OF ALL EJB ........ 76
6.6 TEST WITH TWO DATABASES ........ 80
6.7 TEST WITH TWO PARTITIONS AND TWO DATABASES ........ 82
6.8 TEST BY DISABLING THE LOG FILES WITH TWO PARTITION AND TWO DATABASES 85
6.9 DISCUSSION OF THE SISTERS RESULT ........ 87
6.10 SUMMARY ........ 89

**CHAPTER 7: TEST ON HELIX – THE SUPERCOMPUTER ........ 90**

7.1 INTRODUCTION ........ 90
7.2 JVM HEAP SIZE TEST ........ 90
7.3 TEST USING THE DEFAULT DATA SOURCE POOLING VALUES ........ 94
7.4 TEST USING OPTIMISED DATABASE POOLING ........ 96
7.5 SUMMARY ........ 98

**CHAPTER 8: PERFORMANCE ANALYSIS AND DISCUSSION ........ 100**

8.1 INTRODUCTION ........ 100
8.2 SCALABILITY ANALYSIS FOR SISTERS AND HELIX ........ 100
8.3 BOTTLENECK ANALYSIS ........ 104
8.4 PERFORMANCE TUNING FOR THE CURRENT TESTING SYSTEM ........ 106
8.5 FURTHER PERFORMANCE IMPROVEMENT DISCUSSION ........ 110
*8.5.1 Scaling the database ........ 110*

*8.5.2 Scaling the JBoss application server* ........ *112*
8.6 POSSIBLE USE FOR COMMERCIAL APPLICATION ........ 115
8.7 SUMMARY ........ 117
**CHAPTER 9: CONCLUSION ........ 119**
9.1 INTRODUCTION ........ 119
9.2 CONCLUSION ........ 119
*9.2.1 Contributions* ........ *119*
*9.2.2 Conclusion* ........ *120*
9.3 FUTURE WORK ........ 121
**REFERENCE: ........ 124**

# Table of Tables

Table 2.1: Comparison of basic features of J2EE and .NET

Table 2.2: Comparison of more critical features of J2EE and .NET

Table 6.1: Throughput of ECperf as function of the txRate

Table 6.2: JVM Heap Value vs. Maximum TPS Test

Table 6.3: Final Transaction output VS. txRate. (JBoss-Number = 1)

Table 6.4: Final Transaction output VS. txRate. (JBoss-Number = 1)

Table 6.5: Testing result of cluster all EJB on Sisters

Table 6.6: testing result of cluster all EJB using First Available load-balancing

Table 6.7: test result with 2 databases

Table 6.8: Transaction output with 2 Partitions and 2 Database systems

Table 6.9 is the transaction output with 2 partitions and 2 databases and disabled log

Table 7.1: JVM Heap Value VS. TPS Test

Table 7.2: Transaction output VS. JBoss Number

Table 7.3: Transaction Output VS. JBoss number in Helix

Table 8.1: Transaction Output in Sisters and Helix (JBoss=1)

Table 8.2: Transaction Output in Sisters and Helix (JBoss=2)

# Table of Figures


Figure 2.1: The CORBA object invocation mechanism

Figure 2.2: the J2EE application architecture

Figure 2.3: .NET Framework

Figure 4.1: The ECperf Architecture

Figure 5.1: Example configuration for the Centralized Workload

Figure 5.2: Example configuration for the Distributed Workload

Figure 5.3: Architecture for Centralized workload using Sisters

Figure 5.4: The architecture for distributed workload using Sisters

Figure 5.5 The hardware architecture for JVM heap value test in Sisters

Figure 5.6: Cluster of only Session Beans

Figure 5.7: Cluster of all Enterprise Beans

Figure 5.8: Clustering all EJB with two databases

Figure 5.9: Test with two partitions and two databases

Figure 5.10: EJB Replication with 2 JBoss Partition & 2 Databases

Figure 5.11: The hardware architecture for JVM heap value test in Helix

Figure 5.12: The hardware architecture for ECperf Test in Helix

Figure 6.1: Throughput as a function of the txRate (-Xmx = 180MB)

Figure 6.2: Maximum throughput as a function of the JVM Heap Size

Figure 6.3: The Transaction Output (TPS) VS. Client Number

Figure 6.4: The Transaction Output (TPS) VS. JBoss Number

Figure 6.5: Transaction Output VS. JBoss number when clustering all EJB

Figure 6.6: Transaction output VS. JBoss number

Figure 6.7: Transaction Output using the distributed architecture

Figure 6.8 Transaction Output with 2Partitions and 2 Databases

Figure 6.9: Test with 2 partition, 2 DB & disable log file.

Figure 6.10: All testing results for the sisters

Figure 7.1: Maximum throughput as a function of the JVM Heap Size

Figure 7.2: Transaction Output as a function of the JBoss number

Figure 7.3: Transaction Output VS. JBoss number in Helix

Figure 8.1: Helix and Sisters transaction output with one Jboss

Figure 8.2: Helix and Sisters transaction output with 2 Jboss

Figure 8.3: Example C-JDBC architecture

Figure 8.4: RAIDb-0 example

Figure 8.5 RAIDb-0-1 example

Figure 8.6 Architecture with JBoss and PostgreSQL cluster

# Table of Abbreviations

| | |
|---|---|
| ANSI | American National Standards Institute |
| API | Application Programming Interface |
| BBop | Benchmark Business Operation |
| BSD | Berkeley Software Distribution |
| CCM | CORBA Component Model |
| CICS | Customer Information Control System |
| CLR | Common Language Runtime |
| CMP | Container-Managed Persistence |
| COM | Component Object Model |
| CORBA | Common Object Request Broker Architecture |
| COTS | Commercial off-the-shell |
| CPU | Central Processing Unit |
| CSIRO | Commonwealth Scientific & Industrial Research Organization |
| DBMS | Database Management System |
| DCOM | Distributed Component Object Model |
| EJB | Enterprise JavaBeans |
| HPC | High Performance Computing |
| HTTP | Hypertext Transfer Protocol |
| IMS-TM | Information Management System Transaction Manager |
| INRIA | French National Institute For Research In Computer Science And Control |
| J2EE | Java 2 Platform, Enterprise Edition |
| JDBC | Java Database Connectivity |
| JDK | Java Development Kit |
| JMS | Java Message Service |
| JMX | Java Management Extensions |
| JNDI | Java Naming and Directory Interface |
| JRE | Java Runtime Environment |
| JSP | Java Server Pages |
| JVM | Java Virtual Machines |
| LAN | Local Area Network |
| MPI | Message-passing Interface |

| | |
|---|---|
| MPP | Massively Parallel Processing |
| ODBC | Open Database Connectivity |
| OMA | Object Management Architecture |
| OMG | Object Management Group |
| ORB | Object Request Broker |
| PBS | Portable Batch System |
| PHP | Hypertext Preprocessor |
| PVM | Parallel Virtual Machine |
| RAIDb | Redundant Arrays of Inexpensive Database |
| RAM | Random Access Memory |
| RDBMS | Relational Database Management System |
| RMI-IIOP | Remote Method Invocation Over Internet Inter-Orb Protocol (Rmi Over Iiop) |
| SARs | Storage Area Network |
| SFTP | Secure File Transfer Protocol Message-passing Interface |
| SMP | Symmetric Multiprocessing |
| SOAP | Simple Object Access Protocol |
| SPI | service provider interface |
| SSH | Security Shell |
| SUT | System Under Test |
| UDDI | Universal Description Discovery And Integration |
| UNIX | Uniplexed Information and Computing System. (It was originally spelled "Unics.") |
| WAN | Large Area Network |
| WSDL | Web Services Description Language |
| XML | Extensible Markup Language |

# Chapter 1: Introduction

## 1.1 Introduction

This thesis presents a study of the performance and scalability of J2EE applications. In particular, I concentrate on the application server, which is the core component of the J2EE architecture. I use a cluster of JBoss application servers to test how the scalability and performance of a J2EE application is effected.

In this introductory chapter, I start with the motivation of the scalability study and explain why it is important in the business world. Then I give some brief technical review about how scalability can be achieved using current available hardware and software. I explain why my particular study is useful and finally give an overview of the contents in each chapter.

## 1.2 Motivation of scalability study for distributed applications

Large-scale distributed systems are becoming increasingly important in the world, especially with online business activities. The Internet has greatly improved the accessibility to online businesses, and the increased accessibility has promoted ever-increasing e-commerce applications. The performance and scalability of an application is critical for a successful business, as a business application needs to have high performance to achieve competitive advantages over their competitors.

Performance can refer to many aspects, such as scalability, availability, fault tolerance and load balancing. I am particular interested in the scalability of an application. A scalable application has the capacity to serve additional users or transactions without fundamentally altering the application's architecture or program design. If an application is scalable, you can maintain steady performance as the load increases simply by adding additional resources such as servers, processors or memory.

The two most common types of scalability that can be applied to affect the overall application performance are:

- Horizontal scalability: Adding more servers (web, application or database servers) to improve performance.
- Vertical scalability: Adding more physical resources (memory, processors or network cards) to a existing server to improve performance.

The key point of scalability is to decide how well an application will perform when the size of the problem increases. Scalability is not only critical to maintain current system functionality in a changing workload, but also a key factor to guarantee the system can keep up with the growth potential and has the ability to scale to meet future user's demand.

## 1.3 Today's technology support for scalable application

Today's technology has provided high-quality hardware and software to support the development and deployment of applications with good scalability and high performance.

For the computer hardware, we have consistently increasing computing power with the CPU speed doubling every 18 months, while the price of a personal computer is gradually getting cheaper. Various architectures built on PCs have provided fundamental support for high performance computing.

Supercomputers, which are the most powerful computers in the world, are getting more powerful. Beowulf Clusters, which are built using the Commercial off-the-shelf (COTS) components such as PCs, are gradually becoming more important in the supercomputer field [7]. A major merit of a Beowulf Cluster is its significant cost advantages over traditional mainframe supercomputers with similar computing capacity.

The wide adoptions of fast network connections, for either local or large area networks, as well as the Internet technology have enabled reliable communication facilities to support high performance applications. Combined with the supercomputer and the reliable Internet connections, it is much more practical to build a GRID [20], a network of supercomputers using today's technology.

Software has been developed to take advantage of the hardware architecture to achieve high performance and scalability. Using a cluster of application servers for a J2EE application, the application server components such as an EJB can be replicated across a cluster of application server machine. By load balancing the client request to members in the application server cluster, each client can interact concurrently with local copies of the same EJB component. This results in increased accessibility to computing power, thus, an increased application performance and scalability can be achieved.

I am going to investigate the J2EE application performance using open source software. JBoss, the leading open source application server has recently introduced cluster support, which I will use for my study.

## 1.4 Significance of my study

I am going to investigate application scalability using open source software running in the Beowulf Cluster. The advantage of this approach is that I have total control of the resource, because the Beowulf Cluster was built and maintained by our department in the Massey University, and the open source software can be used free of charge with access to the source code.

From a business point of view, I am using one of the most cost effective combined hardware and software for running J2EE applications. Building a Beowulf Cluster cost only 5% to 20% of total cost compared with traditional mainframe supercomputers with the same computing power [21]. The JBoss application server is

free of charge but with most of the features a leading commercial application server provides. PostgreSQL is the most advanced open source database. By running J2EE applications using a Beowulf cluster as the hardware, the JBoss cluster, PostgreSQL as software, I can expect good scalability results. A good scalability result means that the system could be very useful for developing and deploying cost effective commercial applications.

## 1.5 Overall structure of the thesis

The thesis is organised into the following nine chapters:

**Chapter 1** introduces the overall structure of the thesis. I start with the motivation for the scalability study, followed by current hardware and software technology that can be used to build scalable applications. I then give reasons why my particular approach is useful, and finish with the overall thesis structure.

**Chapter 2** presents some of the background knowledge necessary for understanding my study. Three of the most important architectures for building large-scale distributed applications are presented and compared, this information helps to identify why I chose the J2EE architecture for my study. I give some of the related literature review and also state my research hypothesis.

**Chapter 3** gives a detailed description of the hardware architecture of my study. I use the Beowulf Cluster computers in Massey University for my performance study. Starting with the general architectures of various high performance supercomputers, the advantages of cluster-based system are discussed. At last, the helix and sisters clusters in Massey University are introduced in details.

**Chapter 4** covers the software used in the study. I have chosen all software from open source, which I am particularly interested in. I cover the software for running a distributed applications based on J2EE technology. To be more specific, I give some detail about why I choose the integrated JBoss-Tomcat as the application server and web server, the PostgreSQL as the database and the ECperf as my testing application.

**Chapter 5** describes the details of the test design for both the Sisters and Helix. I start with different types of hardware architecture I will use, followed by some detailed information about how to run various test programs in the system. The last part gives my preliminary design selections on the type of test, and briefly identifies the reason for that selection.

**Chapter 6** gives detailed testing procedures for my study in Sisters. I have done various tests based on different hardware architecture, software and the application configurations. For each type of test, I present with details about the test design and implementation procedures. Followed by a test result, Analysis and discussion on these results reveal several important conclusions.

**Chapter 7** gives detailed testing procedures for my study in Helix. Again, I have followed a similar approach used for the Sisters. But the Helix concentrates on some different aspect of my study and reveals some different results as compared with the Sisters.

**Chapter 8** covers the further analysis and discussions based on the results obtained on both Sisters and Helix. I have given a broader view on how to further improve the performance and scalability of my current system. A higher level of discussions about how to improve the current implementation and use better software features and hardware architecture are discussed.

**Chapter 9** gives the conclusion to my study. Based on the analysis of testing results in the previous chapters, I make my final conclusions and show some of the contributions made by my study. I also anticipated the future work in my research field.

## 1.6 Summary

---

I have introduced the overall structure of the thesis. Firstly, I gave the motivation for the scalability study, followed by the technical support that can be used for building a scalable application. I then show why my particular approach is useful. Finally, I listed the major topics of each chapter in the thesis.

# Chapter 2: Background Knowledge

## 2.1 Introduction

This chapter gives an overview of the most important background knowledge that is necessary for my study. First of all, a brief discussion of the current technology for building distributed applications, followed by a comparison of merits and weakness of each approach is given. The conclusions derived from these comparisons gives a good foundation for why I chose the open source application server JBoss as the candidate for study. After that, the performance issues especially the scalability of an application is presented followed by the literature review of the current study of J2EE middleware technology and application performance.

## 2.2 Computing technology for a distributed system

Historically, building a high quality distributed application was a challenging task for a developer, because a lot of tasks need to be addressed.

In each local site, there are often different operating systems. Even with the same operating systems, you have different versions (like Microsoft Windows) or just different flavours (such as Linux) of the same operating systems.

Building a distributed enterprise application on top of the possible heterogeneous operating systems connected with a network needs a lot of effort. For a particular application such as a banking system, you have to consider many issues, which are specific to this application.

Here is a list of possible problems that the developers have to consider when building large business systems [1].

- Remote method invocations: networks method to connect a client and server.
- Naming: To allow entities to be looked up and shared by the whole system.
- Load balancing: Direct client to a server with the lightest load.

- Transparent fail-over: Can a client be rerouted to other servers without interruption of the service? Then, how long will than taken?
- Back-end integration: How to integrate new data with legacy systems that already exist?
- Transactions: How to make the database access smooth, without deadlock, and recovery from transaction failure?
- Clustering: If one server crashes, if its state is replicated across all servers, the client can use a different server.
- Dynamic redeployment: Can software of the system be upgraded without shutting down the machine?
- Clean shutdown: If you have to shut down a server, can you do it in a clean and smooth manner so that the current clients are not interrupted?
- Logging and auditing: In case of failing, is there a log that I can consult to determine the cause of the problem?
- Threading: How to code for multiple client requests simultaneously.
- Object life cycle: How to manage the object creates and destroys cycle inside the server?
- Resource pooling: How can resources inside a server been pooled for reuse by other clients?
- Security: How to ensure only authorised users perform operations that they have rights to perform?
- Caching: How can you store frequently accessed data in the server's memory to avoid repeated retrieval from the database?
- And many more.

Taking a layered approach, all the services in the above list can be categorised as Middleware services [2]. The middleware services provide a set of software that sit between various operating systems and higher-level distributed applications.

Without a properly available infrastructure for building a distributed application, a company usually needs to build their own middleware to meet their specific application requirement. Companies that build their own middleware risk setting themselves up for failure. High quality middleware is extremely complicated to build

and maintain, requires expert-level knowledge, and is completely orthogonal to most companies' core business.

It has been a number of years now since the idea of multi-tier server-side deployment was first introduced. Since then, many middleware services have begun to appear in the market, such as the IBM's IMS-TM (Information Management System Transaction Manager) transaction processing monitor [22] that provides some general middleware service and the CICS (Customer Information Control System) [23] transaction monitor that was made into a suite of middleware products to provide middleware services for web applications.

Middleware such as the transaction monitors provide useful middleware support, but these non-standard technologies have big limitations of addressing only particular problems and are too hard to solve general problems. A standard architecture for server-side components is required to overcome this problem. This architecture needs to define the interface between the application server and the components contained inside the server. The developer should be able to focus on the business logic of the problem to be solved and not need to worry about the middleware services such as resource pooling, networking, security, and so on.

The goal of standardization is for rapid development of server-side deployments, allowing the development of existing middleware but still building portable server-side components. Currently, three of the most important architectures for building distributed applications are: CORBA, J2EE and .NET. The following section will give a brief comparison of these technologies.

## 2.3 Comparing J2EE with the competing technologies

---

### 2.3.1 CORBA

**CORBA: the original architecture**

The first version of the Common Object Request Broker Architecture (CORBA 1.0) was released in October 1991. It is a unifying standard for writing distributed object systems. This standard is completely neutral with respect to operating system, language, network and vendor [3].

The purpose of CORBA is to build a high level standard that is generally available for any distributed applications. CORBA essentially has three parts: A set of invocation interfaces, the object request broker (ORB), and a set of object adapters.

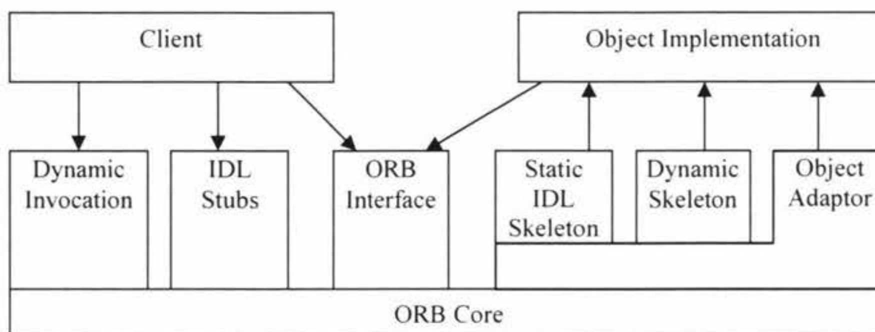


*Figure 2.1: The CORBA object invocation mechanism*

Figure 2.2 [3] displayed the structure of the object invocation in CORBA. To make a request, the client can use the Dynamic Invocation interface (the same interface independent of the target object's interface) or an OMB IDL stub (the specific stub depending on the interface of the target object). The client can also directly interact with the ORB for some functions [3].

The object Implementation receives a request as an up-call either through the OMG IDL generated skeleton or through a dynamic skeleton. The object implementation may call the Object Adapter and the ORB while processing a request or at other times

**CORBA3 with Component Model (CCM)**

At the end of year 2002,the new version of the CORBA 3 standard was completed, which addresses some of the issues of the Object Management Architecture (OMA) and CORBA Component Model (CCM) [3].

The CORBA Component Model (CCM) is a specification for creating server-side scalable, language-neutral, transactional, multi-user and secure enterprise-level applications. It provides a consistent component architecture framework for creating distributed n-tier middleware.

### 2.3.2 J2EE

The Java 2 Platform, Enterprise Edition (J2EE) specification was first released by Sun Microsystems in 1999; it is a complete development platform using Java for server-side programming. Built on top of the J2SE, it has defined a standard enterprise-class platform that will be used to build platform independent, portable, multi-user, and secure applications for server-side deployments written in the Java language.

The J2EE specification has defined a set of Java standard extensions that each J2EE platform must support. They are called the J2EE API. They include: JDBC, RMI-IIOP, EJB, Servlets, JSP, JMS, JNDI and JavaMail.

J2EE is a specification not a product. It has defined a standard way to build server side applications. In order to simplify complexity for building server-side applications, it uses a component-based architecture. There are currently over 60 J2EE application server products, which are implementations of the J2EE specification. Most of them are commercial application servers; such as the Web Logic, Web Sphere and SunONE, and the rest are open source application servers that include JBoss and JOnAS.

The idea behind the J2EE platform is to provide a simple, unified standard for distributed applications through a component-based application model. A complete server-side application is always quite large and complex, the J2EE model tries to break down an application into discreet modules that are each responsible for a specific task, making the application much easier to develop, maintain and understand. Java servlets, Java Server pages and Enterprise JavaBeans are all server-side components for building enterprise applications.

Server-side component architectures allow writing complex business applications without understanding detailed low level middleware services. Because the J2EE platform has provided built in support for all important middleware services such as transaction management, security issues, multi-threaded issues, resource management, dynamic redeployment and much more, the developer can concentrate only on solving the business logic problem.

Figure 2.1 is the basic J2EE application architecture. This architecture provides a very basic framework on how to build enterprise applications using the J2EE technology.

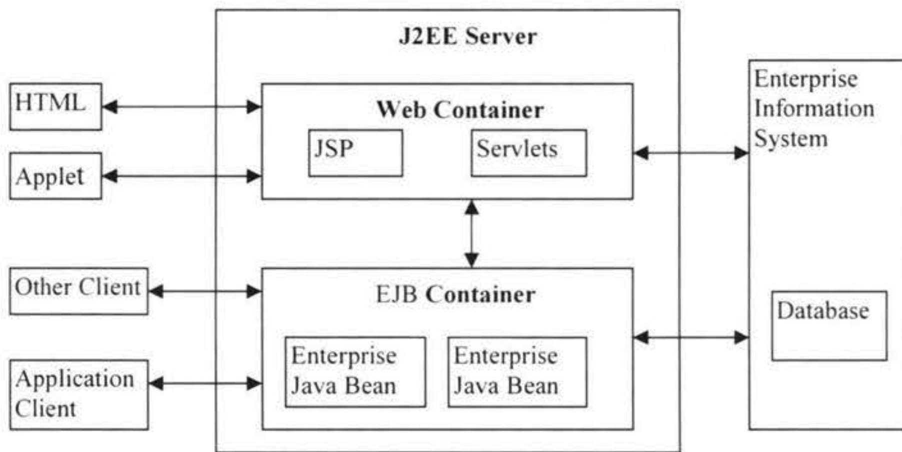


*Figure 2.2*: the *J2EE application architecture*

### 2.3.3 .NET

.NET is a mix of technology, standards and development tools from Microsoft that is used for developing next generation Internet enabled applications. .NET contains the following [36]:

- Windows.NET: windows operating system is the basic software in which all .NET infrastructures will be integrated.
- Office.NET: A new version of Microsoft Office that will have a new .NET architecture based on Internet clients and Web Services.
- ASP.NET is the latest version of ASP. It includes Web Services to link applications, services and devices using HTTP, HTML, XML and SOAP.

- Visual Studio.NET is the latest version of Visual Studio that incorporates ASP.NET, ADO.NET, Web Services, Web Forms, and language innovations for Visual Basic. The development tools have deep XML support, an XML-based programming model and new object-oriented programming capabilities.
- SQL Server 2000 is a fully web-enabled database; it has strong support for XML and HTTP, which are two of the main infrastructure technologies for .NET.
- Internet Information Services 6.0 has significant support for more programming to take place on the server, to allow the new Web Applications to run in any browser on any platform.

The .NET Framework has defined the basic infrastructure for programming in .NET. The major components of the .NET framework are shown in figure 2.3:

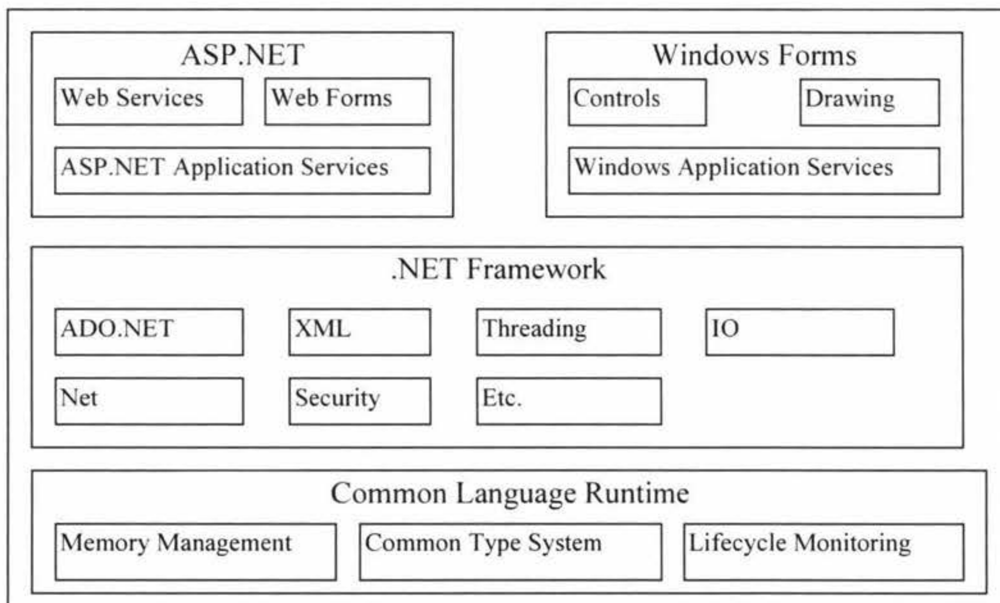


*Figure 2.3: .NET Framework*

At the base is the Common Language Runtime (CLR) that provides a run time environment for .NET applications. It provides services such as the memory management, garbage collection and cross-language support.

Built on top of the CLR are the services (include middleware services) that the .NET Framework provides to the application, such as the Threading, Input and Output, Security, XML and ADO.NET. For example, the ADO.NET provides service to access the database data.

The top layer provides the programming interfaces. Window Forms are a new way to create standard Win32 desktop applications based on the Windows Foundation Classes. Web Forms are a powerful, form-based user interface for develop web based application. Web service provides a mechanism for programs to communicate over the Internet using SOAP that provides functionality similar to COM and DCOM interaction with objects. ASP.NET is the .NET Internet interface portion; it comprises Web Forms and Web Services.

.NET has provided a complete set of solutions that incorporates the operating system, the web server, database engine, the basic application components, as well the tools for development, deployment and running a completed distributed application in the Internet.

### 2.3.4 Comparing J2EE with CORBA

CORBA is a great idea theoretically, but a large specification covering everything is also its weakness to being adopted by the community. CORBA is a high level specification, which lacks details of how to solve a particular type of problem as compared with J2EE.

The Object Management Group (OMG) invented the Common Object Request Broker Architecture (CORBA). It is a unifying standard for writing distributed object systems. This standard is completely neutral with respect to operating system, language, network and vendor [3].

CORBA is just a standard like EJB. CORBA-compliant products, such as Inprise's VisiBroker for Java, IONA'S orbixWeb, implement the CORBA specification.

Using CORBA has some advantages [3]:

- The CORBA is not controlled by one company but has been invented by a consortium of companies.
- CORBA is language-independent, which allows code written in several languages to communicate. This allows for easy legacy integration using CORBA.
- CORBA provides value-added services. The vendors of CORBA products can add optional functionality to enhance the deployments, such as persistence, security and transactions.

Disadvantage of using CORBA include: [3]

- The CORBA standard is slow moving. The openness of CORBA is a drawback for adopting new features. This makes CORBA standard less competitive comparing with other technologies such as Sun's J2EE and Microsoft's .NET.
- CORBA has a steep learning curve. For example, the CORBA specifications are thousands of pages and are quite challenging to master.
- Products developed under CORBA may have incompatible features. Because no one company controls the standard, each company may have different implementations for a CORBA product.

Based on the simple introduction, it's not hard to find out why CORBA standard are still not so widely adopted after so many years. On the other hand, the Java and the J2EE technology have becomes popular quickly.

- Java and J2EE is innovative technology targeting the business application market.
- J2EE has a strictly defined architecture for building multi-tier applications and the middleware that an application server must provide. There are J2EE products compatibility test suits to ensure that all vendors abide to the J2EE standard. But CORBA does not have these features.
- Building enterprise applications using J2EE compatible products can be accomplished rapidly. But CORBA programming is much more tedious with less middleware support.

### 2.3.5 Comparing J2EE with .NET

As CORBA is obvious a less competitive technology as compared with J2EE, the more interesting comparison is J2EE with .NET, two of the most popular technology for building enterprise applications.

.NET is essentially repackaging of existing Microsoft technology to provide distributed, Internet enabled applications and services. For building web-based application, .NET uses an architecture that is quite similar to J2EE.

Thinking about a layered approach, in the presentation layer, the J2EE uses Servlets, JSP, whereas .NET uses ASP.NET. For the business logic tier, the J2EE uses EJB, .NET uses .NET Managed Component. Table 2.1 is a simple comparison of the technical features between J2EE and .NET.

| **Feature** | **J2EE** | **. NET** |
|---|---|---|
| Type of technology | Standard | Product |
| Middleware Vendors | 60+ | Microsoft |
| Interpreter | JRE | CLR |
| Dynamic Web Pages | JSP | ASP.NET |
| Middle-Tier Components | EJB | . NET Managed Component |
| Database access | JDBC, SQL | ADO.NET |
| SOAP, WSDL, UDDI | Yes | Yes |
| Implicit middleware | Yes | Yes |

*Table 2.1: Comparison of basic features of J2EE and .NET*

From the comparison of the implementation features from J2EE and .NET, it can be concluded that they have a quite similar architecture to build web based enterprise applications. More analysis will give a clearer picture about them. Table 2.2 is a further comparison of features that are differences between J2EE and .NET platform.

| Feature | J2EE | .NET |
|---|---|---|
| Vendor's support | The entire industry | Microsoft |
| Platform | Proven platform | Rewrite of exist platform, risk for unknown factors |
| Code portability | Existing code can translate to web service | DNA code can not directly ported to .NET |
| Existing hardware support | Support existing hardware | Not complete supported |
| Portability of code | Good portability of code | Only run in windows system |
| Integrate with legacy system | Better integration using Java Connector Architecture (JCA) | Limited support using Host Integration Server 2000 |
| Supported operating systems | Any operating systems | Windows only |
| Programming language | Java is more mature and with more market share | C# is too new |
| Independent Software Vendor (ISV) and Consulting Company's support | Most of them will choose J2EE for customer's freedom of choice | Choose .NET only useful for Microsoft products user |
| Development tools | A few choices. Overall functionality is good | Visual Studio.NET has relatively better user interface |
| Programming model | More advanced model | Simpler model, likely more productive |

*Table 2.2: Comparison of more critical features of J2EE and .NET*

From the comparison results, these are clearer advantages of J2EE over .NET in a number of issues.

- J2EE has support from all major vendors, but .NET are locked in to Microsoft.
- J2EE is a proven platform for many critical applications, but .NET is a rewrite of an existing platform.
- J2EE can run in any operating systems, but .NET only runs using windows.
- J2EE is better integrated with the legacy systems than .NET.
- J2EE codes are portable between different vendors with only minimal work.
- There is more Independent Software Vendors (ISV) and Consulting Companies to support J2EE as compared with .NET.

My conclusion is: The advantages of J2EE outweigh those offered by Microsoft's .NET for building distributed enterprise application.

## 2.4 Current study of the J2EE application server performance

The popularity of the Internet has introduced huge potential for traditional client-server applications. Access restrictions of the traditional Local Area Network (LAN) or Large Area Network (WAN) on different geometrical areas and the user groups have been greatly eliminated by using the Internet.

The Internet has brought huge opportunities for businesses. Businesses going on-line have much better chance for increasing income. With the easy access to Internet anytime and anywhere in the world, a customer can interact directly with a company; activities such as the on-line transactions will bring direct benefit for both customers and the service providers.

With the strong need for providing web service solutions in the Internet, Companies in the computing industry are working hard to supply all sorts of services to enhance e-business solutions. A suitable infrastructure is the most important framework to enable a better web service.

Currently, the most widely used tools for building distributed multi-tiered applications are J2EE, .NET and CORBA, which have been discussed in the last section. I will concentrate on J2EE application server in my study.

In a typical four-tiered J2EE application that contains client, web server, application server and the database, the application server is the core part. It provides built in, otherwise would be very complicated middleware services such as concurrency, transaction, security and persistence management to support business logic of an application.

Enabling J2EE application servers with many middleware services will simplify the process of developing enterprise applications, because the user can use the services provided by the application server rather than reinvent the wheel. On the other hand, this will add overhead to the overall performance of the application. For example, enabling Entity Bean services will greatly simplify the operation of a database,

instead of using JDBC to explicitly access a database following a tedious and sometimes error prone procedures, the developer needs only do an object lookup, and then perform some object method operation. But making the EJB container handle the complexity of the JDBC tasks will add overhead and is inefficient compared with direct JDBC operations.

Since the first publication of the J2EE specification, there have been enormous interest in the computer industry and the community for adopting this Java based middleware standard that has greatly simplified the development and deployment of multi-tiered enterprise applications and web services. Many standard compliant application servers have emerged in the market place, the strong competition has promoted the vendors to provide high performance, cost-effective, standard-based, easily used and managed development environments incorporated with the application server and web server.

The overwhelming popularity and fast development of the web-based applications in the world has demonstrated both great potential and big challenges for any online businesses. The dynamic nature of the web site and unpredictable nature of the users distribution require a high performance and highly scalable architecture to support an application. The overall performance of a large-scale mission critical application is critical for achieving good level of user satisfaction.

There are some research activities that emphasise the performance and scalability of the J2EE application servers. Though, the application server is the most critical component in J2EE architecture, the performance of a J2EE application is a very complicated issue. The entire application behaviour is the combination of the behaviour of the hardware, the softwares that forms the J2EE platform and the application running on this platform.

The following section will identify and discuss various performance studies related to a J2EE application.

Individual vendors, companies and researches have developed their own testing programs to run on one or more application servers for their special testing purpose.

The Software Architectures and Component Technologies R&D Group from CSIRO has developed a test application called Stock-Online [4] which contains simple but realistic stock transaction systems. There are two different implementations of Session Bean and CMP Entity Beans that allow evaluation scalability and performance of different design choices for different application servers. Cecchet and Marguerite have taken a different approach by implemented an auction site with five versions of EJBs of different design [5], running each version against the same hardware and J2EE platform has revealed big performance gaps between different application implementation methods.

As individual's testing application is only valid for their own purpose, a more general testing program is necessary as a standard benchmark to test the performance of any Standard compliance application servers. A benchmark is normally a widely accepted test program used to compare the performance of hardware and/or software. When comparing benchmark results, it is important to know exactly what the benchmarks are designed to test. For example, Pennbench is a benchmark suite for embedded Java enabled handset devices [24]. Linpack is designed to test performance of a supercomputer [25]. SPECjvm98 is used to evaluate performance for the combined hardware and software aspects of the JVM client platform [26].

ECperf is a middle-tier benchmark designed to test the scalability and performance of application servers and the computer systems that is used to run them. ECperf is designed to reflect a complete heavy weighted, mission-critical real world application by incorporate four domains: Customer Domain, Manufacture Domain, Supplier Domain and Corporate Domain. It is widely accepted as an industrial standard testing program for application server. Many application server vendors have submitted their ECperf result to showcase the scalability and performance of their products since its first publication. More details of ECperf test will be described in chapter 4.

With different testing tools, it is much easier to evaluate performance of a J2EE applications and their supporting platform. The next question is how to improve the performance of J2EE applications.

As mentioned before, a J2EE application performance is related to factors including: the hardware to run the application, the operating system, the J2EE platform (include the Java Virtual Machine, the application server and the web server) and the application design and deployment.

The hardware that is running the whole software system is critical to the performance of a J2EE application. There are different approaches for increasing the performance and scalability of an enterprise application. There is some very basic hardware factors that might affect the performance of a Java program [6], such as: the RAM, cache memory, CPU power, and basic I/O facilities. For a big multi-threaded java program, the user must make sure the hardware is sufficient to support the program. For example, running ECperf test must make sure there is enough RAM during the running process, if many running threads need to be swapped out, the final output of finished transaction will be greatly affected.

There are different hardware technologies that can improve the performance, as well as increase the power inside a machine; there are two ways to increase the hardware power in order to build a more powerful high-performance computer. One of them is using multi-processors. There are many commercial available multi-processor machines with two, four, eight or even more processors in one machine that could greatly increase the computing power of a machine. This is especially true for J2EE application servers, which requires significant computing power. Another way is to build a computer cluster using multiple machines with high bandwidth connections between them. This method is extremely flexible for building highly scalable power computers. Building a clustering system using commercially available personal computers is the most cost-effective way that cost only a fraction compared with commercial computers with similar computing power. As a result, PC clusters are becoming the most widely accepted method to build even the most powerful computer systems in the world [7].

With sufficient hardware, the next major factor related to performance is the software; here I refer to all software, including the operating system, the J2EE platform and the J2EE application running on them. Most developers do not consider the Operating

system, but it is likely to become the bottleneck when running applications on servers [8].

All Java programs run on Java Virtual Machines (JVM), this includes the application servers written with Java Programming Language. Understanding the characteristics of the JVM are the preconditions for tuning the performance of JVM. There are studies on optimising Java code running on JVM with a performance optimised to the level that is comparable with the C/C++ code [9]. cJVM is an interest approach to provide a single JVM running on a cluster of nodes [27]. By running cJVM on a cluster can obtain improved scalability of Java Server Applications by distributing the application's work among the cluster's computing resources.

For the J2EE applications, there are many studies for the purpose of gaining better performance by following good design and implementation guidelines. There are J2EE design pattens [28] that give best practices for the design and implementation of a J2EE application. For a comparison of performance and scalability of different J2EE EJB implementations, five versions of an auction site were implemented [5]. The author concluded that the most important factor in determining performance is the implementation method, where EJB applications using Session Bean only as well as using Java Servlet only performs an order-of-magnitude better than most of the implementations using Entity Beans.

## 2.5 My research approach

Given the fact that the J2EE technology is still new in the market, and the limited number of research activities associated with the J2EE application server, I do not know how scalable a J2EE application in a clustered environment is. The ECperf benchmark results from the industrial leading J2EE application server provider are available, but these results only give the highest value of test against particular highly optimised hardware, operating system and a vendor's J2EE platform, which cannot provide scalability information of their J2EE application server. I want to investigate how scalable a J2EE application server is using a Beowulf Cluster as testing hardware and open source as software.

A very basic technique to increase available services is through data replication. Multi-copies of the same data in multiple machines will enable data to be more accessible to client and thus improve the performance and scalability of the application. Clustering has provided this service and so it has become more widely adopted by the J2EE application server vendors. My hypothesis is: for a properly implemented application server cluster, an application running on it should scale, giving enough hardware and software resources. Ideally, without the overheads such as communication between application servers and unbalanced load distribution, the performance should scale linearly.

I will exam my hypothesis by running a benchmark programme. I will use the J2EE platform as the infrastructure of my study. By using the Beowulf cluster in Massey University Albany Campus as hardware, open source J2EE application server and database as software, I can test how application server cluster can affect the scalability of the testing application program.

The J2EE server software must support clustering, which enables multiple application servers to work together in a distribute environment. There are many J2EE vendors that provide commercial J2EE application environment to support the design, implementation, test and deployment of distributed applications. On the other hand, there are open source application servers that are among the most popular ones in industry, though may not be the best.

I will follow consistently my approach and give the open source a priority when choosing my software. The selection process and main functionality of the application server, web server and the database will be given in chapter 4.

## 2.6 Summary

I have given a brief overview of the three most influential component based architecture: CORBA, J2EE and .NET. By comparing the merits and weaknesses of each approach, I concluded that J2EE is my best choice.

I then introduced how to build distributed applications using the J2EE technology. A literature review gives various studies of the performance of the Java technology with emphasis on the J2EE application server.

From these studies, I formed my research question of investigating the scalability of a J2EE application running on a cluster of JBoss application server. My approach for validating my hypothesis is through program test using the Beowulf cluster supercomputer as hardware, and a set of open source software to form my J2EE platform.

# Chapter 3: Hardware for the test

## 3.1 Introduction

This chapter covers the hardware architecture of the performance testing system as well as some related technologies. The hardware used for the performance test is Beowulf Clusters: the Helix and the Sisters in Massey University. The Helix is a large supercomputer cluster and the Sisters is a smaller cluster.

To help understand the Beowulf cluster, I have firstly reviewed various architectures for building powerful computers especially supercomputers, based on this information, I have discussed why clustering of PC is one of the best choices among these architectures. Lastly, the Massey University Beowulf cluster: the Helix and Sisters are presented with more detail, because they are used extensively for my performance test.

## 3.2 Introduction of supercomputers and Beowulf Clusters

A Supercomputer is a computer that performances significant faster than most of today's computers.

High Performance Computing (HPC) is constantly in need to address the problems that are hard for a general computer to complete in a reasonable period of time. Supercomputers were normally very expensive and were employed for specialized applications that require immense amounts of mathematical calculations. For example, weather forecasting requires a supercomputer. Other uses of supercomputers include animated graphics, fluid dynamic calculations, nuclear energy research, and petroleum exploration.

The supercomputer can be categorized differently according to different criteria. Three major types of supercomputer can be identified as the following according to how memory and processors are used in a system:

- Single memory and single processor: single CPU with one memory space
- Symmetric Multiprocessing (SMP): multiple CPUs that share a common memory
- Massively Parallel Processing (MPP): multiple CPUs each with it's own memory space.

In the list of the world's 500 most powerful supercomputers [7], only SMP and MPP still exist, it is impossible to create a single processor computer with one memory space that is powerful enough to be in the list.

A Beowulf Cluster is a MPP built using commodity hardware components, such as Personal Computers (PC) [10]. The first Beowulf cluster was made in National Aeronautics and Space Administrator (NASA) in American in 1993 [29]. At that time, they needed a supercomputer but could not afford to buy one, so they designed a system with 16 nodes of Intel 100 MHz DX4 processor. The researchers at NASA discovered that their Beowulf Cluster had the computing power to compete with its supercomputer contemporaries, such as the TMC C-5 and the Intel Paragon. Whereas the old system of supercomputing required a huge budget and highly specialized equipment, the Beowulf-Class System required a budget that might be allocated for a computer lab at a college. A Beowulf Cluster consists of a collection of off the shelf motherboards, processors, memory, and network support, and one computer with I/O capabilities

Beowulf Clusters have the following major advantages over traditional supercomputer [30]:

- Significant Lower total cost
- Short assembly time and extremely easy and cheap to expand
- Total control of system: the builder designs the hardware and chooses the software
- Easy to keep up with current technology with a flexible architecture
- Very stable and robust for the user and administrator

Some disadvantages of Beowulf clusters include:

- Communication could be slow compared with proprietary supercomputers
- No official system support, the builder is responsible for everything

The most important feature of a Beowulf Cluster is the cost advantage. The Centre for Advanced Computing Research at Caltech [31] reported in 1998 that a 70 Node Beowulf Cluster whose nodes peak at 1.066 Gflops cost a total of US$152,175, that is US$2174 per node. A typical configured supercomputer with similar processing power cost about $1.5 to $2 millions dollars.

The software used by a Beowulf cluster to support parallel computing is much more mature than before. Both Microsoft Windows and Linux can support Beowulf clusters. The software to perform Inter-process Communications between different machines is also available. A cluster system can use either Parallel Virtual Machine (PVM) or Message-passing Interface (MPI) to make the cluster perform as a single machine to the users. The limitation of the Beowulf cluster is that it only addresses parallel problems efficiently.

I conclude that the Beowulf cluster is both currently and will be in future the most cost-effective architecture for building a supercomputer. By comparing Beowulf clusters with the Internet, the Internet has made the information system accessible to the general public, and the Beowulf cluster has made the supercomputer accessible to most small organizations or even private people.

## 3.3 Massey Beowulf Cluster

Massey University Albany Campus in Auckland currently has two Beowulf Clusters: Sisters and Helix. The Sisters consist of 14 nodes connected with 100Mb/s fast Ethernet adapters. The server node is a dual 667MHz Pentium III with 1GB RAM, and the 13 client nodes using 667MHz Pentium III with 256MB RAM. All the nodes are connected to the server through a switch. The low bandwidth network could be a key bottleneck in the system.

The Helix is a much bigger cluster supercomputer; it ranked Number 304 in the world Top 500 November 2002 List with Linpack Rmax rating of 234.8Gflops [11].

The Helix consists of 66 nodes connected with fast Ethernet adapters. The server node is a dual Athlon MP2200 MHz AMD with 2GB RAM, and the 65 client nodes using dual Athlon MP2200 MHz AMD with 1GB RAM. All the clients are connected to switches that are then connected to the server using 1GB/s fast Ethernet adapters.

The state-of-the-art design and the careful selection of all the hardware parts and software used in the Helix system makes it extremely scalable in performance. The testing results from the Linpack benchmark (processor vs. Rmax rating) shows that the system scales almost linearly [11]. This is due to the high bandwidth, reasonable latency switching and the grid layout of the nodes. The grid architecture is also scalable for adding new processors with row and column size of 23 nodes, which will be equivalent to 1058 processors with an expected Linpack rating of 1.88 Tflops. This rating will put it in the $24^{th}$ positions in the November 2002 Top 500 list.

The basic software used for both Sisters and Helix is quite similar, though the Helix has a stricter security and usage control. The following is a simple description of the major software in the Helix.

Linux was selected as the operating system of Helix instead of the Microsoft Windows. As the major consideration of building cluster system is based on the price consideration, it makes more sense to choose the free software Linux rather than using Windows that has licence fees. As the overall performance of the Linux is comparable to the Microsoft Windows, it is reasonable that the majority of Beowulf cluster systems use Linux. Linux has a better reputation in security compared with MS Windows as it was derived from the UNIX.

Because the Helix is a supercomputer with MPP structure, it is very important to have software to deal with the Inter-Process Communication of the different processes running in the systems. MPI was selected to do this.

For a user to access the clusters anywhere through the Internet, the Security Shell (SSH) is used. To copy files, Secure File Transfer Protocol (Sftp) is used. These software, which use asymmetric cryptography, have greatly enhanced security, though are very convenient to use.

## 3.4 Summary

I have covered in general various architectures for building a supercomputer especially building the Beowulf cluster using the Commercial off-the-shelf (COTS) components. Two examples Beowulf Clusters, the Sisters and the Helix in Massey University were described in some detail.

I believe that the computing power of supercomputers have always been in demand and will continue in the future, because many problems need computing ability that general computers can not provide or might take too long to solve. People will need to rely on supercomputers to solve more and more complicated problems as computing power increases.

Beowulf Cluster's architecture has proved a relatively easy and cost-effective way to build powerful computers. Plus with a complete set of software such as the Linux operating system and MPI, it is becoming more and more powerful. In the near future, the Beowulf cluster will be available to a wider range of research institute, educational institute, the computer industry and even personal users, the availability of large computing power to the general public will become reality soon.

Massey Beowulf Clusters are used extensively in many research and study activities. For my study of scalable application servers, the Helix or Sisters are ideal candidates for providing the necessary hardware that is scalable in performance and support tight cooperation while keeping each individual in the system independent. More detailed information of the software used for my research work is described in the next chapter.

# Chapter 4: Software for the test

## 4.1 Introduction

This chapter will describe the software used for my test. One major consideration for selecting my software is based on the price. As open source software are free to use and are becoming viable in almost all aspects of the software industry, I will pay particular attention to that.

For a typical J2EE application, I have four layers including: the client, the web server, the application server and the databases. A client could be in different forms such as Web Browser, Wireless Devices, Applications, Applet, CORBA Clients through IIOP, or other system through web services technology such as SOAP, UDDI, WSDL, ebXML. I use open source software to form my J2EE platform and database. Using open source application servers on Beowulf Cluster hardware is a very cost effective way to run J2EE applications.

## 4.2 Why choose open source

For most people, Open Source means freedom of choice. There are many different types of Open Source, such as the GNU GPL, BSD, X Consortium and MPL. Though, each of them have differences in details regarding the licence and usage criteria, they do have some common criteria. The most important criteria of Open Source are as following [32]:

- Free redistribution of the original source code
- Source code must be available to the user
- Should allow modification and derived works on the original source
- No discrimination against any people and no discrimination against using the program in a specific field of endeavour.
- And other related criteria.

Though, Open Source has existed for over 20 years, it wasn't a viable choice until recently. With the widespread use of the Internet, the time for worldwide cooperation to create a best possible open source solution is coming. It is now possible to create open source software that is better than traditional proprietary closed model, in which only a few programmers can access and modify the source code.

There are many successful open source softwares already. One of the most successful open source projects is Apache Software Foundation [35]. Among many subprojects in this foundation, the Apache HTTP Web Server is the most widely used in the industry. The most popular application server in industry is JBoss with an annual download of over 2 millions last year [33].

I expect a much better future for open source software through open and cooperative work from people all around the world. Open source J2EE software is discussed in the following section.

## 4.3 Overview of open source software for the J2EE application sever

The open source server side J2EE software includes: web server, application server and the database. Here is the brief introduction to each of them.

**Application Server**

There are many commercial J2EE development and deployment environment such as IBM's Websphere, BEA'S Weblogic, Oracle's Oracle9i application server and Sun's Sun ONE Studio. These products provide standard-compliant completed development environment for the developer and also some level of user support. But all these services are provided with a high charge. On the other hand, there are open source application servers and web servers that provide most of the functionality, which a commercial J2EE platform supports.

The existence of the open source application server has attracted large groups of users from all around the world. Three of the most noticeable J2EE application servers are:

JBoss from the JBoss Group, JOnAS originated from France and OpenEJB. An open source application server can be combined with other open source software to form a complete J2EE platform.

JBoss has a fast growing community for both the developer and users, and the increasing numbers of developers have contributed greatly to the innovation and publication of new versions of JBoss with better features provided to the users.

JOnAS is an Open Source implementation of the J2EE 1.3 specifications and EJB2.0, developed within the open source ObjectWeb consortium. JOnAS is a pure Java™ implementation of this specification that relies on the JDK. The objective of the ObjectWeb is to develop and promote open source middleware software [34].

**Web Server**

There are different web servers, which support different functions. The most basic general web server is the HTTP web server, which support communications with a web browser client. Apache is the most famous HTTP web server. For J2EE applications, I need to use a servlet server. Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and Java Server Pages technologies. Tomcat is the most widely used open source servlet container that is also developed by the Apache Software Foundation [35].

For the convenience of the user, most vendors combined an integrated version of the application server and the web server together. For JBoss users, it is more convenient to use the pre-configured JBoss-Tomcat bundle rather that to download them separately and then to configure them.

**Database**

Nowadays, databases have become very general tools; there are many databases which support different levels of functionality. The range of databases include the easiest ones with only very basic database functionality to more complicated ones that can provide complete support for enterprise level applications.

Even for open source databases, there are various choices for the users. Some of the popular open source databases include: MySQL, PostgreSQL, SAP DB and Firebird.

## 4.4 JBoss application server

I choose JBoss instead of JOnAS simply because it has a much bigger user and developer community.

The objective of my study is the application scalability when using a Cluster of application servers. To realize that goal, I need to understand the architecture of the JBoss server as well as how to implement the different functionalities that JBoss support. I will first introduction the JBoss architecture based on JMX standard, then I will introduce the major features of the JBoss server followed by a detailed description of the JBoss Clustering that I will use extensively in my application, some basic knowledge of JBoss specific features in an J2EE applications will be presented.

### 4.4.1 JBoss structure based on JMX standard

The most important concept in the JBoss server is the JMX architecture, which forms the backbone of the JBoss application server [12].

**JBoss architecture**

JBoss was the first application server that implemented the JMX standard [13]. In the instrumentation level, it implements the micro-kernel architecture of MBean components, each of which implements different J2EE services or other server infrastructure components. In the agent level, JBoss implements a JMX MBean server, which act as the spine of the JBoss server. All the MBeans that provide different J2EE services need to register to this MBean server. In the distributed service level, a JMX HTML adaptor is available to allow access to the Mbean Server's MBeans using a standard web browser.

There are distinct advantages of implement the micro-kernel architecture instead of a monolithic application that contains all the J2EE services at any time. The JBoss

server is extremely adaptable and extendable to fit the users requirement. Adding a new service is a matter of plugging a new MBean to the JMX MBean server, to deploy this service, the administrator only needs to drop a XML file such as the jboss-services.xml that contains the MBean to the deploy directory of the JBoss server. Similarly, removing this file will undeploy the service.

I only introduce the JBoss services that I am interested with, such as the clustering and the farming services in the next section.

### 4.4.2 JBoss Clustering & Naming service

**Clustering in JBoss**

My test will run a cluster of JBoss servers in my Beowulf cluster for the purpose of the scalability and the performance study. To serve this goal, the following sections will introduce the basics of the JBoss clustering and farming service.

As I have described before, a Beowulf cluster is a group of computers that are connected with a high bandwidth network and performs as a single computer. Traditional use of the Beowulf Cluster is for parallel computing that can greatly improve the performance of any parallel program.

For a J2EE application, the concept of clusters have been extended to the software, for a cluster enabled application server or web server, I can run multiple copies of the server that can perform the same set of work simultaneously, by using a good load balancing policy for the clusters of application server, it is theoretically possible to get a linearly increasing scaling of the application servers.

I define a JBoss Cluster as a group of JBoss instance each running in different machines that are configured to accomplish a common goal. The major features that the current cluster implementation of JBoss Application Server supports are as following [38]:

- JBoss instances in the cluster find each other automatically.

- Fail-over and load balancing for many services, include: JNDI, RMI, Entity Beans and the Session Beans.
- HTTP session replication with the integrated web server, such as Tomcat or Jetty.
- Farming service that provides distributed cluster-wide hot-deployment. By deploying to the farm directory of one JBoss instance, it gets deployed to all JBoss nodes in the cluster.

By providing the cluster support to all the server side components, it is possible to get a highly scalable performance for applications running in the J2EE platform. A JBoss cluster introduces the concept of a partition of hardware and software. A partition is a group of JBoss instances that works cooperatively to provide the same service to the client of the application server. A cluster enabled Enterprise Java Bean can be replicated in a JBoss cluster, with the cluster wide JNDI services, smart proxy and the implementation of load balance policy in the JBoss server.

**HA-JNDI in JBoss**

The Java Naming and Directory Interface (JNDI) is an application programming interface (API) that provides naming and directory functionality to applications written in Java™ programming language [19]. It is defined to be independent of any specific directory service implementation. Thus a variety of directories can be accessed in a common way.

The JNDI architecture consists of an API and a service provider interface (SPI). Java applications use the JNDI API to access a variety of naming and directory services. The SPI enables a variety of naming and directory services to be plugged in transparently, thereby allowing the Java application using the JNDI API to access their services.

Java JNDI services are only available within the same JVM, thus for a distributed applications such as my clustered application server test, an extended JNDI model that can be used to lookup objects located in another JVM is necessary.

The default JNDI lookup in local machine will not work properly to support object lookup in the clustered environment. JBoss has implemented HA-JNDI, which is a global, shared, cluster-wide JNDI Context that client can use to lookup and bind an object [38]. Having a HA-JNDI on top of the local JNDI can provide fail-over and load balance to applications running in a JBoss Cluster.

A HA-JNDI property string for known servers can be specified as:

*Java.naming.provider.url=helixl:1100,helix2:1100,helix3:1100*

A HA-JNDI client can locate objects that are bound to a machine using the property string. Further more, if the property string is empty and multicast is supported in the system; automatic discovery through multicast call on the network can be performed using HA-JNDI.

**Farming service**

JBoss supports hot deployment of the assembled J2EE components, such as the EAR, JAR or WAR files. For a running JBoss application server, your can deploy a file or redeploy a file that has already been deployed by copying the file into the deploy directory of the JBoss server.

The farming service is a natural extension of the hot deploy feature to the cluster of JBoss, by copying a file to the server/all/farm directory of one JBoss instance in a cluster, the file will be deployed to all JBoss instances in the cluster.

## 4.5 Choosing the database

There are many open source databases, such as the PostgreSQL, MySQL, Firebird/Interbase and SAP DB. MySQL and PostgreSQL are the most widely used databases. Because of their different design goals from the very beginning, MySQL is more suitable for some applications, while PostgreSQL is more suitable for others depending on the users' requirements. My goal is to identify the major features and differences of them and decide which one is better for my application.

MySQL was originally developed and provided by MySQL AB, a virtual commercial organization [14]. It is an open source relational database management systems that extends the ANSI SQL92 standard. MySQL is considered to be the most widely used database on the Internet.

The design goal of MySQL is to deliver very fast, multi-threaded, multi-user, and robust SQL database that will be used for mission-critical, heavy-loaded production systems as well as single users [16].

Major features of MySQL include:

- Speed: Written in C and C++ with multi-threaded, optimised execution of queries to very fast B-tree disk tables with index compression.
- Portability: with support of all these major operating systems and easy to port data to another operating system using building in program.
- Ability to interface with any programming languages such as PHP, Perl, C/C+, Java, Python, and Tcl, makes it very popular choices for programmers.
- Easy of use: There are complete basic functionalities as well as the users and administrator's tools without many complicated features as compared with commercial databases such as Oracle and DB2. This makes it relatively easy to learn and use and satisfies most users requirements.

PostgreSQL [17] is an open source extension of Postgres, a research project of an Object-Relational Database Management System at the University of California at Berkeley. Due to its long academic history and its rich set of functionality, it is considered the most advanced open source DBMS. The PostgreSQL project is under very active development worldwide from a team of open source developers and contributors.

Major features of PostgreSQL include [15]:

- Standard compliant: Complete support of the core ANSI SQL99 new standard with advanced features.

- Object-Relational DBMS: capable of handling object with complete routines and rules. Examples of advanced functionalities are declarative SQL queries, multi-user support, transactions and query optimisation.
- Client-server model: each database connection is started as a separate process that ensures the completeness of any database transactions.
- Flexible API: a complete API has allowed vendors to provide development support easily for the PostgreSQL RDBMS using interfaces including Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL and C/C++.
- Advanced features: PostgreSQL provide complete functionalities that are comparable with the most advanced commercial DBMS such as Oracle and DB2. There are many features that are not supported by MySQL, for example, the foreign keys, sub-selections, views, stored procedures, constraints, triggers and extensible type system.

With these major feature comparisons, I concluded that MySQL is more suitable for applications that need fast reactions but less complicated functionality, the popularity of the MySQL in the web based applications are a good proof. Its ease of use for both developer and the administrator has also contributed greatly to its popularity. On the other hand, the PostgreSQL is more suitable for advanced users for tasks such as performing more complicated database queries or applications that need better data integrity.

My application will perform large amount of transactions on the data in the ECperf database, the performance and scalability is an important requirement, to ensure data integrity of all these transaction is another factor. Therefore, I choose PostgreSQL because it's transactional and scalability advantages over MySQL.

## 4.6 Choosing ECperf as the testing tool kits

### 4.6.1 Why choose ECperf

ECperf is the industrial standard for test the performance and scalability of middleware in a typical J2EE environment. It is composed of the ECperf Specifications and the ECperf kit. The specification gives detailed description of the ECperf test including the motivation for the ECperf test, information about the application design, details of the workload description, the testing architecture configuration, the scaling and running rules and result disclosure rules. The ECperf kit provides a complete basic java program and the database data for running ECperf. A J2EE vendor or user must modify the Kit with an application specific deployment descriptor and configure it properly to run for particular application server and hardware architecture.

**Brief history of ECperf**

To understand the importance of ECperf as standard for evaluating the performance and scalability of application server, I need first give a brief history of the ECperf. The ECperf version 1.0 was first released in May 2001. Since then, it has attracted many top industrial leading application servers submission of their testing results to showcase the performance of their application servers. Those include the most important J2EE application server vendors, such as IBM, BEA, Sun Microsystems, Oracle and Borland.

The ECperf 1.1 was released in May 2002 as a maintenance version of the ECperf 1.0, since the result of ECperf 1.1 is not allowed to be released publicly, it is more likely to be used by the vendors and the developer as the benchmark for performance test and tuning internally.

SPECjAppServer2001 was released in September 2002 as a result of repackaging of ECperf 1.1 except for some minor changes to the source code, the metric, and the run rules [40]. This benchmark does not restrict the publication of the results as does ECperf 1.1, but users need to pay the licence fee.

SPECjAppServer2002 [40] is basically the same as SPECjAppServer2001 except that the Enterprise Java Beans (EJB) is defined using the EJB 2.0 specification instead of

the EJB 1.1 specification. This new benchmark can then take advantage of the features supported by the EJB 2.0 that include the local interface, CMP relationship between entity beans and the EJB-QL query languages. So, the performance of the new benchmark could be much better than the old one that uses EJB 1.1.

**Introduction of ECperf**

The ECperf test models a complete distributed enterprise level real-world application. It focuses on the test of EJB containers ability to handle the complexities of middleware services such as memory management, connection pooling, caching and activation/deactivation of EJBs.

ECperf simulates a multi-domain worldwide business that incorporates transactions in e-commence, business to business, manufacturing and supply to sales chains. This application is divided into four domains; each manages different business rules and data [18].

- Corporate Domain: maintain information of the customer, supplier and parts.
- Customer Domain: models the customer interaction with the company where a customer can create, delete or check the status of their orders.
- Manufacturing Domain: using "Just-In-Time" manufacturing concepts to maintain customer and orders status.
- Supplier Domain: maintains interaction with the suppliers of the parts.

Figure 4.1 [18] gives the architecture of the ECperf application. The five major parts in the ECperf applications fit into the different layers of J2EE application: The Web Server, Application Server, Database, Driver and Emulator Client. Each of which is run on separate machines.

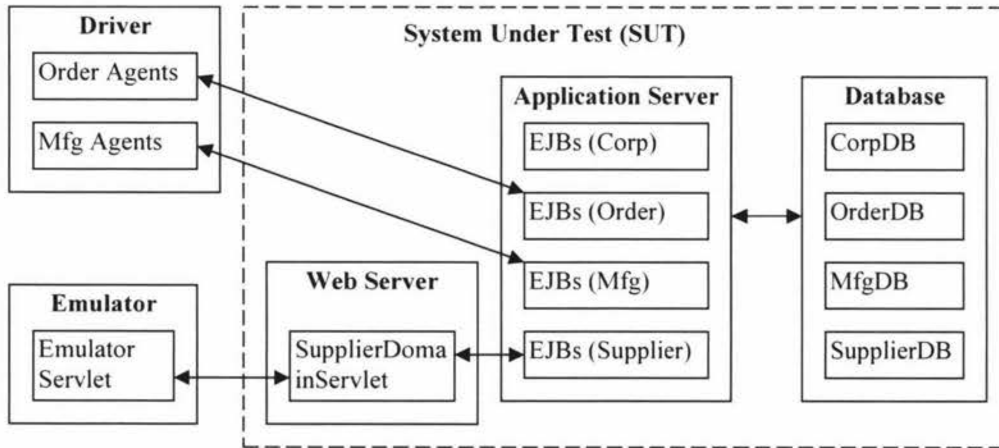


*Figure 4.1: The ECperf Architecture*

**Application Server:** The application server is the core part of the ECperf application, where all the Session Beans that define the business logic and the Entity Beans that represent data in the ECperf application reside.

**Database:** The database is the next most important part of the system. The ECperf application requires that the database provide enough capacity to support all the transactions in the system. There are four separate databases each related to one of the domains in the system. Each of the databases has a few tables. The four databases can be combined to one database for a centralized test.

**Driver:** the driver is a java application program. The driver runs in a multithread mode with each thread acting as one client based on the input transaction rate value.

**Emulator:** The emulator is basically a web-based interface that can interact with the ECperf application. The emulator allows manual interaction with the application one transaction each time, thus is not a necessary component for a formal ECperf test. It is useful for test each single function in the system.

**Web Server:** the web server is normally an important part for a J2EE application, particular an online application. However, the ECperf emphasises the application server performance and the driver client can interact with the Enterprise Java Beans directly. So, the web server is only useful for the Emulator client.

The transactions in the ECperf application are essentially related to the activities of process customer or distributor's order, manufacturing process of products and parts supplied by the supplier. Transactions in the Order Domain include: Create new orders, Change of an existing order, get order information and cancel an order. Transactions in the Manufacturing Domain include: get a large order and start processing a work order based on it, create a new work order and cancel the work order.

Each of the transactions in the ECperf application is called a Benchmark Business Operation (BBop). The performance of the ECperf is measured by the average finished transactions per minute that is BBop/minute. I will test with Cluster of application servers and measure the maximum output that is related to a particular hardware and software setting. With these results, I can evaluate the scalability of the application server I use.

## 4.7 Summary

I have described software used for my test in detail. For a multi-tiered J2EE application, I have chosen PostgreSQL as my database server, JBoss as the application server, Tomcat as the servlet server and the web server. I use the integrated version of JBoss-Tomcat as my server product in order to simplify my usage.

With this testing platform, I have carefully chosen the testing program, instead of building my own program that might only address limited issues; I have selected the ECperf, the industrial standard benchmark program for the application server.

# Chapter 5: Test design

## 5.1 Introduction

This chapter give details of the test design. Firstly, I will show two types of hardware architecture for the performance test, the centralized and distributed testing systems. Secondly, I will give brief information about how to run various test programs in the testing system. In the last section, detailed design analysis and architecture for various test in both the Sisters and the Helix are given.

## 5.2 Two types of the testing architecture

As I have stated before, the ECperf application is formed with four domains: the Corporation, Customer Order, Manufacturing and the Suppliers domain. Each of them contains data that represent the work in that domain. To run the ECperf benchmark, there are two possible architectures: the centralized and distributed workload architecture, which is defined based on how those domains are encapsulated into the databases [18].

### 5.2.1 Centralized workload architecture

The Centralized workload is a testing architecture where all the four domains in the ECperf are combined into a single database. Figure 5.1 shows an example layout of the Driver client and the System Under Test (SUT) components for the Centralized Workload. The SUT comprise all components, which are being tested, this include network connections, application servers and databases [18].

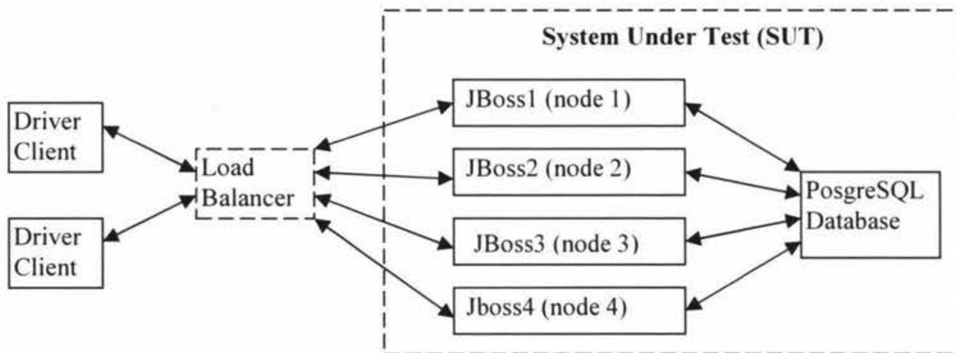


*Figure 5.1: Example configuration for the Centralized Workload*

In the example centralized workload configuration, each Driver Client will run at one machine, each JBoss will also run at a dedicated machine, which forms a cluster of JBoss servers. The database will run on a dedicated machine that will interact simultaneously with all JBoss instances. All those machines are connected through the network. In my case, they are connected through the high-speed network adapters.

### 5.2.2 Distributed workload architecture

The distributed workload is a testing architecture where all four domains in the ECperf are distributed into more than one database. Figure 5.2 shows an example layout of the Driver client and the SUT components for the Distributed Workload.

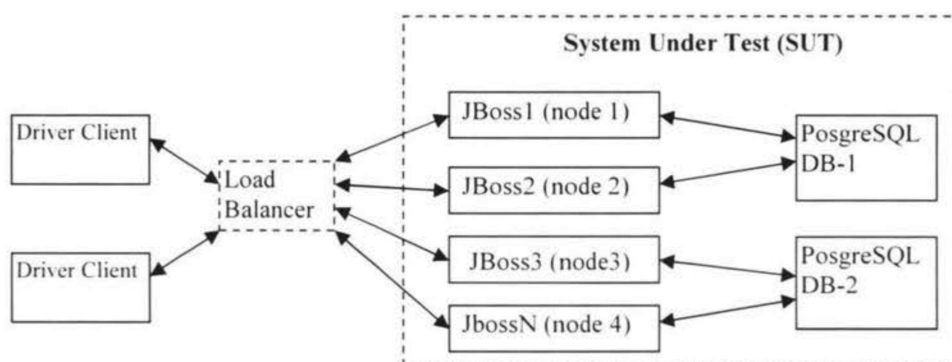


*Figure 5.2: Example configuration for the Distributed Workload*

The hardware and software architecture in the example Distributed Workload configuration is similar to the Centralized Workload configuration except that the database is split into two separate databases, with each running on its own machine.

I will show in the later section of this chapter, for the clustered JBoss architecture, I can further develop the distributed workload architecture by creating more than one partition in the JBoss cluster. For example, to have two JBoss clusters, each of them serving different EJBs. Through this schema, I can further improve the performance of the ECperf application.

I will use both the centralized workload and the distributed workload as my testing architecture. I will show later how these different architectures will affect the performance of the benchmark program.

## 5.3 Testing programs and related configuration

A cluster-enabled configuration of JBoss will start JBoss with all services-enabled including the Cluster support [37].

To use PostgreSQL, the source code must be downloaded and compiled with appropriate parameters.

Ant [41] was used in the test for several tasks. It is used to compile the ECperf source code, to package the ECperf application and deploy the application.

I use general Linux shell commands to monitor the workload of the machines used in my testing system, such as the *top* command and the *uptime* command. Because the *top* command gives details of the system resource consumptions, I can easily identify the hardware bottleneck if there are any. I use the Netperf [42] software to measure the network traffic during the testing process.

Testing the ECperf application performance in the JBoss Cluster and PostgreSQL is a tedious task, because there are many different types of test, and each of them has specific hardware architecture and software configurations. I put all the testing related information and the results in the appendix of the thesis. In the appendix, I give brief instructions on how to use the JBoss Cluster and how to use the PostgreSQL databases. I also cover how to configure and run a complete ECperf test as well as related configurations.

A very useful feature of the Linux operation system is the use of the shell commands in this case the Bash shell. Because Bash is a programming language, I can automate my test by combining all these single testing steps into one Bash command file. I then run this file to realize the automatic test.

By following the properly constructed control flows and combining the ant command into my bash file, I have a complete automatic testing file that is flexible enough to be

modified for running different tests. All those necessary parameters for starting an automatic test can be passed in through the Command line in Linux.

## 5.4 Test design for Sisters

In order to achieve logical, concise and useful results, I will give detailed analysis of the hardware and software included in my system. This design process will provide different types of test as well as expected results in the Sisters.

### 5.4.1 Type of planned test

Preliminary research has revealed some of the important factors that can affect the performance of a distributed J2EE application. These factors include resources involved in the application environment: the hardware, software and the network.

The hardware of the Sisters contains many nodes; each of them can perform the same or different functionalities. For example, nodes in a JBoss cluster normally perform the same functions to serve the clients of the application server; a node that is running a database has a different workload compared with a JBoss node. The total numbers of nodes involved in a J2EE application, the distribution of nodes to different layers of the application as well as hardware capacity of each node are all factors that will affect the application performance.

Software is another major factor that affects J2EE application performance. The software used for the test include: the Linux operating system, the Java Virtual Machine, JBoss application server, the Tomcat server, the PostgreSQL database and the ECperf application. The quality and functionalities of software is critical to overall application performance.

The Java Virtual Machine provides the environment where all the Java programmes run, the settings and parameters of the JVM have a big influence on the performance of software including the JBoss application server, the PostgreSQL and the ECperf application.

The JBoss and Tomcat server is where the middleware services are provided to a J2EE application, the functionalities and settings of JBoss will have big influence on the ECperf application.

PostgreSQL provides the database support to the ECperf application that requires high volumes of transaction support. Some of the default parameter settings in the database are likely to become a bottleneck.

Other software such as the Linux and ECperf application do not need to be changed, so these are less likely to be factors that affect the performance of the ECperf application. I will also identify in the later section, that the network in my Sisters and Helix are sufficient for my test.

Through initial test, I have identified several major factors that affect the application performance. I have planned the following different types of testing architecture:

- Test with different JVM heap size value
- Test of cluster of session bean
- Test with cluster of all EJB using the default load-balance policy
- Test with cluster of all EJB using First Available load-balance policy
- Distributed test with cluster of all EJB
- Distributed test with two partitions of JBoss instance

I will give brief design analysis related to these architectures. I will identify the hardware architecture for each type of test, and also some information about the testing process in the following section.

### 5.4.2 Two hardware architectures for testing the sisters

I have introduced in chapter 3 the architecture of the sisters -- the small Beowulf computer. There are 14 nodes in the Sisters, one (it017577) is the server node and the rest (it017578 to it0127588 plus AMD1 and AMD2) are the client nodes.

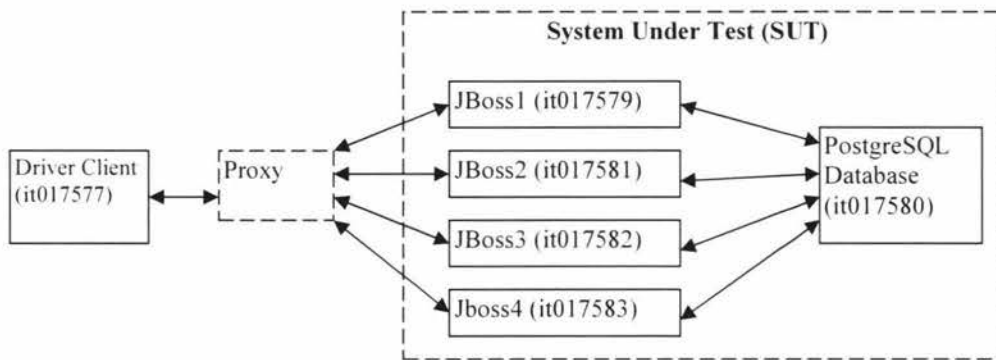


*Figure 5.3: Architecture for Centralized workload using Sisters*

Figure 5.3 gives the centralized architecture for running my test in Sisters. My purpose here is to decide how to deploy and run each layer of the ECperf application on nodes in the Sisters. To ensure the consistence and validity of the testing result, all machines that belong to the SUT must be dedicated for the test, as sharing resource with other program could lead to unreliable or even misleading testing results. On the other hand, the Driver Client provides only a light load and does not belong to the SUT, and can run in a machine shared with other programs, as long as that machine is not overloaded.

As the Sisters is a cluster computer shared by a lot of users and the server node is normally very busy, also because the Driver client is not part of the System Under Test (SUT) in the ECperf application and requires least resources, I can run the client by sharing system resource with other users without affect the testing result. I decide to run the Driver client in the server machine (it017577), which is shared by all the users.

Because there is only one database node to support the entire JBoss cluster node, It's very likely that the database will becomes the bottleneck at some stage of the test. I want to have a node with good computing power such as better RAM, cache and a fast read-write hard disk. Unfortunately, the best node is the server node that is always busy and shared by all the users, all the remaining nodes are the same. So, I choose it017580 node as the database node.

Because all JBoss application servers need to be configured to connect to the database, it makes sense to fix the node that is running the database. In this regards, I do not need to change the database related configurations for the JBoss server and driver client for my test and simplify my configuration work. Another thing to think about is the workload in the database node. Except the server node (it017577) and the Database node (it017580), all the rest of the nodes can be used as the JBoss server nodes when possible.

Sisters' users are required to submit their jobs to client nodes using Portable Batch System (PBS) software [43]. Once one user has used a node, others will not normally be able to use the same machine. This has ensured my repeated test can always use the same system resource.

Following the same analyses as for the centralized architecture, I derive Figure 5.4 for the hardware architecture for the distributed workload. The architecture for the distributed workload is similar to the centralized workload except that the database is split into two separate databases running on two Sisters machines: it017580 and it017584.

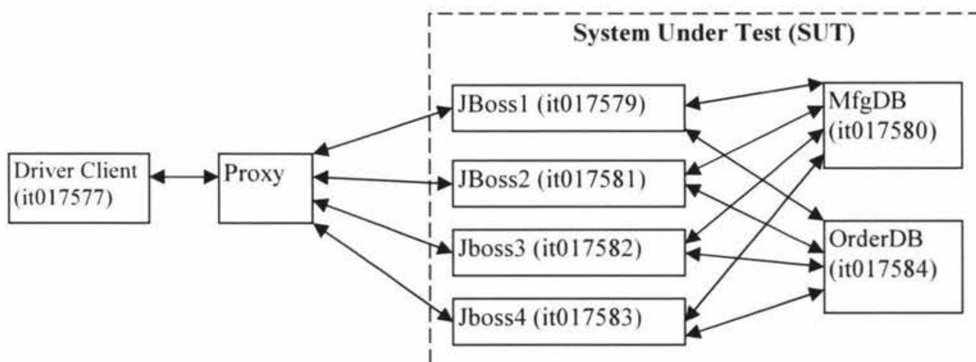


*Figure 5.4: The architecture for distributed workload using Sisters*

## 5.4.3 Test design for JVM test

**Hardware architecture of memory test**

The Java Virtual Machine (JVM) is an abstract computing machine. The JVM is a platform-independent execution environment that converts Java byte-code into a particular machine language and executes it. The performance of an application running on the JVM can be affected by various JVM parameters.

The performance of the JBoss application server is related tightly with the Java Virtual Machine because JBoss is running on top of the JVM. The purpose of the test is to find out the optimal Java Virtual Machine parameters settings that will be used for the later test. I will examine in particular how the maximum and minimum value of the JVM Heap size affects my throughput of the transactions when running ECperf application.

Although, my major emphasis is to test performance of a cluster of application server, I will do the JVM heap size test using only one machine. Because finding out the optimal JVM heap size settings for one machine, I can then apply this to the whole cluster of JBoss servers.

Figure 5.5 presents the hardware architecture of my JVM heap size test. This architecture is a simplified version of Figure 5.3 with only one JBoss_Tomcat server, one Driver client and one PostgreSQL database machines.

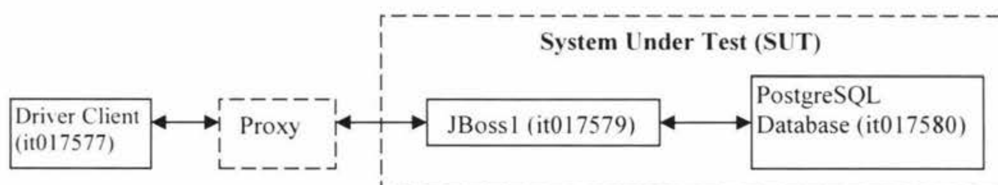


*Figure 5.5 The hardware architecture for JVM heap value test in Sisters*

I have observed when running the tests with the default setting of the JBoss_Tomcat on the Sun's Java Virtual Machine; the performance is very poor and unstable in many cases. When running on the above architecture, even relatively small transactions will cause the Java Virtual Machine in the JBoss server node to throw an *OutOfMemoryException*. This simply implies that the default memory space defined by the JVM, where the JBoss is running is not sufficient for running my program. I will give my analysis and design options for solving this problem.

The JVM heap is the memory space used by all Java program to store objects created at run time. Java's *new* operator creates objects, and memory for new objects is allocated on the heap at run time. In addition to the run time objects, the heap can also

contain memory reserved for the garbage collector and for some other JVM activities [44].

**JVM heap test design**

The overall heap size is normally set by the following two parameters [45]:

- -Xms: Specify the initial size, in bytes, of the memory allocation pool. Must be a multiple of 1024 and greater than 1MB. The default value for Sun's JVM is 2MB.
- -Xmx: Specify the maximum size, in bytes, of the memory allocation pool. This value must be a multiple of 1024 greater than 2MB. The default value for Sun's JVM is 64MB.

Though, there is no value that a programmer can use to control the increment of the JVM heap size. The heap memory will change during the run time if the running Java programs require more memory [45].

The default JVM heap value is set for general programming and is not optimised for a large application that requires many system resources. Thus, I need to test to find out the best JVM heap size. Giving the total RAM of a machine to the JVM heap may not be the best choice, as JVM is not the only program that needs to consume system resources.

Particularly, I need to design a series of tests that will find a pair of –Xms and -Xmx values that can generate the best ECperf test result. How do I choose these values?

I decide to choose the same value for –Xmx and –Xms, that is: -Xms = -Xmx. By assigning the same values to the maximum and minimum heap size, I can get the maximum heap value at the very beginning and eliminate the overhead for the system to increase or decrease the JVM heap size during the run time.

The default value of the Sun's JVM heap size is: -Xms = 2MB, -Xmx = 64MB. The maximum memory available for my sister's nodes is 256MB. So, I choose the

following values as my JVM heap size values as: The default, 64MB, 100MB, 140MB, 180MB, 220MB, 256MB. I will examine which pair of Java heap value can produce the best possible transaction output for ECperf application.

The details of the testing procedures and the result data analysis will be given in the next chapters.

### 5.4.4 Test design for Clustering of Session Beans

**Architecture and design**

I have discussed in the last section the Java Virtual Machine parameters that could affect the ECperf application performance in my system. I will now discus the performance test using the Clustered JBoss. Figure 5.3 of the Centralized workload architecture in the previous section gives the hardware system for running the JBoss Cluster.

In the system, there is one Driver Client machine that is running a multi-threaded java application to emulate many simultaneous client's access to the EJBs running on a cluster of JBoss application servers each running in one node. The Session EJBs in the application are replicated across the application server to provide high availability and fail-over feature support.

The basic theory of a cluster of EJB is to replicate the EJB instance to multiple machines. A JBoss Cluster has provided support for clustering of EJBs. For example, when clustering of Session Bean in this test, each JBoss machine in the JBoss cluster has a copy of local Session Bean, which represents the same Session Bean Object. Through the replication of the Session Bean, the client of the Session Bean can easily use a local copy of the Session Bean that can greatly increase the availability of the Session Bean object and thus improve performance of the overall application. Figure 5.6 gives a graphic representation of clustering of a Cart Session bean and Order Session bean in a JBoss Cluster with two nodes. Please note: the Order Entity Bean is not clustered and has not been replicated in the JBoss cluster, this entity bean is only available in the machine where the ECperf application is deployed.

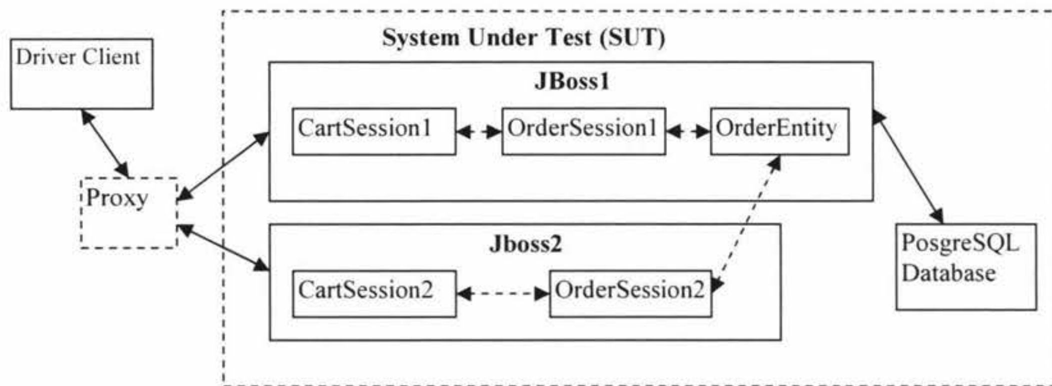


*Figure 5.6: Cluster of only Session Beans*

With the centralized workload architecture and clustering of only the Session Beans, I will exam the best possible transaction output for different JBoss numbers in the JBoss Cluster, which are 1, 2, 4, 6 and 8 JBoss. I can see how the JBoss numbers in the JBoss affect the scalability of the whole testing system.

## 5.4.5 Test design for Cluster of all EJB

**Design and architecture**

I will cluster all EJB here instead of only clustering Session Beans. I expect an improved transaction output by clustering all EJBs compared with clustering of only the session beans.

The architecture used for test is illustrated in Figure 5.3. I actually use the same hardware architecture and software as the test with clustering of only Session Beans. The only difference here is the deployment descriptor I used to define my EJB clustering.

The basic theory of clustering EJB is through replication of one EJB instance to multiple JBoss machines. In the last section, I only clustered Session Beans. I will also cluster all Entity Beans for my new test. Through clustering of all EJB, each JBoss machine in the cluster has local copies of all Entity Beans and Session Beans. Because the Entity Bean is the object representation of one row of data in the relational database, I expect the database related operations such as access speed to

the data will be increased compared with only clustering Session Beans thus increasing the overall transaction output.

Figure 5.7 gives graphic representation of the clustering of the Cart Session bean, Order Session bean and Order Entity Bean in a JBoss Cluster with two nodes. In this schema, not only are all the session beans, but also all the entity beans are replicated to each JBoss instance in the JBoss Cluster, thus the network overhead associated with accessing an entity bean is eliminated.

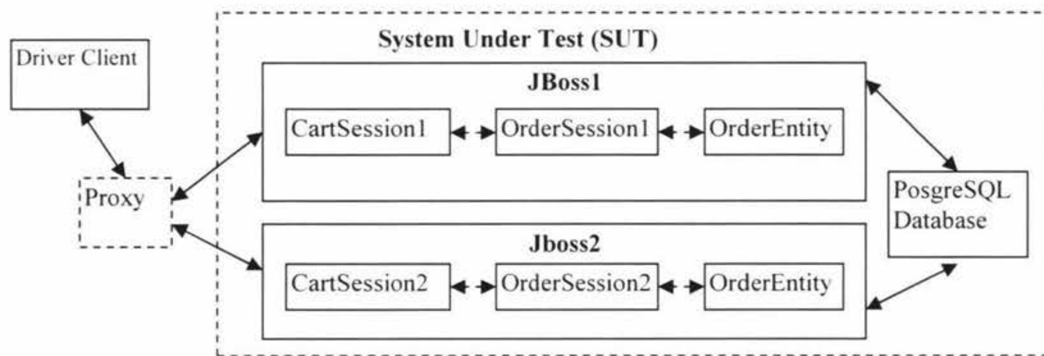


*Figure 5.7: Cluster of all Enterprise Beans*

With the centralized workload architecture, I expect better output performance and scalability by clustering of all EJBs compared with clustering of only the Session Beans. As the number of JBoss serves increase, I expect the overhead associated with the replication to slowly decrease performance, so affecting the scalability.

## 5.4.6 Test design for two databases

**Design and architecture**

The centralized workload is very likely lead to bottlenecks in the PostgreSQL database, because many machines in the JBoss Cluster need to interact with the centralized database concurrently. To solve this problem, I will use the distributed workload architecture for the ECperf so as to improve the transaction output.

As introduced earlier, the ECperf application is divided into four domains, the corporate, the manufacturing, the customer and the supplier domain. The data in the database can also be grouped accordingly. The only exception is the data in the utility

section, which acts as helper classes to all four domains. With this in mind, I split the ECperfDS database into the following two databases:

- The OrderDS database, which contains the data in corporate domain, the manufacture domain and the utility domain.
- The MfgDS database, which contains the data in manufacture domain and the supplier domain.

Figure 5.4 of the distributed workload architecture in the previous section is used as the hardware system for my test. In the testing system, The MfgDB database running in machine it017580, the OrderDB database running in machine it017584.

Figure 5.8 is the EJB replication graph with two databases in the system. All the EJB are replicated in all members in JBoss Cluster, but the data that represent these EJB are split into two separate databases. During the testing process, the Driver client still interacts with all JBoss servers, but a JBoss server needs to interact with two databases depending on what data need to be accessed.

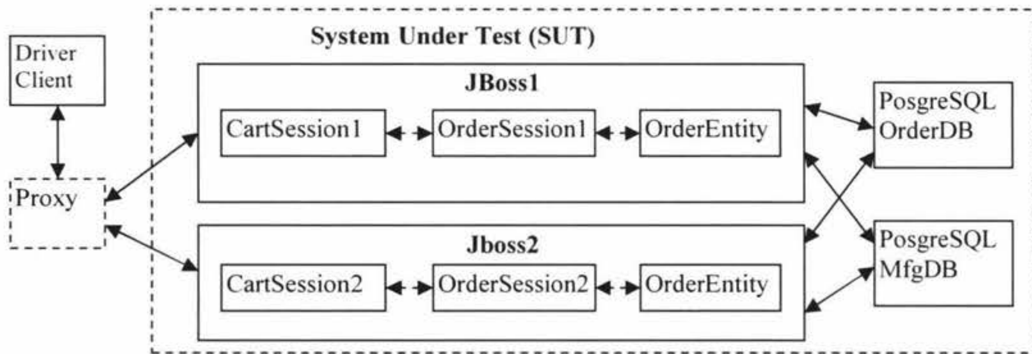


*Figure 5.8: Clustering all* EJB *with two databases*

With the distributed workload architecture and clustering of all EJBs, I can see what level of scalability can be achieved by comparing the results with that of the centralized workload architecture.

## 5.4.7 Test with Two partitions and two databases

**Design and architecture**

The clustering of EJBs is a good strategy to improve system performance, but too many copies of one EJB object may not be desirable. As there are overheads associated with the JBoss Cluster, the performance gain of the application may not be proportional to the number of copies of the EJB objects in the JBoss Cluster. The overhead associated with the clustering of EJBs could be communication between members in the JBoss cluster and action to synchronise access to an EJB to maintain data integrity in the database.

For a better performance, I will do further test with two separate JBoss partitions. Figure 5.9 gives the architecture of the ECperf applications running with two partitions and two databases.

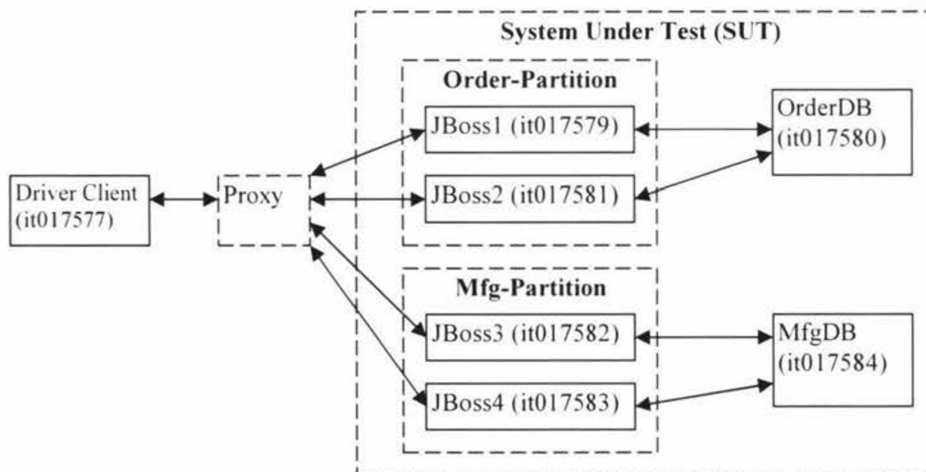


*Figure 5.9: Test with two partitions and two databases*

In this architecture, the JBoss cluster is divided into two separate partitions. There is no communication between members in the two partitions; only members in the same partition will interact with each other. As in the last distributed test, there are two separate databases that are used for data in different domains of the ECperf application; each database only interacts with JBoss servers from one partition.

Figure 5.10 gives an example testing architecture of clustering of all EJBs, using First-Available load balance policy and with two JBoss partitions. In my example test

with four JBoss instance. JBoss1 and JBoss2 belong to Order Partition, JBoss3 and JBoss4 belong to Manufacturing Partition. I have the Order Database (OrderDB) that contains data belonging to Corporate Domain and the Customer Domain; and the Manufacturing Database (MfgDB) that contains data belonging to Manufacturing Domain and the Supplier Domain.

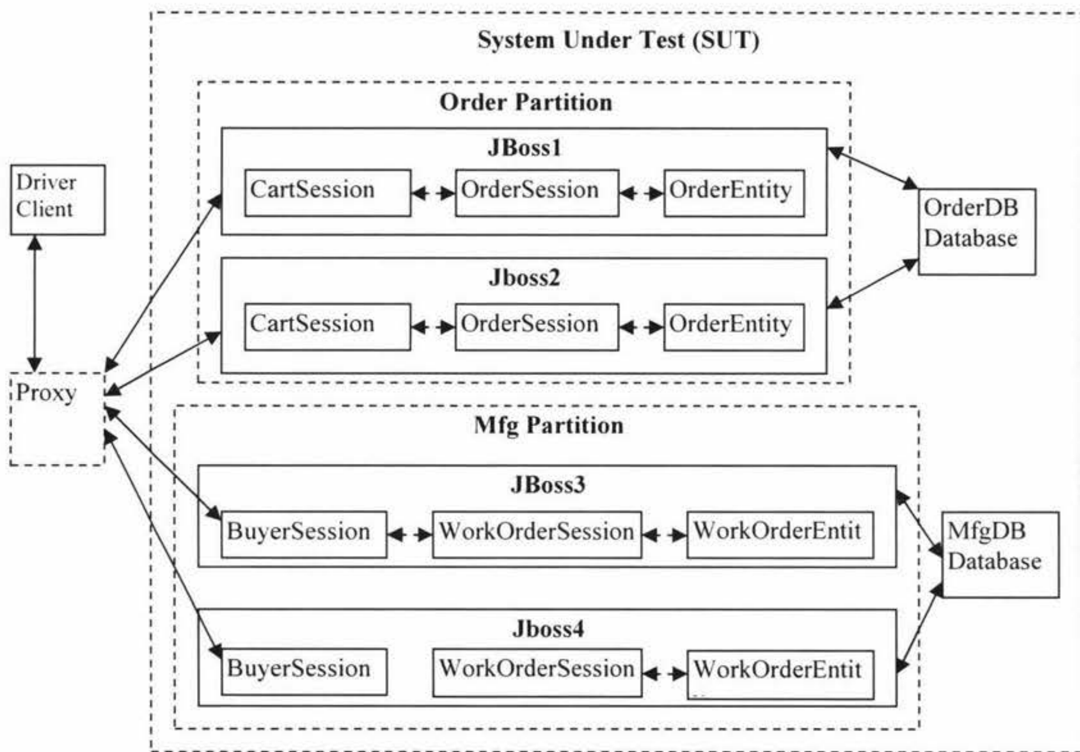


*Figure 5.10: EJB Replication with 2 JBoss Partition & 2 Databases*

An EJB will only be replicated in JBoss members that belong to one Partition. The EJB such as the Cart Session Bean, Order Session Bean and Order Entity Bean are only replicated in JBoss1 and JBoss2 that belongs to Order Partition, and are stored in OrderDB database only. Similarly, Buyer Session Bean, Work Order Session Bean and Work Order Entity are only replicated in JBoss3 and JBoss4 that belongs to Mfg partition, and are stored in MfgDB database.

In my test architecture, the client will still interact with all members in the JBoss Cluster. A client's request will only be directed to a JBoss server in one partition where the particular Session EJB exists, and JBoss servers in one partition only interact with one of the two databases.

By using two separate partitions in the JBoss Cluster, each JBoss partition only has half the number of JBoss members compared with using only one JBoss partition, thus effectively reduce the number of local copies of an EJB. I anticipate reduced Entity EJB persistence overhead as well as the communication overhead between JBoss cluster members, will improve the ECperf application performance. This technique relies on a balance workload between the two partitions, which is the case with ECperf.

## 5.5 Test design for Helix

The following section will give the design and architecture information of test on Helix, the supercomputer.

### 5.5.1 Type of planned test

As described before, factors that affect the ECperf application performance in the JBoss Cluster are essentially the same; these are the hardware, the software, the network and application running in the system.

The fact that the Helix contains many more nodes than Sisters has given the chance to run the ECperf programme in a bigger JBoss cluster. I will test with up to 32 machines in the JBoss cluster using Helix. As there are so many machines in the system, there are extra overheads and complexity introduced.

Helix and Sisters are essentially the same type of Beowulf cluster; ECperf tests on them use the same type of hardware architecture and the software. The only difference is that Helix nodes are more powerful and Helix contains more nodes than Sisters.

Based on these similar characteristics. It's not necessary to repeat all types of test I have planned in the Sisters. I will concentrate on a few experiments on Helix.

The Java Virtual Machine test in Helix is interesting, because a Helix machine provides 1024MB memory instead of 256MB in a Sisters machine. A JBoss Cluster will still be the major software feature in Helix to examine scalability of the running ECperf application.

In summary, I have designed the following types of test in Helix:

- Test with different JVM heap size value
- Test with cluster of all EJB
- Test with cluster of all EJB using the optimised database pooling in JBoss

I will give brief design and analysis related to these architectures. I will identify the hardware architecture for each type of test, and also some information about the testing process.

### 5.5.2 Test design for JVM test

As for the Sisters, I will exam how the JVM machine setting will affect the performance of the Clustered application.

Because a Sisters node has only 256MB total RAM, the scalability results of the JVM test is not very good. Doing JVM test in a Helix node with total RAM of 1000MB will gives better indication of how JVM heap size will affect the transaction output of my program. The JVM test will find out the optimal Java Virtual Machine settings that will be used for the later test.

Figure 5.11 presents the hardware architecture of my JVM heap size test. Here, the Driver client, JBoss_tomcat server and the PostgreSQL database are running on separate machines. I only use one JBoss machine in the test, because the optimal JVM heap size settings for one machine can be applied to all machines in the JBoss Cluster.

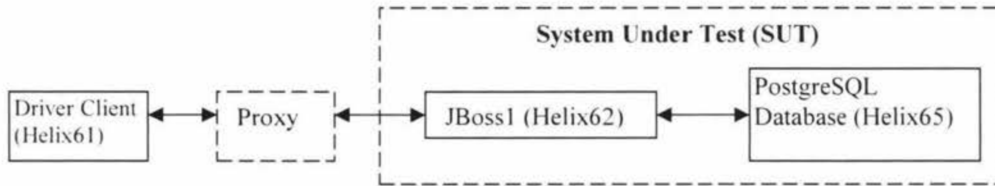


*Figure 5.11: The hardware architecture for JVM heap value test in Helix*

I choose –Xms = -Xmx (the default setting will be an exception). My test case will include the following JVM heap size values: the default heap space, 100MB, 200MB, 300MB, 400MB, 600MB, 800MB and 1000MB. For each of these values, I will test by gradually increasing the value of txRate (that represent the number of clients) in the ECperf testing program. The JVM test will give a good indication of how JVM heap size values affect the scalability of the ECperf application in Helix.

## 5.5.3 Test design for using the default DB pooling in JBoss

Figure 5.12 gives example hardware architecture in Helix with 4 JBoss in the JBoss Cluster.

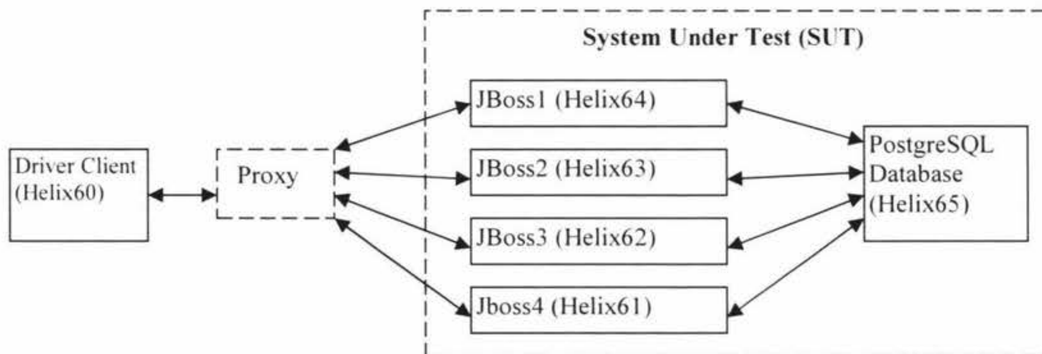


*Figure 5.12: The hardware architecture for ECperf Test in Helix*

In the system, there is one Driver Client machine that running a multi-threaded java application to emulate many simultaneous client's access to the Session EJBs running on a cluster of JBoss application servers each running in one node. The EJBs in the application are replicated across the application server to provide high availability and fail-over feature support.

As the testing results in Sisters have shown that cluster of all EJBs will performance better than cluster of only session beans. I will not do tests with clustering of only the

session beans, instead, I will do the test for clustering all Enterprise Java Beans using the default data source pooling in the JBoss servers.

The Enterprise Java Bean replication schema is displayed in Figure 5.7, where all session beans and entity beans are replicated to all JBoss members in the JBoss Cluster.

With the centralized workload architecture and clustering of all EJBs, I will examine the best possible transaction output for different JBoss numbers in the JBoss Cluster, which are with 1, 2, 4, 8, 12, 16, 24 and 32 JBoss. It can be interesting to see what kind of scalability can be achieved with a large number of JBoss in the JBoss Cluster.

### 5.5.4 Test design using optimised database pooling in JBoss cluster

My pilot test in the Helix has shown that the database pooling is a very important factor for the output of my program. In this test, I will explore the optimal database pooling values in the JBoss application server in each machine. Also, I will disable the JBoss log file, as the Sisters results have shown it is a useful factor to improve the application performance.

The hardware architecture in Figure 5.12 will still be used for this test; the EJB replication schema will still be the same as displayed in Figure 5.7, and also the same testing algorithm will be used.

As I expect better transaction output for this test compared with using the default database pooling in the JBoss Cluster, the PostgreSQL server is very likely to become the bottleneck with relatively smaller number of JBoss in the JBoss Cluster. So, the experiments will run with up to 8 JBoss members in the JBoss Cluster.

## 5.6 Conclusion

I have given some details of the design options for my test. I have presented two types of testing architecture: the centralized workload and the distributed workload architecture. The major difference of them is the centralized architecture uses only

one database, while the distributed architecture uses more than one database. More detailed design and the architecture for various test in both the Sisters and the Helix were discussed using different architectures and Enterprise Java Bean replication schema.

# Chapter 6: Test on Sisters

## 6.1 Introduction

In this chapter, I will present various test results for the Sisters. For each type of test, I will start by giving a brief introduction of each test, followed by the test procedure, and a discussion of the test result. A graph that gives the final test result will be displayed for each type of test. Finally, I will discuss the transaction results, the performance, scalability and the bottlenecks for each type of test.

I will do the following tests:

1. Test with different JVM heap size value
2. Test of clustering of session beans with default connections in the databases
3. Test with optimised database connections
4. Test with clustering of all EJB with the default load-balance policy
5. Test with clustering of all EJB with first available load-balance policy
6. Distributed test with clustering of all EJB
7. Distributed test with two partitions of JBoss instances
8. Distributed test with two partitions and disabled log file

## 6.2 Test with different JVM heap size value

**Testing procedure**

The JVM heap space test uses one of the Sisters client machines that have a total memory of 256MB. To make sure other factors will not become possible bottlenecks in the testing system; I use the optimised connections between the JBoss application server and the database, which I will examine later in the chapter.

Using the hardware specified in the Figure 5.5, I have done the JVM heap test using values of the default, 100mb, 140mb, 180mb, 220mb and 256mb. Here is my testing algorithm:

*For –Xms equals –Xmx with value of (default, 100, 140, 180, 220, 256mb)*
*Do test with transaction rate = 1, 2, 4, 8, 12, 16, 20, 24, 28, 32*
*(Repeat three times for each transaction rate value)*
*End*

In the algorithm, for each pair of the –Xms and –Xmx values, I get a sequence of tests result that corresponds to the txRate of 1, 2, 4, 8, 12, 16, 20, 24, 28 and 32. I repeat the test for each txRate value three times. By increasing the value of txRate in the ECperf testing program, the number of threads, each simulating a client is also increased.

I have repeated each identical test for three times. I take the ECperf Metric output for each test from the ECperf summary file. I calculate the average of the three tests as the final output.

As I have a lot of tests to complete, I have to make each test period shorter than the standard minimum test time; this is acceptable for my purpose. Also, I relaxed the output result by recording only the total transaction output even when there are some errors in the output. Because the final output represents client request that have been processed properly, they are all valid transaction output. All tests in my research are under the same conditions.

**JVM Heap space Testing Result**

For every single test, output of 8 files are produced, they are:

- ECperf.summay
- Orders.summary
- Mfg.summary
- Orders.detail
- Mfg.detail
- ords.err
- plannedlines.err
- loline.err

From the ECperf.summary file, I obtain the value of the ECperf Metric; this gives the number of Transactions per second. For each pair of JBoss-Number and txRate value,

I have obtained three valid outputs of the ECperf Metric. I take the average of these three as the final output.

Table 6.1 is an example result where –Xmx = 180MB. For each txRate value, I have a correspondent transaction output (TPS) value when running the ECperf application in the system. As discussed, the TPS value in the table is the average value of three valid test outputs.

| TxRate | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 16 | 20 | 24 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| TPS | 103.6 | 198.8 | 413.4 | 528.6 | 566.7 | 578.3 | 490.3 | 533.7 | 496.7 | 406.4 | 425.4 |

*Table 6.1: Throughput of ECperf as function of the txRate*

The graph of transaction output versus txRate that corresponds to each –Xmx value can be drawn. Figure 6.1 gives the Relationship of txRate versus Transaction Output when the JVM heap value equals 180MB.

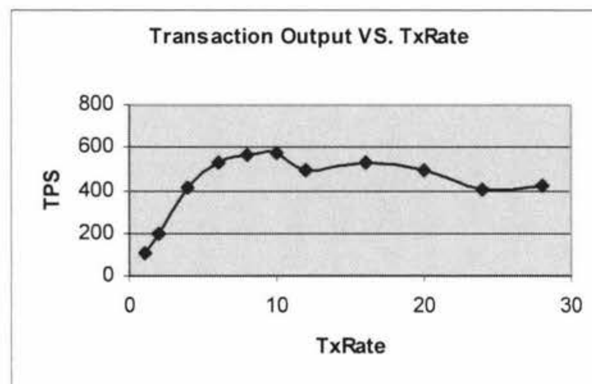


*Figure 6.1: Throughput as a function of the txRate (-Xmx = 180MB)*

The transaction output increases when the transaction Rate increases from 1 to 10, and txRate = 10 actually gives the highest output. The output decreases when the transaction rate increases after txRate = 10. The reason for this is obvious. Since the JBoss console gives an *OutOfMemoryException*, this simply means that the memory used by the JBoss application server is too small. The given JVM heap value of –Xmx = 180MB needs to be increased.

I can get a similar graph as Figure 6.1 for each pair of JVM heap value, which will not be displayed here.

For each JVM Heap value, I have a maximum value that is the maximum throughput I can get for that Heap value. Taking the –Xmx and Maximum TPS pairs, gives the results shown in table 6.2.

| JVM heap (-Xms = -Xmx) | 64 | 100 | 140 | 180 | 220 | 256 |
|---|---|---|---|---|---|---|
| TPS | 313.7 | 536.5 | 571.5 | 578.3 | 557.1 | 552.7 |

*Table 6.2: JVM Heap Value vs. Maximum TPS Test*

The results in table 6.2 are shown as a graph reporting the Transaction Per Second as a function of the maximum JVM Heap (-Xmx) in Figure 6.2.

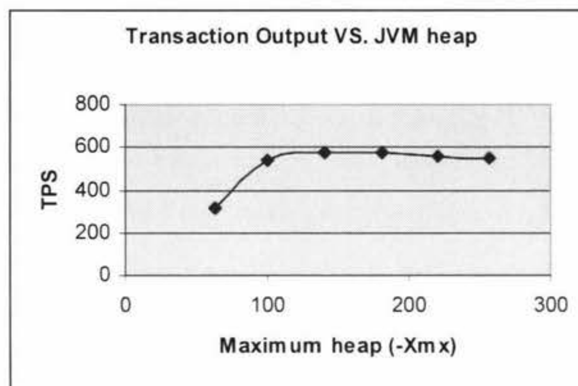


*Figure 6.2: Maximum throughput as a function of the JVM Heap Size*

**Result Analysis, further discussions and conclusion**

Figure 6.2 shows how JVM heap size will affect the performance. With the increase of the JVM heap size value, the transaction output is increased. When the value of JVM heap size is smaller than 100MB, the output increases significantly. Above 100MB the performance increment rate is slower until a top value is reached. After that, the transaction output decreases slowly when the JVM heap size increase.

Figure 6.2 shows that when the JVM heap size value is from 45% to 100% of the maximum memory available in a machine, the transactions output is pretty close to the top output. The maximum transaction output was produced when the JVM heap size value is 180MB. I will use this value for the remaining ECperf tests.

This is a reasonable result. Because the JBoss application server is running on top of the JVM, the size of memory space in the JVM available to JBoss will be an important factor related to how much work a JBoss can do. Given more JVM heap size means more memory space available to store objects created by the application running on JBoss, the number of transactions will increase with more resources available.

As JBoss is not the only program that is running on the JBoss node machine, there are other programs such as the operating system and related programs running together with the JBoss application server. To get best performance from the machine,
Some system resources need to be reserved for them. So, when the memory consumption of the JBoss application server reaches a point that is close to the maximum memory available in the machine, the overall performance of the application will decrease slowly while the memory consumed by JBoss increases. If other operating programs do not have enough memory resources and start running too slow, that will slow down the whole system, thus affecting the overall performance of the JVM.

A small JVM heap value could greatly restrict application output with a high workload. Giving the maximum possible JVM heap will generally increase the application performance, but does not guarantee to produce the best output. The best JVM heap value for a particular application needs to be tested, and should normally between the ranges of 50% to 100% of the total RAM of a machine.

I will identify the bottlenecks encountered when doing the JVM heap value test. For the default JVM value, the memory used by the JVM is obvious too small for my application; the *OutOfMemoryException* clearly shows that the JVM heap value is the bottleneck in the system. This is true when the JVM heap value is smaller than 140MB.

After the JVM heap value reaches 140MB. The memory space defined by the JVM heap value is no longer a bottleneck in the system. I can see from the Linux *top* command, that the JBoss node in the system has becomes the new bottleneck. The CPU usage in the JBoss node is 100% (including the users plus the systems CPU

usage). As this CPU usage keeps consistent in the system, the Job queue in the JBoss server keeps growing until the whole system becomes slower and slower and eventually to a state where the performance is significantly reduced.

Sun has suggested some of the parameters in the JVM that could affect the performance of the Java program running with the Sun's JVM [45]. I have tried other JVM values, such as the NewSize and MaxNewSize for garbage collection, AggressiveHeap that affect the working of JVM heap size, but none of them seems to have an obvious effect on the transaction output of my ECperf program. Therefore no further testing results related to them are presented.

The optimal JVM heap value of -Xms = -Xmx = 180MB is used for the test and keeping other JVM parameters as default values.

## 6.3 Test with clustering of only session beans

### 6.3.1 Preliminary tests using the default connections in PostgreSQL

**Testing procedure**

The testing hardware architecture for running this test was given in Figure 5.3 and the design analysis for test of clustering of only the session beans was given in section 5.4.4.

The default database connections in PostgreSQL database is used for the test. The default allowed maximum number of connections to the database is 32, in which, two of them are reserved for the super user, and thus the actual maximum allowed concurrent connection from the client to the database is 30 [17].

I will test the scalability of the J2EE application server. For this purpose, I need to get the highest output for a particular JBoss Cluster. I have chosen the following testing algorithm as a general testing procedure. These tests have been automated for the purpose of convenience.

*For JBoss-Number = 1, 2, 4, 6 and 8*

*Do test with transaction rate (txRate) = 1, 2, 4, 8, 12, 16, 20, 24, 28, 32*
*(Repeat three times for each txRate value)*

*End*

**Test result analysis**

Preliminary testing results do not display expected scalability when the JBoss numbers in the JBoss Cluster is increased. Instead, when using the default PostgreSQL database connection, whenever testing with two and more JBoss in the JBoss cluster, I got the following error message consistently in the PostgreSQL log file:

*FATAL: Sorry, too many clients already*
*FATAL: Non-superuser connection limit exceeded*

I have got similar peak output results when the JBoss number is more than 2 in the JBoss Cluster, which will not be displayed here. The default number of allowed concurrent connections in the PostgreSQL database has become an obvious bottleneck in my test with clustering of only Session Beans.

On the JBoss side, Database pooling is used to interact with the databases. The default minimum and maximum database pool values in JBoss application server are 5 and 25 respectively [37]. Resource pooling is a very useful method for optimising the performance of an application server. The JBoss application server uses database connection pooling to optimise the usage of precious database connections. When an enterprise bean requests a connection, the container fetches one from the database pool and assigns it to the bean. Because the connection has already been established, the bean can quickly reuse a connection. An EJB may release the connection back to the database pool after each database call, thus many beans can share the same connection sequentially.

2 JBoss servers can provide total of maximum-pooled connection of 50. As the maximum allowed concurrent connections in the PostgreSQL database is only 30. I can expect the above-mentioned error message once the JBoss cluster tries to establish more than 30 concurrent connections.

Further tests using maximum allowed connections of 1024 in PostgreSQL have greatly improved the output performance of the system (see the next section). For example, over 70% increases for the peak output was observed using optimised database connections compared with using the default one, when testing with JBoss Cluster with 4 members.

For the rest of the tests in Sisters, I will use 1024 connections, that is the maximum allowed by the Linux operation system without increasing available shared memory in the Operating System. The following command is used to start the PostgreSQL database.

*Postermaster –I –N 1024 –B 2048 > logfile 2 > &1 &*

Using connections of more than 1024 will result in "*Maximum Shared memeory errors*". That means the PostgreSQL has tried to use memory space that exceeds the default allowed values in the Linux operating system. The solution to this is for the administrator to adjust the maximum-shared memory allowed to each program and might also need to adjust the shared system semaphore value as well [46]. I do not need to do this, as maximum connections of 1024 are already enough for the remaining tests in Sisters.

## 6.3.2 Test by using optimised connections in PostgreSQL

**Design the testing procedure**

The testing hardware architecture and the design analysis for test of clustering of only the session beans has been discussed in the last chapter.

As mentioned in the early section of this chapter, these tests have been automated for the purpose of convenience. I have chosen the following testing algorithm as a general testing procedure.

*For JBoss-Number =1, 2, 4, 6 and 8*
*Do test with transaction rate (txRate) = 1, 2, 4, 8, 12, 16, 20, 24, 28, 32*
*(Repeat three times for each txRate value)*
*End*

**Result**

Table 6.3 shows the test result when there is 1 member in the JBoss Cluster:

| TxRate | 1 | 2 | 3 | 4 | 6 | 8 | 12 | 16 |
|---|---|---|---|---|---|---|---|---|
| TPS | 101.2 | 198.8 | 304.2 | 403.2 | 597.5 | 605.7 | 609.8 | 599.8 |

*Table 6.3: Final Transaction output VS. txRate. (JBoss-Number = 1)*

Figure 6.3 shows the transaction output that is the function of the transaction rate (txRate) in the testing program. The number of clients can be calculated in the following formula: number-of-client = txRate * 8.

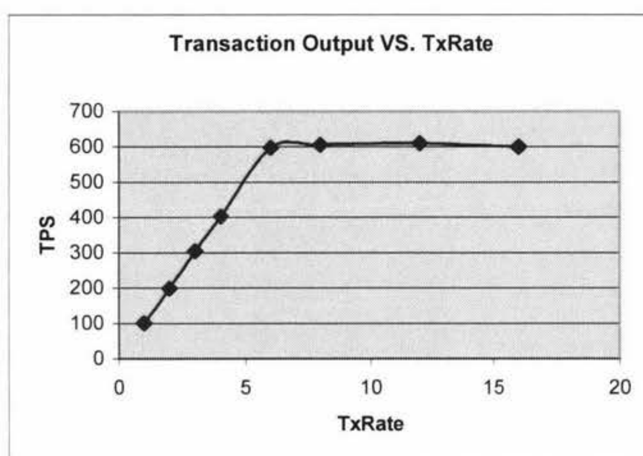


*Figure 6.3: The Transaction Output (TPS) VS. Client Number*

The graph shows that when the user number is less than 48, the Transaction output of the program increases with the increase of the Client Numbers. After that, the output is almost flat even when the Client Numbers increase. This means that the system has reached a bottleneck. By checking the system resource consumption of each machine in the SUT, I observed that a CPU bottleneck is actually reached in the JBoss machine.

I will take the biggest output value as the final Transaction Output for one JBoss server. I got: *TPS = 609.8*

Following the same steps, I will get the final TPS against each JBoss-Number of server as in the Table 6.4.

| JBoss No. | 1 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| TPS | 609.8 | 989.7 | 1584.6 | 1993.4 | 1998.4 |
| % of change | | 62.3% | 60.1% | 25.8% | 0.3% |

*Table 6.4: Final Transaction output VS. txRate. (JBoss-Number = 1)*

These results are shown in the graph Figure 6.4, with the X-axis representing the Number of JBoss in the test, Y-axis represent the Maximum output.

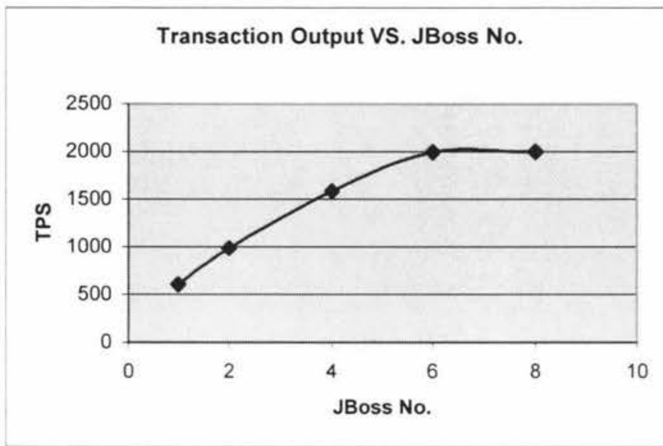


*Figure 6.4: The Transaction Output (TPS) VS. JBoss Number*

**Result analysis, discussion and conclusion**

From Figure 6.4, it can be seen that when the JBoss number is between 1 to 6, the transaction output of the application increases almost linearly, with a factor of about 62.3% to 50.6% increase of performance when the JBoss number is doubled, that is by doubling the JBoss number from JBoss-Number = 1 to JBoss-Number = 2, the performance increase is 62.3 percent. When the JBoss number becomes bigger, the percentage of the performance increase is smaller. When the JBoss number is increased from 6 to 8, the performance increases is very small. This indicates a bottleneck is reached when JBoss-Number equals 6. I will discuss this in the following section.

For the SUT with only one JBoss instance, when the highest output is created, the bottleneck is the JBoss application server. Because the CPU usage of the JBoss

machine is 100% and the job queue of the CPU identified by load average value is also very high. This shows that the full hardware capacity of the JBoss machine has been used to process the ECperf program and the hardware bottleneck has been reached.

For JBoss number equals two and four, the same thing happens for the JBoss application server as that with only one JBoss in the SUT. One important factor that needs to be identified is the load balance problem. The default Round Robin load balance policy [38] for clustering of EJB in JBoss does not seem to works well with high client request load. I have observed from the Linux utility command that in a high volume of workload, the job queue in the node with the longest queue becomes longer and eventually crashes.

The reason for this is reasonably clear with the Round Robin policy. The Driver Client distributes EJB lookup to all JBoss servers one after other using the Round Robin algorithm. Because each call can contains different types of transaction requests and may not have equal workload, it is natural that some JBoss nodes have more workload than others during the testing process. For a test with high transaction rate, when a JBoss node becomes overloaded, it takes longer to process the same client request than other JBoss nodes. But the Round Robin will not know that and continue dispatching the workload in the same way. In the end, the heavy loaded JBoss node will have an increasingly longer job queue and eventually crash.

For the case where the JBoss node equals six, I have observed a different bottleneck neck in the SUT. The PostgreSQL database seems to become the bottleneck. When the ECperf output is close to the maximum values, the job queue on the database machine identified by the *top* command of the Linux operating system keeps high during the whole testing process. Also the CPU usage is 100% during most of the testing process. When the number of JBoss nodes is six, the PostgreSQL becomes a bottleneck, even when the JBoss nodes increase to eight, the actual performance output does not increase at all.

When the targeted txRate is bigger than the txRate, which gives the maximum output, the output variation under the same testing conditions is very big. That is to say, I

could have transaction outputs with big differences under the same testing conditions. This could be explained by the highly unbalanced workload between each JBoss instance in the JBoss cluster, which is caused by the default Round Robin load balance policy. Under a heavy workload, one or more JBoss in the JBoss Cluster becomes overloaded and does not work properly. As the testing process is highly dynamic, each experiment can produce a different transaction output that is dependent on the workload distribution.

I conclude that clustering of only Session Bean on JBoss cluster could provide good scalability up to six JBoss instances. However, the scalability is restricted due to the Bottleneck generated in the PostgreSQL database when the JBoss number is six. When the JBoss number varies from 1 to 4 in the JBoss Cluster, the bottleneck in the system is the CPU capacity in the JBoss machines.

## 6.4 Test with clustering of all EJB

**Testing procedure**

Clustering of only Session Beans provides some level of scalability, however, the transaction output could be restricted based on the fact that the Entity Beans have not been replicated. In this section, I am going to test clustering all EJB in the ECperf application.

As analysed in chapter 5, the testing architecture for Clustering of all EJB is exactly the same as clustering of only the Session bean, but clustering all EJB makes each node in the JBoss Cluster have a local copy of all EJB as illustrated in Figure 5.7. I expect increased transaction output as compared with the test of clustering of only session bean.

The testing process is exactly the same as testing of clustering of session bean.

**Result**

Table 6.5 gives the final result of test of Cluster of all EJB in the JBoss Cluster using Sisters.

| JBoss | 1 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| TPS | 615.1 | 1056.4 | 1723.5 | 1966.3 | 2286.5 |
| % of change | | 71.6% | 63.1% | 14.1% | 16.3% |

*Table 6.5: Testing result of cluster all* EJB *on Sisters*

These results are shown in the graph figure 6.5, with the X-axis representing the Number of JBoss server in the test, and the Y-axis representing the Maximum output against each JBoss value.

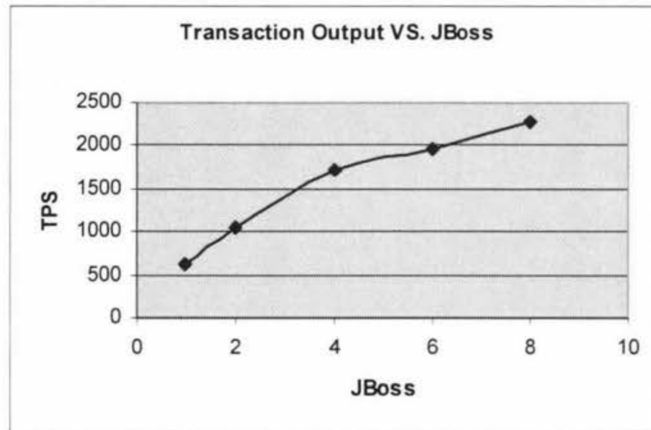


*Figure 6.5: Transaction Output VS. JBoss number when clustering all* EJB

**Result analysis, discussion and conclusion**

From the Figure 6.5, it can be seen that when the JBoss number varies from 1 to 4, the transaction output of the application increases almost linearly, with a factor of 71.6% and 61.3% respectively when doubling the number of servers. However, the transaction output increase from 4 to 6 is much smaller, with only 14.1% increased output from 4 to 6, and 16.3% from 6 to 8.

There is a clear performance improvement as compared with the testing result of clustering only Session Beans. The peak performance has also increased from 6 to 8 JBoss servers when testing with clustering of all EJB.

The performance and scalability has improved by clustering all EJB instead of clustering only Session Beans. The availability of the EJBs will be better when clustering all EJB, thus generating a better transaction output.

I have observed similar bottlenecks as in the testing of clustering of only Session Beans. For the test with only one JBoss instance, when the workload is heavy, the CPU usage of the JBoss machine is 100% and the job queue of the CPU identified by load average value is also very high, thus CPU capacity in the JBoss Cluster is the bottleneck. This bottleneck can be identified for JBoss instance numbers equals one, two and four.

For JBoss server equals six, both the PostgreSQL databases and some of the JBoss machines with higher workload seem to be overloaded. When JBoss nodes equal eight, the database becomes the bottleneck. When the ECperf output is close to the maximum values, the job queue in the database node keeps high and CPU usage is mostly 100% during the whole testing process.

Again, I have observed a problem created by the default Round Robin load balance policy. Some JBoss nodes have much higher job queues than others when the workload is heavy in the system.

I concluded that a good scalability has been achieved by cluster of all EJB. Better performance and scalability can be achieved by clustering all EJBs as compared with that of clustering of only Session Beans. The database is certainly the bottleneck for relatively larger JBoss Clusters. The load balance policy is a problem with heavy workload.

## 6.5 Test with First-Available load policy for clustering of all EJB

**Testing procedure**

The default Round Robin load balance policy has been used for all previous tests. Round Robin dispatches workload to each machine in the system one after another,

this schema will guarantee each machine gets the same number of job items disregarding the actual weight of each job.

Starting from JBoss 3.2, a new load balance policy called First Available becomes available for Clustered EJB. Under the first available policy, one of the available target nodes is selected randomly from the list of JBoss nodes and is used for every call [38].

I will test using the First Available load balance policy instead of the default one in this section. I will see how the workload distribution and transaction output can be affected by a different load balance policy.

Exactly the same test as with Clustering of all EJB in the last section will be run. I use the same hardware and software as illustrated in Figure 5.3. I also use the same testing procedure as described before.

**Result**

Table 6.6 is the final test result for clustering of all EJB with the First Available load balance policy.

| JBoss | 1 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| TPS | 615.3 | 993 | 1712.7 | 2217.4 | 2309.2 |
| % of change | | 61.4% | 72.5% | 29.5% | 4.1% |

*Table 6.6: testing result of cluster all* EJB *using First Available load-balancing*

Figure 6.6 shows transaction per seconds as the function of number of JBoss by using the data in table 6.6.

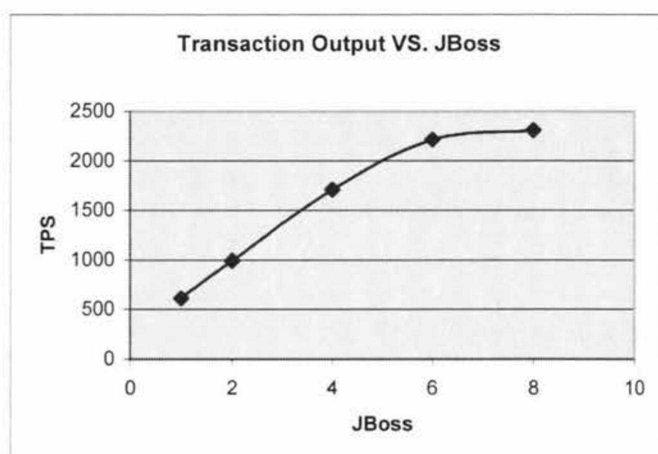


*Figure 6.6: Transaction output VS. JBoss number*

**Result analysis, discussion and conclusion**

In the graph, when the JBoss number varies from 1 to 6, the transaction output of the application increases almost linearly, with a factor of between 59% to 72.5% performance increases, whenever the JBoss number is doubled. When the JBoss number is increased from 6 to 8, the performance increase is very small. This indicates a bottleneck is reached when JBoss-Number equals 6.

Comparing the test result with that using the default Round Robin load balance policy, there is an obvious performance gain when the JBoss number equals 6. Recall for using Round Robin policy, I have both PostgreSQL and some of JBoss nodes become the bottleneck. By using the First Available load balance policy, I have got better load distribution in the JBoss Cluster, and thus effectively decreased the chance of a JBoss server being the bottleneck in the system.

For the SUT with only one JBoss instance, the CPU capacity of the JBoss machine is the bottleneck. Similar bottlenecks can be identified for JBoss instance numbers equals two and four. The load balance problems still exist, but the load distribution is much better than that of using the default one. For the number of JBoss node equals six and eight, because of better load distribution in each of the JBoss instances, the PostgreSQL database is the only bottleneck.

A good scalability result using the First Available load balance policy has been achieved. A better load balance policy can effectively improve the performance of a J2EE application by keep a better distributed workload in the JBoss Cluster, this has been proved by the results using the First Available load balance policy compared with that of using the default Round Robin policy. The database is certainly the bottleneck when JBoss number is more than six in the testing system.

**Comparison of load balance policies**

By comparing the testing result using first available load-balance policy with that of the default load-balance policy, the first available load-balance policy gives a much better result when the txRate value is high.

The First Available load balance policy can achieve a better transaction output than that of using the default Round Robin policy. Using First Available policy has created two distinct output differences:

1. The best throughput is better than that of using the default load balance policy.
2. The output after the peak is much more stable than that of using the default policy.

These two output characteristics can be explained with the load distribution result of using different load balance policy. Because the First Available policy can results in better distributed workload across the JBoss Cluster, the chance of one JBoss node becoming overloaded is much smaller than using the default Round Robin policy. So, I can achieve better transaction output under heavy workload.

Using the default Round Robin load balance policy, when one JBoss has been overloaded and becomes slow, the Round Robin algorithm will continue to distribute workload to each machine one after another. In the end, the overloaded machine will becomes slower and slower, with the job queue becomes longer and longer until this machine crashes. The Round Robin policy does not provide a chance for an overloaded node to recover.

The First Available algorithm can improve this situation in some way. Because the workload is distributed randomly, for a machine with higher workload, if new jobs

that are dispatched to this machine is less than to other machine, there is a chance for heavy loaded machine to decrease its workload, this is a big improvement as compared with the round robin algorithm that continues to dispatch workload to the machines in sequence one after another. However, the unbalanced workloads still exist because of the random distribution of workload in the JBoss Cluster. I will discuss possible implementation of better load balance policies such as a dynamic algorithm and weight-based algorithm in chapter 8.

Although, both the Round Robin and the First Available load balance policy cannot provide a really good workload distribution across machines in the JBoss Cluster, I will use the First Available load balance policy, which performs better under higher workload.

## 6.6 Test with two databases

**Testing procedure**

There are two problems in the previous testing results. The load balance problem has been partially solved by using a different load balance policy. Another problem is the PostgreSQL database problem, because the hardware in the database machine has become the bottleneck, it's not possible to gain better performance by using just one database. This can be improved by using a distributed testing architecture instead of using centralized architecture. The database is split into two separate databases each containing part of the centralized database.

I use the hardware and software as illustrated in Figure 5.8. My testing procedures are slightly different compared with the centralized test. After splitting one centralized database into two separate databases, I will examine how the workload distribution and transaction output can be affected.

Figure 5.8 is the EJB replication graph with two databases in the system. All the EJB are replicated in all members in JBoss Cluster, but the data that represent these EJB

are split into two separate databases. During the testing process, the Driver client still interacts with all JBoss servers, but the JBoss servers need to interact with the two databases depending on what data need to be accessed.

**Result**

Table 6.7 is the final result for test with two databases.

| JBoss | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| TPS | 1004.8 | 1710.4 | 2343.1 | 2907.4 |
| % of change | | 70.2% | 37.0% | 24.1% |

*Table 6.7: test result with 2 databases*

These results are shown in the graph figure 6.7; with the X-axis representing the Number of JBoss server in the test, and the Y-axis represent the Maximum output against each JBoss value.

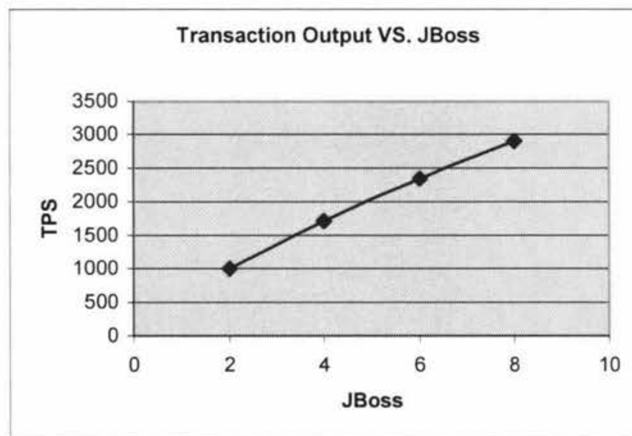


*Figure 6.7: Transaction Output using the distributed architecture*

**Result analysis, discussion and conclusion**

Figure 6.7 shows a consistent performance gain when the JBoss number is increased from 1 to 8. The application output increase is almost linear, with a factor of from 70.2% to 74% whenever the JBoss number is doubled.

By comparing the result with that of using one database, these are significant performance gains with two databases

During the test, the JBoss machine is always the bottleneck from one to eight JBoss in the JBoss cluster. When the JBoss servers are six or eight, the load balancing is a problem. Even with the First Available load balance policy for all EJB, the workload differences between each JBoss can become very large.

By using the distributed test architecture, the hardware bottleneck when using a centralized database has been solved. Using two databases has effectively distributed the database workload to two machines and thus solved the database bottleneck for a centralized database.

A linear scalability could be achieved using a distributed architecture for the ECperf test using Sisters. A much better performance and scalability have been reached by using a distributed test rather than that of using centralized test. With two separate PostgreSQL databases, the database always has enough capacity to deal with the workload from the JBoss Cluster, thus making the JBoss Cluster the only bottleneck in the system, this leads to linear scalability results. However, the load balance policy is still a problem even using the First Available load balance policy.

## 6.7 Test with Two partitions and two databases

**Testing architecture**

The clustering of EJB is a good strategy to improve system performance, but too many copies of one EJB object may not be desirable. As there are overheads associated with the JBoss Cluster, the performance gain of the application may not be proportional to the copies of the EJB objects in the JBoss Cluster. The overhead associated with the clustering of EJB includes communication between members in the JBoss cluster and action to synchronise an EJB access to maintain data integrity in the database.

For a better performance, two separate JBoss partitions will be created, and each of them interacting with a different database. Figure 5.9 in the last chapter gives the testing hardware architecture with two JBoss partitions and two databases.

In the architecture, the JBoss cluster is divided into two separate partitions. There is no communication between members in two partitions; only members in the same partition will interact with each other. Only EJB that belong to the Order Database will be replicated in the Order Partition, only EJB that belong to the Manufacturing Database will be replicated in the Manufacturing Partition.

Figure 5.10 in the last chapter gives examples of how Enterprise Java Beans were replicated to one of the two partitions in the JBoss Cluster; it also shows how each JBoss server interacts with just one of the two databases. For example, EJBs such as Cart Session Bean, Order Session Bean and Order Entity Bean are replicated to the JBoss servers that belong to the Order Partition, and the JBoss servers in the Order Partition only interact with the Order Database (OrderDB). In short, the client will still interact with all the JBoss machines, but a client's request will only be directed to a JBoss server in one partition where the particular Session EJB exists.

By using two separate JBoss Clusters, the numbers of local copies of an EJB are effectively reduced. Reduced EJB persistence overhead as well as the communication overhead between JBoss cluster members should improve the ECperf application performance.

I have chosen the following testing algorithm for this test.

*For JBoss-Number =1, 2, 4, 6 and 8*
*Do test with transaction rate (txRate) = 1, 2, 4, 8, 12, 16, 20, 24, 28, 32*
*(Repeat three times for each txRate value)*
*End*

**Result**

Table 6.8 is the final result of the ECperf applications running with two JBoss clusters and two separate databases.

| JBoss number | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| TPS | 1148.8 | 1928 | 2590.5 | 3280.5 |
| % of change | | 67.6% | 34.4% | 26.6% |

*Table 6.8: Transaction output with 2 Partitions and 2 Database systems*

These results are shown in the graph figure 6.8, with the X-axis representing the Number of JBoss servers in the testing program, and the Y-axis representing the Maximum output against each JBoss value.

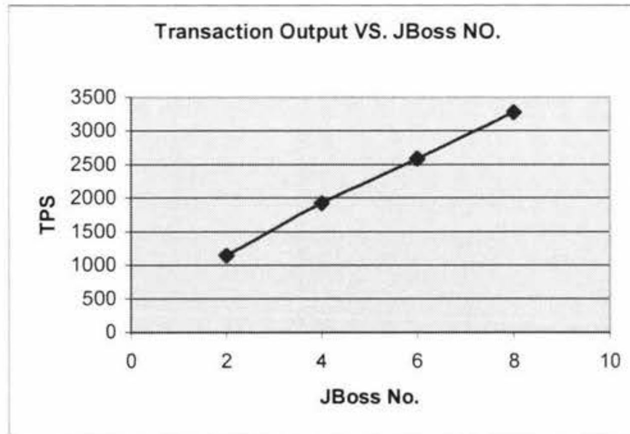


*Figure 6.8 Transaction Output with 2Partitions and 2 Databases*

**Result analysis, discussion and conclusion**

Figure 6.8 shows consistent performance gains when the JBoss number is increased from 2 to 8. The application output increase is almost linear, with a factor 67.6%, 68.8% and 79.8% whenever the JBoss number is doubled. This trend indicates strong scalability for testing systems with larger numbers of JBoss in the JBoss Cluster.

By comparing the result with that of using two databases and only one partition, there is a significant performance gain when using two partitions and two databases.

The same bottlenecks as presented with the test of one partition and two databases still exist. The JBoss machine is always the bottleneck from two to eight JBoss in the JBoss cluster. Even with the First Available load balance policy, the differences of workload between members in JBoss Cluster can become very large under heavy workload.

I conclude that linear scalability can be achieved using a distributed architecture with two partitions and two databases in the system. Improved performance has been achieved as compared to a distributed test with only one partition. With two JBoss

partitions and two separate PostgreSQL databases, the database always has enough capacity to deal with the workload from the JBoss Cluster, thus making the JBoss Cluster the only bottleneck in the system, this leads to linear scalability results. However, the load balance policy is still the problem even using the First Available load balance policy.

## 6.8 Test by disabling the log files with two partition and two databases

**Design of the test**

Log files are very important for identifying errors while running a program and getting useful history information for a user. However, for a high workload application, because of the high volume of transactions in the system, a large amount of information will be written to the log file and thus consume a lot of system resources.

Disabling the writing of data to the PostgreSQL log files and writing information to the JBoss application server log file, and only writing error messages to the console window should further improve performance.

All the testing architecture and the procedures are the same as testing with two partitions and two databases.

**Result**

Table 6.9 is the final result of the ECperf applications running with two JBoss clusters, two separate databases and disabling the log files.

| JBoss number | 2 | 4 | 6 | 8 |
|---|---|---|---|---|
| TPS | 1140.8 | 2091.3 | 2983.2 | 3672.9 |
| % of change | | 83.3% | 42.6% | 23.3% |

*Table 6.9 is the transaction output with 2 partitions and 2 databases and disabled log*

The result are shown in the graph figure 6.9, with the X-axis representing the Number of JBoss servers in the test, and the Y-axis representing the Maximum output against each JBoss value

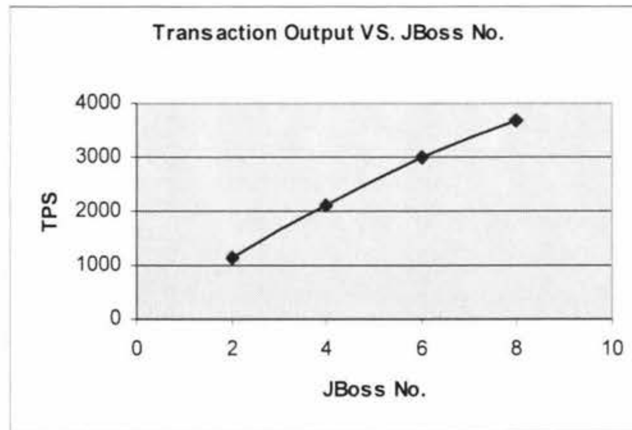


*Figure 6.9: Test with 2 partition, 2 DB & disable log file.*

**Result analysis, discussion and conclusion**

Figure 6.9 shows a consistent performance gains when the JBoss number is increased from 2 to 8. The application output increase is almost linear, with a factor of 69.9% and 83.3% when the JBoss number is doubled. This trend indicates strong scalability for testing systems with a larger number of JBoss in the JBoss Cluster.

Comparing the result with that of using two databases and two partitions, I can see a significant performance gain by disabling the log file. The default log file is useful because it records most of the system activities, but when the workload is very heavy, writing a lot of information to the log file, consumes a lot of system resources and can becomes a big overhead for the JBoss application server. So, disabling log files in the JBoss application servers has given a performance improvement.

The JBoss machines are always the bottleneck from two to eight JBoss in the JBoss cluster. The databases have no problems handling the workload. An unbalanced workload distribution in JBoss Cluster is still a problem when the workload is heavy.

I conclude that linear scalability can be achieved using a distributed architecture with two partitions, two databases and disabled log file. Improved performance has been achieved as compared with distributed testing with two partitions. The JBoss

members in the JBoss Cluster are the only bottleneck in the system. However, the load balance policy is still a problem even using the First Available load balance policy.

## 6.9 Discussion of the Sisters Result

**Initial conclusion from the Sisters result**

Figure 6.10 gives transaction per seconds as the function of number of JBoss in the JBoss Cluster. I have presented six different testing results:

- Test of Clustering of only Session Bean
- Test of Clustering of all EJB using Round Robin load balance policy
- Test of Clustering of all EJB using First Available load balance policy
- Test using two databases
- Test using two databases and two JBoss partitions
- Test using two databases, two JBoss partitions and disabled log file

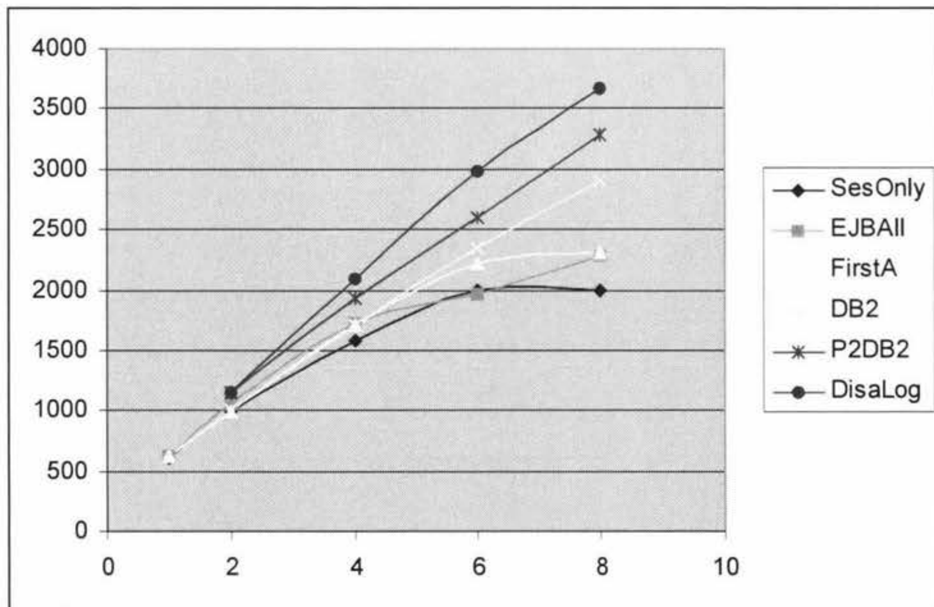


*Figure 6.10: All testing results for the sisters*

**Scalability analysis for the Sisters results**

Good scalability results are shown with all types of test in Sisters. For the centralized test, the database becomes the bottleneck when JBoss number is six in the JBoss Cluster; the potential scale up to more than eight JBoss Cluster is not big. For a distributed test with two databases, I have nearly linear scalability when the JBoss number increase from one to eight. The scalability results have implied good scalability potential for systems with more than eight JBoss. These results have been further improved with partitioning the JBoss cluster and disabling extensive log file outputs.

**Performance Bottleneck analysis based on sisters result**

Whenever the transaction output does not increase with the client workload, there is a bottleneck in the system. The bottleneck could be any resources in the testing system, such as the operating system, the Java Virtual Machine, the JBoss application server, the PostgreSQL database, the hardware and the network resources.

By identifying the bottleneck in the test system, a suitable solution can be found and further improvements made to the performance. One interesting point is that finally the bottlenecks can never be resolved, because whenever one bottleneck is solved, another system resource will become the new bottleneck when the workload is higher.

Detailed description of the bottlenecks and solutions for each test are given in each section. Comparing all testing results in Figure 6.10, improved performance has been achieved when each solution was applied.

## 6.10 Summary

Various tests of the ECperf application running on the Sisters have been completed. A centralized test with Java Virtual Machine heap size parameters, with the default database connections, with the clustering of session bean, with clustering of all EJBs using the default load balance policy and with cluster of all EJB using the First Available load balance policy have all been completed.

I have also run distributed test with two databases, with two databases plus two partitions and with disabled log file in both JBoss and the PostgreSQL databases.

With all these tests, very good scalability results have been achieved for 1-4 JBoss server. For each test, when the JBoss number in the JBoss cluster increases, the workload that the whole system can handle also increases. Good scalability up to 8 JBoss server has been achieved for distributed workloads.

In the process of identify the system bottlenecks and finding the solutions, the flexibility that the J2EE provides can be seen. By using different deployment architectures, improved performance can be achieved without changing the source code. When moving from one application server vendor to another, the only file that needs to be changed is the deployment descriptor of the ECperf application in XML format. The flexibility of Beowulf Cluster allows hardware resources to be increased by adding a JBoss instance machine into the JBoss Cluster and solving any problem associated with the JBoss application server.

# Chapter 7: Test on Helix – The supercomputer

## 7.1 Introduction

This chapter will examine the performance of the ECperf application in the Helix, the supercomputer in Massey University. Due to the similarity of the test in the Sisters and the Helix, as well as the limitation of the hardware resources available in Helix only a subset of the experiments will be discussed. The experiments to be presented in this chapter are:

- JVM heap size test running one JBoss instance
- Test using the default JBoss connection pool value
- Test using optimised database pooling in JBoss

I will also give performance analysis and identify the bottlenecks based on each test result.

## 7.2 JVM heap size test

**General hardware analysis in Helix**

I have introduced in chapter 3 the architecture of the Beowulf supercomputer: the Helix. There are 66 nodes in the Helix, one (Helix0) as the server node, and the rest (helix1 to helix65) as the client nodes.

The test program is a distributed application that needs to run in at least three layers: the Driver Client (or the Emulator), the JBoss_tomcat server and the Database. Figure 5.12 in chapter 5 gives the hardware architecture of my test in Helix.

The basic requirement for the ECperf test is to use dedicated hardware for the System Under Test (SUT), the SUT include all the JBoss server machines and the

PostgreSQL database machine. As the Helix is a supercomputer shared by a lot of users and also because it is normally busy, the different components must be carefully chosen.

As the Driver client is not part of the SUT, and also requires the least resources, it will run on Helix0 – the server machine.

Choosing the database node needs careful consideration, because all JBoss servers need to be configured to point to a database in a particular machine. A fixed database machine can allow minimum tests. I choose Helix65 as the database node, as it is most unlikely to be used by others. All the remaining Helix nodes, from Helix1 to Helix64 might be selected as the JBoss server nodes when possible.

**Design of the JVM Heap Test**

As for the Sisters, the JVM machine setting on the helix will affect the performance of the Clustered application. The purpose of the test is to find out the optimal Java Virtual Machine settings that will be used for the later test. Although, my major emphasis is to test performance of a cluster of application servers, it's not necessary to do that for JVM heap size test. Once the optimal JVM heap size settings for one machine are found, it can then be applied to the whole cluster of JBoss servers.

Figure 5.11 in chapter 5 gives the hardware architecture of the JVM heap size test. Here, the Driver client, JBoss_tomcat server and the PostgreSQL database are running one separate machines.

The JVM heap space test used one of the Helix client machines that has a total memory of 1000MB.The Min & Max heap size are set to be equal, that is –Xms=-Xms (the default setting will be an exception). The test cases will include the following: using the default heap space, 100MB, 200MB, 300MB, 400MB, 600MB, 800MB and 1000MB. For each of these values, the value of txRate in the ECperf testing program will be steadily increased, this will increase the number of ECperf client that interact with the ECperf server.

Here is the test algorithm:

*For –Xms equals –Xmx with value of (default, 100, 200, 300, 400,600, 800, 1000)*
*Do test with transaction rate = 1, 2, 4, 8, 12, 16, 20, 24, 28, 32*
*(Repeat three times for each transaction rate value)*
*End*

In the algorithm, for each pair of the –Xms and –Xmx values, I get a sequence of tests result that corresponds to the txRate of 1, 2, 4, 8, 12, 16, 20, 24, 28 and 32. I repeat test for each txRate value for three times. By increasing the value of txRate in the ECperf testing program, the number of threads, each simulating one client is also increased.

**JVM Heap space Testing Result**

For each of the JVM heap space values, there is a maximum transaction output. The Maximum transaction output (TPS) VS the JVM heap space values (-Xmx) are shown in table 7.1.

| -Xmx | 64 | 100 | 200 | 300 | 400 | 600 | 800 | 1000 |
|---|---|---|---|---|---|---|---|---|
| TPS | 100.9 | 359.6 | 946.8 | 1228.7 | 1743.9 | 2140.1 | 2170.9 | 2095.6 |

*Table 7.1: JVM Heap Value VS. TPS Test*

These results are also shown in graph figure 7.1.

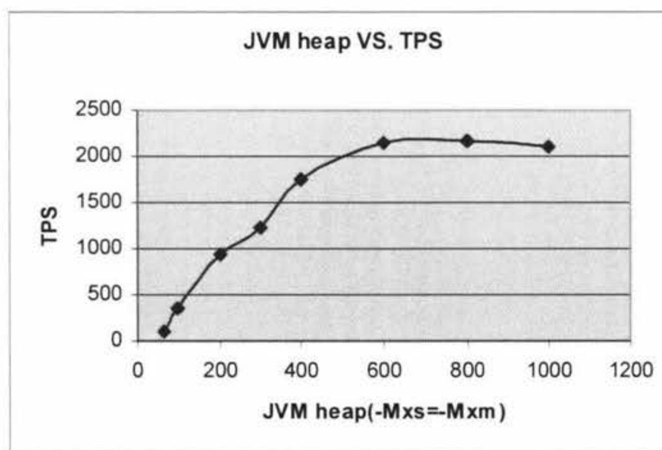


*Figure 7.1: Maximum throughput as a function of the JVM Heap Size*

**Result Analysis, further discussions and conclusion**

Figure 7.1 shows how the JVM heap size will affect the performance. With the increase of the JVM heap size value, the maximum transaction output is increased. When the value of JVM heap size smaller than 600MB, the rate of increase of the output is good. Then the output increment rate is slower until a maximum performance value is reached. After that, the transaction output decreases slowly when the JVM heap size increases.

Figure 7.1 shows that when the JVM heap size value is from 60% to 100% of the maximum memory available in a machine, the transactions output is close to the maximum output. The maximum transaction output was produced when JVM heap size value is 800MB. This value will be used for the remaining ECperf tests.

This is a reasonable result. Because the JBoss application server is running on top of the JVM, the size of memory space in the JVM available to JBoss will be an important factor related to how much work a JBoss can do. Given more JVM heap size means more memory space available to store objects created by the application running on JBoss, the transaction will increase with the more resources available.

When the heap value is less than 400MB the bottleneck is the size of the memory. After the JVM heap value reaches about 500MB. The memory space defined by the JVM heap value is no longer a bottleneck in the system. The utility command *top* shows that the JBoss node in the system has becomes the new bottleneck. The CPU usage in the JBoss node is 100% (including the users plus the systems CPU usage). As this CPU usage keeps consistent in the system, the Job queue in the JBoss server keeps growing until the whole system becomes slower and slower and eventually to a state of giving a very small output Figure 7.1 shows that the output keeps going down after the best output is achieved with a heap size of 800MB.

The optimal JVM heap value of -Xms = -Xmx = 800MB is used for the remaining tests and keep other JVM parameters as default values.

## 7.3 Test using the default data source pooling values

**Design of the test of clustering all** EJBs

In the system, there is one Driver Client machine that is running a multi-threaded java application to emulate many simultaneous client's access to the EJBs running on a cluster of JBoss application servers each running in one node. The EJBs in the application are replicated across the application server to provide high availability and fail-over feature support.

As the test results in Sisters have shown that cluster of all EJBs will performance better than cluster of only session beans. I will not test with clustering of the session beans only, instead, I will test clustering of all Enterprise Java Beans using the default data source pooling in the JBoss servers. The EJB replication schema is given in Figure 5.8 where all EJBs are replicated to each JBoss member.

To test the scalability of the J2EE application server, I need to get the highest output for a particular JBoss Cluster. The following test algorithm is used. As discussed in the early section of this chapter, these test have been automated for the purpose of convenience.

*For JBoss-Number = 1, 2, 4, 8, 16, 24, 32*

*Do test with transaction rate (txRate) = 1, 2, 4, 8, 12, 16, 20, 24, 28, 32*

*(Repeat three times for each txRate value)*

*End*

**Result**

Table 7.2 gives the final result of test of Cluster of all EJB in the JBoss Cluster using helix.

| JBoss | 1 | 4 | 8 | 16 | 24 | 32 |
|---|---|---|---|---|---|---|
| TPS | 811.5 | 1643.6 | 2337.6 | 3035 | 3446.5 | 3744.5 |
| % of change | | 102.5% | 42.2% | 29.8% | 13.6% | 8.6% |

*Table 7.2: Transaction output VS. JBoss Number*

The results are shown in figure 7.2 with the X axis representing the Number of JBoss servers in the test, and the Y axis represent the maximum output against each JBoss value.

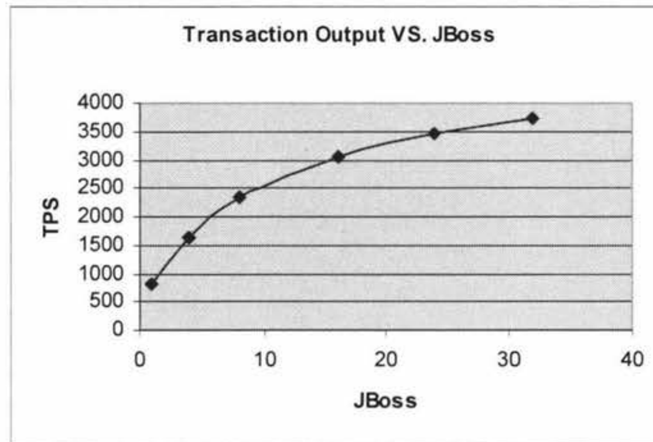


*Figure 7.2: Transaction Output as a function of the JBoss number*

**Result analysis, discussion and conclusion**

Figure 7.2 shows that when the JBoss number is increased from 1 to 32, the transaction output of the application increases. However, the rate of increase in the performance is actually decreased when the JBoss number becomes bigger. I have achieved 51.3%, 51.3%, 42.4%, 29.8% and 23.4% performance increase when the JBoss number is doubled with 1, 2, 4, 8, 16 and 32. The performance output increase is almost flat when the JBoss number is increased from 24 to 32, with only a slight 8.6% increase in transaction output. This indicates a bottleneck is reached when JBoss-Number is more than 24. I will discuss this in the following section.

For the SUT with only one JBoss instance, when the highest output is created, either the JBoss or the PostgreSQL has become the hardware bottleneck, because the CPU and RAM usage in both machines have not reach 100% limit. The PostgreSQL also does not reach the maximum number of connection limit. However, the default number of maximum connections allowed in the JBoss application server, the MaxSize which defines the maximum allowed database connection pool numbers in the JBoss server is 25. This can be a bottleneck as shown in the next section.

For the case where the JBoss node equals 24 and 32, I have observed a different bottleneck neck in the SUT. The PostgreSQL database seems to become the

bottleneck. When the ECperf output is close to the maximum values, the job queue on the database machine identified by the *top* command of the Linux operating system keeps high during the whole testing process. Also the CPU usage is close to 100% during some of the testing process. When the number of JBoss nodes is 24, the PostgreSQL becomes bottleneck, even when the JBoss nodes increase to 32; the actual performance output only increases slightly.

I concluded that clustering of all EJBs using JBoss cluster in Helix could provide good scalability up to 24 JBoss instances. However, the scalability is restricted due to the Bottleneck generated in the PostgreSQL database when the JBoss number is 24. When the JBoss number is from 1 to 16 in the JBoss Cluster, the bottleneck in the system is the default maximum allowed database connection pool value in the JBoss machines.

It should be noted that except for some poor load balancing effects the JBoss server node were not overloaded in these tests.

## 7.4 Test using optimised database pooling

**Design of the test of clustering all** EJBs

The test performance on the Helix can be improved by using a better data source pooling parameters in the JBoss Cluster and also disabling the JBoss log file. The same hardware architecture as previous test is used. The following test algorithm is used for this test

*For JBoss-Number = 1, 2, 4, 8*

*Do test with transaction rate (txRate) = 1, 2, 4, 8, 12, 16, 20, 24, 28, 32*

*(Repeat three times for each txRate value)*

*End*

**Result**

Table 7.3 gives the final result of test of clustering of all EJB in the JBoss Cluster using helix.

| JBoss | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| TPS | 2355.4 | 3628 | 3680 | 3640 |

*Table 7.3: Transaction Output VS. JBoss number in Helix*

The results are shown in figure 7.3 with the X-axis representing the Number of JBoss server in the testing program, and Y-axis representing the Maximum output against each JBoss value.

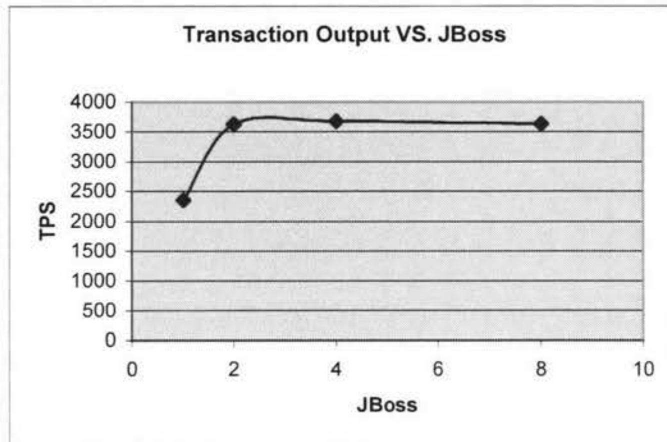


*Figure 7.3: Transaction Output VS. JBoss number in Helix*

**Result analysis, discussion and conclusion**

Figure 7.3 shows that when the JBoss number is from 1 to 2, the transaction output increases 54%, but this output is almost the maximum output, as a similar output is obtained when the JBoss number is 4 and 8. This has indicated a bottleneck has reached when the JBoss number is 2.

Comparing the results with results from the last section, the performance has improved by using the optimal data source pooling values. With one JBoss in the JBoss cluster the output increased 190.2%, and output increased 120.8% when the JBoss number is 2 in the JBoss cluster as compared with the default database polling values. These results have shown the default database pooling value is a bottleneck when running the ECperf application in Helix.

In the test, the hardware such as the CPU and the RAM in the testing system do not seem to be the bottleneck. However, the Job queue in the PostgreSQL database is high

during some of the testing period. The database is the bottleneck during the test. Due to the bottleneck in the PostgreSQL database, it is natural that the ECperf application output does not increase even when the JBoss number in the JBoss cluster is increased.

## 7.5 Summary

Three experimental tests of the ECperf application running at the Helix were completed. Tests were run with different JVM heap space values, a centralized test with the default JBoss database pooling values and test with optimised values were also run.

The testing results have shown good scalability for testing output. The JVM test showed the optimised JVM heap size is critical for ECperf application output running in a cluster of JBoss application server. When using default database pooling in the JBoss members in the JBoss Cluster, a scalable result was reached up to 32 JBoss members, this indicated very good performance potential for a cluster of JBoss servers however the database polling bottleneck severely limited the maximum performance achieved. The test using optimised database pooling in the JBoss member has indicated scalability until the database becomes the bottleneck in the testing system. It is disappointing that this bottleneck was reached with just 2 JBoss servers.

When the JBoss number in the JBoss cluster increase, the workload that the whole system can handle also increases, thus achieved scalability with the JBoss cluster.

Several factors such as the database connection pooling and the database scalability might affect the system performance, Although a Beowulf cluster provides some flexibility for distributed deployment, it also has some restrictions on software that need particular good hardware resources. One example is the database, because there is only one database machine in the system, it is easy to become the bottleneck due to

the hardware resources available in that machine. Ideally the database system hardware should be better than the JBoss Cluster machines.

# Chapter 8: Performance Analysis and Discussion

## 8.1 Introduction

In this chapter, scalability analysis based on the results from both Sisters and Helix is presented, followed by a discussion of the identified performance bottlenecks for the system. A general discussion about how to improve J2EE application performance through the application design architecture, component implementation, application deployment, system hardware, software and networks, is also given.

Techniques on how to improve the ECperf applications performance by changing the hardware and software used for the JBoss server and the PostgreSQL database are discussed. The chapter finishes with an architecture that can be further developed for generating better performance. Finally, suggestions of how my research results might be applied to the commercial world for running high performance J2EE application in a cost effective way are presented.

## 8.2 Scalability analysis for Sisters and Helix

**Concept of Scalability**

Scalability is the ability to provide quality service to the client as the workload increases. These are two ways to achieve scalability for a J2EE application.

Vertical scalability is achieved by upgrading hardware resources to existing system; such as increasing the capacity of the memory, CPUs and Cache. Achieving vertical scalability does not require a change to the system architecture or application running on the system. An increased performance can be achieved, but reliability and availability may not increase.

Horizontal scalability is achieved through duplicating the server (application server and web server) or database resources. By using a cluster of application servers in the

system, the system architecture becomes more complex to manage, but with the added performance, availability and fault tolerance also improves. Hardware or software must be used to achieve load balancing for the cluster of application servers or databases.

**Vertical Scalability (JBoss=1)**

Vertical and horizontal scalability have been used in the tests on the Sisters and Helix. Table 8.1 gives the test results with the same hardware and software architecture in Sisters and the Helix. The system contains one Driver Client machine, one JBoss machine and one PostgreSQL database machine.

| txRate | 1 | 2 | 4 | 8 | 12 | 16 | 20 | 24 | 28 |
|---|---|---|---|---|---|---|---|---|---|
| TPS Sisters | 103.6 | 198.8 | 413.4 | 566.7 | 490.3 | 533.7 | 496.7 | 406.4 | 425.4 |
| TPS Helix | 101.6 | 201.4 | 409.6 | 798.2 | 1221.8 | 1623 | 2030.4 | 2355 | 2095.8 |

*Table 8.1: Transaction Output in Sisters and Helix (JBoss=1)*

Figure 8.1 gives the final test results on the Sisters and Helix on the architecture, which includes one Driver client machine, one JBoss application server machine and one database machine.

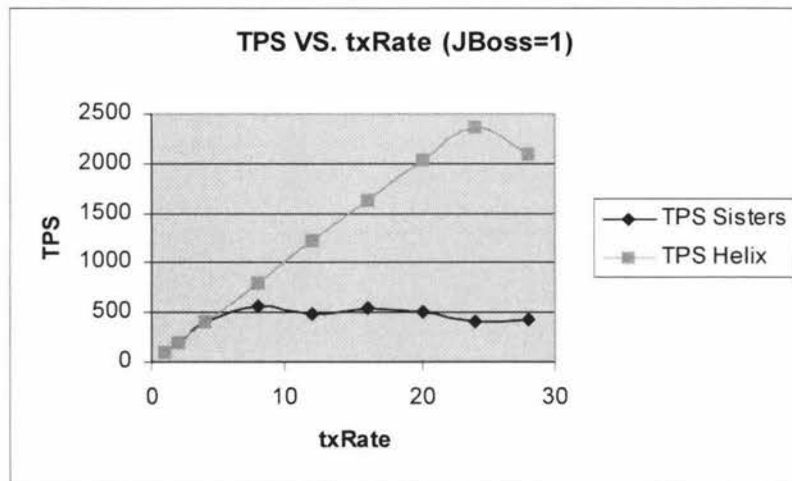


*Figure 8.1: Helix and Sisters transaction output with one JBoss*

Although the same test architecture is used for both the Helix and Sisters, the hardware capacities of the two systems are different. The Helix consists of Athlon MP2200 MHz AMD with 1GB RAM for each of the 64 nodes, and the Sisters

consists of 667MHz Pentium III with 256MB RAM. Because each machine in Helix has better CPU and RAM compared with the Sisters machine, vertical scalability is displayed when comparing the result of Helix and Sisters with the same testing hardware architecture.

Figure 8.1 shows the test results with only one JBoss node in the SUT, for this kind of test, because the JBoss node is the bottleneck when the system reaches its highest output, I can safely ignore the fact that the hardware used for PostgreSQL database and Client Driver node in Helix is better than that in Sisters. The better CPU and RAM in the application server can improve the performance of the ECperf application.

The test results show that a 415% performance improvement results when running the same test in Helix as compared with that in Sisters. This result has proved that vertical scalability can be achieved by upgrading the CPUs, RAM or other hardware resources. Identifying the bottleneck in the system is again important because upgrading the hardware resources that are not the bottleneck of the system may not be helpful at all. For example, the database has enough capacity to deal with the test with only one JBoss node in the system, using a Sisters node or Helix node does not matter.

**Vertical scalability (JBoss=2)**

Table 8.2 gives the test results with the same hardware and software architecture in Sisters and the Helix. The system contains one Driver Client machine, two JBoss machine and one PostgreSQL database machine.

| txRate | 1 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TPS Sisters | 98.3 | 405.9 | 802 | 1056 | 960.9 | 930.8 | | | | | | |
| TPS Helix | 103 | 404.8 | 797 | 1213 | 1611 | 2005 | 2428 | 2807 | 3209 | 3587 | 3620 | 3645 |

*Table 8.2: Transaction Output in Sisters and Helix (JBoss=2)*

Figure 8.2 shows the final test results of Sisters and Helix on the hardware architecture, which includes one Driver client machine, two JBoss machines in JBoss Cluster and one database machine.

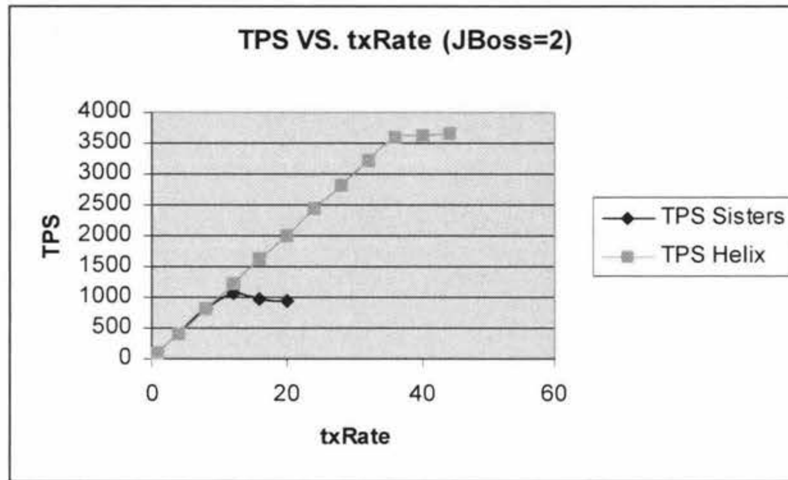


*Figure 8.2: Helix and Sisters transaction output with 2 JBoss*

The database and the client machine factors can be ignored based on the fact that the JBoss machine is the bottleneck in the system. The test results show a 371% performance improvement results when running the same test in Helix as compared with that in Sisters. This result has further proved vertical scalability by upgrading the CPUs, RAM of the hardware resources. With properly identified hardware bottlenecks in the system and applying solutions by upgrading the hardware resource, good vertical scalability is achieved.

**Horizontal Scalability Result**

Horizontal scalability has been tested in the Sisters and the Helix. The results show that Sisters can achieve very good scalability with the increase of the JBoss number in the JBoss cluster. However, hardware is not the only important factor that affects the system output. The software factors also contribute a lot to the J2EE application server performance output. These factors include the Operating System, the Java Virtual Machine, the JBoss application server, the database capacity and the ECperf application deployment strategy.

The Helix also shows a very good potential for horizontal scalability. Unfortunately, the hardware in the PostgreSQL becomes the bottleneck even when the JBoss node

equals 2 in the JBoss cluster. The hardware capacity for the database node is a critical factor for a high performance application that requires high volumes of database access through both read or write calls.

## 8.3 Bottleneck Analysis

Because of the complexity of the whole system, there are many factors that might become a bottleneck with a high volume of workload on the sisters and helix. These factors are mainly related with the system hardware type, the software used and the application deployment settings. Some of the bottlenecks can be solved with out adding extra resources, while other can only be solved with extra hardware resources.

**Hardware bottleneck**

The hardware capacity is the most fundamental factor for supporting high performance J2EE applications. Based on the fact that only a constant maximum output capacity can be achieved with a particular hardware, it is normally impossible to accomplish better performance for applications running on particular hardware architecture. However, with the Cluster of JBoss application servers, it is now possible to solve hardware bottlenecks that occur in the application server or web server by running a cluster of JBoss servers. Running JBoss clusters on a Beowulf Cluster has simplified the deployment of the application, but also provided very flexible hardware architecture that can scale in a cost effective way.

In the Sisters, I have seen the CPU of the JBoss machine becomes the bottleneck when there's only one or two JBoss in the testing system. Increasing the total JBoss numbers in the JBoss Cluster can effectively solve the problem by distributing the workload to each member in the Cluster.

**Software bottleneck**

The software performance is a complicated problem and related directly with the implementation of the source code, such as the programming language efficiency, the architecture and the quality of the source code. Different software can display

dissimilar features of flexibility to support applications that requires performance and scalability. Good software provides adjustable parameters to support an application that requires better performance and scalability.

Some of the software factors become the bottleneck when doing the test. For the Java Virtual Machine, I have seen "*OutOfMemeory*" error message when using the default JVM setting, I can solve this problem by simply redefining the Java heap value of –Xmx to allocate more memory to the heap space. But the allowed JVM heap size is restricted by the RAM space available in the machine where the JVM running. It is counterproductive to allocate JVM heap size more than available RAM value.

The PostgreSQL database also has parameters to adjust its behaviour. I have seen error message of "*FATAL: Non-superuser connection limit exceeded*" when using the default allowed number of connection in the PostgreSQL database with higher workload. I can easily solve this problem by defining the maximum allowed number of connections and associated buffering size. The maximum allowed connection wouldn't be more than 1024 for my Linux operating system, as the maximum-shared memory defined by the Linux kernel cannot be exceeded unless changed by the administrator. The transaction rates specified in my experience did not require more than 1024 connections.

The JBoss application server has flexible clustering support. When a J2EE server component such as EJB is clustered in the JBoss Cluster, each JBoss in the group owns a copy of the EJB component, with the load balance mechanism provided by the JBoss application server, the system can provide very good scalability and performance. Most bottlenecks associated with using one JBoss server can be solved by using JBoss Clustering.

**Network bottleneck**

The network was most likely to become a bottleneck historically due to the speed limitation of transferring data. With today's higher bandwidth network using 1Gbps network adaptors, the network can normally provide enough speed to support communications between each node in the system.

For the sisters, I have a 100Mbps network connection; even this can support the ECperf testing application without problems. For the Helix with 1Gbps network connections, the communication between nodes in the Helix is very good. I do not anticipate any network bottleneck between any of the machines running the ECperf application. To test the network traffic, I have used an open source network benchmark application: the Netperf [42] to test my network traffic during various testing conditions. For an ECperf test with 32 JBoss nodes in the testing system, the peak total traffic volume for the busiest node (the database node) is about 180Mbps. This testing result using the Netperf application has proved that the network in the system is sufficient for my test.

## 8.4 Performance Tuning for the current testing system

Performance tuning requires consistent effort. Increased performance is normally achieved by identifying bottlenecks and by finding a way to solve the problem.

To achieve good performance and scalability, a broader view of most important factors in the system that could affect a J2EE application performance must be given

A J2EE application performance is related to resources in the system. These factors include generally the following:

- The application design, assembly and deployment options
- The hardware capacity in the system
- The operating systems support to the application
- Java Virtual Machine
- JBoss cluster features
- JDBC connections and resource pooling
- PostgreSQL databases functionality
- Database Storage capacity

**Application design options**

The effects of bad application design and development cannot be easily overcome with system tuning, or by making additional computing resources available to the application. Therefore, application related best practices should be followed from the start [46].

A pattern is a proved best practical solution to a common recurring problem. Using design pattern in J2EE applications is particular useful as it is a new and still maturing technology.

There are several different J2EE patterns that are used to address special problems. For example, Session Façade is a very basic EJB pattern that wraps the entity bean layer in a layer of session bean [28]. Clients only need access to the business method exposed by the session bean instead of the entity bean, thus simplify the application design. Using Session Façade can also reduce the network overhead by combining all the Entity Bean interaction into the session bean.

Using J2EE patterns or best practices in all levels of the application can make a clear design and implementation as a well as a high performance output.

**Hardware capacity**

Hardware capacity, such as the CPU, RAM and cache size is critical for application performance. To achieve high performance output, a computer system must provide hardware capacity that is large enough. Any hardware bottleneck can restrict the whole application's scalability.

CPU speed limits how much work a machine can handle in a given time of period. The CPU usage in any machine in the system should be below 100% during the application-running period.

The CPU must deliver its data at a very high speed. The regular system RAM cannot keep up with that speed. Therefore, a special RAM type on the processor called cache is used as a buffering temporary storage. The fastest cache RAM is called L1 cache. The next layer is the L2 cache. When the CPU needs data, it first checks the fastest

source — L1 cache. If the data is not there, the CPU checks the next-fastest source — L2 cache. If the data still cannot be found, a time-consuming search of the slower RAM is required.

Getting data from the L2 cache is about 10 times faster than from the RAM, getting data from the RAM is over 100 times faster than that of from the hard disk [47].

Luiz have show that online applications that have significant instruction and data locality can benefit with large off-chip caches [48]. The distribution of cache in a computer system also affects application performance. Caches shared by multi-processors can increase the cache hit and improve application output [49].

In conclusion, the L2 cache, CPU and RAM are critical for application performance. A system that does not provide sufficient hardware resources can greatly restrict performance of the application due to a hardware bottleneck in the system.

**Java Virtual Machine memory allocation**

The JVM is critical for the portability of a Java based applications across different platforms. It is essentially a software virtual machine that translates the intermediate Java Byte Code into a particular machine code that can be run in a local machine. However, this portability does come with some performance penalty, as the default parameter values of the JVM do not provide optimised performance for most applications.

My JVM test results in both the Sisters and the Helix have demonstrated the important of the Heap space values. For the output in the Helix, the maximum output using the optimal –Xmx and –Xms values have achieved 20 times as many transactions as compared to using the default JVM heap size values. Other factors such as the parameters that define the JVM garbage collection could also affect the performance of Java applications.

In general, to optimise the application performance, careful analysis of the resource requirements of a particular java application must be made, and optimised JVM parameters provided to match the application requirement.

**JBoss cluster feature and the application deployment**

How an application is deployed in a system can greatly affect the performance of the application. This is especially true when there is an application server cluster in the system. JBoss clusters provide a mechanism to scale an application output by distributing the workload to member machines in the Cluster.

My test results in both the Sisters and the Helix have provided good scalability results by running a JBoss cluster compared with that of only one JBoss machine in the system. By deploying multiple copies of the same EJB to each member machine, increased accessibility is achieved and the performance is greatly increased by sharing work between the JBoss Cluster member machines. I have seen linear scalability results when a well-tuned JBoss Cluster is used in the system.

**JDBC connection and resource pooling**

Different resources pooling such as the database pooling and thread pooling are used to improve the performance of an application.

In database pooling, existing database connections are reused to eliminate the overhead associated with expensive database connection creation and destruction. A better performance can be achieved through this technique. The results show how the database pooling affects output results. When the JDBC connections becomes the bottleneck, increasing the default values of the maximum database pool size will improve the output results

Thread pooling is another example of resource pooling. The JVM supports multiple threads of executions for better performance, however, two many threads running concurrently can lead to resource bottleneck and degrade the system performance. Using threads pooling allows the user to set the maximum numbers of concurrent threads and optimise the performance of the system. Current version of JBoss does not support thread pooling; this seems to be the reason for the unstable performance when there are a large numbers of concurrent threads running in a JBoss machine.

**Other factors**

There are others factors that affect a J2EE application performance but may not directly related to ECperf test results. These include but are not limited to the Java garbage collection, best practices for use of servlets/JSP and session management, application assembly, HTTP server configuration, processor throughput, the application server and servlet queuing and parameter tuning and the database configuration.

## 8.5 Further performance improvement discussion

It should be possible to further improve ECperf performance by introducing new factors that are related to the database and the JBoss application server.

### 8.5.1 Scaling the database

**Clustering of Database: C_JDBC discussion.**

In the J2EE application, these are four tiers: the client, the web server, the application server and the database tiers. Good performance has been achieved for the web server and application server tiers using JBoss and Tomcat cluster. The bottleneck for a large system is the database. Using a cluster of databases, a much better overall output for my ECperf application is expected.

Recently, open source software called C-JDBC has been developed that can be used to build a cluster of databases. C-JDBC is the research result of French National Institute For Research In Computer Science And Control (INRIA) [50]. It is a databases cluster that allows any Java application to transparently access a cluster of databases through JDBC without the need to modify existing client applications, application servers or database server software.

When the database in a J2EE application becomes the bottleneck or single point of failure, C-JDBC can be used to resolve the problem. Adding database nodes and balancing the load among these nodes can achieve increased performance and scalability. Similarly, high availability and transparent fail-over can be achieved.

Figure 8.3 gives the C-JDBC architecture. A generic JDBC driver called C-JDBC driver is provided to the client. This driver forwards the SQL request to the C-JDBC controller that balances them to a cluster of databases. The controller uses Redundant Arrays of Inexpensive Database (RAIDb) to provide a virtual database to the DB client. The underlying databases are distributed and replicated among several nodes.

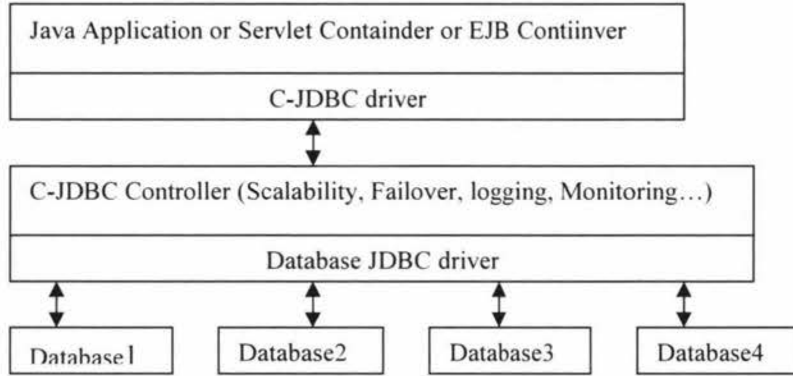


*Figure 8.3: Example C-JDBC architecture*

There are three basic RAIDb architectures in the C-JDBC framework. Figure 8.4 shows an example of a RAIDb-0 configuration. In the example, different tables in the database are distributed to different backend nodes; this can achieve improved scalability but not fault tolerance.

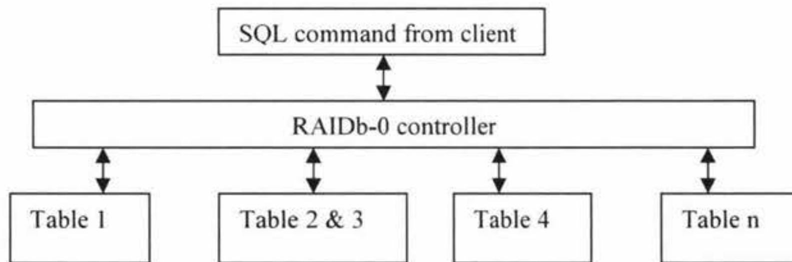


*Figure 8.4: RAIDb-0 example*

RAIDb-1 provides full replication of the database on the backend. This schema provides best fault support, since the whole database is available from any of the backend database nodes, however RAIDb-1 does not provide improved performance since the write to database need to be broadcast to all nodes. Some performance improvement on reads can be achieved.

A nested RAIDb system can provide both improved scalability and fail over support. Figure 8.5 shows a RAIDb-0-1 example. The top level RAIDb-0 can be used to achieve good scalability and fault tolerance is achieved on each of the two partitions using a RAID-1 controller.

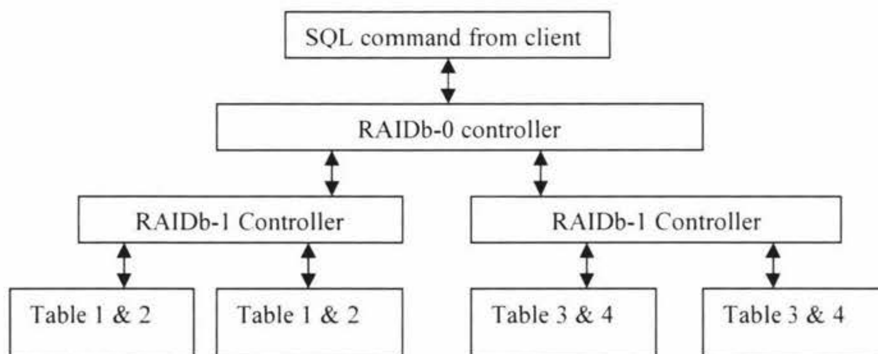


*Figure 8.5 RAIDb-0-1 example*

## 8.5.2 Scaling the JBoss application server

To improve the JBoss application server performance, it is necessary to extend some of the cluster features so as to improve the system performance. Some of the problems I have identified for current JBoss cluster architecture and suggested possible solutions are discussed below.

Recent versions of JBoss include JBoss2.4.4, JBoss3.0.6, JBoss3.2.1 and JBoss3.2.2, and each version is integrated with the Tomcat Servlet server. JBoss2.4.4 does not support the cluster feature; it was used in the very early stage of the study. JBoss3.0.6 is the earliest stable JBoss3.X version, but has problems supporting reliable deployment of J2EE applications in a Cluster environment. JBoss3.2.1 has problems for JBoss Cluster node members to communicate properly, whenever a member is suspended temporally, it cannot effectively rejoin the JBoss Cluster again. This is due to a bug when implementing the JBoss Cluster using JavaGroup [33] as the communications software in JBoss Cluster. JBoss3.2.2RC1 was used in these tests, which works well.

JBoss was implemented using JMX framework, which is very well structured. However, this does not necessary give a performance advantage compared with other application servers. Research has revealed big performance differences for using different application servers. An auction site using a session façade implementation has recorded 267% more peak output when using JOnAS application server compared with using JBoss application server [5].

As JBoss Clustering was only supported in recent months, the overall performance of the JBoss Cluster, how well it support the features such as scalability and fail-over need to be carefully tested.

Some of the problems associated with the JBoss application server, include the load-balance policy, JBoss Partitioning support and the transaction support for EJB in a JBoss Cluster. Some general JBoss features such as thread pooling also need to be implemented for better performance.

**Using better load-balance policy**

Current JBoss cluster servers that work cooperatively in an application suffer from the load-balancing problem.

The test results show that the client workload cannot be balanced well when using the default Round Robin load balance policy. In a high workload, the workload in one or two JBoss could be much higher than other machines, but the Round Robin algorithm will continue to distribute workload to each machine one after another. In the end, the machine with the higher workload will becomes slower and slower, with the job queue becoming longer and longer until this machine does almost nothing.

The First Available algorithm can improve this situation in some way. Because the workload is distributed randomly, for a machine with higher workload, if the new jobs that are dispatched to this machine are less than to other machine, there is a chance to have decreased workload for a machine that is overloaded. However, there is still a problem of unbalanced workload, because random distribution could also make things worse in some cases.

A better performance can be achieved by having a better load balance policy. If the workload distribution can be decided in a dynamic way according to the workload in each server machine, then there is much bigger chance that the load will be better balanced. For example, WebLogic application server has implemented a weight-based algorithm [51], each node in the cluster can have pre-assigned weight of workload. By improving this algorithm to assign weights dynamically to each member machine during the working process based on the current load of the machine, a nearly perfect load balance in the JBoss Cluster should be achieved.

**A better resource pooling such as the thread pooling**

JBoss has the problem of managing large number of concurrent threads because there's no definition of thread management in the JBoss application server. When the number of threads in JBoss is too large, the JBoss server will work much slower than normal, in the mean time it will not refuse any new connection requests, thus leading to a crash of the system. This is a long existing problem for JBoss [5].

Thread pooling is the way to solve this problem. A properly defined thread pooling will allow threads to be allocated and returned to the thread pool algorithm for the purpose of reuse, thus improving the performance of the application server.

In a thread pooling implementation, the maximum and minimum number of threads allowed can be defined, and maximum idle time can be defined to prevent a thread being used too long without being released. A working thread is assigned for each client request, but only for the duration for that particular request. Also threads in the pool can be dynamic changed to meet the client's requirement [52].

A server can define a maximum number of threads that can be allocated to handle client requests. If there are no threads available in the pool and the maximum number of threads has already been created, the request will block until a thread currently in use has been released back into the pool, thus avoiding overloading or crashing the server.

**A better partition strategy**

Currently JBoss clustering does not support sub-partitions. This is a big restriction for applications that need to be deployed in a separate partition for performance purpose [38].

Another problem for the current partition schema is the communication between different partitions, there's no communication between different partitions and thus very difficult to deploy one application to more than one partition.

A well-defined sub-partition schema and communication channels between them can be used to solve the problem. With the flexibility of sub-partitioning and allowing communication between different partitions in an application, the application can be deployed more easily to support higher performance requirements. The partitioning of the JBoss server machine and the EJB should be designed to simplify use but with good support for applications that need flexible deployment.

Note: the experiment in chapter 6 used two sperate partitions rather than two sub-partitions. This was suitable as in this application no communication between portions was required.

## 8.6 Possible use for commercial application

In today's business environment, budgeting is restricted for almost all IT projects, in the mean time; performance expectation is higher based on the customer's requirement. So, a highly desirable system needs to provide high performance with low expense.

Though, J2EE is one of the best architectures to build multi-tiered distributed system that is widely supported by leading industrial companies, a vender specific J2EE platform is usually associated with expensive hardware cost and software licence fees, thus the total ownership cost for a high performance J2EE application is high.

The alternative of good quality open source software with a large group of user and developer support, to the general users, especially users with limited budget are large. JBoss is one of the examples of successful open source software that can effectively decrease the cost of software ownership.

For the hardware, a typical mainframe high performance computer system that is necessary to run a large-scale application is usually expensive. Beowulf clusters have effectively solved this problem by building high performance computer using Commercial off-the-shelf (COTs) components. A Beowulf Cluster built using PCs only spends about 5% to 20% of cost compare with commercial product with similar computing power.

The tests discussed in this thesis run on a very cost effective architecture. The Beowulf Cluster as the hardware, the open source JBoss as the application server and the open source PostgreSQL as the database. Gives one of the lowest cost high performances J2EE application platforms.

The test results have proved that this architecture is very good for implementing enterprise solutions that can provide a system with high availability, fault tolerance, high scalability and performance.

This solution is very practical for businesses with a limited budget but who still want to provide high performance computing solutions. To make the solutions more reliable, a careful analysis of the system requirement is very important. A good design of the application and careful evaluation of the system performance expectation need to be specified.

A Beowulf cluster runs Linux as hardware, the open source Clustered JBoss and Tomcat as the application server and web server, the C-JDBC with PostgreSQL as the database cluster can be used commercially to achieve good performance and scalability. Although, the C-JDBC has not been tested in this study, the distributed experience with 2 databases indicates significant performance improvements can be achieved.

Further more, adding Storage Area Network (SANs) to the database cluster will enable support of better database access. SANs represent an emerging networking technology that connects servers and storage (disks) at gigbaud speeds. Some of the critical functionality such as fail-over, High Availability can be achieved without affecting the client of the system [39].

*Beowulf Cluster + Linux + JBoss-Tomcat Cluster + PostgreSQL Cluster + SANs.*
I believer this is one of the most cost effective architectures for running a commercial high performance application. Combining with the vertical and horizontal scalability concept a very high performance can be achieved with little cost.

Figure 8.6 shows a example suggested hardware architecture with JBoss cluster and PostgreSQL cluster to provide better scalability.

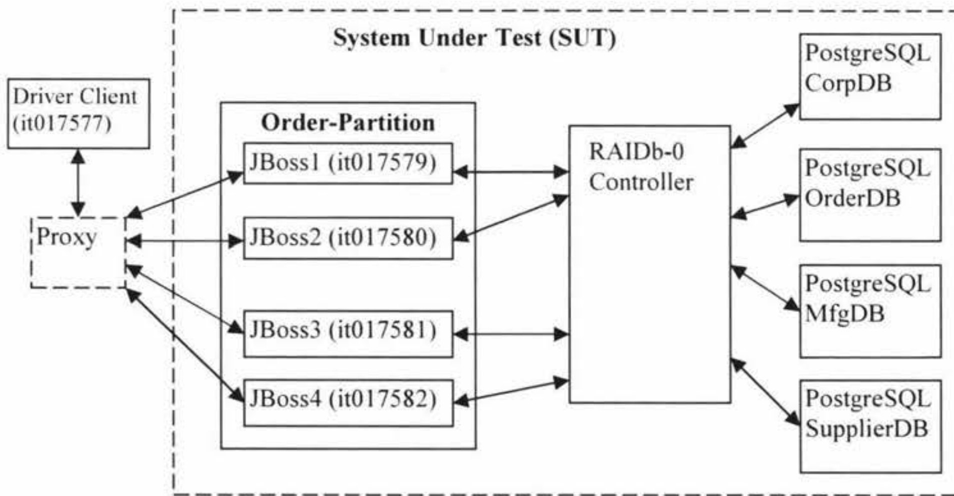


*Figure 8.6: Architecture with JBoss and PostgreSQL cluster*

## 8.7 Summary

By comparing the test results in Sisters with Helix, which are run using the same testing architecture, good vertical scalability has been demonstrated. The test results in both Sisters and Helix have also shown good horizontal scalability when increasing the number of JBoss in the JBoss Cluster.

The possible bottlenecks in the test system have been identified and also solutions to them proposed. The bottlenecks could be any resources in the test system, such as the hardware (RAM, CPU speed and Hard Drive write speed), Software (JVM heap size, application server connection pooling and database connections), network, operating system and application configuration. By carefully identifying the bottleneck, a solution that can further improve the performance of the whole system under test can be found.

I discussed some of the most important factors that might have a large affect on J2EE application and discussed how to achieve good performance output based on factors such as application architecture design, implementation method, deployment, system hardware, software and networks.

Techniques that could be implemented to further improve the J2EE application performance have been discussed including the C-JDBC architecture that can be used for building a scalable and fault tolerance database. Some of the problems that exist in the current JBoss application server have been identified and possible solutions to these problems given. Finally an improved architecture with support of JBoss Cluster and database cluster has been given.

# Chapter 9: Conclusion

## 9.1 Introduction

Scalability tests of application server have been presented for a small and a large system. These results have been discussed in chapter 8 and future: improvements proposed.

This chapter provides a final conclusion to the thesis. I will identify what has been achieved by my work and contribution of my research. Finally further work building on this study is proposed.

## 9.2 Conclusion

### 9.2.1 Contributions

This research study has shown that good scalability result can be obtained for a clustered application server using JBoss. Vertical scalability is achieved by using hardware resources such as RAM and CPU in the application server machines. A more general scalable result has been achieved by increasing the application server numbers in the cluster.

The Beowulf cluster and open source application provide a very cost effective hardware and software architecture to support high performance J2EE applications. By using a Beowulf Cluster, I have high performance hardware that can be used easily to support scalable applications. I have a JBoss and Tomcat integrated server that can be used to support an application server Cluster. A JBoss cluster provides cluster support to all the application server components, such as the cluster of Stateless session bean, Stateful session bean, and Message Driven bean. Tomcat provides a cluster of Servlet sessions. When the J2EE server component is replicated in the application server cluster, a scalable performance can be achieved for the application running on the JBoss server.

Open source software is a suitable choice for running a commercial J2EE application, because it can be used free of charge for developing and deploying applications. Combined with cheaper hardware such as a Beowulf cluster running on Linux operating system, the potential total cost of ownership of an application could be much lower than that of using commercial J2EE software running on hardware that is specifically designed for the application server and database.

The advantages of using open source in the research study are clear. The free software that can be used without restriction, a big open source user and developer group that can be used to communicate on line are a major advantage for a research project. An especially important point of open source is the free access to the source code. Modifying and even adding source code for a particular functionality are easy to do since the source code is available; this is almost impossible when using proprietary software.

### 9.2.2 Conclusion

The J2EE application server is critical for enterprise level applications. By running the ECperf application, the industrial standard J2EE application server test program using an application server cluster, several different performance results have been obtained. The test results have shown good scalability of transaction output for the test system.

Vertical scalability is achieved by only increasing the hardware capacity such as the CPU power or RAM space. By comparing the Sisters and Helix result with the same architecture, I have seen good vertical scalability.

Horizontal scalability is achieved by increasing the number of application servers in the JBoss cluster, from the test results in both Sisters and Helix; I have seen good horizontal scalability. When the application server number in the cluster increases, the transaction output is also increases.

There are always bottlenecks in the system that prevents the system achieving a better transaction output. Hardware can easily become the bottleneck, such as the CPU and RAM. When one of them reaches 100% usage, the system is unlikely to get better output.

Software is also important for the system, all software in the system, the Linux operating system, the Java Virtual Machine, the application server and the database are all likely to be the bottleneck.

Other factors also important for the application performance, the application itself and the deployment are likely to contribute greatly to the application output. From the test results, significant differences can be seen due to the different deployment properties of the same application.

I have seen good horizontal scalability and vertical scalability for the ECperf application. With different architectures in either the application server or database, big differences for the transaction output are obtained. By careful analysis of the system bottlenecks and using a flexible cluster strategy for both the application server and the database, a much better output can be achieved. Some ideas for further performance improvement are presented in the following section.

## 9.3 Future Work

Based on the analysis from this thesis, there are three major elements that can be done in the future: the database cluster implementation, the JBoss cluster feature improvement and performance monitoring tools.

Since the database was one of the bottlenecks, a database cluster plus the Storage Area Network could further improve application performance. Any application that requires a high volume of database access needs a database system that provides high scalability, availability and performance. As I have discussed the C-JDBC can build scalable databases using PostgreSQL cluster to increase database capability.

JBoss clustering does support high availability and scalability. High availability can be achieved by fail over to another JBoss node when one of them fails. Scalability can be achieved by distributing workload to each node in the JBoss cluster. However, there are some problems associated with the JBoss cluster implementation.

Load balancing is a big problem in high workload. JBoss currently support basically two types of load balance policy: the default Round Robin and the First Available load balance policy. Both of them work fine with lower workload but fail to achieve good balanced load in high workload. A better strategy for load balancing such as using a dynamic load balance that distributes load based on current workload in each machine should be tested. A well-balanced workload in the JBoss cluster can effectively improve the performance of the whole J2EE application.

Resource pooling is good way to improve performance by reusing the existing resources. JBoss connection pooling is a good example of reusing the database connections in the connection pool without the need to create and close database connections for each database call. Thread pooling, which has not been supported by current JBoss, is a problem for JBoss. Without properly defined thread pooling, such as the maximum allowed thread for a JBoss, the JBoss can keep on receiving client requests. If the client numbers are too big, the JBoss will be overloaded reaching a state that works significantly slower than normal and finally crash. Adding thread pooling can make JBoss work faster by reusing existing threads. Also, extra clients can be put in the queue so that JBoss can work in an efficient way without crashing.

To improve J2EE application performance requires identifying the bottlenecks in the system, a good program that can give detailed statistics of the system resources usage will help significantly. Monitoring that performance will enable a more accurate report of the system resource usage and help to accurately identify the bottlenecks in the system under test. By further analysing bottlenecks in the system, the application can be tuned for a better performance.

I expect to achieve a high performance using Beowulf Cluster that is comparable with that using a mainframe supercomputer. Commercial high performance computers that

are designed for J2EE application usually have better hardware such as large RAM and Cache that can achieve high level of vertical scalability, but a Beowulf Cluster is usually been built for general purpose computing and with fixed hardware capacity in each nodes. A careful study needs to be done to identify how these hardware differences affect the performance output of a test program.

# Reference:


1. Roman, E. (2001). Mastering Enterprise JavaBeans (Second Edition). *John Wiley & Sun Inc.*
2. Tanenbaum, A., Van Steen, M. (2002). Distributed Systems: Principles and Paradigms. *Prentice Hall.*
3. The Object Management Group (OMG), "Common Object Request Broker Architecture: Core Specification", OMG Document: formal/2004-03-12
4. Commonwealth Scientific & Industrial Research Organization (CSIRO). Evaluating J2EE Application Server:Version 2.1. http://www.cmis.csiro.au
5. Cecchet, E., Marguerite, J., Zwaenepoel, W. (2002). Performance and Scalability of EJB Applications. *Rice University.*
6. Shirazi, J. (2000). Java Performance Tuning (2nd Edition)
7. Top 500 computers. //Top 500 web site
8. Yoo, H., Ko K. (2000). Operating System Performance and Large Servers. *Sun Microsystems, Inc.*
9. Singhal, S., Nguyen, B., Redpath R. (2002). Building High-Performance Applications and Services in Java: An Experiential Study. *IBM T. J. Watson Research Center*
10. Mark Baker (ed.) Cluster Computer White Paper, December 2000, to be downloaded from: http://www.clustercomputing.org/cluster_white_paper.pdf
11. Barczak, A., Messom, C., Johnson, M. (2003). Performance characteristics of a Cost-Effective Medium-Sized Beowulf Cluster Supercomputer. *Research letters in the Information and Mathematical Sciences in Massey University.* Volume 5, June 2003.
12. Java Management Extensions Instrumentation and Agent Specification, v1.2. *Sun Microsystems.*
13. Lindfors, J., Fleury, M. (2002). JMX: Managing J2EE with Java Management Extensions. *SAM Publishing.*
14. Wildenius, M., Axmark, D. (2002). MySQL Reference Manual. *O'Reilly Community Press.*
15. Worsley, J., Drake, J. (2002). Practical PostgreSQL (O'Reilly UNIX). *Command Prompt Inc.*
16. MySQL AB. MySQL. http://www.mysql.com

17. PostgreSQL. PostgreSQL documentation. http://www.postgresql.org
18. Sun Microsystems, Inc. ECperf Specification version 1.1. http://java.sun.com/j2ee/ecperf/
19. Rosanna Lee, Scott Seligman. (2000). JNDI API Tutorial and Reference: Builing Directory-Enabled Java ™ Applications. *Sun Microsystems.*
20. Smallen, S.; Crine, W.; Frey, J.; Berman, F. Combining workstations and supercomputers to support grid applications: the parallel tomography experience.*Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th*. 1 May 2000. Pages: 241-252
21. Guest, M.F., Sherwood, P. (2002). Computational chemistry applications: performance on high-end and commodity-class computers. In *High Performance Computing Systems and Applications, 2002. Proceedings. 16th Annual International Symposium on,* 16-19 June 2002. Pages 290-301.
22. Froidevaux, W.; Murer, S.; Prater, M. The mainframe as a high-available, highly scalable CORBA platform. *Reliable Distributed Systems, 1999. Proceedings of the 18th IEEE Symposium on* , 19-22 Oct. 1999, Pages 310 – 315
23. Benda, M. Middleware: any client, any server. *Internet Computing,* IEEE , Volume: 1 Issue: 4 , July-Aug. 1997 Pages 94 -96
24. G. Chen, M. Kandemir, N. Vijaykrishnan, M.J.Irwin. PennBench: A benchmark suite for embedded Java. Penn State University. http://www.cse.psu.edu/~mdl
25. Tani, K.; Aoki, T.; Matsuoka, S.; Ohkura, S.; Uehara, H.; Aoyagi, T. First light of the Earth Simulator and its PC cluster applications. *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on* , 23-26 Sept. 2002. Pages: 175.
26. Rajan, A.S. Shiwen Hu. Rubio, J. Cache performance in Java virtual machines: a study of constituent phases. In *Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on* 25 Nov. 2002. Pages 81 –90.
27. Aridor, Y.; Factor, M.; Teperman, A. cJVM: a single system image of a JVM on a cluster. *Parallel Processing, 1999. Proceedings. 1999 International Conference on* , 21-24 Sept. 1999. Pages 4-11.
28. William Crawford, Jonathan Kaplan. J2EE Design Pattern. *O'REILLY.*

29. Sterling, T.; Becker, D.; Warren, M.; Cwik, T.; Salmon, J.; Nitzberg, B. An assessment of Beowulf-class computing for NASA requirements: initial findings from the first NASA workshop on Beowulf-class clustered computing *Aerospace Conference, 1998. Proceedings., IEEE , Volume: 4* , 21-28 March 1998. Pages 367 –381.
30. Sterling, T. Launching into the future of commodity cluster computing. *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on,* 23-26 Sept. 2002. Pages 345.
31. The centra for advanced computing research (CACR). http://www.cacr.caltech.edu/
32. Open Source Initiative (OSI). http://www.opensource.org
33. JBoss group. JBoss. http://www.jboss.org
34. ObjectWeb. JOnAS. http://www.objectweb.org/
35. Apache Software Foundation. Tomcat. http://www.apache.org/
36. Microsoft .NET home. http://www.microsoft.com/net/
37. Scott Stark and the JBoss Group. JBoss Administration and Development. *JBoss Group, LLC.*
38. Sacha Labourey, Bill Burke. JBoss Clustering. *JBoss Group, LLC.*
39. Milanovic, S.; Petrovic, Z. Building the enterprise-wide storage area network. *EUROCON'2001, Trends in Communications, International Conference on.* , Volume 1, 4-7 July 2001. Pages 36 –139.
40. Standard Performance Evaluation Corporation (SPEC). SPECjAppServer. http://www.specbench.org
41. Apache Software Foundation. Apache Ant. http://www.apache.org/
42. Hewlett-Packard Company. Netperf. http://netperf.org
43. Adabala, S.; Kapadia, N.H. Interfacing wide-area network computing and cluster management software: Condor, DQS and PBS via PUNCH. *High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on,* 1-4 Aug. 2000. Pages 306-307.
44. Radhakrishnan, R.; Vijaykrishnan, N.; John, L.K. Java runtime systems: characterization and architectural implications. *Computers, IEEE Transactions on,* Volume: 50 Issue: 2 , Feb. 2001. Pages 131-146.
45. Sun Microsystems Inc. Java Virtual Machine (JVM). http://java.sun.com.
46. Red Hat. Red hat Linux Manuals. http://www.redhat.com

47. Intel Corporation. Intel CPU. http://www.intel.com/

48. Luiz A. Borroso, KOUROSH Gharachorloo, Edouard Bugnion. Memory System Characterization of Commercial Workloads. In *Proceeding of the 25$^{th}$ annual international symposium on computer architecture,* pages 3-14, June 1998.

49. Karlsson, M.; Moore, K.E.; Hagersten, E.; Wood, D.A.; Memory system behavior of Java-based middleware. *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.* , 8-12 Feb. 2003. Pages 217 –228

50. Emmaunel Cecchet, Julie Marguerite, Mathieu Peltier. C-JDBC User's Guide. *French National Institute For Research In Computer Science and Control (INRIA)*

51. BEA Systems, Inc. Weblogic application server. http://www.bea.com

52. Borland Software Corporation. Borland Enterprise Server, AppServer Edition. http://www.borland.com