A Dynamic Networked Browser Environment for Distributed Computing

by

Luke Joseph Fletcher, BSc

A dissertation submitted to the

School of Computing

In partial fulfilment of the requirements for the degree of

Bachelor of Computing with Honours

UNIVERSITY OF TASMANIA

November 2002

DECLARATION

I, Luke Joseph Fletcher, certify that this thesis contains no material which has been accepted for the award of any other degree or diploma in any tertiary institution, and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Signed: ---------- Date: ----------

ABSTRACT

Many organisations have a large number of computers with varying usage patterns. Some of these machines at different locations are often free from time to time leaving them to do very little useful computation or none at all. It is at these times that this dynamically changing environment of machines can be used for a more useful task.

This project reports the development and feasibility testing of a dynamic distributed computing environment. This is achieved by making use of ubiquitous web browsers to harness these underutilised computers. Therefore taking the idea of distributed computing away from the traditional paradigm of fixed hosts to which it is often associated.

ACKNOWLEDGEMENTS

Firstly, I wish to thank my supervisor Dr. Vishv Malhotra for his support and invaluable assistance throughout the year with this thesis.

To all my workmates at the School of Computing, a very special thanks go out to you for your continued support, advice, and often welcome distractions. To Julian, Andrea, Ian, Mark, and Nicole thank you for your interest and help with everything. In particular I would like to thank my fellow tech support comrades Dave and Terry for your help, suggestions, and proofreading of various bits of this thesis as well as your excellent thesaurus skills! Also to Young, for the opportunity to both study and work with such a great group of people like everyone at the School.

To all my fellow honours students from the last two years, thanks for your help, fun and often bizarre times. From the LAN parties, BBQ's to Disco Bowling it was a great time! In particular I would like to thank Edward, Pauline, Ben, Richard, Chris, Adam, and Chloë for your help and just being there to chat whenever.

Finally, a huge thank you to all of my family and friends. Thank you to my parents, Jim & Judy, and my brothers, Stuart & Chris for your continued love and support and especially for your help and assistance in the last year while I have been working on this thesis.

Luke Fletcher
November 2002

TABLE OF CONTENTS

Chapter 1 INTRODUCTION AND OVERVIEW

1.1 Motivation

Organisations such as the University of Tasmania have a large number of computers with varying usage patterns. Some of these machines at different locations are often free from time to time leaving them to do very little useful computation or none at all. The majority of these computers are networked together or connected to a medium such as the Internet, and therefore it is at these times that this dynamically changing environment of machines can be harnessed for a more useful task.

This thesis is an exploration of this concept, in which we will try to harness the free machines through the use of distributed computing and web browsers, in an attempt to develop a feasible dynamic distributed computing environment.

1.2 Distributed Computing

Farley (1998) suggests that distributed computing can be considered in terms of "*breaking down an application into individual computing agents that can be distributed on a network of computers, yet still work together to do cooperative tasks*".

The concept of "distributed computing" has gained popularity within the computing industry in the last decade. This is primarily due to the fact that it is now considered a viable alternative for processing large datasets or applications, compared to using traditional and expensive supercomputers.

1.3 Thesis Aims

This thesis aims to show that a NoW (Network of Workstations) where each computer is running a web browser can provide a flexible platform for distributed processing. Therefore aiming to prove that distributed processing is possible using Java within a dynamic and constantly changing environment, rather than the traditional paradigm of fixed hosts.

To prove this concept of a dynamic distributed computing environment this thesis will present the development of a distributed computing system which includes both

server and a dynamic client base. Therefore endeavouring to show that this solution is a feasible alternative to the more traditional concept of distributed computing.

1.4 Thesis Structure

This thesis document consists of five chapters and three appendices.

Chapter Two will present a review of the relevant literature, in particular looking at the popular approaches to distributed computing, both past and present, as well as the issues and current trends.

Chapter Three presents the development and the steps used in developing a dynamic distributed computing environment written in the Java programming language.

Chapter Four will present and discuss the results obtained based on the performance of the dynamic distributed computing environment developed in Chapter Three.

Finally, Chapter Five will conclude this thesis with a discussion as to whether the proposed dynamic distributed environment is a feasible alterative to fixed host distributed computing.

The appendices contain a full source code listing of both the server and client developed for the dynamic distributed computing environment. They also contain a listing of the task-set data used in testing the distributed system and acquiring the results presented in Chapter Four.

Chapter 2 LITERATURE REVIEW

Individuals, business and organisations throughout the world are turning to the concept of distributed computing as a viable alternative to the traditional and expensive supercomputers used to perform both simple and complex computational tasks. This review will look at the popular approaches to distributed computing, both past and present, and the mechanisms that can be used to achieve a distributed environment. Also investigated are the issues and current trends of distributed computing.

2.1 Distributed Computing

Distributed Computing is a complex process, to which there a many ways to achieve the results required for a distributed system or application. This section will investigate more closely the popular methods used in distributed computing, starting with the original Remote Procedure Call (Tanenbaum & Steen, 2002) and then moving through to technologies such as CORBA (OMG, 2002) and Java RMI (Waldo, 1998) & (Ahuja & Quintao, 2000).

Overall distributed computing has closely followed the traditional approach of the procedural paradigm that was first introduced with the Remote Procedure Call (RPC). However in recent years there has been a strong shift towards distributed object based systems. CORBA and Java RMI are examples of recent distributed systems that have adopted this model. Objects provide a natural means of transmission of data in object orientated languages such as C++ and Java, this therefore provides developers of distributed systems with an ideal foundation to build their distributed environments on.

2.1.1 Remote Procedure Call (RPC)

Tanenbaum & Steen (2002, p. 68) credit Andrew Birrell and Greg Nelson in 1984 for first proposing the concept of Remote Procedure Calls (RPC). At the time it was a new and innovative way of approaching the concept of distributed computing. Most earlier distributed systems where based on explicit message transmission between processes, however Birrell and Nelson proposed a way in which distributed applications could be designed to call procedures located on other machines running

another distributed application. This procedure, the callee, would then run while the caller waited, and upon completion the callee would then return its results back to the caller application.

The main focus of RPC was to make the process of distributed computing transparent to the actual applications, meaning that an application would not know the difference between a local or distributed procedure call. This transparency is achieved through the use of client and server stubs, which are procedures designed specifically for making requests to, and receiving responses from, procedures in a distributed application. This concept is summarised by Tanenbaum & Steen (2002, p. 72) as:

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's OS sends the message to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls the local OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

The basic concepts of a remote procedure call are simplistic and this contributes to their continued success. This concept has since been expanded upon and now forms the basis of many of the current models of distributed computing in modern programming languages.

2.1.2 Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture, or as it is more commonly known CORBA, has provided the computing industry with what Orfali & Harkey (1998, p. 3) describe as "the next generation of middleware" for distributed systems. It is an Object based system, however it is not a distributed system in itself, but forms the specification of one. These specifications where drawn up by a non-profit organisation called the Object Management Group (OMG), which includes more then 800 members, many of them the most successful players in the computer industry.

Distributed systems are often criticised because of interoperability problems when integrating distributed applications together, especially those which are written in

different languages. Schaaf & Maurer (2001, p. 73) suggest that the primary aim of CORBA is to overcome these problems by providing:

- Access to services
- Discovery of resources and object names
- Error handling, security policies, and
- Language and platform neutrality

Ideally CORBA has the potential to provide distributed applications with a *Homogeneous* environment; this is where both the client and server's environment is compatible with each other. Heterogeneity issues are therefore hidden, for example different programming languages and differences in the physical computers or architectures. This homogeneous environment is provided by what is known as the Object Request Broker (ORB), which forms the core of a CORBA distributed system by providing a middleware to allow communication between objects and their clients.

The ORB provides a "programming library suitable for the programming language used in a given software development project" (Schaaf & Maurer 2001, p. 73), therefore enabling CORBA to remain language neutral. This is in part due to the *Interface Definition Language* (IDL) which specifies all objects and services in CORBA. IDL provides "a precise syntax for expressing methods and their parameters" (Tanenbaum & Steen 2002, p. 496), these methods can then be invoked from a programming language which provides CORBA bindings, such as for example C, C++, and Java.

Another of CORBA's strongest points is the fact that it is an open standard. It is a specification that has been developed with the input from a large proportion of the computing industry and therefore the base services provided by the original CORBA core have been expanded upon and improved over time. Each organisation that develops an implementation of CORBA for their distributed system can expand upon the features and specifications that they specifically want to target, although this does have the potential to introduce compatibility issues.

2.1.3 Distributed Component Object Model (DCOM)

Distributed Component Object Model (DCOM) is an object-based distributed system that in many regards is very similar to CORBA. However CORBA was developed as an open specification and organisations that where involved in this process included many of the largest within the industry, for example Sun Microsystems. However one organisation that was missing from this committee is Microsoft. This is because Microsoft developed their own standard, DCOM, which originated from COM, the underling technology developed by Microsoft for their operating systems from Windows 95 onwards.

COM forms the basis for many of the operations carried out in the Microsoft operating systems, and this is achieved by providing a mechanism to "support the development of components that can be dynamically activated and that can interact with each other" (Tanenbaum & Steen 2002, p. 527). DCOM is basically an extension of this and adds the ability for a process to communicate with components that exist on another physical machine, while also adding a layer of transparency to the distributed system.

Both CORBA and DCOM make use of an IDL, however the DCOM implementation is a more specific IDL called Microsoft IDL (MIDL) which generates standard layout binary interfaces which are specific to the implementation and methods of DCOM, thus it is still very much a closed distributed system. It has been specifically designed for use with Microsoft operating systems and applications running under these operating systems, therefore it is not portable to other platforms. However it is by far one of the most widely used systems and this is primarily due to the success and widespread use of Microsoft operating systems in networked environments today.

2.1.4 Distributed Computing Environment (DCE)

The Distributed Computing Environment (DCE) is a specific RPC based system developed by the Open Software Foundation (now known as The Open Group). Initially DCE was designed for use with UNIX systems only, but it has since been ported to a number of popular server and desktop operating systems such as Windows NT. Similar to CORBA in certain regards, Tanenbaum & Steen (2002) suggest that DCE acts as a middleware system between existing networked operating

systems and a distributed system. The basic idea is that existing machines can have the DCE software installed and then be able to run distributed applications, without affecting other areas on those systems. The Open Group now sells the source for DCE to organisations so they can develop their own implantations and adapt the system to their specific purposes.

DCE, like other distributed systems still relies heavily on the Interface Definition Language (IDL). Among others, the IDL through the use of client and server stubs provides the bridge between the application and the distributed system.

Because DCE is acting as middleware and is built upon the RPC system where remote services can be accessed by calling a local procedure, existing code can be modified to run in a distributed system using DCE with very little or in some cases no changes. This also has the added advantage that because the clients and servers act independently of one another they can be written in two different programming languages and still be able to act in a distributed environment.

2.1.5 Java Distributed Computing

Java was designed with the potential of being an Internet programming language, and as such it provides a powerful foundation for the development of distributed systems and applications. In particular the core reliability, simplicity and architecture independence Java provide along with built in networking, multithreading, security and support for Java applets make it an ideal choice for distributed programming.

Another attribute of Java which is crucial to its success in distributed computing programming environments is that it is a pure object-orientated programming language. As already mentioned objects provide a natural means of transmission for data within Java and therefore Farley (1998) suggests that the development of a distributed system in Java can simply be thought of as "distributing its objects in a reasonable way, and establishing networked communication links between them using Java's built-in networked support".

This section will discuss three of the features in Java which can be used to develop distributed systems, these being Java RMI, Java Sockets & URLs, and finally Java Servlets.

2.1.5.1 Java RMI

Java Remote Method Invocation (RMI) is an object based distributed system which was first introduced in 1997 by Sun Microsystems in version 1.1 of the Java Development Kit (JDK). At the time Java was still only in its infancy and had no real support built-in for distributed computing, except for the basic services provided by Java Sockets. Thus, the main aim of RMI was to allow for the development of distributed applications from within Java without the use of third-party software sitting on either the client or server side of the system, for example CORBA.

At the basic level, Java RMI like many other distributed systems has originated from and is similar to other RPC based systems. This is because it allows for an object to invoke method calls on another object that may be residing on a different machine, however it also allows a distributed application to transport whole objects across a network, something that is not possible in traditional RPC based systems. However Waldo (1998, p. 5) suggests that beneath the surface it has a number of "differences in the programming model, capabilities, and interactions".

Unlike other distributed systems, RMI makes an important assumption that moves away from the common idea, although there are exceptions such as CORBA, of distributed systems working in an environment of heterogeneity. This assumption is that RMI operates in an environment where both the client and the server are running on a Java virtual machine, and that the objects within the distributed system are written in the Java programming language, therefore creating a homogeneous environment. This removes issues normally associated with heterogenic environments, while also allowing distributed applications to take advantage of the "dynamic nature" of Java by automatically creating (or activating) remote objects on demand. It is, however, possible to add a layer of heterogeneity to RMI with the use of CORBA, as RMI provides full compatibility with CORBA distributed systems through the use of RMI-over-ORB (Internet inter-ORB protocol).

The homogeneous environment in which RMI operates also means that it does not require the services of the machine-neutral IDL that many distributed systems use. Instead it uses a Java interface to declare the remotely accessible interfaces. Developers in heterogenetic environments can however use the Java-to-IDL mapping

which allows for the development of CORBA-based Java applications without prior knowledge of IDL.

The main driving force behind RMI is the client interfaces (stubs) to remote objects within the client and server. These exist in the stub/skeleton layer and provide similar services to stubs in RPC based systems. The RMI system also maintains a remote object registry that allows objects to be registered on the network and a client can then easily retrieve a remote objects stub reference by using its reference name. This process is supported by a backbone which is the JRMP (Java Remote Method Protocol) that allows for network communication between the remote objects and a client.

Jim Waldo (1998, p. 7), a senior engineer with Sun Microsystems and project manager for the distributed programming infrastructure for Java, best describes RMI in comparison to other distributed systems by saying:

> *"In most existing systems, the result of writing an IDL interface is a static wire protocol, which defines the way the stub of one member of the distributed computation will interact with the skeleton that belongs to another part of the distributed computation. In the RMI system, the interaction point has moved into the address space of the client of a remote object and is defined in terms of a Java interface. That interface's implementation comes from the remote object itself, is dynamically loaded when needed, and can vary in remote objects that appear, from the client's point of view, to be of the same type (because the client only knows that remote objects are of at least some type)."*

2.1.5.2 Java Sockets & URLs

Java is a programming language that was designed to take advantage of the latest networking capabilities, and as a result it is a platform independent language that has three main areas of networking support, these being applets, RMI, and sockets. While not normally classed as a distributed system by itself, sockets are almost always used in conjunction with Java RMI for example, to provide the means of communication for a distributed system. Java Sockets follow similar models to those sockets (Sun Microsystems, 2002) found in other programming languages and therefore are for most part the same across all languages with networking capabilities.

A socket is "one end-point of a two way communication link between programs running on the network" (Sun Microsystems, 2002). At the basic level sockets use TCP which provides a point-to-point communication channel for client/server applications that then can be used read and write across the socket that has been bound to the connection.

The procedure for how a socket connection is made is that a client will attempt to connect to a server on a particular port; if this is successful the server will accept the connection and will create a new socket bound to a different port. This allows the connection to be made and reading and writing to occur over the socket while the server can continue to listen for incoming requests on the original port. This process is illustrated in Figure 2.1.

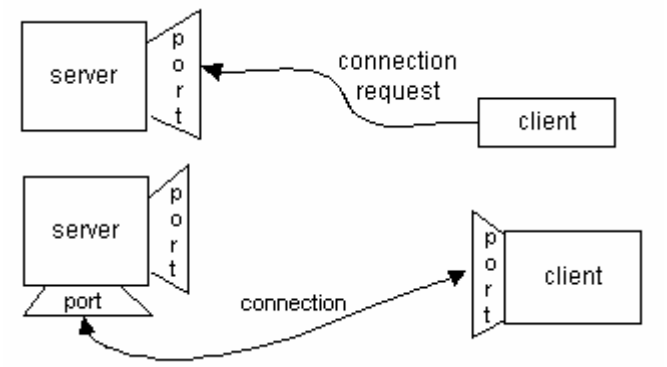

Figure 2.1 – The process of establishing a connection between two hosts using sockets (Sun Microsystems, 2002)

2.1.5.3 Java Servlets

Hunter & Crawford (1998) suggest that Java Servlets can in many ways be thought of as Java's answer to Common Gateway Interface (CGI) scripts commonly found on many web servers in use throughout the Internet today. They provide the facility for dynamic content on web pages and are commonly used for creating web based applications. An example of a common Java servlet based web application could be a simple web page counter that keeps track of the number of visitors to a web page, or another could be a more complex online shopping system which allows visitors to select items for purchase and place them in an online "shopping cart".

The core Java API does not support servlets by default; rather servlets are supported by a specific add-on Servlet API made available for use with Java by *Sun Microsystems* or as part of the *Java2 Enterprise Edition*. However all features of the

core Java API are supported by servlets, therefore anything that can be developed using a traditional Java application can be developed for use with or as a servlet.

Servlets are server extensions that sit on top of many common web servers such as Apache and provide a "Java class that can be loaded dynamically to expand the functionality of a server" (Hunter & Crawford, 1998, p. 6). Servlets run inside the Java Virtual Machine (JVM) of the server, with all computation occurring on the server side of the system, for example in the case of a counter the servlet calculates the visitor information and the result is displayed on the client web browser. Therefore support for Java is not required on the client side web browser for traditional servlet based applications. Figure 2.2 provides an overview of the typical servlet life cycle and shows how servlets are multithreaded and able to handle multiple requests from clients at the same time, i.e. concurrent sessions.

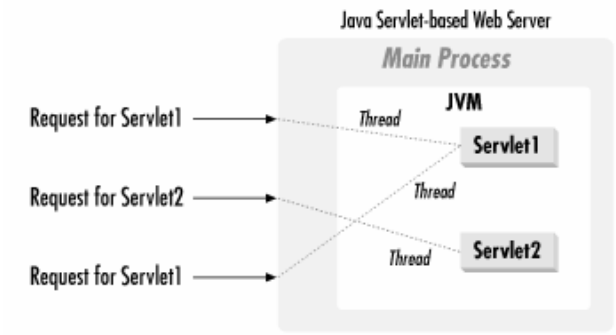

Figure 2.2 - The servlet life cycle (Hunter & Crawford, 1998, p. 6)

Servlets are not commonly associated with distributed applications or systems and therefore this is an area that has had very little research conducted on it in the past. This thesis will focus on this area, and section 3.2 will discuss the concept of servlets and their use in distributed computing.

2.2 Web Browser Architecture

A web browser is the most popular method in use today to view information from the Internet. It acts as the "main client software residing at the user end, which interacts with a web server to retrieve documents for display on the clients screen" (Web Browser Architecture, pp. 1-2). The process of retrieving documents from the server also requires the browser to interpret what it has downloaded and then display the expected result. The common way this is achieved is through the use of the mark-up language known as HTML, which pages designed for the Internet are usually coded in.

However this only provides basic web page viewing capabilities. The baseline architecture of a web browser is constantly being expanded to provide new services. The result of this means that the architecture can be broken up into three primary components, the controller which provides interactions between the client and the user, the service modules which provides drivers for connecting to particular protocols such as HTTP (Hypertext Transfer Protocol) and FTP (File Transfer Protocol), and finally the interpreter module which provides the drivers for the browser to interpret the incoming document. These three levels of architecture are shown in Figure 2.3.

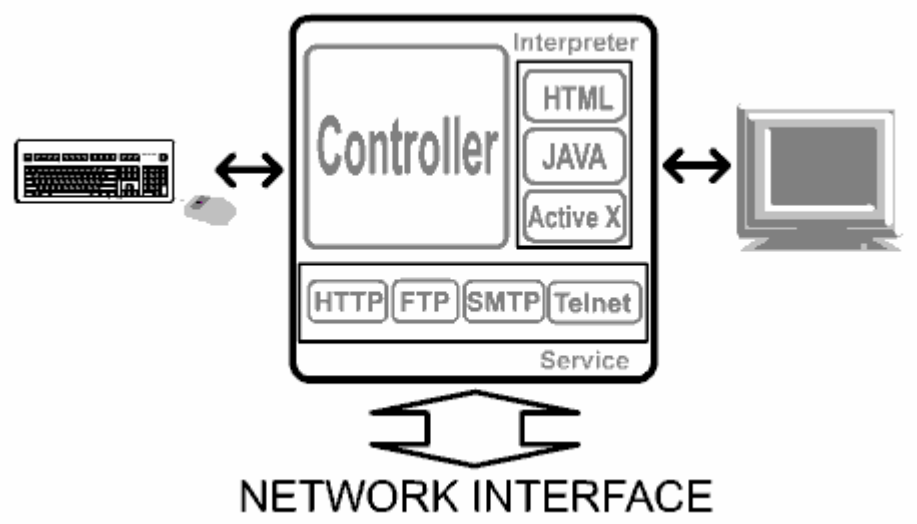

Figure 2.3 – The architecture of a Web Browser (Web Browser Architecture, pp. 1-2)

The interpreter module provides the bulk of services to a web browser, at the basic level it receives a document from a web server and as the name suggests interprets it using the required driver and then displays it on the users screen, most commonly

this would occur with HTML documents. However there is the potential for other documents such as Microsoft's Active X and in particular Java Applets

2.2.1 Java Applets

The support for Java Applets within the web browser architecture is an area of particular interest in terms of distributed computing. Java applets are a Java programs that rather then running as a traditional application, runs from within the web browser environment. The Applet runs in the web browser through the Java runtime environment which is installed as a plug-in, this plug-in follows the standard Java runtime environment produced by *Sun Microsystems*.

A Java applet is very similar to a Java application in the features that it provides, including Java RMI and Sockets. Many of the distributed environments mentioned earlier require that a client be written using the features of a particular distributed language. This client then needs to be installed on a physical machine which then allows it to connect to a distributed application server and take part in that application. However Java Applets have an advantage in this area as a distributed client can be written as a Java applet, and then downloaded to a client machine on demand and run from within the web browser environment. This eliminates the need for any client software to be installed while still providing all the features of a distributed system. This concept will be discussed in further detail in Chapter 3.

Security of course is a major concern with any distributed system, and it is an area that needs to be addressed correctly when using applets in this way. This area will be discussed more thoroughly in section 2.3.1.

2.3 Issues in Distributed Computing

While the idea of a distributed application or system running in a distributed environment may seem like a viable alternative to the more traditional approaches, it also introduces new issues that need to be addressed when implementing a system. Firstly the issue of security is one of the most difficult issues involved, as for example a distributed application has the potential to be exposed to outside parties compared to an application that was just run on a stand alone system.

Also scalability (ensuring a distributed system has the resources to service the demand), the robustness and reliability, efficiency, usability, and transparency from clients are all important issues that need to be addressed in a distributed environment. This section will discuss these issues in further detail.

2.3.1 Security

Security is arguably the biggest issue when developing a distributed system. This is because by connecting a client to a distributed system not only exposes it to the risks of being connected to a medium such as the Internet but it also opens it to risks from within the distributed system. The main areas at risk are common to all areas of computer security, these being confidentiality, integrity, and accessibility. Therefore, there are primarily two important issues when looking at security from these regards, firstly there is the security of the communication between the users or processes, and secondly the concept of access control or authorisation to a distributed system.

Authorisation is a crucial issue when looking at security and it is often approached by using secure channels and encryption between the users or processes. This will provide a secure channel that allows the system to "mutually authenticate the communicating parties, and protect messages against tampering during their transmission" (Tanenbaum & Steen 2002, p. 489). Access control is the other important security issue; for example authorising a particular user or process to take part in a distributed system. This can usually be achieved with the use of access control lists.

Of course some systems are more at risk then others, simple distributed applications may not require such means of security while others, which may be involved in, for

example banking information, may need high levels of security to ensure data is not compromised. The level of security often relies on the distributed system in use. Consider Java applets in a distributed system for example, special considerations need to be taken into account to ensure that firstly it is secure and secondly that it does not allow the applet to utilise resources outside of the sandbox (a secure area that allows an applet to run while not normally having access to local machine resources).

Security in Java and applets in particular has been significantly improved in recent years with the applet security model in Java2 becoming a "malleable system that could be expanded and personalized on an applet-by-applet basis" (McGraw & Felten, 1999 p. 81). This new security model and also the introduction of access control gives administrators more power over controlling security in Java or applet based distributed applications.

2.3.2 Scalability

Scalability is an important design goal for any type of programming, but it can add a layer of complexity to distributed systems especially those which span the world with the aid of technologies such as the Internet.

According to Tanenbaum & Steen (2002, pp. 10-13) there are three areas of which the issue of the scalability of distributed systems can be measured. These are scalable with respect to size, geographically scalable, and administratively scalable. Scalable with respect to size is crucial for a distributed system and it deals with how easy it is to add extra users to a system and the effect that this will have on that system. The key point with size is that a system needs to have enough resources to prevent a bottleneck occurring as the numbers of users grow.

Geographically scalable generally only affects those systems which are using wide area networks (WANs) as this adds a level of unreliability to the system. This issue is made even more complex with the fact that most WANs have services that contain components which are centralised in a particular location, increasing the distance that a distributed system has to cover and therefore increasing reliability issues. In contrast distributed systems which are designed for use in a local area network

environment (LAN) tend to be more reliable because they are functioning on an "in-house" environment.

Finally the administration issues with distributed systems can in fact be the most complex. This is especially an issue with large systems with huge numbers of users. The system needs to be designed in a way that protects each of the individual users from attacks within the distributed system.

2.3.3 Other Areas

There are a number of other areas that are important when dealing with a distributed system; these include the reliability, efficiency, usability, and transparency. Each of which will be discussed in the following.

2.3.3.1 Reliability & Robustness

Reliability and robustness are similar to the scalability of a distributed system, as it is important that a bottleneck is not created when a distributed system grows larger. It is also equally important that a system be reliable, for example if a client fails the system can continue to operate normally and is able to recover any lost work that may not have been completed by the client which failed. This also applies to servers on the system whereby if a server fails the system should have redundancy to allow it to continue to operate normally, minimising downtime.

2.3.3.2 Efficiency

Efficiency is a crucial area of distributed computing for a number of reasons. Firstly it needs to be efficient is such a way that it justifies the reasons for having the distributed system, i.e. it is futile to have a distributed system for something that could have been done easier and quicker on a single workstation. Efficiency is also important in the structure of a distributed system, for example ensuring that servers don't become overloaded and therefore affect the performance of the system, this is usually overcome using load balancing.

2.3.3.3 Usability

The usability of a distributed system is also an issue that needs to be considered as this is how administrators and end users will interact with the system. Ideally the system should hide all issues related to structure of a distributed system from the user. This is usually achieved through the use of graphical user interfaces which

allow users to interact with the specific parts of the system, for example client front end interfaces or a server administration interface.

2.3.3.4 Transparency

As mentioned earlier, it is important that a distributed system "is able to present itself to users and applications as if it were only a single computer system" (Tanenbaum & Steen 2002, pp. 5-7). Therefore it is important that the system has the ability to hide the factors of access, location, migration, relocation, replication, concurrency, and failure from end users.

2.4 Current Trends

In September 1991 the *Computer* periodical published an article titled "*Computer networks and distributed systems*" that looked closely at the areas of distributed computing and the trends that it was expected to take over the coming decade. In the article Larry D. Wittie (1991, pp. 67-76) narrowed down these trends into three dominate themes:

- Tapping the immense data-carrying potential of optical fibres (Very high speed optical communication).
- Efficiency using tightly coupled networks of thousands of computers (Fast parallel and distributed computers).
- Making network access inexpensive so many people will buy the services (Ubiquitous networks).

As predicted, these where the three prominent areas that have effected distributed computing in the last ten years and it is still constantly evolving and adapting as new technologies are developed. High speed networks have helped this growth and many individuals and organisations are now starting to turn to the idea of distributed computing as a functional and cost effective alternative to the traditional supercomputers used for solving large problems. Of course one factor that has contributed to the success of distributed computing is the fast growth of the Internet in recent years. The Internet provides a strong backbone in which distributed systems can be built, whether they are local systems or a worldwide system.

The idea of ubiquitous computing is perhaps the most recent advancement in the computing industry and as such is one that is going to have a large effect on the path that distributed computing takes from here. The idea of ubiquitous computing can be described as "invisible, everywhere computing that does not live on a personal device of any sort, but is in the woodwork of everywhere" (Brumitt 2000, pp. 41-43). As computers continue to grow in popularity the idea of a ubiquitous environment for distributed computing where any number of devices can come together to solve or complete a task is going to help shape distributed systems over the coming years.

2.4.1 Areas of work in Distributed Computing

Distributed systems are in use throughout much of the world today and the popularity of this type of system is constantly growing. This is primarily due to the fact that a distributed system can in many cases result in computing power that greatly exceeds the technological specifications and power of, for example, a single processor system, or expensive supercomputer. Access to this level of computing power has resulted in the number of major distributed system projects increasing.

Research and work in the area of distributed systems has also been accelerating in recent years, partly due to the rapid adoption of Internet technologies. Distributed computing is a very wide ranging area of work and research is being conducted into many of its fields, ranging from the latest object-based distributed systems to the less popular alternatives. Each type of distributed system has its advantages and disadvantages when applied to a particular task, so developers today have a number of possibilities to choose from when deciding the best system to use.

This section will look at several of the areas of research into distributed computing that are closely related to the focus of this thesis, in particular the use of Java as the backbone for distributed systems.

2.4.1.1 Java Applet in Massively parallel computing

The concept of JAM, Java Applet in Massively parallel computing, is research that is being conducted at the School of Computer Science at the Florida International University in the USA. The aim is to achieve high performance distributed computing using the Internet with distributed object technology. Yan & Chen (1999, p. 1) describe basic paradigm of the research as:

> "*A group of server machines that host a regular web server and a task server, and any number of client machines. Any computer equipped with a Web browser can join the client pool to contribute to the computing. This requires the user of the computer to visit the Web server, either by clicking on a hyperlink embedded in a Web page or by opening an URL address that actually leads to a Java Applet stored in the server*"

The driving force for the research is to provide a dynamic distributed computing environment using less expensive workstation clusters rather then traditional and expensive supercomputers. Taking advantage of Internet growth, they have

proposed a distributed system that is based on Java sockets and the use of Applets in web browsers to act as the client to the distributed system server. This research has shown a number of advantages:

- There is a potentially huge client space, as any computer with an Internet connection and a web browser can take part in the system.
- It is also removes the overhead normally associated with distributed system clients of setting up the client and establishing the connection.
- Heterogeneous computing is achieved as any computer can take part so long as it has a web browser equipped with Java
- There is a highly dynamic client pool where any computer can join or leave the distributed system at any time.

Figure 2.4 provides an overview of the computing scheme proposed by this research. It suggests that a server would act as a task manager, distributing parts of the overall task out to clients and also acting as a solution composer to build an overall solution from the results each client returns. At the client level, within the applet there will be a communication layer which provides the socket interface and also a problem solving layer which will work on tasks received from the server.

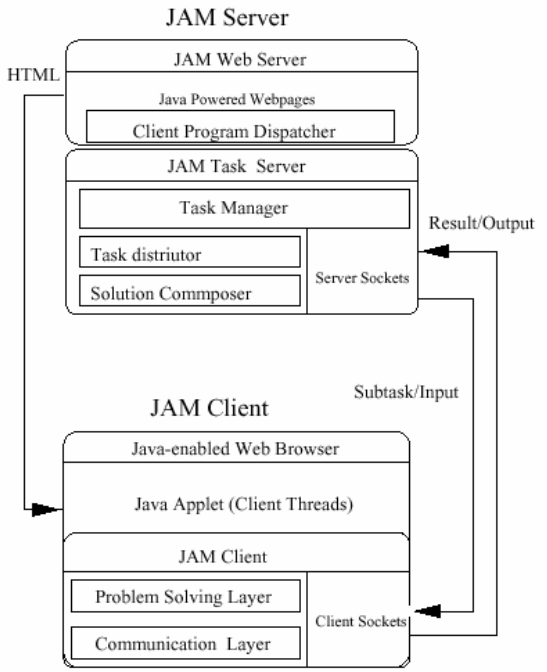

Figure 2.4 – The computing scheme of JAM (Yan & Chen 1999, p. 3)

Potential problems that have been suggested, and are common to most distributed systems include client fault tolerance, and in particular the security issues involved with the client machines. Also suggested as potential problems with a distributed system of this type are three generic barriers to achieving linear speedup in parallel

systems, these are "*startup* (the time needed to start a parallel operation), *interference* (the slowdown due to access to shared resources), and *skew* (the level-off of the speedup as the number of parallel process increases and reaches a point where the size of each subtask is too small such that further partition of the task yields little gain in parallelism" (Yan & Chen 1999, p. 5).

2.4.1.2 SETI@home Project

SETI@home is one example of a project that is currently making use of distributed computing. The project is maintained by researchers at the Space Sciences Laboratory of the University of California, Berkeley and is involved in the search for extraterrestrial intelligence (SETI). Researchers use a 305 meter telescope to collect terabytes of data from the radio signals' emanating from space, this data then needs to be analysed to determine if it is just noise from natural sources of radio emission or if it is in fact a possibility of extraterrestrial intelligence.

This task is potentially huge as "even a small portion of the radio spectrum would require more computational power then is available in the largest existing supercomputer" (Korpela et al. 2002, p. 78). However the process can be achieved through the use of distributed computing as the data collected can be broken up into "work units" which are frequency bands that are basically independent of one another.

The distributed system that is required to analyse this data is huge, therefore the research team make available a client that users can download from the Internet. These users can then install the client, which has been ported to most popular operating systems, on their computers which will then retrieve work units from the server. In effect the users join the distributed system and donate computer time CPU cycles to analysing this data, usually running as a screen saver when the computer is unattended. Once a data unit has been analysed the results are then returned to the server for further analysis.

SETI@home is an example of one of the largest successful distributed computing projects in existence today with more then 2.4 million users downloading the client and taking part in the distributed system.

There are a number of other distributed projects similar to SETI@home which are currently being undertaken with the aid of the Internet. Another example is the *Distributed.net RC5 Project*. This is a distributed project that attempts to find cracks in current encryption protocols (particularly RC5-64) commonly in use throughout the world today. The project is achieved by using distributed clients that attempt to "crack" encryption keys.

2.4.1.3 Mobile Agent Technology

Mobile agents are a technology that has been used for some time with distributed computing. Basically they are a distributed system that is capable of checking if a client machine is available and then distributing a distributed task to that system if it is willing to take part. There are a number of these agent systems available commercially and being successfully used within the computing industry, such as *General Magic's Odyssy*, *IBM's Aglets*, and *ObjectSpace's Voyager*.

There has been a recent surge in interest in distributed mobile agent systems, Kiniry and Zimmerman (1997, p. 21) credit this to the widespread adoption of Java, and in particular several of Java's features, saying that "serialisation, remote method invocation, multithreading, and reflection – have made building first-pass mobile agent systems a fairly simple task".

With the widespread use of mobile agents there are also a number of research projects being conducted at various institutions throughout the world to extend the features provided by agents.

An example of one such project being conducted by the Department of Computer Science at the University of Brasilia is to use mobile agent technology to improve scheduling of applications executed in a cluster environment. This is achieved by the use of load balancing and acquiring more precise information of a client agent's workload before distributing a task to that client, for example ensuring a heavily loaded client does not receive a long and complex task to process. Dantas, Lopes, & Ramos (2002) suggest that their results indicate "that it is possible to spend less elapsed time when considering the execution of a parallel application using the agent approach".

2.5 Summary

The use of distributed computing within the industry has exploded in recent years, and as a result distributed environments are no longer only a viable option for those setting up expensive clusters or supercomputers. With the massive growth of the Internet, distributed computing has been brought to the desktop of all computer users.

Farley (1998) suggests that there are three common motivations for using distributed systems and applications, these being:

- *Computing things in parallel by breaking a problem into smaller pieces enables you to solve larger problems without resorting to larger computers. Instead, you can use smaller, cheaper, easier-to-find computers.*
- *Large data sets are typically difficult to relocate, or easier to control and administer located where they are, so users have to rely on remote data servers to provide needed information.*
- *Redundant processing agents on multiple networked computers can be used by systems that need fault tolerance. If a machine or agent process goes down, the job can still carry on.*

Research underway in these areas of distributed computing is for the most part proving successful with many examples of practical distributed systems being developed. The SETI@home project is a prime example of a well designed distributed project used for analysing a large dataset. Another example is the work proposed by Dermoudy (2002), which considers how parallel execution, load distribution and speculative evaluation can be used in speeding up of the execution of functional programs in an attempt to provide linear speed-ups.

Since this literature review was carried out a number of other distributed research projects have been identified including the "*POPCORN project*" (Camiel, London, Nisan, & Regev, 1997) and the "*Jaguar project*" (Wang & Wang, 2002). These projects investigate the concept of distributed computing using Java over the Internet. Another relevant area of research is the "*Web-based Distributed Topology Discovery of IP Networks project*" (Lin, Wang, Wang, & Chen, 2001). This research is of particular interest as it uses Java servlets and the concept of distributed computing to investigate a web based distributed network management architecture for discovering the topology of networks.

Chapter 3 METHODOLOGY

Distributed Computing is often traditionally associated with a paradigm of fixed hosts; this approach is where a distributed system has a fixed number of clients taking part in a distributed task. In many cases these clients may have special hardware or software requirements in order to take part in this distributed system.

The following chapter presents the steps used in developing a dynamic distributed computing environment written in the Java programming language. This environment is designed to make use of computers with varying usage patterns which are connected to the Internet, thus, providing a distributed computing client base which is dynamically changing.

This provides a dynamic distributed computing environment where an administrator of a large distributed task can submit a task-set to a server and then allow dynamic clients to connect to the server to retrieve individual tasks for processing. These clients will then return the result of the task processing back to the server for viewing by the administrator or for further processing. The processes involved in these interactions between the server and clients will be discussed in further detail later in this chapter.

Also discussed is a description of an example problem that a distributed system such as the one presented here was originally intended to solve. The development of this system is intended to not only solve this problem but could easily be adapted to solve a range of other problems.

3.1 Crossword Problem

The first step in developing any distributed system is defining what the distributed system is going to be used for and in particular the problem that the system is attempting to solve.

An important consideration in choosing a problem for a distributed system is the partitioning or “breaking up” of the overall problem into small sections or tasks that can then be distributed and solved among clients before being brought together to

form an overall result. These tasks may be totally independent of other tasks or may rely on the results of previous tasks in order for them to be solved successfully.

In the case of this work, the problem chosen to develop the distributed system around was that of a crossword. Crosswords are traditional problems that date back to the early twentieth century and the particular angle that this work will look at is that of a crossword solver or put more simply word guessing. Crosswords are made up of intersecting words, therefore the partitioning of an overall crossword into smaller tasks is a relatively easy process, as each task will form a word from the crossword.

The focus of this work is not primarily this problem; rather it is the development of a dynamic distributed system that can be used to solve this problem or similar ones. Therefore it is assumed for the sake of simplicity that each particular task, or word, of this crossword is independent of the other tasks, and therefore does not rely on the previous results. In this case this means that each task is a unknown word from the crossword and has a word length and at least one or possibly more known letters in certain positions in that word.

The work to be undertaken on each individual task is to take what information is known about this word and compare it to an extensive word dictionary, producing a list of possible words that fit the information that is known. While this task is relatively simple it has aspects which make it a computationally long, and intensive process which therefore allow for accurate results to be achieved when part of a distributed system.

3.2 Distributed System

As previously established the language of choice for development of the dynamic distributed computing environment was the Java programming language. A number of key factors contribute to this choice over other languages used for distributed environments, the most crucial of these being the "reliability, simplicity, and architecture neutrality" (Farley 1998) of Java.

Java was not only built from the ground up to include the standard features available in other similar programming languages, but for it to be an Internet programming language as well. As a result the networking, security and multithreaded operations included in the language make it an ideal choice for a web browser based dynamic distributed system operating in an environment which is heterogeneous.

Two areas of Java are particularly well suited for this type of system, these being Java RMI which is included in the core Java API, and Java Servlets which are available as a standard extension to the core API. Both of these approaches have both benefits and disadvantages when being used to develop a system of this type. This work however will concentrate on the approach of Java Servlets, firstly because they are not usually associated with the area of a distributed computing environment of this type, and secondly because there is currently research being conducted within the school of a similar nature into Java RMI and its use in a web based distributed environment.

3.2.1 Java Servlets

Java Servlets have already been discussed in detail in the previous chapter (section 2.1.5.3). However it is important to note here why they are suited to the development of this type of distributed system. Servlets are primarily found in web applications and dynamic content, for example an online book store with shopping cart facilities, and are similar in nature to the more traditional CGI (Common Gateway Interface) scripts often found on web servers.

Servlets are generic server extensions that sit on top of many of the major web servers providing full support for Java in dynamic web applications. All computation occurs on the server side of the system while the results appear on the client end, commonly a web browser. Because servlets inherit the entire core Java

API they too provide the full networking capabilities required by a distributed environment.

3.2.2 Distributed Environment

The focus of this work is to develop a Java servlet based task system, including a server which receives task requests from Java applet based clients, most likely running inside the Java Virtual Machine (JVM) of a web browser. These clients will then receive a task from the server, in the case of the crossword problem an unknown word with some letters and positions identified from a crossword, and can then preform the required computation on the task and return the results back to the server which processes the results. An overview of this process is shown in Figure 3.1 below.

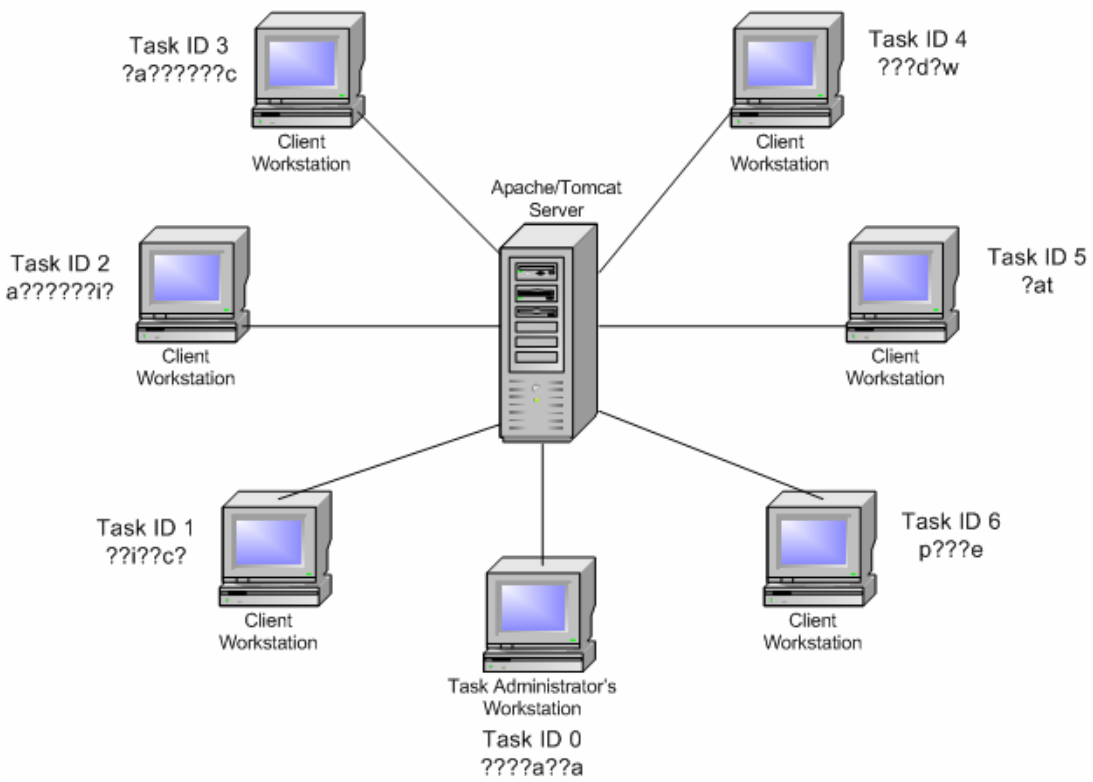

Figure 3.1 – The Java servlet based distributed server, with connected Java applet based clients.

The distributed system can basically be broken up into three separate components: The Server, Client Workstation and Task Administrator's Workstation. The roles of each of these components are outlined below.

The Server – Set of servlets cooperating together to form the basis of the distributed task environment.

Client Workstation – Java applet based client workstation which includes a graphical user interface for user interaction with the task.

Task Administrator's Workstation –Task and system administration comes from a normal client accessing a special set of web pages designed to retrieve and upload task information to the server. This workstation also provides all the features of the normal client applet, thus it can also take part in the current distributed task.

The development of a distributed system of this type was approached as a three step process, firstly to develop a standalone client to solve the problem, secondly to develop a server to manage the distributed system, and thirdly to adapt the standalone client to interface with a server and work in the web based environment. Both the client and server designs will be discussed in detail in the following sections.

3.2.3 Standalone Client

This stage involved the development of a single isolated component of the distributed application that is a totally independent Java application which processes unknown word information and produces a wordlist. This standalone application will form the basis of the client in the distributed environment. Figure 3.2 outlines the model used.

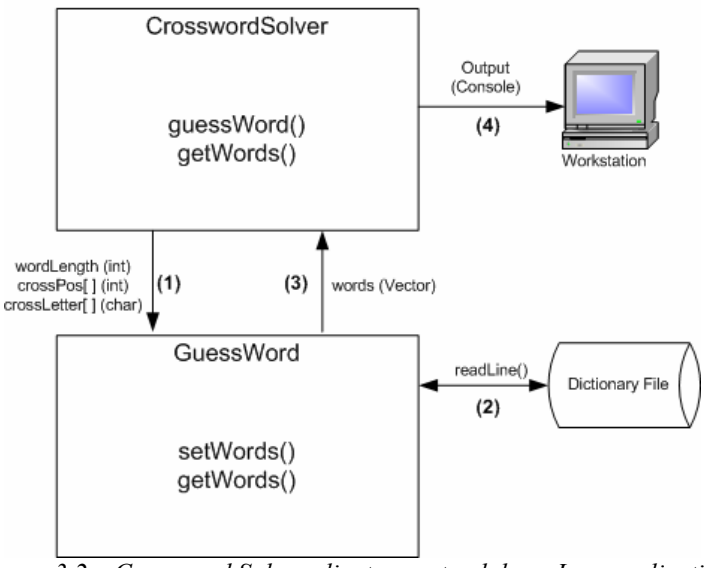

Figure 3.2 – Crossword Solver client as a standalone Java application

The client is made up of two classes, `CrosswordSolver` and `GuessWord`. `CrosswordSolver` is the main class of the application and because the client was

designed with the intention of later being integrated into a client system which can connect to a server all word information was simply hard coded. The `GuessWord` class is the problem or task solver section of the client; it receives word length and known letter information from `CrosswordSolver` (Figure 3.2 - 1). A connection is established to a dictionary file which contains approximately 81,000 different English words each on a new line of the file (Figure 3.2 - 2). Each line of the dictionary is read and compared to what is known about the current task. If a word from the dictionary matches (i.e. it is the right length, and has known letters in the correct positions) it is stored in a results vector which is returned back to `CrosswordSolver` once all words in the dictionary have been checked (Figure 3.2 - 3). Once the results are returned they are displayed on the user's workstation via the Java console (Figure 3.2 - 4).

While this client is simple it provides the basic client task solving abilities that can be adapted into a Java applet based client that will work within the distributed environment.

3.2.4 Distributed Java Servlet Server

The server for the distributed computing environment forms the basis of this work. It is a servlet based server which provides the task and result management features as well as providing an interface for the client applets to connect to and take part in the current distributed task.

There are a number of possibilities in the way the system could be developed, particularly in regards to the protocol used for the clients to connect to the server. Two possibilities where considered; these being the development of a specialised protocol which could provide all means of communication between the servlet and the client, or the use of the existing HTTP protocol that servlets already commonly use to communicate.

The client's interaction with the server would only be a brief exchange of data, i.e. receiving task data or posting task results. Therefore the HTTP protocol would suit the requirements as it is a stateless protocol where "a client, such as a web browser, makes a request, the web server responds, and the transaction is done" (Hunter & Crawford 1998, p. 15). This means that the exchange of data between client and

server could be simplified using the standard HTTP GET requests to retrieve data and POST requests to send data.

Therefore the server is not tied to a custom proprietary protocol and allows it to interface with any client that supports the standard HTTP requests and responses in a way that the server understands.

The physical hardware being used to develop this server needs to be sufficient to provide accurate testing results of the distributed environment. Therefore the server chosen for this testing environment was a Sun SPARCstation 5 with 128MB RAM and 9GB hard disk. The server is running the Sun Solaris 9 operating system which includes the Java2 1.4 SDK and is compatible with the add-on Servlet 2.2 API being used to develop the servlets. An Apache web server is required to be installed for serving the standard web pages that make up the system, also installed is the Apache Tomcat server for deploying the servlets. In a real world distributed environment the hardware requirements for this system would be far greater.

The servlets which make up the server need to provide the following functionality:

Client communication and task distribution – provide the facility for clients to connect to the distributed task system server and request a task for processing.
Client database – the ability to store information such as the hostname and IP address of connected clients.
Task database – the ability to store information about tasks. For the problem discussed in this work this will take the form of a task string such as "*??i??c?*" for the word "*science*". Also stored will be information about the task progress, i.e. allocated or unallocated, completed or uncompleted and the details of the client processing the task.
Task result management – the ability to store the task results returned from clients and mark each task as being completed once the results are received successfully.
Task redundancy – verify that results have been returned for each task. If a client fails and results are never returned, then this task should then be reissued out to another client for processing.

Handle multiple requests – the system needs to be able to accept multiple clients attempting to connect at the same time. By default servlets are multithreaded; therefore this issue can for the most part simply be handled by the servlet API.

The proposed development is four cooperating servlets to handle the above mentioned functionality. These servlets were developed using version 2.2 of the Java Servlets API and receive requests from clients using the standard HTTP requests, GET and POST. The servl1ets will communicate with each other via the same means. This form of Interservlet communication is possible by the use of the `HttpMessage` class used in the client and described in more detail in section 3.2.5. Figure 3.3 below provides an overview of both the client interactions with the server and interservlet communications.

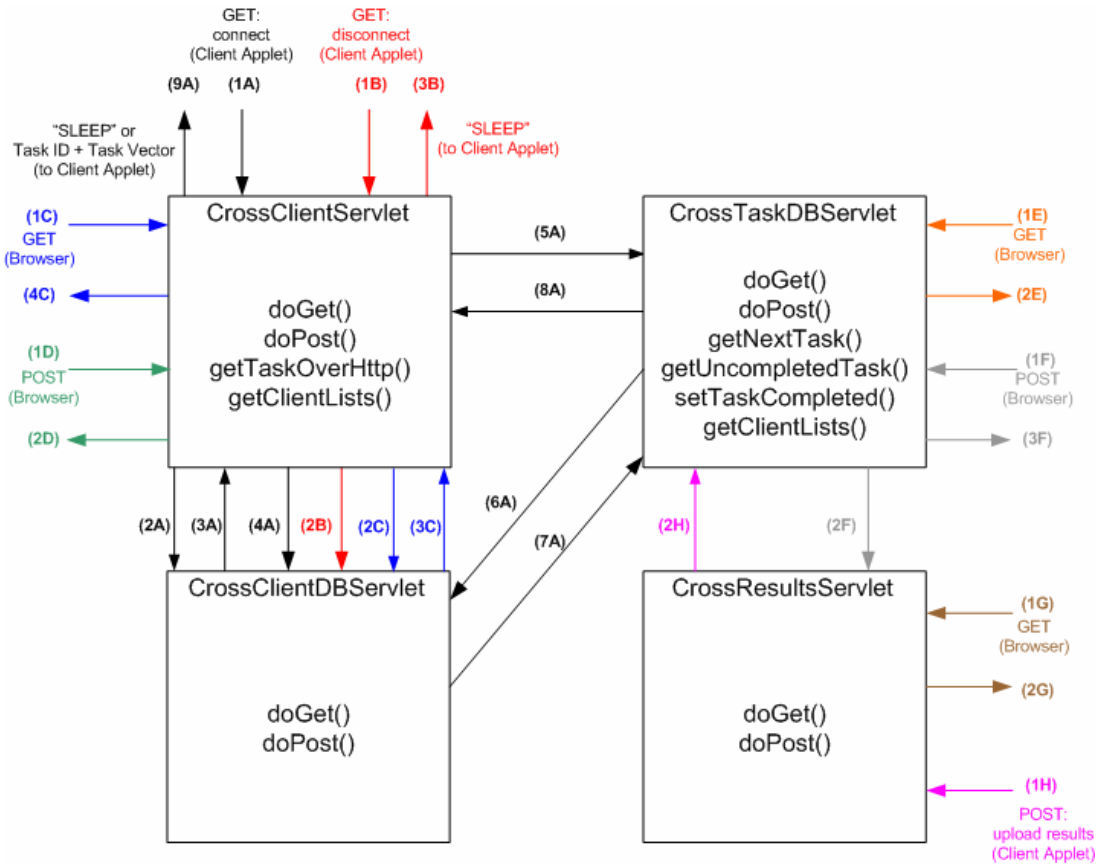

Figure 3.3 – The distributed task server showing the four primary servlets, client interactions with the server and interservlet communications

Sections 3.2.4.1 to 3.2.4.8 below provide a listing of the client/server and interservlet communication scenarios that occur within the distributed environment using the server model discussed.

3.2.4.1 Server Scenario A: Client connects to the system

1. A client applet makes a connection to the server via `CrossClientServlet` and using a GET request, passing the parameter "*connect*" (Figure 3.3 - 1A & Figure 3.4 - 1).
2. Using a GET request `CrossClientServlet` makes a connection to `CrossClientDBServlet` (Figure 3.3 - 2A) requesting the current connected client lists. Two vectors are returned (*clientList* & *clientListHostName*), one containing client host name information, the other containing client IP address information (Figure 3.3 - 3A).
3. The client which is attempting to connect has it's details compared to those currently in the list – if it is a new client it's details (hostname and IP address) are transmitted back to `CrossClientDBServlet` using a POST request (Figure 3.3 - 4A).
4. `CrossClientServlet` compares the current client information to the vector (*clientAuthList*) of clients authorised to take part in the distributed system. If the client is not found in the list the servlet transmits a "*SLEEP*" message contained in an object back to the client as the response to the original GET request (Figure 3.3 - 9A & Figure 3.4 - 2) and this scenario continues no further until the client attempts to connect again after sleeping. Otherwise if a client is authorised a GET request is sent to the `CrossTaskDBServlet` requesting a task for the client (Figure 3.3 - 5A).
5. `CrossTaskDBServlet` first checks the task database for unallocated tasks, if one is found it is marked as allocated and the requesting client's details are stored against that task. The task is converted to an object and then transmitted back as a response to the GET request from `CrossClientServlet` (Figure 3.3 - 8A).
6. Otherwise if no unallocated tasks are available the servlet then checks for uncompleted tasks (i.e. no results returned as yet), if one is found a GET request is sent to `CrossClientDBServlet` (Figure 3.3 - 6A) requesting the current client lists (Figure 3.3 - 7A). If the client who originally received the uncompleted task is no longer connected to the system the currently requesting client's details are recorded against that task and it is then converted into an object and transmitted back as a response to the GET request from

`CrossClientServlet` (Figure 3.3 - 8A). If however the original client is still connected to the system it is assumed that the client is still processing the task and the servlet searches for another uncompleted task.

7. If no unallocated task and then no uncompleted task is found the servlet responds to the `CrossClientServlet` GET request with a "*-1*" indicating that no task is available at this time (Figure 3.3 - 8A).
8. The `CrossClientServlet` analyses the response to its GET request and if it has received a new unallocated task or an uncompleted task it responds to the original client GET request with the task ID and the actual task vector, both stored in separate objects (Figure 3.3 - 9A & Figure 3.4 - 2). If however the servlet received "*-1*" it returns "*SLEEP*" contained in an object as a response, indicating that no tasks are available to the client at this time (Figure 3.3 - 9A & Figure 3.4 - 2).

Figure 3.4 below illustrates a typical exchange between the client and server.

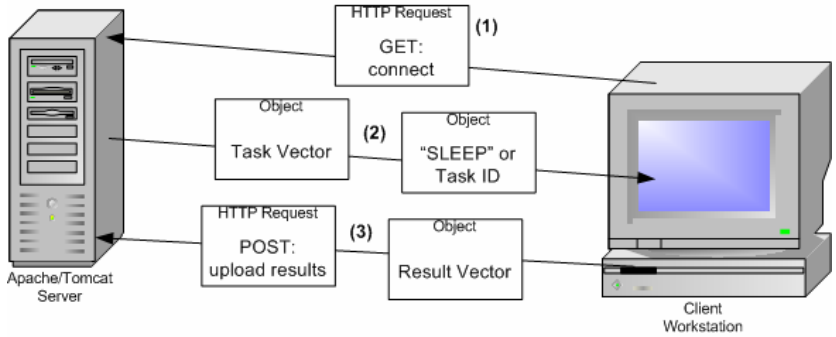

Figure 3.4 – The communication between the distributed task server and client. Showing the HTTP requests and the responses sent as a result of these requests.
In the case of (2) the task vector object is only sent if the first object was a task ID, otherwise the "SLEEP" object means that no task is available and it is all that is transmitted in this instance.

Figure 3.5 below illustrates the object that is sent to a client containing a typical task vector.

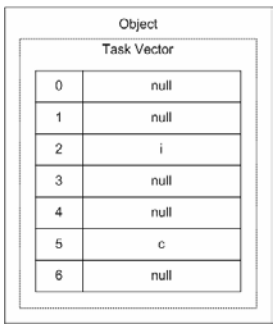

Figure 3.5 – The object transmitted between the server and client containing a task vector.
In this case the unknown word task string is "??i??c?" for the word "science".

3.2.4.2 Server Scenario B: Client disconnects from the system

1. A client applet makes a connection to the server via `CrossClientServlet` and using a GET request, passing the parameter "*disconnect*" (Figure 3.3 – 1B).
2. `CrossClientServlet` processes the disconnect request and sends a POST message to `CrossClientDBServlet` with the details of the client stored in an object (Figure 3.3 - 2B). The servlet receives the POST request and removes the client's details from the client hostname and IP address vectors.
3. `CrossClientServlet` responds to the original disconnect GET request passing a "*SLEEP*" message contained in an object (Figure 3.3 – 3B). The "*SLEEP*" object is returned to the client to acknowledge that the disconnect request has been received successfully.

3.2.4.3 Server Scenario C: Browser requests client interface

1. A web browser makes a connection to the server via `CrossClientServlet` and using a GET request, passing no parameters. (Figure 3.3 – 1C).
2. Because no parameters where passed with the request, the servlet interprets this as a request for the client selection interface (see section 4.3.4.2).
3. The servlet generates the HTML page and passes it back to the web browser as a response to the original GET request (Figure 3.3 – 4C). During the HTML page generation the servlet will make a connection to the `CrossClientDBServlet` requesting (using a GET request) the connected client vectors (Figure 3.3 – 2C). This data is received (Figure 3.3 – 3C) and used to populate the currently connected clients HTML form component.

3.2.4.4 Server Scenario D: Browser posts data from client interface

1. A web browser makes a connection to the server via `CrossClientServlet`.
2. Using the POST request it transmits data collected from the client selection interface (Figure 3.3 – 1D). This data will be list of clients authorised to take part in a task or a request to select all clients as being authorised.
3. The servlet receives the data and stores the information in either the *clientAuthList* vector or the *selectAllClients* boolean variable.

4. The servlet acknowledges that the authorised client data has been received and generates a HTML page (listing the selected clients) and passes it back to the web browser as a response to the original POST request (Figure 3.3 – 2D.)

3.2.4.5 Server Scenario E: Browser requests task list

1. A web browser makes a connection to the server via `CrossTaskDBServlet` and using a GET request, passing no parameters. (Figure 3.3 – 1E).
2. Because no parameters where passed with the request, the servlet interprets this as a request for a listing of the current tasks in the database.
3. The servlet generates the HTML page and passes it back to the web browser as a response to the original GET request (Figure 3.3 – 2E).

3.2.4.6 Server Scenario F: Browser posts task data

1. A web browser makes a connection to the server via `CrossTaskDBServlet`.
2. Using a POST request (Figure 3.3 – 1F) it transmits data collected from the task management interface (see section 4.3.4.3). This data will either be a list of new tasks to be added to the task database or a request to clear all existing tasks in the system.
3. The servlet receives the data and first checks to see if the user selected the clear task database option. If so the task database is cleared and the servlet sends a blank POST request to `CrossResultsServlet` informing it to clear the task results database (Figure 3.3 – 2F).
4. Otherwise the data being posted is new task strings and this is stored in the task database vector (*taskVector*).
5. The servlet generates a HTML page (listing the new tasks added or a message saying the database has been cleared) and passes it back to the web browser as a response to the original POST request (Figure 3.3 – 3F).

3.2.4.7 Server Scenario G: Browser requests task results

1. A web browser makes a connection to the server via `CrossResultsServlet` and using a GET request, passing no parameters. (Figure 3.3 – 1G).
2. Because no parameters where passed with the request, the servlet interprets this as a request for a listing of the current task results in the database vector (*wordVectorResults* & *taskIDVectorResults*).

3. The servlet generates the HTML page and passes it back to the web browser as a response to the original GET request (Figure 3.3 – 2G).

3.2.4.8 Server Scenario H: Client Applet transmits task results

1. A client applet makes a connection to the server via `CrossResultsServlet`, using a POST request with a parameter informing the servlet that results are about to be transmitted.
2. The client transmits an object containing a vector with includes the task ID information and the results of a task (Figure 3.3 – 1H & Figure 3.4 - 3).
3. The object is received and the task ID is extracted from the vector leaving only the results for that particular task which are then stored within the result database vector (*wordVectorResults)*.
4. The *taskIDVectorResults* vector is used to store the task IDs of each task in the order they are received by the server.
5. The servlet then sends a POST request to `CrossTaskDBServlet` informing it that the task has completed and the number of results received for that task. (Figure 3.3 – 2H).
6. `CrossTaskDBServlet` receives this message and then marks the specified task as being completed.

Figure 3.6 below illustrates an example result object transmitted between the client and server

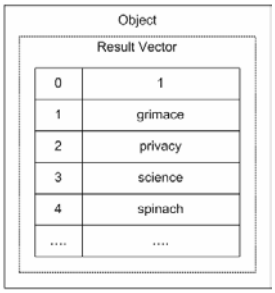

Figure 3.6 – The object transmitted between the server and client containing a set of task results. The object contains a vector of results, of which the first element is the task ID for the task

3.2.5 Java Applet based Client

Once the standalone client had been developed the process of adapting the client to work in a distributed environment was relatively straightforward. The first step was to take the `CrosswordSolver` class and convert it from an application to a Java

Applet with a graphical user interface (GUI) which allows the end-user easier access in controlling the client as well as providing status information (see section 4.3.4.4).

As already mentioned the server has been designed to use standard HTTP requests to communicate with clients. This means that the client needs to be adapted so that when requesting a task to process it sends a GET request to the server, and then when returning results uses a POST request to transmit the result data. This functionality is provided through the use of the third party class - `HttpMessage`. The `HttpMessage` class has been developed by Hunter, J. for the O'Reilly & Associates publishing group and provides a general-purpose class for HTTP communication between applets and servlets, and in the case of the server interservlet communications between servlets.

`HttpMessage` allows multiple GET and/or POST requests to be sent to a servlet at a specified URL. Data can be transferred with these requests as serialized Java objects, serialisation being a feature of the *java.io* package and is the ability to "convert an object into a stream of bytes that can later be deserialized back into a copy of the original object" (Flanagan 1999). Serialization makes it possible to transmit the object via the HTTP protocol, and is ideal for the transmission of task data and results, both of which are contained in vectors. The `HttpMessage` class is described in further detail in Hunter & Crawford (1998, pp. 289 - 296).

The client has no specific hardware requirements, except that of the Java2 Runtime environment (version 1.3 or 1.4) which is required for the applet Hardware will however affect the speed at which the client can process a task. A Java compatible web browser or applet viewer is also requited to view and interact with the client.

In addition to GUI and server connectivity the distributed version of the client needs to provide the following functionality:

Allow the client execution to sleep – Because the server may not always hold a ready supply of tasks to send to a particular client it will transmit a "*SLEEP*" message back. The client needs to be able to receive this message and then pause the execution of the client for a certain amount of time, in the case of this client 30 seconds. This functionality can be provided by the creation of another thread which

controls all server communications and task solving. This means that this thread could be put to sleep without affecting the overall applet and its GUI components.

Easily adaptable to another task – The client should be easily adaptable to any type of task not just the Crossword problem. This is achievable because in this case the `GuessWord` class does the bulk of the task processing, while `CrosswordSolver` provides the applet interface and servlet communications. Therefore meaning that the `GuessWord` class could simply be replaced with another class for a new type of task, however some minor changes to `CrosswordSolver` would be unavoidable. Figure 3.7 below provides an overview of the modified client which now runs within the distributed environment.

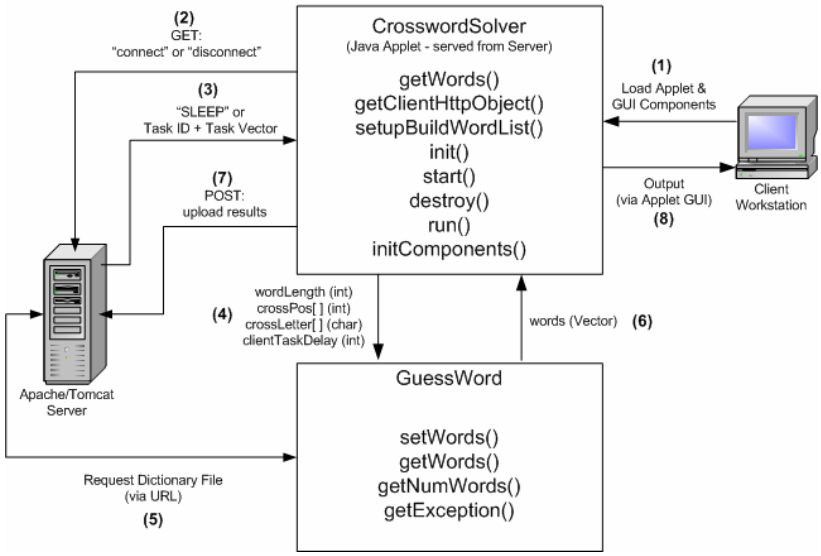

Figure 3.7 - Crossword Solver client operating within the distributed environment

The basic structure of the standalone client still exists; however the `CosswordSolver` class has been modified to not only work as an applet, but to communicate with the server using HTTP requests. The `GuessWord` class had very little change except that it now reads its dictionary file from a web page located on the server which originally served the applet. The reason this change was required is because Java security by default does not permit applets access to the local file system, and the only network connections that can be made by an applet are back to the originating server. This also applies to the servlets that the applet needs to communicate with, therefore the client applet must be served off the same server as the distributed server.

An example of a basic client/server exchange follows:

1. A client workstation would visit the web page to load the client applet (Figure 3.7 - 1).
2. The user would click the "*Connect*" button and the applet would send a GET request with the parameter "*connect*" to `CrossClientServlet` on the server (Figure 3.7 - 2).
3. The server will as a response to the GET request either return an object containing the string "*SLEEP*" in which case there are no tasks available for this client and it will sleep for 30 seconds before repeating the GET request again, or the server will return an object containing a task ID followed by another object contain a task vector (Figure 3.7 - 3).
4. The task vector will be converted into the form `GuessWord` is expecting (word length and known letter information) and then passed to the `GuessWord` class for processing (Figure 3.7 – 4).
5. `GuessWord` will establish a connection back to the server requesting the web page containing the word dictionary (Figure 3.7 – 5).
6. Each line of the dictionary is read and compared to what is known about the current task. If a word from the dictionary matches (i.e. it is the right length, and has known letters in the correct positions) it is stored in the results vector which is returned back to `CrosswordSolver` once all words in the dictionary have been checked (Figure 3.7 – 6).
7. The task ID of the current task is appended to the first element of the result vector for identification purposes.
8. `CrosswordSolver` then makes a POST request to `CrossResultsServlet` passing the results vector back to the distributed server (Figure 3.7 – 7).
9. Finally the client will check if the user requested to see the results of the task via an option on the client interface. If so the results will be displayed on the client interface (Figure 3.7 – 8) and a GET request will be sent to the server with the parameter "*disconnect*" informing the server that the client should be disconnected from the distributed system (Figure 3.7 – 2).
10. However if the user did not ask to see task results then task statistics will be displayed on the client interface (Figure 3.7 – 8), and the client will then start

the process again attempting to connect and retrieve a new task (Figure 3.7 – 2). This process will continue until the user selects the "*Disconnect*" button on the client interface.

3.2.6 Implementation

This section provides an overview of the classes and methods used in developing the distributed server and client

3.2.6.1 Server

The server requires that the GET and POST requests in `CrossClientServlet`, `CrossTaskDBServlet`, and `CrossResultsServlet` are synchronised in such a way that allows only either one GET or POST request per servlet to occur at any one time. Java provides this ability by default through the use of synchronisation and any requests that occur when another is already in progress will simply be queued.

The reason for this requirement is that the servlets use global variables for storing task information and if multiple requests are allowed to occur at the same time it increases the chance of the information being transferred getting mixed together. This only has a limited effect on the performance of the server as the GET and POST requests are normally only brief connections.

The source code for the Java servlet based distributed server developed in this work is provided in Appendix A of this thesis. The following is an overview of the implementation used for the server.

CrossClientServlet

This servlet is the first point of contact for a client with the server. It manages all communication with the clients as well as providing access control to authorised clients and distributing tasks.

doGet() – Method called by the servlet when it receives a GET request. Receives connect requests from clients and sends out tasks to the client or sends a sleep message if no task is available. Also allows clients to be disconnected from the system and displays the client selection interface on a client browser.

doPost() – Method called by the servlet when it receives a POST request. Gets the POST request from the client selection interface and stores the authorised client details.

getTaskOverHttp() – Retrieves a task from the `CrossTaskDBServlet` (new task or uncompleted) and returns the task ID of the task or "*-1*" if no task available to this client

getClientLists() – Retrieves the connected client list vectors from `CrossClientDBServlet`.

CrossTaskDBServlet

This servlet manages the task database vectors. The servlet receives requests from `CrossClientServlet` to retrieve tasks from the database, as well as receiving new tasks being added into the database via the task management interface.

doGet() – Method called by the servlet when it receives a GET request. Retrieves tasks from within the database vectors or displays the current tasks on the task management interface.

doPost() – Method called by the servlet when it receives a POST request. Allows the servlet to receive new task information (or clear task request) from the task management interface, as well as or receive update requests from `CrossResultsServlet` to mark a task as being completed (i.e. results received).

getNextTask() – Retrieves the next unassigned task from the database vectors. Takes the client ID of the client requesting the task and returns the task ID of a task or "*-1*" if no task available.

getUncompletedTask() – Retrieves the next uncompleted task from the database vectors where the client that originally received the task is no longer connected to the system. Takes the client ID of the client requesting the task and returns the task ID of a task or "*-1*" if no task available.

setTaskCompleted() – Marks a task a being completed if a POST request from `CrossResultsServlet` is received by doPost() indicating that a task has been completed successfully.

getClientLists() – Retrieves the connected client vectors from `CrossClientDBServlet`.

CrossClientDBServlet
Servlet for storing information about clients connected to the distributed system.

doGet() – Method called by the servlet when it receives a GET request. Retrieves the client vectors and sends them back to the requesting servlet.
doPost() – Method called by the servlet when it receives a POST request. Receives requests to add clients to the vectors, as well as remove client details from the vectors.

CrossResultsServlet
The clients communicate with this servlet to send the results of tasks after processing has been completed.

doGet() – Method called by the servlet when it receives a GET request. Displays the current results stored in the database vectors.
doPost() – Method called by the servlet when it receives a POST request. Stores new results sent to the servlet from the client into the results database vector. Also clears the result database vectors if a request from `CrossTaskDBServlet` is received.

3.2.6.2 Client
The source code for the Java applet based distributed client developed in this work is provided in Appendix B of this thesis. The following is an overview of the implementation used for the client.

CrosswordSolver
The `CrosswordSolver` class is the main class of the client applet and provides all GUI components and networking capabilities. This class is mostly independent of the specific task that is currently being distributed via the system.

getWords() – Retrieves the task results from `GuessWord` and posts the results to the server.
getClientHttpObject() – Connects to the server and attempts to either download a task or disconnect the client from the server, depending on what the user selected from the applet interface.

setupBuildWordList() – Takes the task vector sent from the server and converts it into the form `GuessWord` is expecting – an integer equal to the word length and two arrays with known letter information (actual letters, and the letter positions).
init() – Initialises the applet.
start() – Creates and starts a new thread for the processing to occur in. This allows the client to sleep when no tasks are available while still allowing the GUI components to function normally.
destroy() – Called when the applet is closed – automatically sends the disconnect message to the server.
run() – Main method which provides the client processing order by calling the other methods, interpreting the results and updating the user interface.
initComponents() – Initialises the GUI components.

GuessWord
`GuessWord` is the main task solving class and is called by `CrosswordSolver`. This class is specific to the current crossword problem.

setWords() – Opens a connection to the dictionary web page and compares the task with each word in the dictionary. Adding results to the results vector where appropriate.
getWords() – Retrieves the task results vector.
getNumWords() – Retrieves the number of words returned for the current task.
getException() – Retrieves any exception that may have occurred. Required to display error messages on the client interface, e.g. "*Connection timed out*".

3.3 Summary

This chapter has proposed a dynamic distributed computing environment which is designed to make use of computers with varying usage patterns connected to a medium such as the Internet, thus, providing a distributed computing client base which is dynamically changing. Also proposed was a crossword problem which can be applied to distributed environment.

The next chapter will present the results obtained through implementing and testing the proposed environment.

Chapter 4 RESULTS AND DISCUSSION

The aim of this chapter is to establish the testing procedures and present the results obtained based on the performance of the dynamic distributed computing environment presented in the previous chapter.

In particular this chapter will present results which will allow justification to be made as to whether the proposed dynamic distributed environment is a plausible alterative to fixed host distributed computing.

4.1 Testing

One of the primary objectives in distributed computing of any type is to take a task which is computationally long or complex and distribute it among client computers for processing. This results in the reduction of the need for large and expensive supercomputers which are normally required for a task with a large and complex data set. Therefore the techniques of testing a distributed system, such as the one presented in the previous chapter are primarily based on the performance and efficiency of the system overall.

This section outlines the techniques used in testing the performance of the dynamic distributed computing environment when processing a standard task-set based on the crossword problem.

4.1.1 Distributed Environment

The primary purpose of the work presented in this thesis is the development of a dynamic Java based distributed computing environment which operates with a primarily unknown client base. Therefore in testing the performance of this system a number of factors have to be investigated such as the number of clients processing tasks, the time taken for an entire task-set to be processed, and the overall server performance.

To accurately measure the time taken for a task-set to be processed the distributed server was modified to record the time in which the distributed task is started (the point at which the task administrator authorises a list of clients to take part in the distributed task) and also the time at which the last task from the current task-set is issued to a client. The time taken to process all tasks provides a performance result

for the distributed environment based on the number of clients which took part in the processing.

Therefore performance measurements are first taken with just one (1) client connected to the system; this provides a benchmark result for comparing the time taken to process an entire task-set. This result can then be compared to the performance of the system with 2, 3, 5, 10 and finally 20 computers processing tasks from the same task-set.

An important consideration in the performance of a distributed system of this type is heavily based on the clients. The ability of each client being able to process its tasks will have an overall effect on the performance of the system as a whole. Therefore for testing purposes two client bases of 20 machines each where chosen. The first of these being PC based DELL Pentium 3 866 MHz computers, each with 256MB of RAM and running Microsoft Windows 2000. Each machine was running Netscape 6.2 which includes the Java2 1.3.1 runtime environment used for running the distributed client applet. The other clients were Apple 700 MHz PowerPC G4 based iMacs, each with 256MB of RAM and running MacOS X 10.1. On these clients the distributed client was run within Microsoft Internet Explorer 5.2 which also includes the Java2 1.3.1 runtime environment. All clients where connected to a switched 100 Mbps local area network (LAN).

Overall server performance is also of interest. Therefore the server's CPU and Network utilisation are also monitored to provide statistics of the server performance based on the number of clients connected to the distributed system.

4.1.2 Crossword Task-Set

While the crossword problem presented in this work is not a computationally complex problem, it provides a good means of testing the performance of the system based on the time taken to process each task and produce a results wordlist.

In order to provide accurate results the same standard task-set was used to perform all testing of the distributed system. This task-set was a standard list of 250 randomly selected words each with a varying degree of known letter and word length information. When compared against the dictionary file by a client these words produce results ranging from one (1) through to approximately 350 possible matching words. A listing of this task-set is provided in Appendix C of this thesis.

4.2 Results

This section presents the results obtained using the distributed system following the testing procedures outlined in the previous section. As also previously mentioned the task-set used in obtaining results is a set of 250 words with varying degree of known letter and word length information.

Test Results (Apple 700Mhz PowerPC G4 based iMac)						
Clients	**Run 1** (seconds)	**Run 2** (seconds)	**Run 3** (seconds)	**Run 4** (seconds)	**Run 5** (seconds)	**Average** (seconds)
1	362.684	360.025	361.638	379.713	343.687	362
2	296.547	303.722	290.624	296.163	290.721	296
3	267.122	263.33	236.335	270.804	267.786	261
5	245.158	250.452	253.739	252.118	255.794	251
10	238.222	232.053	236.556	221.638	229.061	232
20	213.376	226.361	224.352	216.314	224.575	221

Table 4.1 – Average total task-set time for between 1 to 20 iMac based clients

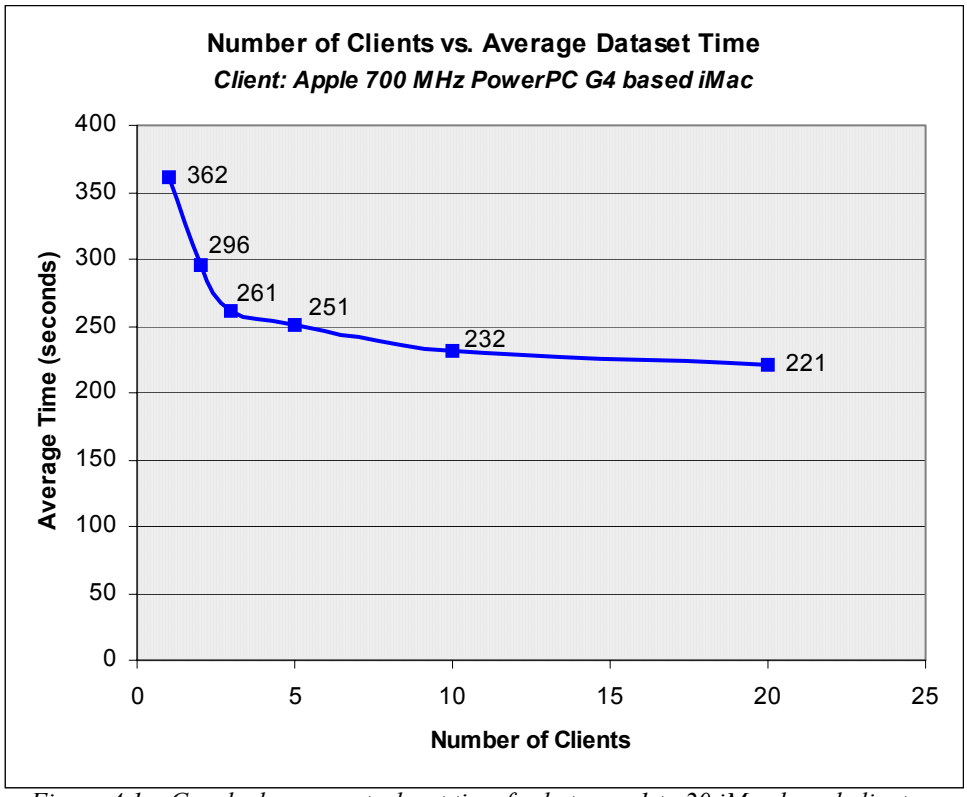

Figure 4.1 – Graphed average task-set time for between 1 to 20 iMac based clients

Table 4.1 shows the time taken for an entire task-set to be processed by the distributed system with between 1 to 20 clients taking part. In order to gain accurate results for each number of clients connected the tests where repeated five times and then an average was calculated of total time taken. On average in this test a client is

able to retrieve a single task from the server, process the task and return the results to the server in approximately 1.5 seconds. This figure does vary slightly as more clients connect to the system and the server load increases, as described in section 4.3.1.

Figure 4.1 presents a graphed version of the average task-set time versus number of clients. As shown there is a large decrease in time required to process the results when 20 clients are connected to the system compared to a single client (approximately 141 seconds difference). The results show that as more clients take part in the distributed system the total time required reduces. They also show that the decrease in time is relatively linear for between 1 to 3 clients; however this is not true overall. From 5 clients through to 20 the time still decreases however at a significantly lower rate with the difference in performance between 5 and 20 clients only being approximately 30 seconds, compared to that of the difference between one (1) and 3 clients being approximately 101 seconds. This suggests the performance of the server is being affected by the number of clients connected to the system and this will be discussed in detail in section 4.3.1.

The previous results where obtained by using iMacs as the client base. In order to see if the results are effected by the clients which are connected to the system the tests where repeated again but this time using PC based computers. On average in this test a client is able to retrieve a single task from the server, process the task and return the results to the server in approximately 1.2 seconds, but again this does vary. Table 4.2 below presents the results obtained from these tests.

Test Results (DELL PIII 866Mhz based PC)						
Clients	**Run 1** (seconds)	**Run 2** (seconds)	**Run 3** (seconds)	**Run 4** (seconds)	**Run 5** (seconds)	**Average** (seconds)
1	321.121	318.888	315.821	317.044	320.579	319
2	278.878	290.531	290.179	279.731	282.369	284
3	267.856	275.161	285.296	287.721	282.15	280
5	250.531	260.099	261.852	257.324	255.341	257
10	231.596	246.08	244.454	242.627	238.3	241
20	219.22	231.471	226.37	229.366	225.559	226

Table 4.2 – Average total task-set time for between 1 to 20 PC based clients

Figure 4.2 provides a graphed representation of the average time taken; which shows a similar result to those obtained using the iMac clients. Interestingly these results are closer to being linear then the previous tests. This could be attributed to the fact that these clients where able to process tasks quicker, with a single PC client on average only taking 319 seconds to process the task-set compared to a single iMac client taking on average 362 seconds.

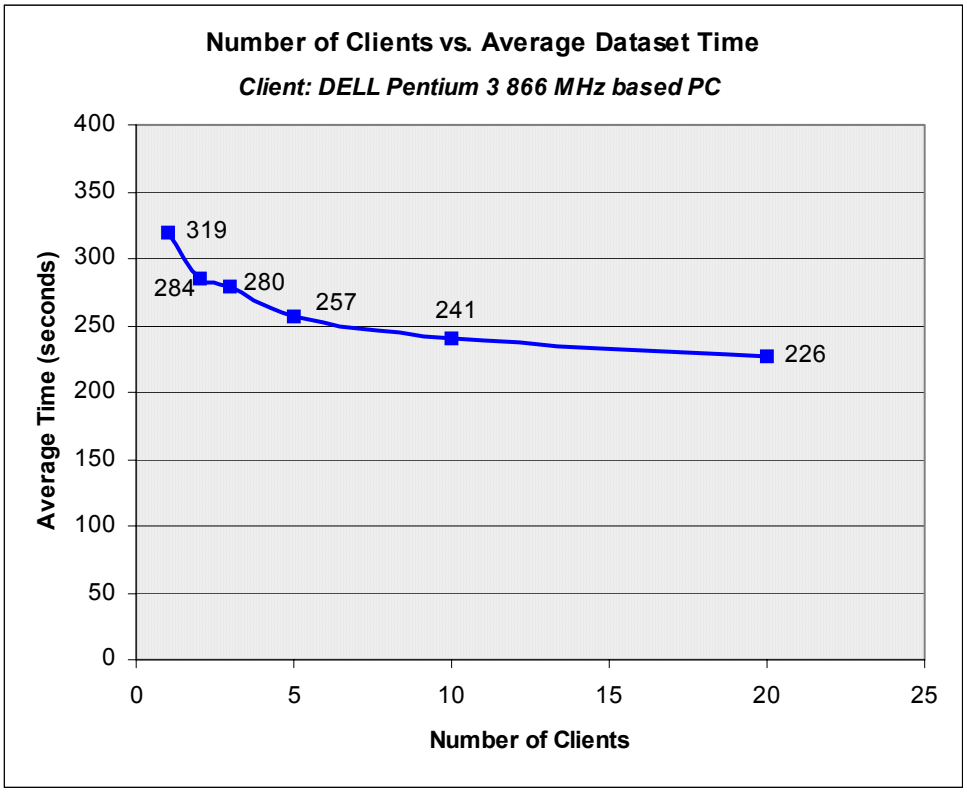

Figure 4.2 – Graphed average task-set time for between 1 to 20 PC based clients

However, there is a considerable slow down in the processing speed of the task-set after 3 or more PC based clients have connected to the system. With the data set taking slightly longer to be processed in its entirety then it did when compared to the iMac clients. This again suggests that the server performance is beginning to become an issue as the number of clients on the system increase, particularly as these clients where able to process tasks faster and therefore put a heavier load on the server as they request tasks and transmit results. This is also supported by the fact that the difference between 1 and 3 clients (approximately 39 seconds), and between 5 and 20 clients (approximately 31 seconds) was only 8 seconds compared to the same result for the iMac clients which was 71 seconds.

To further test this theory the client applet was modified to receive a variable called a "*client delay*". This was designed to slow the task processing speed down by making the client go into a loop for a certain number of times as specified by the task administrator. This delay had no effect on the results produced by the client except that the time taken to produce these results was increased. This was added with the intention of reducing the load placed on the server because the clients wouldn't be continually connecting to request tasks and transmit results.

Test Results (Apple 700Mhz PowerPC G4 based iMac) - with delay						
Clients	**Run 1** (seconds)	**Run 2** (seconds)	**Run 3** (seconds)	**Run 4** (seconds)	**Run 5** (seconds)	**Average** (seconds)
1	2417.18	2378.315	2379.735	2376.443	2483.639	2407
2	1211.952	1220.587	1214.635	1212.791	1246.16	1221
3	826.01	831.039	833.069	827.028	829.22	829
4	636.03	636.512	621.996	634.866	626.582	631
5	507.942	516.966	503.581	505.991	516.365	510

Table 4.3 – Average total task-set time for between 1 to 5 iMac based clients with a client delay of 5000

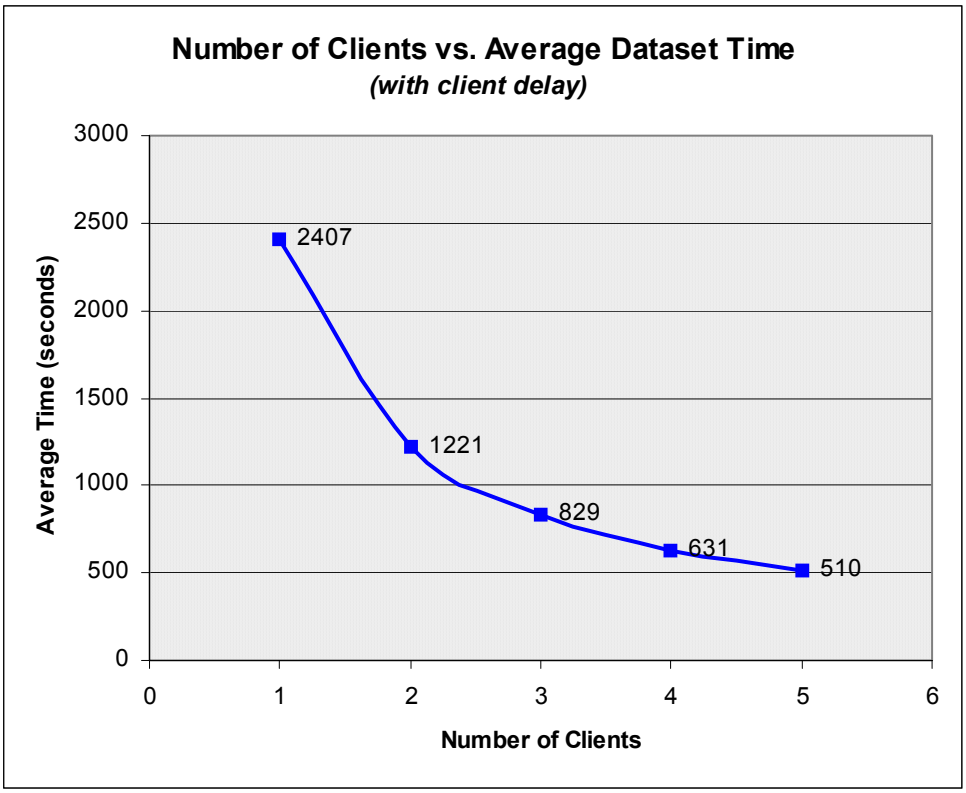

Figure 4.3 – Graphed average task-set time for between 1 to 5 iMac based clients with a client delay of 5000

Table 4.3 above shows the results obtained from running the tests again on the iMac clients for between 1 to 5 clients connected to the system, and with the client looping

(client delay) 5000 times. This on average increased the time an iMac client was able to retrieve a single task from the server, process the task and return the results to the server to approximately 10 seconds.

The results show that by increasing the task complexity the average time taken for a single iMac client to process the task-set was increased from 362 seconds to 2,407 seconds. Figure 4.3 shows that this had a slight effect on the overall performance of the system with the results produced suggesting a more linear trend then the original iMac client tests. However the results also show that as the client number increased there was still a slowdown trend of the distributed system performance, again bringing the performance of the server into question.

4.3 Discussion

This section will discuss the results presented in section 4.2, paying particular attention to the factors which affect the performance of the dynamic distributed environment. Specifically this section will investigate the issues of server load and synchronisation which where discovered, while also discussing the security of the system and the user interfaces implemented.

It should be noted that the test results presented in the previous section where conducted in a switched local area network (LAN) environment. Therefore issues of network congestion could be ignored, however further testing is required in this area and this will be discussed in further detail in Chapter 5.

4.3.1 Server Load

As previously discussed, during the testing of the distributed system it was found that as the client number increased the performance of the overall system decreased. While the performance still improved overall, the level of improvement decreased (flattened out) with each extra client added to the system. While it is reasonable to assume that the time taken would eventually flatten out as the overall performance can only be improved to a certain point, this behaviour was observed with only a small number of clients connected.

This questions the server performance, especially upon visual observation of the clients which when processing tasks in an environment with for example 20 clients connected, took considerably longer to retrieve tasks from the server and access the dictionary file used for building the wordlist. This results an increase in time taken to process tasks by in some cases up to 20 to 30 seconds.

This claim is further supported by clients occasionally having "*connection timed out*" errors when attempting to connect to the dictionary file. This error was simply caught by the client which then attempts to connect again, with the task usually then being processed successfully. Because the dynamic distributed system could potentially have an unlimited number of clients connected the Apache web server which was used to serve the dictionary file to the clients was set up to have a dynamic pool of *httpd* processes waiting to process requests from clients. If there are more requests then there are available processes extra ones are loaded to cope

with the demand. The Apache Tomcat server which serves the servlets and processes requests from clients works in a similar way and operates in multithreaded environment meaning that it is able to process requests from multiple clients at the same time.

Therefore the tests where run again, but this time with server CPU and Network Utilisation being monitored to see if the server's performance was being affected by the demand of a large number of clients connected to the system. The results of the CPU utilisation tests are presented below.

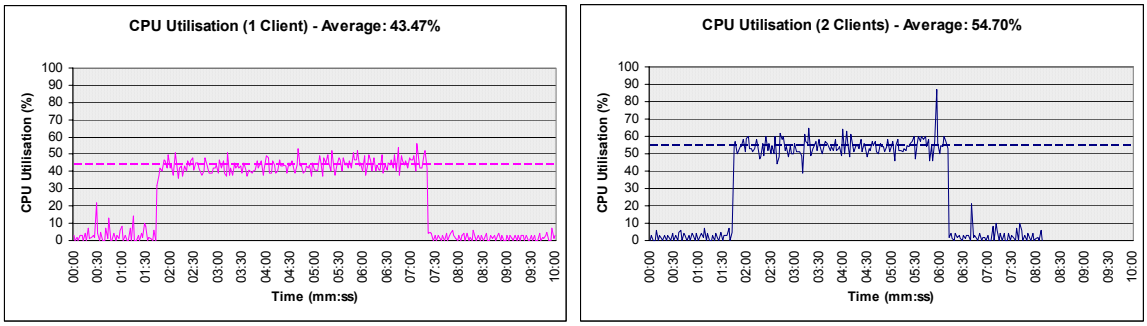

Figure 4.4 – CPU Utilisation for the server with 1 and 2 clients connected

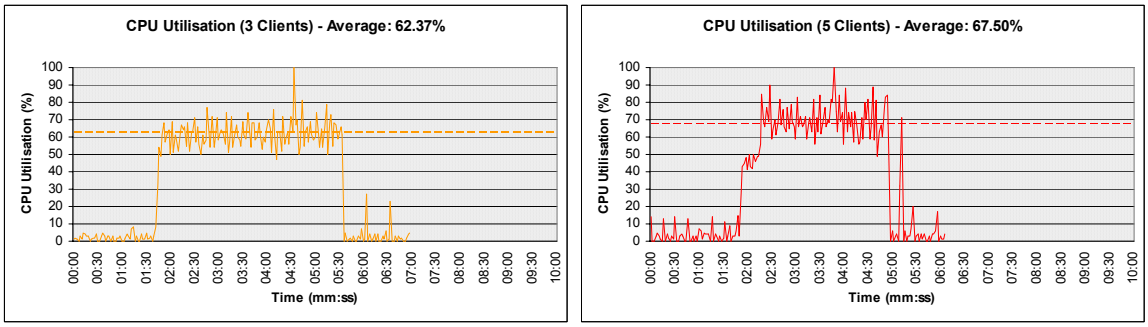

Figure 4.5 – CPU Utilisation for the server with 3 and 5 clients connected

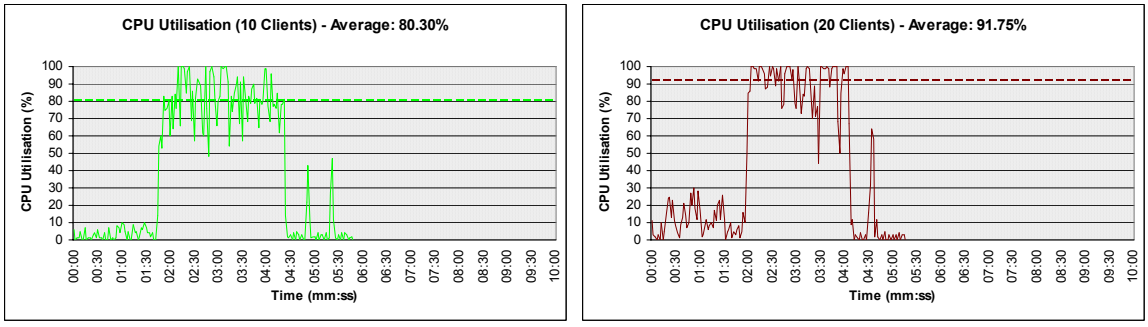

Figure 4.6 – CPU Utilisation for the server with 10 and 20 clients connected

CPU utilisation was monitored over a period of 10 minutes with the distributed task commencing at approximately 1 minute and 30 seconds. The server is not used for any other purpose apart from the distributed system and therefore it remains relatively idle at times when not servicing the requests of clients. Figure 4.4 shows that the average server CPU utilisation for a single client was 43.47% and this then

increased to 54.7% by adding a second client to the system. Figure 4.5 shows that when a third client was added the average utilisation increased to 62.37% and at one time peaked to approximately 100%. Adding a fifth client increased the average utilisation again to 67.5% with several peaks reaching the 80% to 100% range. Finally Figure 4.6 shows that with 10 clients connected the average utilisation is increased to 80.3% and with 20 clients connected the average utilisation is approximately 91.75%, but also reaching 100% at several points during the task-set processing.

All charts show that there is a considerable increase in utilisation once the task processing commences and this then drops back to less then 10% once the task-set is completed. For the most part the utilisation during this idle time remains at is less then 10% however there are several peaks, especially in the case of 20 clients. This can be attributed to multiple clients waking from sleep and attempting to connect to the server at the same time, requesting a task and then being sent the "*SLEEP*" message from the server. The clients are designed to sleep for a period of 30 seconds, Figure 4.6 with 10 clients connected perhaps shows this the clearest with a peak at approximately the 5 minute mark and then again at the 5 minute 30 seconds mark.

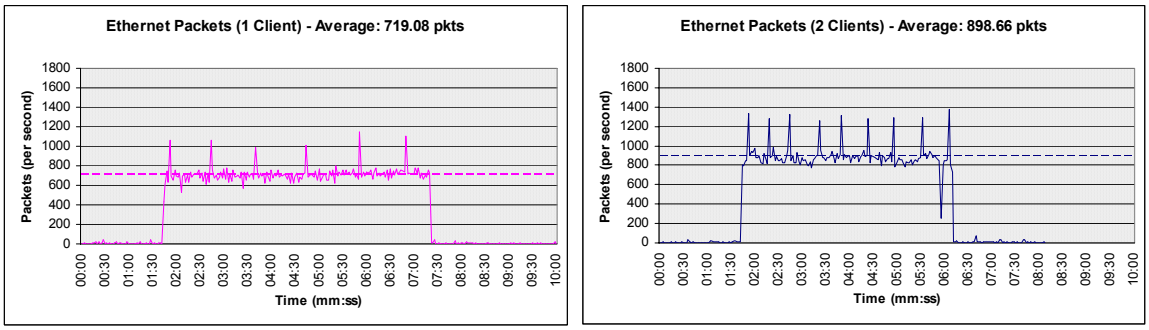

Figure 4.7 – Network Utilisation for the server with 1 and 2 clients connected

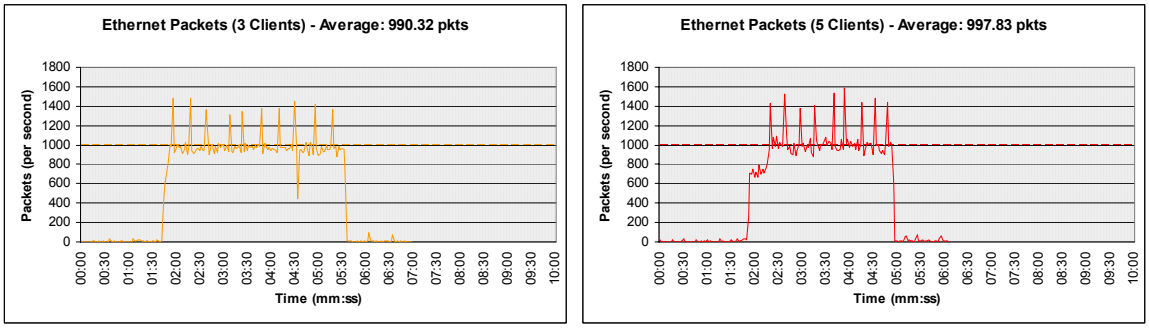

Figure 4.8 – Network Utilisation for the server with 3 and 5 clients connected

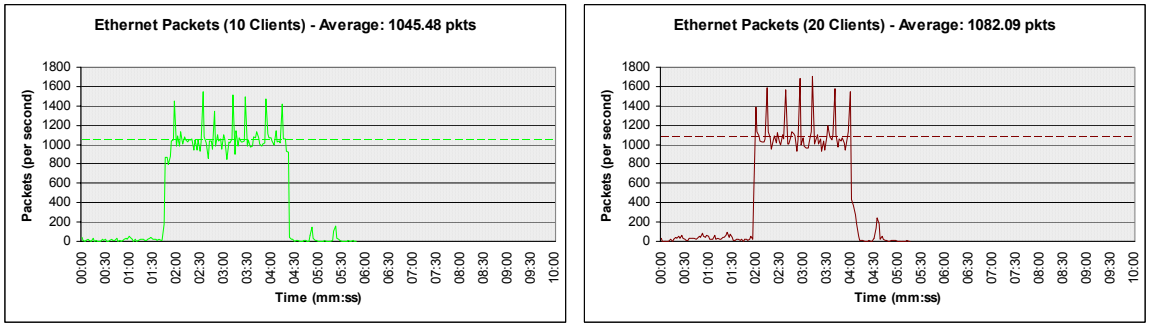

Figure 4.9 – Network Utilisation for the server with 10 and 20 clients connected

Figures 4.7 through 4.9 show that the servers network connection was also heavily utilised during the task-set processing. This is measured in the number of Ethernet packets being transmitted by the server and increases on average from 719.08 packets per second for a single client to 1082.09 packets per second for 20 clients with several peaks reaching in excess of 1,400 packets per second. While this places a heavy load on the network bandwidth of the server, it is not saturating the connection.

The results shown here prove that the server is placed under a heavy load as more clients connect to the distributed system. The CPU utilisation results especially show that the server is only just able to cope with the demand that 20 clients place on it. Adding further clients to the current server configuration would continue to follow the trend that has been observed, while still processing task-sets marginally faster the overall performance of the distributed system would decrease. Thus, it is feasible to suggest that if the client number continued to increase the performance would eventually flatten out or get worse, i.e. task-sets would take the same amount of time to process no matter how many clients were connected or it may begin to take longer for task-sets to be processed as the server is unable to cope with the load being placed on it.

It is arguable that with a more complicated problem and task-set the clients would take longer to process results and therefore reduce the load placed on the server, however the server load issue would still exist it just wouldn't be as apparent until large numbers of clients connected. Thus, there are two possible solutions to this problem, the first of which would be to run the servlets on a higher capacity server. This would increase the maximum number of clients which could connect to the system at any one time. Preliminarily testing was conducted of the distributed

system on a higher capacity server and this showed promising results. However, increasing the capacity of the server will only increase the performance of the distributed system so far. In a real world situation there may be thousands of clients connecting to the system at any one time, therefore again possibly overloading the server.

The second solution would be to introduce load balanced servers through the use of Enterprise Servlets and the *Java2 Enterprise Edition*. This introduces support for "large-scale web sites—high traffic, high-reliability sites that have extra demands for scalability, load balancing, and failover support" (Hunter & Crawford 2001). This would result in the distributed systems duties being distributed across multiple backend servers, therefore solving the issue of one server taking the entire load of the system.

4.3.2 Synchronisation

Initial testing of the distributed system highlighted a potential problem that when multiple clients were connected to the system at the same time, for example when posting processed results, there was an increased chance that the data being transferred could get mixed together. This is due to the fact that the servlets use global variables for storing task information, task results and client details. This problem is what is commonly described as a *race condition*, Allen Holub (1998) describes this as being "a situation whereby two threads simultaneously contend for the same object and, as a consequence, leave the object in an undefined state".

As mentioned in the previous chapter this problem was overcome using Java synchronisation, whereby the servlets which contain global variables are synchronised in such a way that allows only either one GET or POST request per servlet to occur at any one time, and any requests that occur when another is already in progress will simply be queued.

While this only has a limited effect on the performance of the server it is not an ideal solution. This is primarily because it introduces the notion of a "*caravan effect*" where clients begin to queue behind each other while they wait for the current client to complete it's communications with the server. In the case of the crossword problem, because the time required to process a task is approximately the same per

task it means that there is a possibility that rather then clients accessing the server at staggered intervals they will all attempt to access it at the same time. This is because they where all previously queued together, therefore again effecting the load and performance on the server.

A potential solution to this problem is the removal of all global variables in the servlets, and the integration with a shared access database system. This would potentially eliminate any need for synchronisation in the servlets as well as allow for far great control over the tasks, results, and client data which is stored in the database.

4.3.3 Security

As mentioned in Chapter 2 the issue of security is of large importance to the area of distributed computing. This was especially the case with the dynamic distributed system presented in this work, which uses Java applets as the client end of the distributed system. Java uses what is known as a sandbox to run Java applets on a local machine; this provides a secure area that allows an applet to run while not generally having access to the local machine and limited access to network resources.

As all communication between server and client occurs over the HTTP protocol it was possible to ignore the complexities of Java security policies and signed applets because the Java applet sandbox allows by default network connections to be made back to the originating server. Therefore because the servlets are run on the same server as the web server distributing the client applet, communication can occur between the server and client without any special consideration to modifying the default Java security policies.

Consideration to the security of the system was also given in allowing the task administrator control over which clients can take part in a distributed task. This was achieved by allowing clients to connect to the system, however approval for each client is required before it will receive a task from the server. Other areas of security such as encryption of the data being transferred, and confirming the integrity of the results returned from clients where not considered as they fall outside the scope of this work.

4.3.4 User Interfaces

During the development of the dynamic distributed system special consideration was given to the way in which users would interact with the system. Thus, graphical user interfaces where developed for each area where the task administrator would need to interact with the servlets and also a graphical front-end to the client applet. These interfaces will be briefly discussed in the following.

4.3.4.1 Distributed System Interface

The Distributed System interface is the front door to the distributed environment. It provides access to all server functions such as task, client, and results management, as well as access to the Java applet based client.

The interface is a HTML web page made up of three frames, with title banner across the top, links down the side and the main frame in the centre which is where all the above mentioned functions are displayed. The pages are served from the Apache web server.

Figure 4.10 - Distributed System Interface

4.3.4.2 Client Selection Interface

The Client Selection Interface is displayed in the main frame of the Distributed System Interface, and is a HTML web page automatically generated by a GET request to `CrossClientServlet`.

It provides the task administrator with access to a list of available clients which can then be selected to take part in a distributed task. The results of this selection are submitted to `CrossClientServlet` via a POST request.

Figure 4.11 – Client Selection Interface

4.3.4.3 Task Management Interface

The Task Management Interface is displayed in the main frame of the Distributed System Interface, and is a static HTML web page served by the Apache web server.

It allows the task administrator to add either a single or multiple tasks to the task database, as well as clear the contents of the database. The contents of the HTML form on this page are submitted via a POST request to `CrossClientDBServlet`.

Figure 4.12 – Task Management Interface

4.3.4.4 Java Applet Client Interface

The Java Applet Client Interface is the front end to the client based applet. It is served by the Apache web server and can be displayed by directly accessing a URL or via the Distributed System Interface.

It provides the user with easy access to connect to the distributed system as well as displaying the current client status and the results or task statistics of any previous tasks processed by this instance of the client.

Figure 4.13 – Java Applet Client interface

4.4 Summary

The results presented in this section show that using a dynamic distributed computing environment produces a significant performance increase in processing a distributed task compared to the results produced by a stand-alone workstation. It also shows that as more clients are added to the distributed system the performance of the overall system continues to increase but at reduced amounts.

Ideally the results should suggest linear speed-ups with the assumption that the time taken would eventually flatten out as the overall performance can only be improved to a certain point, this behaviour however was observed earlier than expected and further tests show that the server has become a bottle-neck to the overall distributed system.

Chapter 5 CONCLUSION AND FURTHER WORK

5.1 Summary

This thesis has presented the development of a dynamic distributed computing environment which has the primary aim of making use of under-utilised computers which are connected to a medium such as the Internet. This provides a client base that is a dynamically changing environment of computers which can be harnessed for a more useful task through distributed computing.

In conclusion, the results presented in this thesis show that the concept of a dynamic distributed computing environment is a feasible alternative to the traditional paradigm of fixed hosts commonly used in distributed computing. In particular, these results show that in a dynamic environment it is possible to achieve performance increases in processing a distributed task as more clients are added to the system. It is also suggested that through the use of a higher capacity server or load balanced servers the overall performance of the dynamic distributed environment will be further increased.

Further to this, the aim of this thesis was achieved through the use of a technology that is not usually associated with the area of distributed computing, Java Servlets. This provides a solid foundation for future research into the use of Java Servlets in not only distributed systems of this type but other more traditional distributed environments, for example database replication.

5.2 Further Work

The development of the dynamic distributed computing environment presented in this thesis is still in its early stages. There is much more work that can be carried out to improve both what the overall system provides and the performance it delivers. The areas of further work that have been identified during the development of this work are outlined below.

5.2.1 Testing

The results presented in this thesis were conducted in a controlled environment using a switched local area network (LAN), and therefore did not take into account the

factors of network congestion or different network mediums such as dial-up modem connections. Preliminary testing conducted over other mediums showed that the system is capable of operating in these environments; however further testing is required to gauge the performance.

As mentioned previously in section 4.3.1, preliminary testing indicated increased overall performance of the distributed environment when using a higher capacity server. Further testing is required in this area to firstly compare the performance with the results presented in this thesis, and secondly to benchmark a dynamic distributed computing environment against other similar distributed systems.

5.2.2 Task & Client Interactions

In the current implementation of the dynamic distributed computing environment using the distributed crossword problem all tasks are treated as individual units. Therefore, currently in order for one task to complete successfully it doesn't have to retrieve information from the results of previously completed tasks. Obviously in the case of a real world distributed crossword problem the results of previous tasks would give clues as to the known letters in other tasks.

Further investigation is therefore required in this area so that depending on the current distributed problem being solved there are interactions between the clients and also previous results. As an example if there are three tasks currently loaded in the system, *A*, *B*, and *C*, in order for task *C* to be completed successfully it has to wait for the results of task *A* and *B* to be returned before it can process the task and return results.

5.2.3 System Reliability

System reliability is of major importance in any distributed system, in particular in regards to fault tolerance and system redundancy, and this is an area of the dynamic distributed computing environment that needs further investigation. Currently the system provides a level of fault tolerance through checking to see if a task is uncompleted and if the client originally processing that task is no longer connected to the system. In that case the task is then reissued out to a new client for processing again. Improvements required to this model include making the server poll connected clients rather then relying on the client informing the server that it is

disconnecting, and also in the case that a client does disconnect part-way through processing a task it returns the results of its processing up to that point, rather then that processing being lost.

Task redundancy is also an area that needs further investigation, as currently each task is only issued to one client and the results returned are assumed to be correct. This can be improved through data checks to ensure that results returned are correct, for example in the expected format. Also tasks could be issued to multiple clients to increase the chance of successful results being returned to the server.

Finally, system redundancy issues require further investigation, as previously mentioned in regards to a higher capacity server, and the development of load balanced servers to process requests from clients.

5.2.4 Advanced Distributed Task

The final area of further work identified in this thesis is the development of an advanced distributed task. A real world dynamic distributed computing environment such as the one presented in this work would be an expensive and complex system to establish. Therefore in order to make the system viable it is necessary to develop a distributed task which justifies the expensive of the distributed system.

The current crossword distributed problem is an example problem that provided a good base to test the distributed environment on. However in reality this problem would not justify this type of distributed system, and could simply be processed by a single stand-alone workstation or server. Rather this type of system would be optimal for similar problems to those processed by the *Distributed.net RC5* project and *SETI@home* project discussed earlier.

LIST OF REFERENCES

Ahuja, S. & Quintao, R. 2000, 'Performance Evaluation of Java RMI: A Distributed Object Architecture for Internet Based Applications', *Proceeding of the Eighth International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, 29 Aug.-1 Sept., pp. 565-569

Brummitt, B., et al. 2000, 'Ubiquitous computing and the role of geometry', *IEEE Personal Communications*, vol. 7, no. 5, pp. 41-43.

Camiel, N., London, S., Nisan, N. & Regev, O., "*The POPCORN Project -- An Interim Report Distributed Computation over the Internet in Java*", site viewed 31/10/2002
URL - http://www.scope.gmd.de/info/www6/posters/721/poster.html

Dantas, M.A.R., Lopes, J.G.R.C. & Ramos, T.G. 2002, 'An enhanced scheduling approach in a distributed parallel environment using mobile agents', *High Performance Computing Systems and Applications, 2002. Proceedings. 16th Annual International Symposium on*, pp. 177- 181

Dermoudy, J. R., 2002, "*Effective Runtime Management of Parallelism in a Functional Programming Context*", PhD Thesis, School of Computing, University of Tasmania

Farley, J. 1998, *Java Distributed Computing*, O'Reilly & Associates, Inc., Sebastopol

Flanagan, D. 1999, *Java in a Nutshell*, Third Edition, O'Reilly & Associates, Inc., Sebastopol

Holub, A., 1998, "*Programming Java threads in the real world*", site viewed 20/10/2002
URL - http://www.javaworld.com/javaworld/jw-10-1998/jw-10-toolbox_p.html

Hunter, J & Crawford, W 1998, *Java Servlet Programming*, O'Reilly & Associates, Inc., Sebastopol

Hunter, J. & Crawford, W. 2001, *Java Servlet Programming*, Second Edition, O'Reilly & Associates, Inc., Sebastopol

Kiniry, J. & Zimmerman, D. 1997, 'A Hands-On Look at Java Mobile Agents', *Internet Computing*, IEEE, Vol.1, Iss.4, pp. 21- 30

Korpela, E., et al. 2001, 'SETI@home—Massively Distributed Computing for SETI', *Computing in Science & Engineering*, vol. 3, no. 1, pp. 78-83.

Lin, H., Wang, Y., Wang, C. & Chen, C., 2001, 'Web-based distributed topology discovery of IP networks', *Information Networking, 2001. Proceedings. 15th International Conference on*, pp. 857- 862

McGraw, G. & Felten, E. 1999, *Securing Java*, John Wiley & Sons, Inc. Canada

OMG, *The Common Object Request Broker: Architecture and Specification*, revision 3.0, OMG Document formal/02-06-02, Object Management Group, Franingham, MA, July 2002

Orfali, R. & Harkey, D. 1998, Client*/Server Programming with JAVA and CORBA*, John Wiley & Sons, Canada

Schaaf, M. & Maurer, F. 2001, 'Intergrating Java and CORBA: A Programmer's Perspective', *IEEE Internet Computing*, Janurary/Feburary Edition, pp. 72-78.

Sun Microsystems, "*All about Sockets*", site viewed 15/5/2002, URL – http://java.sun.com/docs/books/tutorial/networking/sockets/index.html

Tanenbaum, A. & Steen, M. 2002, *Distributed Systems Principles and Paradigms*, Prentice Hall, New Jersey.

Waldo, J. 1998, 'Remote procedure calls and Java Remote Method Invocation', *IEEE Concurrency*, vol. 6, no. 3, pp. 5-7, July 1998.

Wang, S. & Wang, W., "*Jaguar: A distributed computing environment based on Java*", site viewed 31/10/2002
URL - http://parallel.iis.sinica.edu.tw/cthpc/7th.html

"*Web Browser Architecture*", posted 10/2/2001, site viewed 15/5/2002,
URL – http://wwwtech.comp.polyu.edu.hk/int_comp/chap_4/ch4_3.htm

Wittie, L.D. 1991, 'Computer networks and distributed systems', *Computer*, vol. 24, no. 9, pp. 67-76.

Yan, L. & Chen, C. 1999, 'JAM: High Performance Internet Computing with Massive Java Applets', *Electronic Commerce and Web-based Applications/Middleware, 1999 Proceedings, 19th IEEE International Conference on Distributed Computing Systems*, Austin, TX, USA, pp. 3-8.

APPENDIX A

JAVA DISTRIBUTED NET SERVER

Source Code Listing:

CrossClientServlet

CrossClientDBServlet

CrossResultsServlet

CrossTaskDBServlet

APPENDIX B

JAVA DISTRIBUTED NET CLIENT

Source Code Listing:

CrosswordSolver

GuessWord

APPENDIX C CROSSWORD TASK-SET

Task ID 0: ????a??a (9 results)
Task ID 1: ??i??c? (13 results)
Task ID 2: a??????i? (40 results)
Task ID 3: ?a??????c (19 results)
Task ID 4: ???d?w (4 results)
Task ID 5: ?at (12 results)
Task ID 6: p???e (41 result)
Task ID 7: a?ra????ne?? (1 result)
Task ID 8: ?cad??i?a?l? (1 result)
Task ID 9: ?????room (12 results)
Task ID 10: b??ch??? (16 results)
Task ID 11: o??i???i?? (5 results)
Task ID 12: p???ed (92 results)
Task ID 13: mi??ion???? (1 result)
Task ID 14: ?l???h (19 results)
Task ID 15: ?r???le?? (28 results)
Task ID 16: z??? (15 results)
Task ID 17: re????ed (124 results)
Task ID 18: ???put?? (6 results)
Task ID 19: s??p (16 results)
Task ID 20: ???pac? (1 result)
Task ID 21: ???it?r (12 results)
Task ID 22: ??????s??y (65 results)
Task ID 23: ?????ward (6 results)
Task ID 24: ??eak?? (11 result)
Task ID 25: ??ai? (28 results)
Task ID 26: ???ice (14 results)
Task ID 27: ??gh? (14 results)
Task ID 28: k??????d (11 result)
Task ID 29: ??????mm??g (2 results)
Task ID 30: ?a?a (12 results)
Task ID 31: ?i?? (341 result)
Task ID 32: m?? (29 results)
Task ID 33: ??m?? (170 results)
Task ID 34: s?? (48 results)
Task ID 35: ???work??? (3 results)
Task ID 36: ??art???? (21 result)
Task ID 37: ?n?c?s (4 results)
Task ID 38: long????? (8 results)
Task ID 39: ????rou?ly (12 results)
Task ID 40: t???k???r (2 results)
Task ID 41: e??m (1 result)
Task ID 42: ??hib??i?? (4 results)
Task ID 43: pi??a??e (3 results)
Task ID 44: ????z? (29 results)
Task ID 45: sn??ze (2 results)
Task ID 46: ???p???t (50 results)
Task ID 47: un?????w?r??? (1 result)
Task ID 48: u??er???? (53 results)
Task ID 49: ????i??t? (129 results)
Task ID 50: ???o???e (84 results)
Task ID 51: ???n??h (8 results)
Task ID 52: zu??h??i (1 result)
Task ID 53: ab???ma? (1 result)
Task ID 54: ??sor??i?? (4 results)
Task ID 55: a??e??o?y (1 result)
Task ID 56: ????done (1 result)
Task ID 57: over???? (87 results)
Task ID 58: ox???n (1 result)
Task ID 59: p?p?r?a?k (1 result)
Task ID 60: ??p?????y (28 results)
Task ID 61: ????a??a (9 results)
Task ID 62: ??i??c? (13 results)
Task ID 63: a??????i? (40 results)
Task ID 64: ?a??????c (19 results)
Task ID 65: ???d?w (4 results)
Task ID 66: ?at (12 results)
Task ID 67: p???e (41 result)
Task ID 68: a?ra????ne?? (1 result)
Task ID 69: ?cad??i?a?l? (1 result)
Task ID 70: ?????room (12 results)
Task ID 71: b??ch??? (16 results)
Task ID 72: o??i???i?? (5 results)
Task ID 73: p???ed (92 results)
Task ID 74: mi??ion???? (1 result)
Task ID 75: ?l???h (19 results)
Task ID 76: ?r???le?? (28 results)
Task ID 77: z??? (15 results)
Task ID 78: re????ed (124 results)
Task ID 79: ???put?? (6 results)
Task ID 80: s??p (16 results)
Task ID 81: ???pac? (1 result)
Task ID 82: ???it?r (12 results)
Task ID 83: ??????s??y (65 results)
Task ID 84: ?????ward (6 results)
Task ID 85: ??eak?? (11 result)
Task ID 86: ??ai? (28 results)
Task ID 87: ???ice (14 results)
Task ID 88: ??gh? (14 results)
Task ID 89: k??????d (11 result)
Task ID 90: ??????mm??g (2 results)
Task ID 91: ?a?a (12 results)
Task ID 92: ?i?? (341 result)
Task ID 93: m?? (29 results)
Task ID 94: ??m?? (170 results)
Task ID 95: s?? (48 results)
Task ID 96: ???work??? (3 results)
Task ID 97: ??art???? (21 result)
Task ID 98: ?n?c?s (4 results)
Task ID 99: long????? (8 results)
Task ID 100: ????rou?ly (12 results)
Task ID 101: t???k???r (2 results)
Task ID 102: e??m (1 result)
Task ID 103: ??hib??i?? (4 results)
Task ID 104: pi??a??e (3 results)
Task ID 105: ????z? (29 results)
Task ID 106: sn??ze (2 results)
Task ID 107: ???p???t (50 results)
Task ID 108: un?????w?r??? (1 result)
Task ID 109: u??er???? (53 results)
Task ID 110: ????i??t? (129 results)
Task ID 111: ???o???e (84 results)
Task ID 112: ???n??h (8 results)
Task ID 113: zu??h??i (1 result)
Task ID 114: ab???ma? (1 result)
Task ID 115: ??sor??i?? (4 results)
Task ID 116: a??e??o?y (1 result)
Task ID 117: ????done (1 result)
Task ID 118: over???? (87 results)
Task ID 119: ox???n (1 result)
Task ID 120: p?p?r?a?k (1 result)
Task ID 121: ??p?????y (28 results)
Task ID 122: ????a??a (9 results)
Task ID 123: ??i??c? (13 results)
Task ID 124: a??????i? (40 results)
Task ID 125: ?a??????c (19 results)

Continued…

...continued from previous

```
Task ID 126: ???d?w (4 results)
Task ID 127: ?at (12 results)
Task ID 128: p???e (41 result)
Task ID 129: a?ra????ne?? (1 result)
Task ID 130: ?cad??i?a?l? (1 result)
Task ID 131: ?????room (12 results)
Task ID 132: b??ch??? (16 results)
Task ID 133: o??i???i?? (5 results)
Task ID 134: p???ed (92 results)
Task ID 135: mi??ion???? (1 result)
Task ID 136: ?l???h (19 results)
Task ID 137: ?r???le?? (28 results)
Task ID 138: z??? (15 results)
Task ID 139: re????ed (124 results)
Task ID 140: ???put?? (6 results)
Task ID 141: s??p (16 results)
Task ID 142: ???pac? (1 result)
Task ID 143: ???it?r (12 results)
Task ID 144: ??????s??y (65 results)
Task ID 145: ?????ward (6 results)
Task ID 146: ??eak?? (11 result)
Task ID 147: ??ai? (28 results)
Task ID 148: ???ice (14 results)
Task ID 149: ??gh? (14 results)
Task ID 150: k??????d (11 result)
Task ID 151: ??????mm??g (2 results)
Task ID 152: ?a?a (12 results)
Task ID 153: ?i?? (341 result)
Task ID 154: m?? (29 results)
Task ID 155: ??m?? (170 results)
Task ID 156: s?? (48 results)
Task ID 157: ???work??? (3 results)
Task ID 158: ??art???? (21 result)
Task ID 159: ?n?c?s (4 results)
Task ID 160: long????? (8 results)
Task ID 161: ????rou?ly (12 results)
Task ID 162: t???k???r (2 results)
Task ID 163: e??m (1 result)
Task ID 164: ??hib??i?? (4 results)
Task ID 165: pi??a??e (3 results)
Task ID 166: ????z? (29 results)
Task ID 167: sn??ze (2 results)
Task ID 168: ???p???t (50 results)
Task ID 169: un?????w?r??? (1 result)
Task ID 170: u??er???? (53 results)
Task ID 171: ????i??t? (129 results)
Task ID 172: ???o???e (84 results)
Task ID 173: ???n??h (8 results)
Task ID 174: zu??h??i (1 result)
Task ID 175: ab???ma? (1 result)
Task ID 176: ??sor??i?? (4 results)
Task ID 177: a??e??o?y (1 result)
Task ID 178: ????done (1 result)
Task ID 179: over???? (87 results)
Task ID 180: ox???n (1 result)
Task ID 181: p?p?r?a?k (1 result)
Task ID 182: ??p?????y (28 results)
Task ID 183: ????a??a (9 results)
Task ID 184: ??i??c? (13 results)
Task ID 185: a??????i? (40 results)
Task ID 186: ?a??????c (19 results)
Task ID 187: ???d?w (4 results)
Task ID 188: ?at (12 results)
Task ID 189: p???e (41 result)
Task ID 190: a?ra????ne?? (1 result)
Task ID 191: ?cad??i?a?l? (1 result)
Task ID 192: ?????room (12 results)
Task ID 193: b??ch??? (16 results)
Task ID 194: o??i???i?? (5 results)
Task ID 195: p???ed (92 results)
Task ID 196: mi??ion???? (1 result)
Task ID 197: ?l???h (19 results)
Task ID 198: ?r???le?? (28 results)
Task ID 199: z??? (15 results)
Task ID 200: re????ed (124 results)
Task ID 201: ???put?? (6 results)
Task ID 202: s??p (16 results)
Task ID 203: ???pac? (1 result)
Task ID 204: ???it?r (12 results)
Task ID 205: ??????s??y (65 results)
Task ID 206: ?????ward (6 results)
Task ID 207: ??eak?? (11 result)
Task ID 208: ??ai? (28 results)
Task ID 209: ???ice (14 results)
Task ID 210: ??gh? (14 results)
Task ID 211: k??????d (11 result)
Task ID 212: ??????mm??g (2 results)
Task ID 213: ?a?a (12 results)
Task ID 214: ?i?? (341 result)
Task ID 215: m?? (29 results)
Task ID 216: ??m?? (170 results)
Task ID 217: s?? (48 results)
Task ID 218: ???work??? (3 results)
Task ID 219: ??art???? (21 result)
Task ID 220: ?n?c?s (4 results)
Task ID 221: long????? (8 results)
Task ID 222: ????rou?ly (12 results)
Task ID 223: t???k???r (2 results)
Task ID 224: e??m (1 result)
Task ID 225: ??hib??i?? (4 results)
Task ID 226: pi??a??e (3 results)
Task ID 227: ????z? (29 results)
Task ID 228: sn??ze (2 results)
Task ID 229: ???p???t (50 results)
Task ID 230: un?????w?r??? (1 result)
Task ID 231: u??er???? (53 results)
Task ID 232: ????i??t? (129 results)
Task ID 233: ???o???e (84 results)
Task ID 234: ???n??h (8 results)
Task ID 235: zu??h??i (1 result)
Task ID 236: ab???ma? (1 result)
Task ID 237: ??sor??i?? (4 results)
Task ID 238: a??e??o?y (1 result)
Task ID 239: ????done (1 result)
Task ID 240: over???? (87 results)
Task ID 241: ox???n (1 result)
Task ID 242: p?p?r?a?k (1 result)
Task ID 243: ??p?????y (28 results)
Task ID 244: ????a??a (9 results)
Task ID 245: ??i??c? (13 results)
Task ID 246: a??????i? (40 results)
Task ID 247: ?a??????c (19 results)
Task ID 248: ???d?w (4 results)
Task ID 249: ?at (12 results)
```