Proceedings of the IASTED International Conference
PARALLEL AND DISTRIBUTED COMPUTING AND NETWORKS
February 17-19, 2004, Innsbruck, Austria

NETWORK OF BROWSERS – A MULTI-PROCESSOR COMPUTER

Luke Fletcher and Vishv Malhotra
School of Computing, Private Box 100
University of Tasmania, Hobart Tas 7001 Australia
{luke.fletcher, vishv.malhotra} @utas.edu.au

Abstract

The paper describes an experimental system in which we linked together a number of computers over the Internet to form a multi-processor computer system. The arrangeme nt uses Java-enabled web-browsers as the tool for adding available computers in the multi-processor system. The number of processors in this system can grow and shrink dynamically as the computers join or leave the system.

Key Words

Multi-processor computer, Internet, Java, Web server, HTTP.

1 Introduction

As computational power becomes cheaper and more pervasive, we continue to seek even more computational power at even lower costs for a variety of new and innovative applications ranging from important economical, scientific and life-saving medical experiments to recreational computation [1]. Notwithstanding the unfulfilled demand for computation, at any point in time we have hundreds of thousands of computers around the world that are under-utilised. A typical computer is used for only a small fraction of the time duration over which it could be used.

The under-utilisation of the available computational power results from many reasons. Foremost of these reasons is our inability to locate free computers. Security concerns are also significant deterrence in this regard. Owners of the computers are reluctant to hand controls of their computers over to strangers who need these computational resources. At the same time clients/users of these facilities, if they can lease them, need to feel secure before they transfer their sensitive data and programs to a computer environment that an anonymous benefactor may have made available.

The paper describes a network of browsers (NoB) that combines the available free computers into a huge multi-processor computer system. In deed, it allows the individual computers to join the multi-processor computer system when they are available. At the same time when the owners of these computers want to use their computers for their own work, they withdraw their computers from the arrangement. Thus, the multi-processor computer system continues to grow and shrink as free computers join in to become its processing units and the busy computers leave and cease to be its processing units. The owners of the participating computers decide if, when and for how long they donate time on their computers to the multi-processor system.

The network of browsers (NoB) uses the ubiquitous internet technology – the humble web browser – to match the demand for computation power against the available computational resources. Further, the existing and future security measures in the Internet domain provide and will continue to improve and enhance the trust environment for those who make available their computational resources to the others. At the same time, those who make use of these resources will be assured about the protection of their data and software.

An experimental prototype of a network of browsers (NoB) was developed in the School of Computing, University of Tasmania. The paper describes the system and presents initial experiences and analysis from this experiment in running a simple concurrent program to solve a crossword problem.

In section 2, we briefly summarise the aspects of Internet technology that have bearing on the network of browsers (NoB) developed in this university. Section 3 describes the system. Section 4 provides a brief description of the crossword problem that we used to test the system. Experimental performance results are also presented in section 4. We conclude the paper in section 5 by making some concluding remarks and listing the directions we wish to work on to make the system more usable.

2 Background: Web Browsers, Web Servers and Java

The Internet is comprised of three kinds of components: browsers, servers and the network.

The network provides the communication media that allows any browser to interact with any server. The predominant protocol for interaction between a browser and server is through Hypertext Transfer Protocol (http) [10]. The protocol is memory-less and comprises of two steps. A browser sends a request for an html document to a specific server. The server responds to the request by sending the requested document. The response concludes the interaction and the browser and the server are free for their next intersection with the other players on the World-wide Web (WWW).

The browsers are usually enhanced, with a Java plug-in, to run Java byte-codes called applets [2, 3, 4]. The plug-in executes Java applets received from the servers. At the same time, modern servers too can be extended by means of Java servlets and other server-side extensions, for example, CGI. A servlet is Java byte-code running alongside a server. The server passes certain http requests to a designated servlet when it receives them from a browser. Thus, these requests are handled by a specially tailored program (servlet). The servlet-browser interaction may use a protocol that maintains state over a sequence of request-response interactions (transaction) between them.

Another ingredient of interest in the construction of the networks of the browsers (NoB) is the Java programming language and associated Java virtual machine. Java programs are translated into Java byte-code that can be run on interpreters called Java Virtual Machines (JVM). This enables the java programs to run independent of the underlying hardware and virtually on any computer. Java plug-in for a browser is a JVM capable of running with the browser. Thus a Java enabled browser can run an applet when it receives it from a server in response to a request. Traditionally, these applets are intended to provide sophisticated and powerful application-specific interactive interfaces to the browser users.

In addition to the three components that provide obvious functionality to our system, we are also interested in the Java2 security model [5]. It provides fine-grained security guarantees for distributed applications in an environment where objects and code migrate over the network. Though the model is well integrated into the Standard Java Development Kit (SDK), we are not yet aware of a plug-in that implements the fine-grained security model of Java2. For the present we can only rely on traditional sand-pit security model and its extension to signed applets. In this model the imported code (applets) run in the browser host with a small set of rights considered safe. Browsers such as Internet Explorer and Netscape allow for the users to vary some rights of the external code to suit their needs. As opposed to this scheme, also known as Java 1.0 security model, Java 1.1 uses a model in which appropriately signed applets are fully trusted. We are confident that future developments in Java will implement Java2 security model. Java2 security model allows for rights of the classes to be defined independent of other classes. A fine grained security policy will be useful to individually set the rights of the applets and also of the other objects on the applets. Thus, enabling the network of browsers (NoB) to be more secure and providing greater confidence and security assurances to those loaning their computers as processing units of the multi-processor computer system and to those using it to run their applications.

3 Network of Browsers (NoB)

A prototype network of browsers (NoB) was built as an honours project [6] and preliminary performance results were obtained using a simple crossword problem. As shown in Figure 1, the three main actors in the prototype system are:

i. *Client*: A client is a user (or a computer) who is interested in execution of a suitably devised program on the prototypical network of browsers (NoB) computer.

ii. *Server*: The server is a well advertised site that acts as a broker between the clients and the donors of computer times.

iii. *Donors*: The donors are owners of computers on the internet who volunteer to make their computers available for executing clients' programs. In what follows we often use the word donor to mean the donor's computer.

A client contacts a broking server to load their program components, called tasks, on the server system. The client also provides data and establishes execution dependencies between the tasks. A sever receives the tasks from a client and then waits for donor computers to contact it to volunteer to execute the tasks.

A donor volunteers to execute tasks by starting a Java-enabled web-browser and accessing a specified web page on the server. The server returns an html page with suitably coded applet to the browser. All further interaction with the server for executing the tasks on the browser's Java plug-in is a responsibility of this applet.

A donor will be able to reclaim the computer by simply closing the browser. The same effect is achieved if the browser is moved to a different web page as it stops the controlling applet.

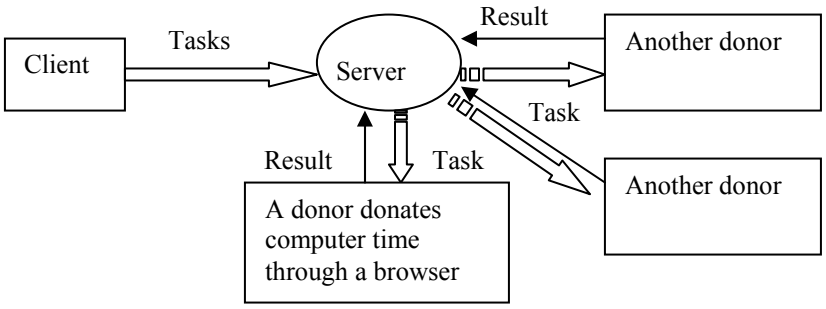


Figure 1: The web server acts as a broker to distribute computation tasks to Java enabled web-browsers.

As is clear from Figure 1, the computation intensive tasks are passed on to the donors' computers where they are executed as applet objects in the Java-enabled browsers. However, the task coordination and execution dependencies are handled by the server. The prototype server interacts with other components of the system exclusively through http methods GET and PUT. The primary responsibilities of the server are:

i. *To receive tasks and inter-task dependency information from the clients wishing to use the system for their computations*: Each task is assigned a unique identifier (TID) to track it. In our prototype system, we have not yet implemented an interface for this purpose. In our current implementation the tasks and their dependencies are loaded manually.

ii. *To receive computer time donations from the volunteering donors:* A donated computer runs a web-browser and loads a pre-specified html page carrying coordination applet to joint the multi-processor system. The server adds the IP address of the donated computer to its database of the current donors. The information will be used to track the completion of tasks sent to the donor for execution.

iii. *To distribute tasks to the donors for execution:* When a donor computer requests a task for execution (GET), a task is located in the available task database and sent to the browser for execution. The server tracks the assigned task, by storing the IP address of the computer executing the task and the task unique identifier in its allocated tasks database.

iv. On completion of a task, the result is sent (PUT) back to the server. The result returned, to the server, by a completed task is recorded against the task number. The allocated tasks database is also updated.

v. An important task that the server needs to perform is to track tasks lost during execution. The tasks may be lost due to errors in transmission and machine failures. A donor withdrawing their computer while executing a task also causes the task to be lost during execution. The recovery procedure used is described in section *3.1 Recovering the Lost Tasks*.

The server has been implemented using a set of four cooperating servlets. All requests for new tasks to execute are received by servlet `CrossClientServlet`. This servlet is also responsible for sending the tasks to the browsers for execution. A number of tasks requests may be made concurrently from different donors. To achieve good response the servlet passes the requests to background servlets for processing. All inter-servlet communication is performed using http methods GET and PUT.

The task database is maintained by servlet `CrossTaskDBServlet`. The servlet maintains the status of each task as *unallocated*, *allocated*, and as *completed*. When requested by the servlet `CrossClientServlet`, it finds and returns the next task to be executed.

Servlet `CrossClientDBServlet` maintains the status of donor computers currently participating in the Network of Browsers (NoB).

Finally, servlet `CrossResultsServlet` records the completion of the tasks by recording their results.

3.1 Recovering the Lost Tasks

A program is executed when all its tasks have executed. As the processing units (donors' computers) of the Network of the Browsers (NoB) can join and leave the system asynchronously, there shall be allocated tasks that are lost as the computers running them are withdrawn. To

counter this problem, servlet `CrossTaskDBServlet` uses a simple strategy to find the next task for execution. First it tries to locate an *unallocated* but not *completed* task for execution on the requesting browser. If it does not have an *unallocated* task, it sends a task already *allocated* to a browser for execution on the least recently allocated basis. The motivation for this policy is to use the available processing time to run a task that has either been lost or allocated to a slow computer.

As the tasks may be re-allocated without verifying that they have actually been lost, certain tasks may send in multiple successful completions. The servlet `CrossResultsServlet` silently ignores additional result(s) after it has recorded the first result against a task identifier (TID).

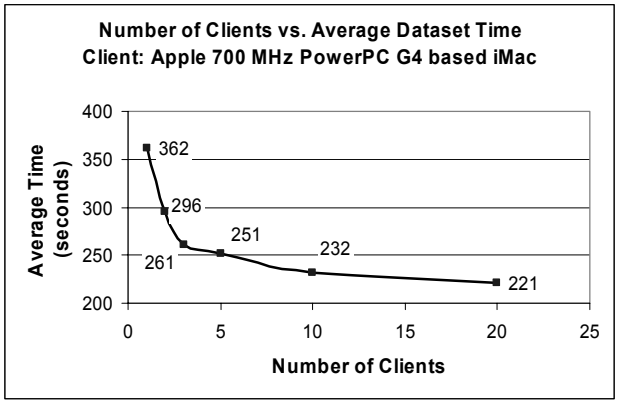
If neither an *unallocated* task nor an *allocated* task remains in the task database a wait request is sent back to the browser applet. The applet will try again after 30 seconds.

4 Feasibility and Performance

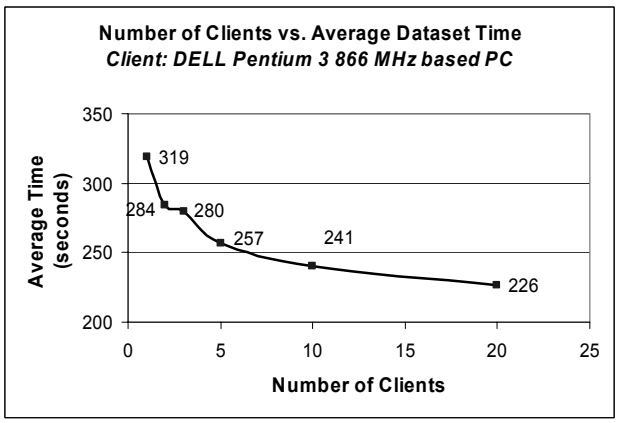
A prototype was built as described in the previous section. To test the system's effectiveness and performance, we defined a version of crossword puzzle solver. This test program was chosen as it is easily split into parallelisable tasks – each clue can be passes as a parallel task. Thus, the lowest level task is the following problem: given an array of characters, with some characters in the array already known, match the array against the words in a dictionary. The task returns a collection of words that can complete the array. By choosing the suitable length of the array, known characters in the array and number of words in the dictionary one can tailor the computational demands of a task to suit the availability of resources during the experimental runs of the Network of Browsers (NoB).

To obtain performance results for the system, we used two test platforms of donor computers. These donor computers consisted firstly of a number of Apple 700 MHz PowerPC G4 based iMacs running MacOS X 10.1, and secondly PC based DELL Pentium 3 866 MHz computers running Microsoft Windows 2000. The browser used on these test systems was Microsoft Internet Explorer 5.2 on the iMac based systems and Netscape 6.2 on the PC based systems. All donor systems were using the Java2 1.3.1 runtime environment used for running the distributed client applet, and had 256MB of RAM installed. The test components used were simply a small collection of arrays of words from a dictionary of approximately 81,000 different English words. These arrays were repeated several times to obtain measurable time durations for the system.

The server side of the system was based on an Apache Tomcat [8] server implemented on a Sun SPARCstation 5 with 128MB RAM and running the Sun Solaris 9 operating system which includes the Java2 1.4 SDK. The server and the client computers were connected by a switched 100 Mbps local area network (LAN) within the School of Computing.



(a) iMac based donors



(b) PC based donors

Figure 2: Graphs show the average execution time for a task set over 1 to 20 donor computers.

For this problem comprising of a collection of similar sized parallel tasks, Figure 2 shows the average execution time for a task set. The system showed nearly linear speed up to 3 computers – from average time for the iMac based donors of 362 seconds to complete the tasks using one donor to 261 seconds with 3 donors, and from 319 to 280 for the PC based donors. Thereafter there was only a marginal improvement to about 221 seconds (iMac) and 226 seconds (PC) with 20 donors. The server side network traffic also showed corresponding saturation of number of packets per second.

To test the system, in an environment free from server side saturation, we artificially varied the perceived task

size by artificially modifying the execution speed of these parallel tasks. We were able to extend the linearity region to a larger number of computers in the network of browsers (NoB). Figure 3 presents a graph for one such case.

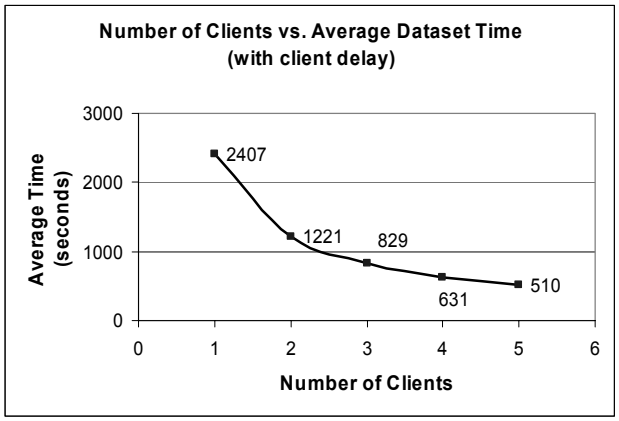


Figure 3: Graph showing average execution time for a task set executing over 1 to 5 iMac based donors after artificially modifying the execution speed

One of the first, albeit an obvious, lessons learned from the prototype is that the server side processing can easily become a bottleneck as demonstrated by the server CPU utilisation shown in Figure 4. If each task presents a significant computational load, the demand on the server from frequent requests will be alleviated. An environment in which each task is a significant computational load with diverse time requirements is more common. The network of browsers (NoB) would be expected to return even better performance improvements for such mix of tasks.

To formalise these ideas, consider an idealised problem which can be cast into k equal sized sub-tasks. We further assume a pool of homogeneous donor computers. Let,

C Computation time needed on a donor computer to run the problem

s Server time needed to send a task to a donor and to record the result when returned.

c Set up and wind up time on the donor computer to run a task

d Number of participating donor computers

k Number of sub-tasks created for the computation problem

a Ambient load on the server as fraction of CPU time.

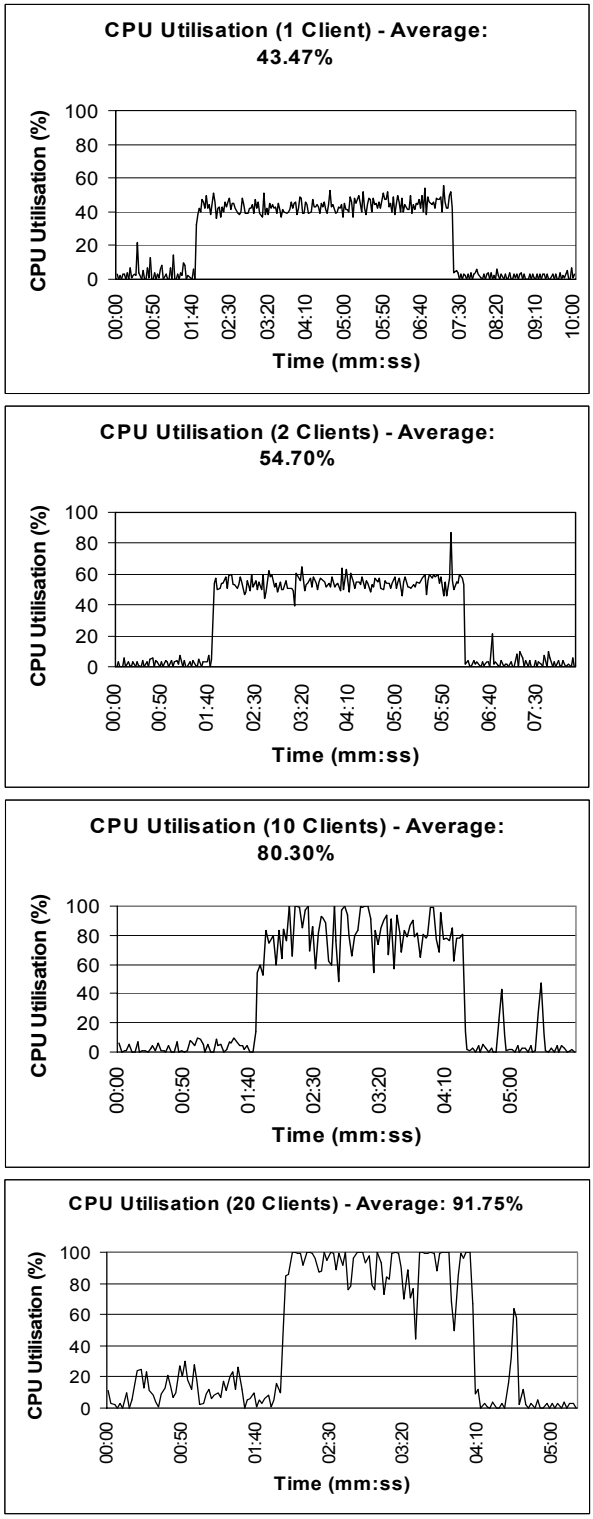


Figure 4: Server CPU Utilisation for the server with 1 (top), 2, 10 and 20 (bottom) donors connected to it. Server load is comprised of load due to computational problem and the ambient server load.

The speed up, σ, shall be constrained by the following limit:

$$\sigma \le \frac{C}{\max\left\{\frac{s\,k}{1-a}+\frac{C}{k}+c, \left\lceil\frac{k}{d}\right\rceil\left(\frac{C}{k}+c+\frac{s}{1-a}\right)\right\}}$$

The first term in the argument of function `max` caters for the case where the server is loaded heavily and is the limiting resource. The second term caters for the case where the computation is donor bound.

We have not yet tried configuring the server to ease the server-side processing bottleneck that occurs when a barrage of browser requests is received over a short period. However, it is expected that this could be done using standard techniques such as by having a pool of higher capacity servers available to receive requests from the donor computers.

5 Conclusions and Further Work

The prototype has shown that it is indeed possible to harness the capability of unused computers on a network through fairly simple development on the Internet. Superior performances should be possible with well-known strategies for handling server-side saturations. However, building a robust system that is able to keep a number of computers on the network of browsers (NoB) active simultaneously is an important task that we hope to explore soon.

The servlets can run multi-threaded computation. However, at this stage we have implemented the prototype using uni-threaded servlets. The approach avoids many synchronisation issues. Multiple threads of executions in each servlet will significantly improve the response of the servlet but would require more elaborate strategies to synchronise the use of shared resources.

Though we have implemented a rudimentary recover strategy for handling problems that occur as the computers are withdrawn from the network of browsers (NoB) by their owner or network problems, yet other causes of failures need further consideration.

A suitable interface for submitting programs for execution is also an important item that needs early attention. Without such an interface it is not easy to submit tasks for execution on this supercomputer.

It is worth noting that a number of similar efforts are accessible on the web [1, 11, 12]. However, these efforts are designed to solve high impact, albeit, computation intensive problems through sever side design tailored to the problem. Typically, the donors need to download code that runs on their computers. Philanthropy motivates the donors to overcome their concerns for computer security to let the downloaded software run on their machines. We seek a system capable of receiving programs from "ordinary clients" for execution; and, seek to overcome security concerns of the computer donors by restricting the downloaded code to the donor trusted browsers.

References

[1] Search for Extraterrestrial Intelligence SETI@home, http://setiathome.ssl.berkeley.edu/ (accessed on 27 November 2003)

[2] J. Gosling, B. Joy, G. Steele and G. Bracha, *The Java™ Language Specification*, Addison-Wesley, Boston, 2000.

[3] M. Campione, K. Walrath, The Java™ Tutorial Second Edition: Object-Oriented Programming for the Internet, Addison-Wesley, Reading, MA 1998.

[4] M. Campione, K. Walrath, A. Huml and the Tutorial Team, *The Java Tutorial Continued: The rest of JDK™*, Addison-Wesley, Reading, Ma 1998

[5] Li. Gong, G. Ellison and M Dagwforde, Inside JAVA 2 Platform Security: Architecture, API Design and Implementation, Second Edition, Addison-Wesley, 2003

[6] L. Fletcher, *A Dynamic Networked Browser Environment for Distributed Computing*, Honours thesis, School of Computing, University of Tasmania, 2002.

[7] D. Lea, Concurrent Programming in Java™: Design Principles and Patterns, second edition, Addison-Wesley, Boston, 2000.

[8] The Apache Jakarta Project, (accessed on 27 November 2003) http://jakarta.apache.org/tomcat/

[9] The Berkeley NOW Project, (accessed on 27 November 2003) http://now.cs.berkeley.edu/

[10] HTTP: Hypertext Transfer Protocol, (accessed on 27 November 2003) http://www.w3.org/Protocols/

[11] Grid.org (accessed on 27 November 2003) http://www.grid.org

[12] Fight AIDS @ home project (accessed on 27 Novemeber 2003), http://fightaidsathome.scripps.edu