GridWork

Grid Computing focused on High Throughput Computing using JavaScript-based Web client job execution environments

Joanne Marston
UCT Computer Science student
Department of Computer Science,
University of Cape Town,
Rondebosch, Cape Town, 7701
Mrsjoa005@myuct.ac.za

ABSTRACT

Grid computing has the potential to offer a less expensive and faster alternative to high performance computing. This report serves to determine whether grid computing and specifically high throughput computing using JavaScript-based Web client job execution environments is a viable option. It describes the process of setting up a high throughput computing system using a client-server model for processing simple XML files. The process of implementing the system and an evaluation of the system are discussed. The report concludes that high throughput computing is a viable option that is fairly easy to implement. At a university level it could be used to process generic administrative forms.

Categories and Subject Descriptors

J.1 [**Computer Applications**]: Administrative Data Processing – *Education.*

General Terms

Experimentation, Performance, Verification.

Keywords

GridWork, grid computing, high throughput computing, high performance computing, JavaScript-based, web client execution environment.

1. INTRODUCTION

Traditionally, performance computing has focused on high performance computing [1]. However, the problem with high performance computing is that it usually involves building machines that are purpose-built or run software that is exceedingly specific. Grid computing on the other hand involves breaking a complex or long computational problem down into smaller sections, and executing each section on a possibly different machine. This form of computing can thus potentially offer a less expensive and faster solution than high performance computing.

The aim of this research paper is to investigate whether grid computing and specifically high throughput computing using JavaScript-based Web client job execution environments is in fact a viable option. It aims to verify whether a high throughput computing system can be implemented in a simple manner and to measure the performance of such a system.

2. PROCESS

High throughput computing is based on a client-server model. The implementation process is thus broken into two parts, namely the client side and the server side. The server side is further broken down into a sender program (to send files to the client for computation) and a receiver program (to receive the completed data from the client).

The basic idea is that the client sends a request for a file. The receiver program then selects a random file from the server and returns the data to the client. The client then processes the data and sends a request to the receiver program. The completed data is sent to the receiver program where it is stored in a result text file on the server. The receiver program then deletes the corresponding data file off the server so that it will not be processed again.

For the purposes of this project, simple XML (Extensible Markup Language) files were created and placed on the server as test files. Each file is in the outline of an order form, containing 10 products along with a price and quantity value. The file also contains a filename property. The client's job is to multiply the price and quantity values and then send through the product names along with these total values. The filename property is also sent so that the receiver program can use it to delete the appropriate data file.

2.1 Client side implementation

The client portion of this project was implemented as a web page that was able to send and receive data to and from the server. The webpage is simple in design, with the main feature being a button to start the process of receiving data from the server, processing it and sending the results back to the server. There is also a text area in which the results of this process are displayed. Each time a file has been successfully processed a message is added to the text area. Once all files have been completed, a summary of how many files were processed, as well as the time taken, is also displayed.

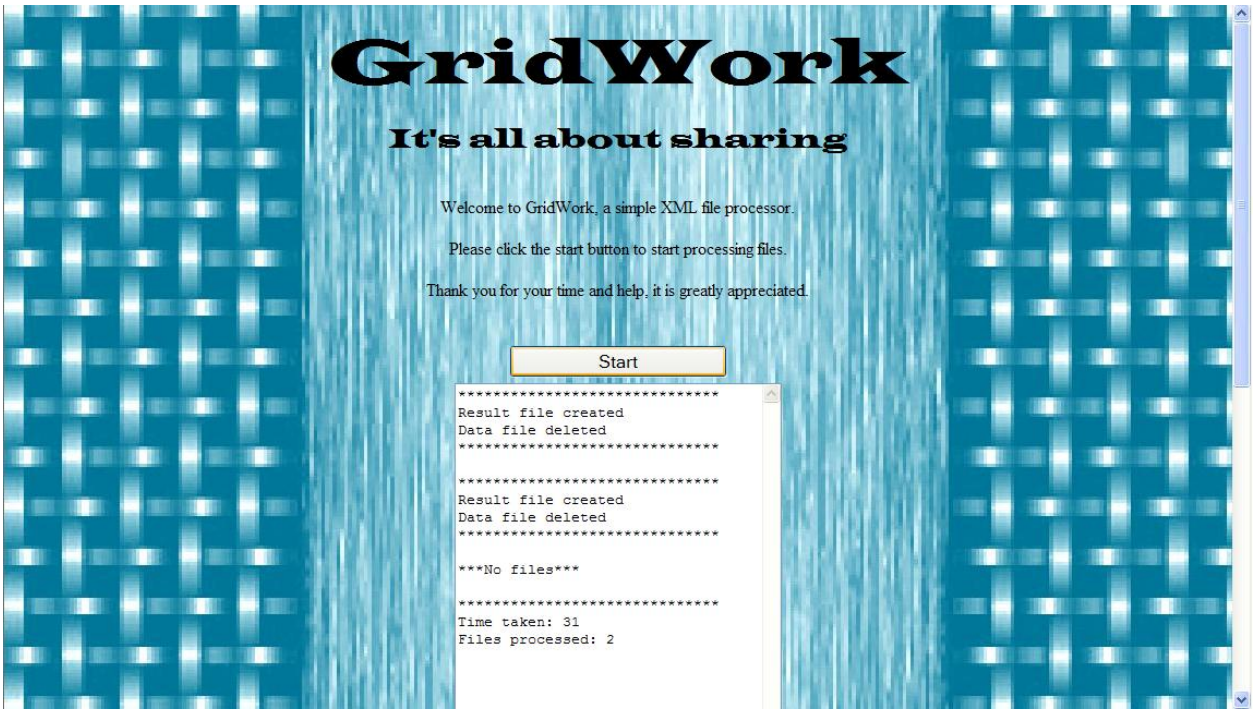

Figure 1: A screenshot of the client side web page.

Communication with the server is done using JavaScript, with transfer being synchronous so that the data can be received and processed immediately before being sent back to the server. This process of receiving files and sending the results to the server is repeated until all files on the server have been processed.

2.2 Server side implementation

Apache Tomcat 7.0.23 Server was used as the server for this project. Two servlets were created as the sender and receiver programs. Their class files were placed in a class folder in the WEB-INF folder of the project's web application folder on the server. They were mapped to appropriate URLs in the web.xml file for the use of communication with the client. The test files were placed in a "Files" folder in the project's web application folder and a "Completed" folder was created for storing the result files.

2.2.1 Sender program

The sender program selects a random file from the "Files" folder on the server. It then reads the contents of this file into a string and sends it as an xml response to the client. If an error occurs reading the files or if there are no files left to read, the program sends an html response with an appropriate message.

2.2.2 Receiver program

The receiver program receives two parameters from the client, the processed data and the name of the data file. The processed data is written to a result file (Result-<filename>.txt), provided that such a file does not already exist. The data file that was used to produce this data is then deleted using the filename parameter. This is to prevent the server from continuously resending the same files for processing. Finally the receiver program sends back the results of what it has done in a plain text response.

2.3 Evaluation

The project was evaluated by placing 501 files on the server and having three computers (One of which housed the server) process the files simultaneously. This was performed five times and the number of files each computer processed, in addition to the time taken was recorded. It should be noted that although only 501 files were available for processing, some files were processed more than once due to the fact that data files are only removed after the processed data has been saved. Thus some files may be sent for processing when they are already being processed by another computer.

The test was also performed five times using only the computer that the server was running on for comparison purposes.

3. RESULTS

Figure 2 below represents the results of testing. It shows the time taken to process a file by each computer. The columns in blue represent that of the three computers that processed files simultaneously, while the red column represents the computer that processed all the files alone.

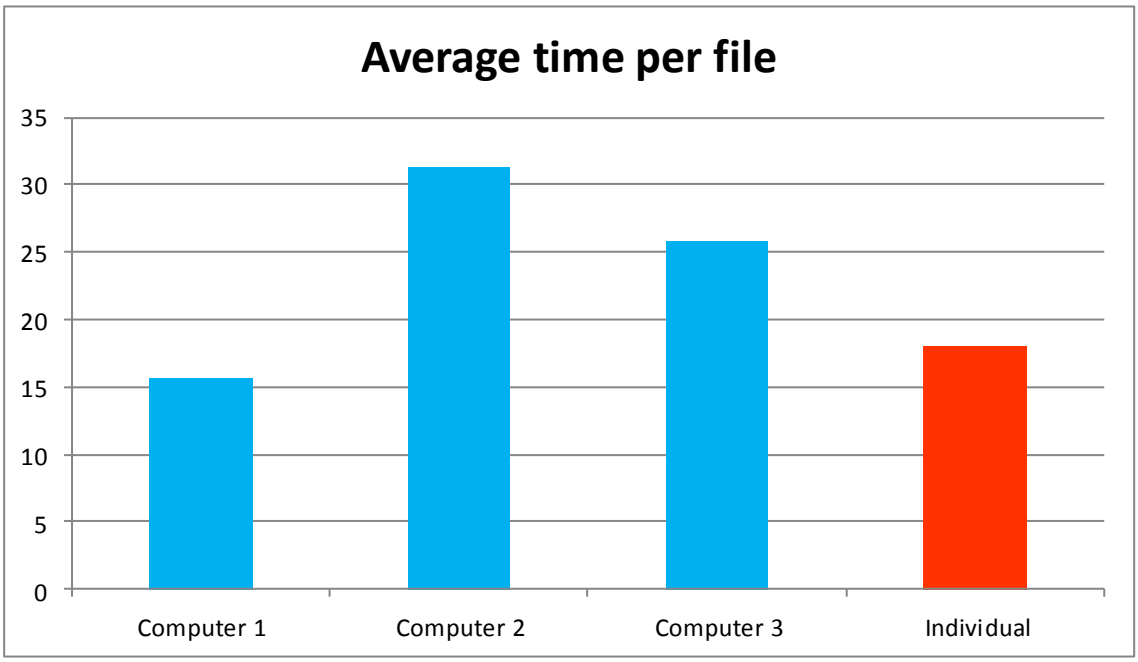

Figure 2: A graph representing the average time taken for each computer to process a file.

4. DISCUSSION

Comparing the time taken for the individual computer to process a file, to the times of the three simultaneous computers, there is not much difference if we consider only one file. If we consider four however, the three computers would be able to process them in 31.36 milliseconds, the time taken for computer 2 to process its one file (Computer 1 would process two files, and computer 3 one file). The individual computer however would take 71.89 milliseconds (4 multiplied by 17.972 ms), more than double the time of the three simultaneous computers.

From this multiple things can be noted. Firstly, the more computers available to process files, the better. Due to the fact that the files are processed simultaneously, more computers means that more files are being processed at the same time, which can greatly reduce the time taken to process all of the files. However it should also be noted that the system is only as fast as the computers it uses. One extremely fast computer would be able to process all of the files a lot faster than a large number of extremely slow computers. The idea behind high throughput computing though is not to have one extremely fast computer, which can be very expensive to build and maintain. The idea is utilize a large number of standard-to-fast computers that are readily available. High throughput computing can thus indeed offer a less expensive and faster solution to high performance computing, it just needs to be implemented in the correct manner, namely using a decent number of capable machines.

It should also be noted that a high throughput computing solution is fairly easy to implement. It is also very flexible; the code for the html file would just have to be altered to fit the format of the xml file being processed. It also scales very well; large amount of files could be processed easily. The JavaScript of the client side would need to converted to AJAX however to avoid the delay while files are being processed.

5. CONCLUSION

High throughput using JavaScript-based Web client job execution environments computing is a fairly easy concept to implement. It can be done in a low cost manner and can considerably accelerate the processing of data or any other task that lends itself towards this idea. It has potential for being implemented on both a large or small scale, with the only major concerns/costs being setup and maintenance/hosting of the server. The fact though that most of the infrastructure of this solution will exist already (i.e. a local area network or machines connected through the internet), is a real advantage. Tertiary institutions or businesses with a large amount of data processing could particularly benefit from high throughput computing. High throughput computing using JavaScript-based Web client job execution environments is thus a very viable option.

6. REFERENCES

1. Cohen, R. 2009. "Embracing Low Performance Computing". [online] Available at: http://www.elasticvapor.com/2009/10/embracing-low-performance-computing-lpc.html [Accessed 14 December 2011].

7. BIBLIOGRAPHY

1. W3Schools, 2012. "W3Schools: Online Web Tutorials". [online] Available at: http://www.w3schools.com [Accessed 17 January 2012].

2. Rose India Technologies, 2011. "A collection of web tutorials". [online] Available at: http://www.roseindia.net/ [Accessed 19 January 2012].

3. The Apache Software Foundation, 2012. "Apache Tomcat". [online] Available at: http://tomcat.apache.org/ [Accessed 9 January 2012].

4. Suman, B. 2009. "Get start with Ajax, Ajax simple example with Servlet, Ajax programming with Servlet". [online] Available at: http://binodsuman.blogspot.com/2009/05/get-start-with-ajax-ajax-siimple.html [Accessed 12 January 2012].

5. OpenJS, 2011. "Using POST method in XMLHTTPRequest(Ajax)". [online] Available at: http://www.openjs.com/articles/ajax_xmlhttp_using_post.php [Accessed 20 January 2012].

6. Murray, G. 2005. "Asynchronous JavaScript Technology and XML (Ajax) With the Java Platform". [online] (2006) Available at: http://www.oracle.com/technetwork/articles/javaee/ajax-135201.html [Accessed 20 January 2012].