2007

Performance Evaluation of Java Web Services: A Developer's Perspective

Je-Loon Yang
University of North Florida

PERFORMANCE EVALUATION OF JAVA WEB SERVICES: A DEVELOPER'S PERSPECTIVE

by

Je-Loon Yang

A project submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

December, 2007

The project "Performance Evaluation of Java Web Services: A Developer's Perspective" submitted by Je-Loon Yang in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by: Date

Signature Deleted

Sanjay Ahuja
Project Director

12/4/07

Signature Deleted

Charles N. Winton
Graduate Director of the School of Computing

12/12/07

Signature Deleted

Judith L. Solano
Director of the School of Computing

12/13/07

ACKNOWLEDGEMENT

This paper is a tribute to the helpful and thoughtful guidance of my adviser Dr. Sanjay Ahuja. I further express my gratitude to my family for unwavering support and understanding, during the many hours I dedicated to achieving this milestone in my life and career.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

With the rapid growth of traffic on the internet, further development of the web technology upon which it is based becomes extremely important. For the evolvement of Web 2.0, web services are essential. Web services are programs that allow different computer platforms to communicate interactively across the web, without the need for extra data for interfaces and formats, such as webpage structures. Since web services are a future trend for the growth of the internet, the tools used for their development are also important. Although there are many choices of web service frameworks to choose from, developers should choose the framework that best fits their applications, based on performance, time, and effort. For this project, we compared the qualitative and quantitative metrics of four common frameworks. The four frameworks were Apache Axis, JBossWS, Codehaus XFire, and Resin Hessian. After testing, the results were statistically analyzed using the Statistical Analysis System (SAS).

Chapter 1

INTRODUCTION

When going on a trip to another state or country, a person usually must buy airplane tickets, rent a car, and make hotel reservations. When dealing with airplane tickets, a person may even have to buy several tickets, due to not having a direct flight available to take the person directly to his or her final destination. Looking up arrival and departure times for connecting flights that will not require a long wait at the airport could be time consuming and frustrating. Therefore, people often seek out travel agents to make arrangements for them. But, what if the agent was actually a virtual agent online [Hendler01]? What if the person just entered into the computer the location he wanted to start from, the destination, the desired time for departure or arrival, and all the information required; and, the computer showed all the results the person could choose from to purchase the tickets? Even better, such virtual agents could provide information on car rentals and hotels near a person's destination and reserve them. A virtual agent could save much effort and time and could also be more accurate than human agents. This kind of agent has already been implemented, although not necessarily using web services, and represents the types of technology that stand to benefit from the development of web services. Figure 1 shows a comparison of travel planning with and without a virtual agent.

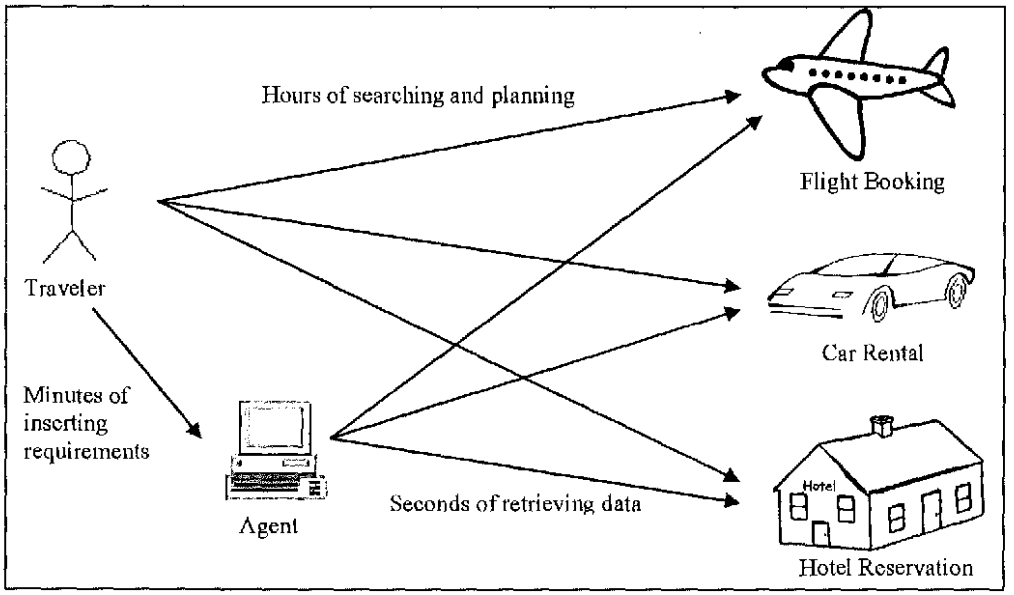

Figure 1: Comparison of Planning Travel With and Without Virtual Agent

Instead of creating a system that browses airline websites to look up flight schedules and then integrates the information, a system requesting the same information through web services is much better for several reasons, one of which would be easier maintenance [Yang02]. The format of websites' pages may change from time to time. For example, if Northwest Airlines wants to add more services to its website once in a while, the format of the webpage would definitely change, making it more difficult for an agent system to retrieve data from it, unless the system had a high degree of artificial intelligence, which is an unnecessary feature. Another reason for web services would be efficiency [McIlraith01]. Even for a webpage that never changes format at all, a travel agency system would have to download all the hypertext pages with much unnecessary data, such as the markups, instead of just retrieving the information needed in a few strings. Web services avoid this problem, since the required information can be easily

called up and only the information requested would be sent back, without the markups. Figure 2 compares the information size of HTML webpages and simple object access protocol (SOAP) envelopes used by web services.

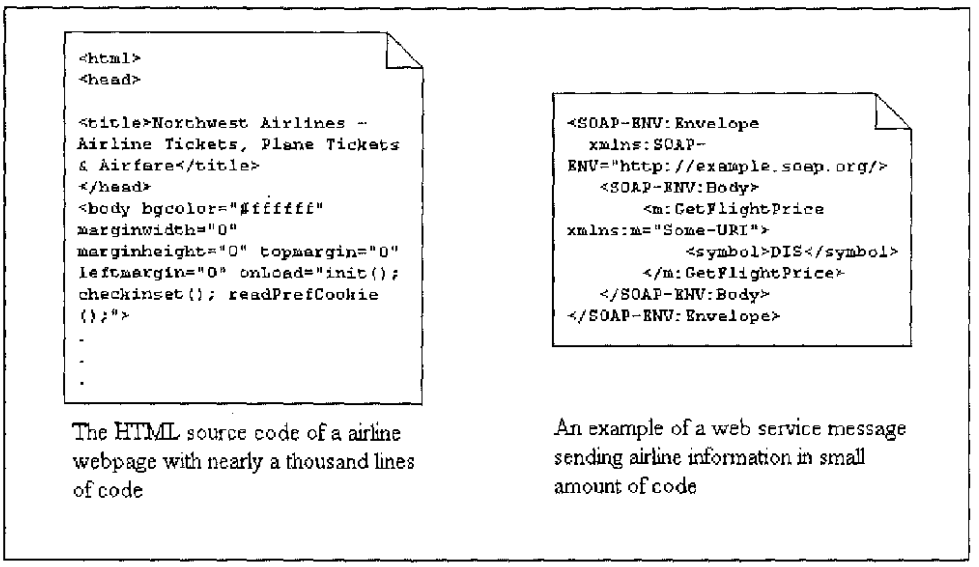

Figure 2: Comparison of Information Between Webpages and Web Services

Instead of developing a web service application from scratch, there are frameworks available, which are mostly free, that can be used to make development much easier. Which of these frameworks would be a better choice for web service application development? This study compares four popular open source frameworks, both qualitatively and quantitatively, by doing several tests and analyses. The four frameworks are Apache Axis, JBossWS, XFire, and Hessian. A more thorough introduction of web services is given in chapter 2. Chapter 3 describes the four frameworks used in this study. In chapter 4, the metrics used to measure the performance of the frameworks are explained in more detail. Chapter 5 introduces the

statistical methods used to analyze the results. In chapter 6, the test results are shown and analyzed. The conclusions are presented in chapter 7.

Chapter 2

WEB SERVICES

Web services are basically software systems designed to support interoperable machine-to-machine interaction over a network [Narayanan02]. In order to allow different computer platforms to communicate with each other, a language that all platforms can understand is needed. A platform-independent language, Extensible Markup Language (XML) [Decker00], performs this role. XML envelopes specified in a SOAP format are passed between client and web services to enable communication.

Web services are divided into three different areas – communication protocols, service descriptions, and service discovery. The specifications for each are currently being developed. The most common specifications for each area are SOAP, the Web Services Description Language (WSDL), and the Universal Description, Discovery, and Integration (UDDI) directory [Curbera02]. Figure 3 shows the architecture of web services based on the three areas..

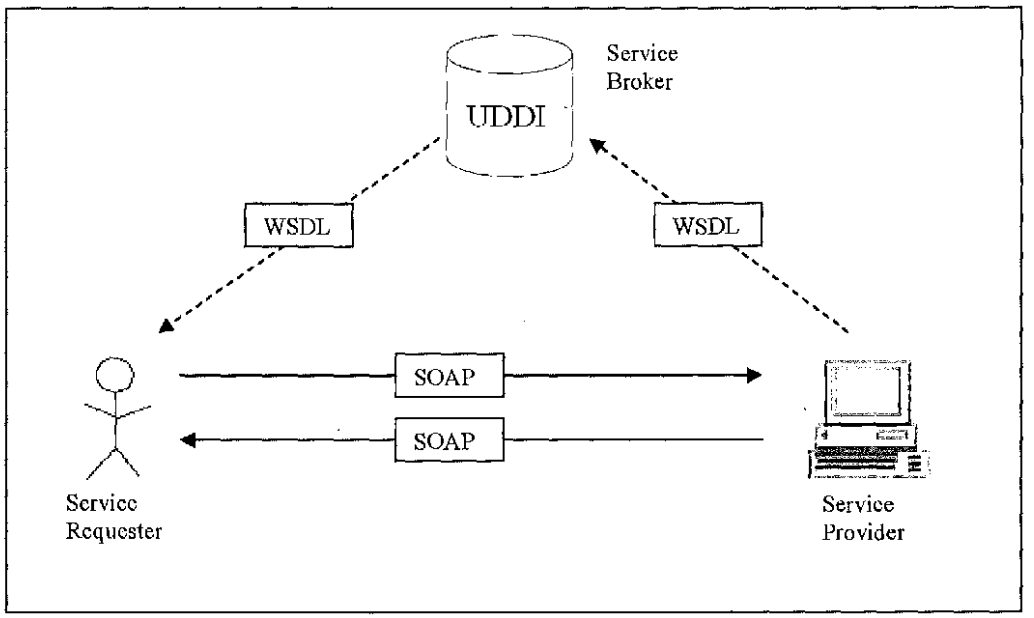

Figure 3: Web Service Architecture

2.1 SOAP

The SOAP protocol specifies communication among web services. Since the Web's nature is actually both distributed and heterogeneous, communication methods for web services has to be platform-independent, international, secure, and as lightweight as possible. XML meets such qualifications effectively, and thus, at present is the best solution for a web service's communication protocol.

The web service's XML-based protocol is used for messaging and remote procedure calls (RPC). Instead of defining a new transport protocol, SOAP works on existing transports such as HTTP, SMTP, and MQSeries. The structure of SOAP messages is quite trivial. It is an XML element with two child elements – one of them containing the

header and the other containing the body [Curbera02]. Both the header contents and body are arbitrary XML elements. Figure 4 shows the structure of a SOAP envelope.

```
<SOAP:Envelope xmlns:SOAP= "http://
schemas.xmlsoap.org/soap/envelope/">
     <SOAP:Header>
          <!-- content of header goes here -->
     </SOAP:Header>
     <SOAP:Body>
          <!-- content of body goes here -->
     </SOAP:Body>
</SOAP:Envelope>
```

Figure 4: Structure of SOAP Envelope

2.2 WSDL

WSDL provides a formal, computer-readable description of web services. Although SOAP enables communication for web services, it does not provide the information about the messages exchanged for the interaction. This is where WSDL comes into play. It describes the interface of web services and provides users the information needed for making SOAP messages. This description language is in XML format and was developed by IBM and Microsoft to describe web services.

Two pieces of information are provided in a WSDL service description: an application-level service description and specific protocol-dependent details [Curbera02]. Users must follow the details, so they can access the web services at their concrete end points. The purpose of separating the information provided by WSDL into the two levels is to

help show common functionality between different end points. This is intended to make development of web service applications easier to understand.

2.3 UDDI

Web services would not be very useful, if only limited services were available. UDDI solves this problem. It is a registry of web services' descriptions, like a phone book for web services in the digital world. The specification allows users to find service providers through a centralized registry of services. There are two basic types [Curbera03] of specifications that define a service registry's structure and operation. One is the definition of the information to provide about each service and the way to encode it; the other is the query and how to update the API for the registry that describes the way such information can be accessed and updated.

Chapter 3

WEB SERVICE FRAMEWORKS

Since web services are designed to transfer data in common ways. Several companies and groups developed web service frameworks for the convenience of web service developers, so they do not need to write a complete web service from scratch. Some of the popular frameworks are Apache Axis, JBossWS, Codehaus XFire, and Resin Hessian. This chapter introduces and discusses these frameworks.

3.1 Apache Axis

Apache Axis (Axis stands for Apache EXtensible Interaction System) is an open source, Java and XML based web service framework created by the Apache Software Foundation (ASF). The foundation is a non-profit corporation that mainly produces software for network use, such as servers and frameworks for servers. Their projects are known to be collaborative, consensus based development processes and free or open source software. The Apache Axis package has an implementation of a SOAP server and application programming interfaces (API) for generating and deploying web service applications [WSA06]. The SOAP engine constructs SOAP processors like clients, servers, and gateways. This allows the servers and clients to communicate through SOAP messages. The API supports a variety of languages. Besides the Java version, a C++ implementation is also available. It allows developers to construct their applications in a variety of ways. The easiest method only requires changing the file

name extension from ".java" to ".jws". The downside of such a method is that it lacks flexibility for further configuration.

3.2 JBossWS

JBossWS is JBoss' implementation of Java 2 Enterprise Edition (J2EE) compatible web services. The framework is designed to fit better in the overall JBoss architecture and is generally more suitable for the specific J2EE requirements for web services. Instead of using the traditional Apache server for this framework, JBoss has a server of its own and suggests using a framework on this server to get the best performance. Similar to ASF, JBoss serves people who focus on open source projects. Their projects emphasize the development of Java Enterprise Middleware [JBossWS07], which are software programs that act like bridges between applications, operating systems, or both.

3.3 Codehaus XFire

Codehaus XFire is a next-generation java SOAP framework. It is a free and open source SOAP framework that allows a user to implement web services with great ease and simplicity. It also provides many features identified in web service specifications, which are not yet available in most commercial or open source tools. It is claimed that Codehaus XFire has higher performance, since it is built on a low memory StAX (Streaming API for XML) based model, but no data confirms this claim [XFire07].

3.4 Resin Hessian

The Hessian binary web service protocol makes developing web services simple and usable without requiring a large framework; thus, developers do not need to spend more time and effort to learn a wide variety of protocols. Since it is a binary protocol, it works well on sending binary data, without any need to extend the protocol with attachments. Java 2 Micro Edition (J2ME) devices like cell-phones and PDAs can use Hessian to connect to web services with better performance, because it is a small protocol [HBWSP06]. Hessian was named after the Hessian cloth, which is the British term for Burlap. Burlap is simple, practical, and useful, but extremely ordinary material – similar to the characteristics of the Hessian protocol. Resin is an open source application server also offered by Caucho that integrates with Hessian.

Chapter 4

EVALUATION METRICS

This chapter examines different factors to be considered when comparing the four frameworks in this project. Some metrics are to determine the performance and efficiency; some are to show the transparency and abstraction.

4.1 Latency

In terms of networks, latency is an expression of how much time it takes for data to be sent back in response to a request [Ching01]. This includes the time for the request to be sent to the server, the time the server spends on processing the task, and the time for the results to be sent back. Figure 5 depicts what this would look like. Network latency is contributed to by many factors, such as propagation, transmission, modem and router processing, and storage delays. Propagation is the time it takes for objects, such as data, to transfer from one location to another at the speed of light. Transmission is the delay from the medium, like optical fiber or wireless networks. Modems and routers take time to check the headers of a packet. The storage delay is the time it takes for the actual hardware, such as hard drives, to store the received data. In this project, the latency was tested with different scenarios, such as requesting 1, 2, 3, 4, and 5 MB of data, and having 1, 5, 10, 15, and 20 clients simultaneously (within the limits of the environment utilized) requesting data. From the results of such testing, trends can be found and compared for each framework.

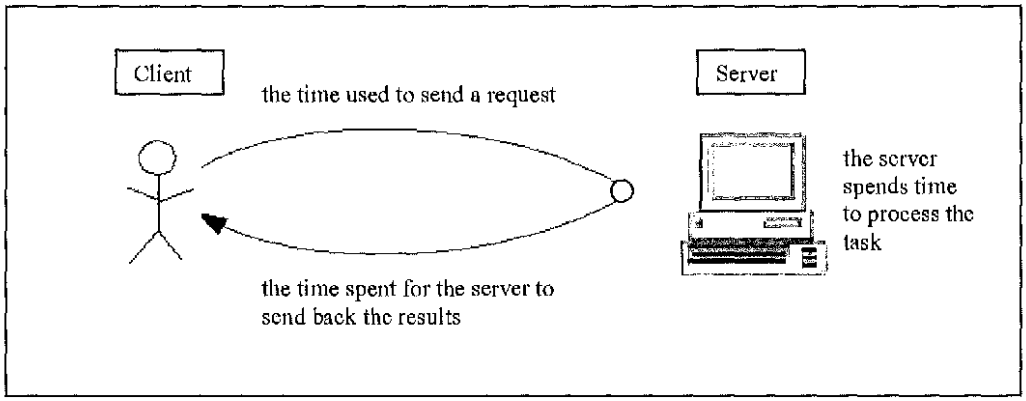

Figure 5: Network Latency

4.2 Throughput

Throughput is the amount of clients or data processed within a certain unit of time [Ching01]. It has an inverse relationship with latency, since scenarios with high latency would result in low throughput, and scenarios with low latency would result in high throughput. By viewing the latency graph, we can only tell the trends of response time, while we can determine the most efficient scenario for a framework through viewing a throughput graph.

4.3 Memory Usage

In computing, the purpose of memory is to temporarily store data for computer calculations. There are a number of memory types, such as cache memory, flash memory, random access memory (RAM), virtual memory, etc., but they are all limited on servers, due to cost and space considerations. A framework that uses less memory

would have the advantage by allowing higher capacity for the server. Figure 6 shows where the memory usage can be viewed in Windows XP.

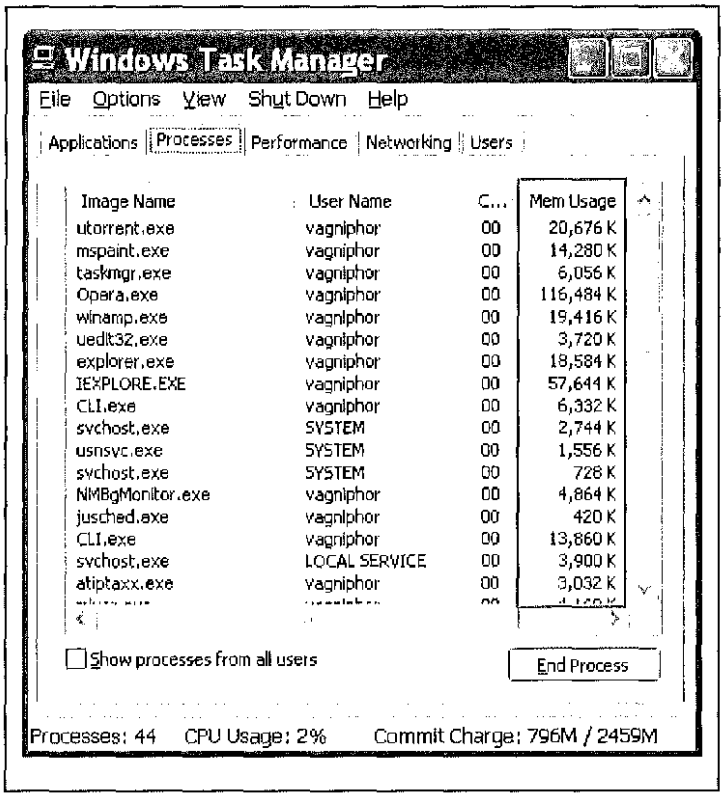

Figure 6: Memory Usage in Windows XP

4.4 CPU Usage

A central processing unit (CPU), also known simply as a processor, is a component in a computer used to interpret program instructions and process data. The CPU is only able to process one task at a time. When there are multiple tasks, instead of finishing a task (potentially waiting for I/O or other system operations) and then going to another, the

CPU is designed to switch to other tasks, if necessary, thus behaving as if it is executing multiple tasks at the same time. However, large tasks might consume a lot of CPU time, which decreases the time scheduled for other tasks. A framework that uses less CPU capacity would allow the server to have more time to execute other tasks. Figure 7 shows where the CPU usage can be viewed in Windows XP.

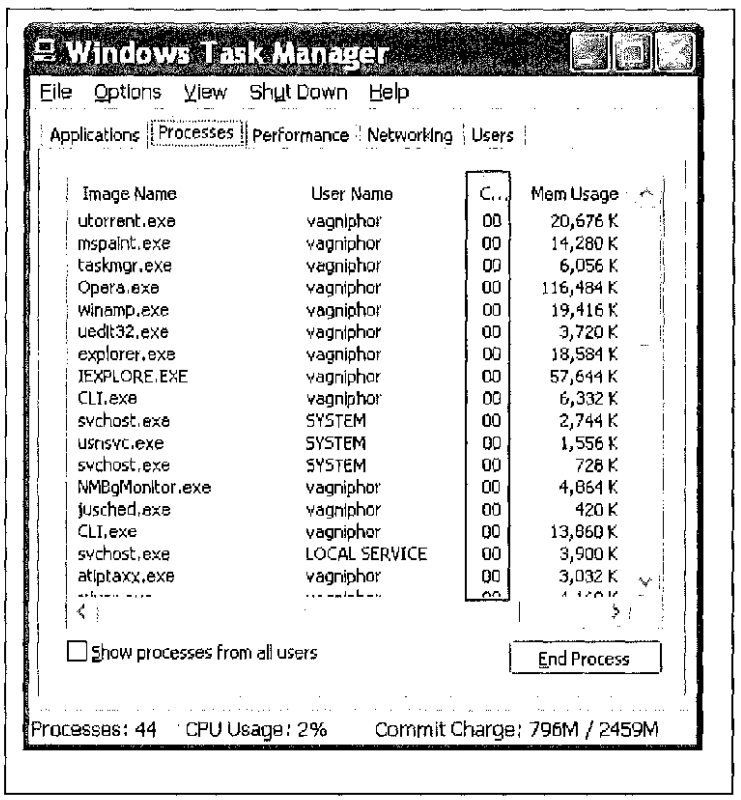

Figure 7: CPU Usage in Windows XP

4.5 Source Lines of Code

The source lines of code (SLOC) used in a framework can indicate the transparency and abstraction of the framework. The main goal of a framework is to save the developer's time and effort, by not requiring the entire code be written from scratch. Thus, the fewer the lines of code required for a framework, the more time and effort it saves. However, lines of code as a metric have evident limitations, since some lines might be long while some lines are short. So, other measures, such as the number and size of files, must also be considered.

Chapter 5

STATISTICAL ANALYSIS METHODS

A method is required for analysis of the test data produced to compare performances. Simply calculating the average response times and graphing them is insufficient; for instance, in looking at average response times of 1.5 and 1.6 seconds, this statistic in isolation doesn't indicate whether there is a significant difference or not. Therefore, statistical analysis methods were required to tell whether or not the difference was significant. In this project, the general linear model (GLM) and two-way analysis of variance (two-way ANOVA) were used for statistical analyses. Furthermore, the Statistical Analysis System (SAS) was used as a tool for generating the calculations for the statistical analyses required.

5.1 The SAS System

The SAS system is statistical analysis software that has a wide variety of statistical modules and procedures. The system uses a fourth-generation programming language (4GL) for code and the programs are composed using three main components [SAS07] – the data step, the procedure step, and the macro language. The data step is for entering data, like inserting the data in the code or reading data in data files. The procedure step is the use of statistical methods and models to analyze the data read in the data step. The macro language is for decreasing the redundancy of functions used again and again throughout the program.

5.2 The GLM Model

The GLM model is a statistical linear model used in general cases. It is the foundation of many statistical analyses, such as t-test, ANOVA, Analysis of Covariance (ANCOVA), etc. To understand how the GLM model works it is easiest to look at the two-variable case [GLM06]. The goal of this analysis is to find a way to accurately describe the information in the plot in Figure 8.

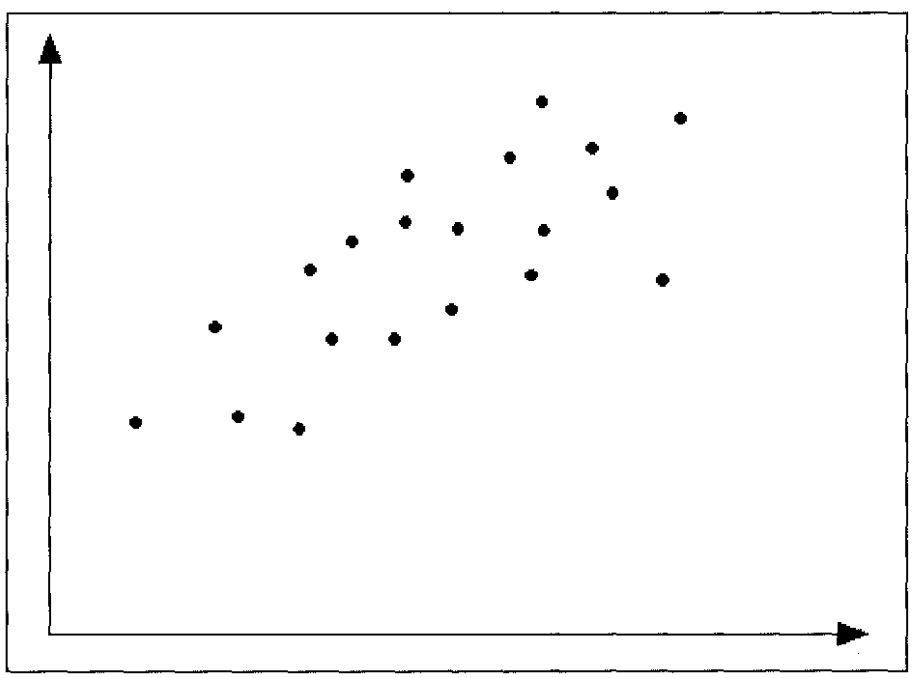

Figure 8: Plot of Two-Variable Example

Using the GLM model, we try to find a straight line closest to all the dots in the plot. Figure 9 shows the corresponding line for this example.

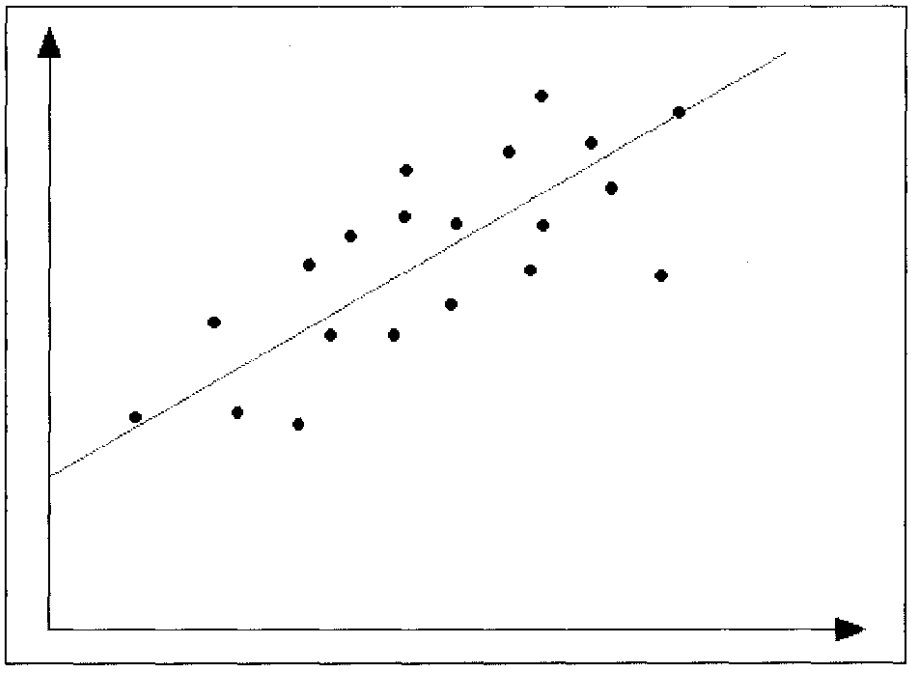

Figure 9: Plot of Two-Variable Example with Straight Line

This line would be written like this: $y = b_0 + b_1x + e$, where y is the y-axis variable, x is the x-axis variable, b_0 is the intercept (the value of y when x equals 0), b_1 is the slope of the straight line, and e is the error. Figure 10 shows the variables in the plot.

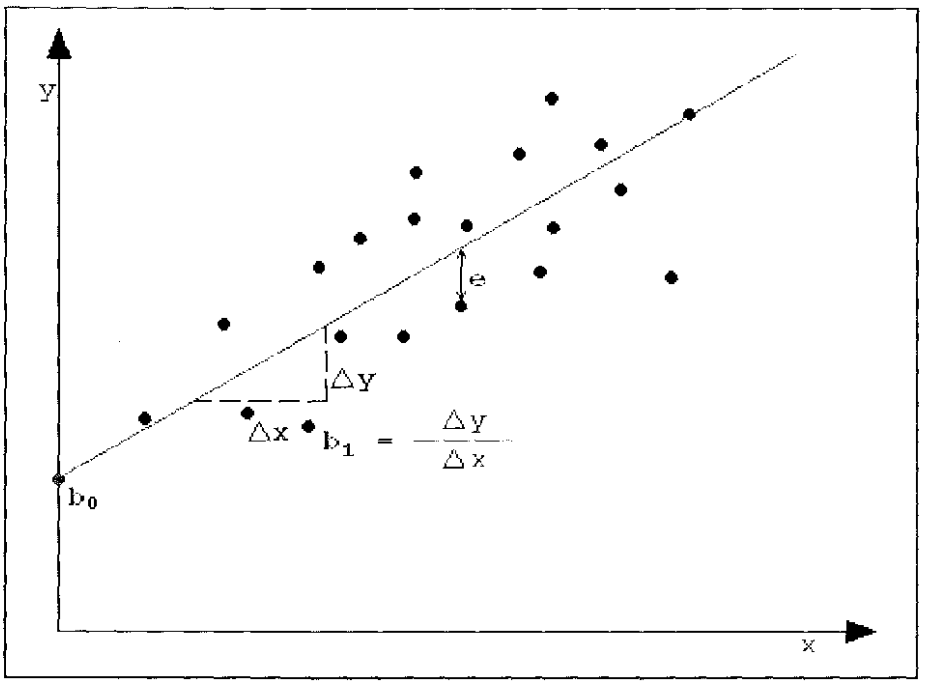

Figure 10: Plot of Two-Variable Example with Straight Line and Variables

By solving for b_0 and b_1, we can get information about this linear approximation for the dots in the plot. In other cases with more than two variables, the formula can be extended like this: $y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \ldots + b_nx_n + e$, where n is the number of variables for the situation. The mechanism for solving such problems is the same as for solving with two variables.

Chapter 6

RESULTS AND ANALYSES

In order to get the best results from SAS, each case was tested 20 times. Since four frameworks were tested by measuring response time for each, when five different numbers of clients (1, 5, 10, 15, and 20) requested data simultaneously, there were 20 different cases. The 20 test times for the 20 different cases resulted in 400 data sets to be calculated by SAS. Besides the number of clients, the amount of data was also considered, so with five different file sizes of data sent (1, 2, 3, 4, and 5 MB) for each of the four frameworks, there were 20 cases with a total of 400 data sets. Response time was measured by recording the time right before invoking the web service and recording the time right after the data requested was received, then subtracting the time difference.

6.1 Results

6.1.1 Client Scenarios

For testing the four different frameworks in different scenarios, web service applications to send out the data were created. To test the performance of the four frameworks based on the number of clients requesting data simultaneously, five scenarios were run (1 client, 5 clients, 10 clients, 15 clients, and 20 clients). Each client

retrieved 1 MB of data. The average response time for each scenario and each framework was recorded for analysis. The results are as shown in Figure 11.

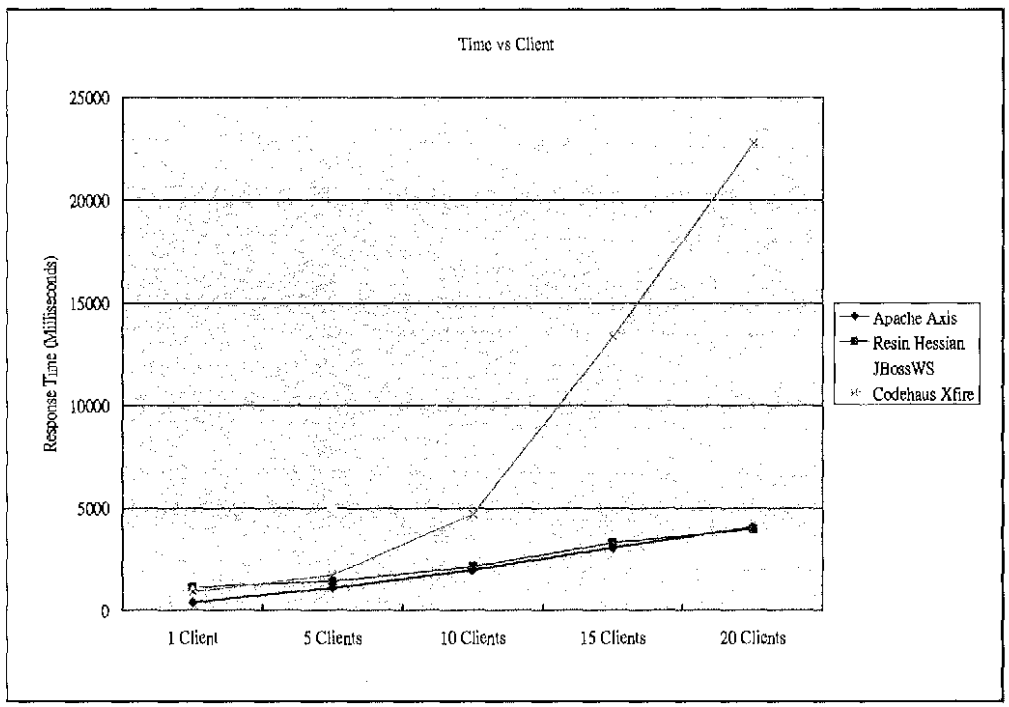

Figure 11: Latency in Client Scenarios

For throughput, Figure 12 shows the average number of clients serviced per second for each scenario and framework, and in particular, the most efficient client scenario for each framework. Apache Axis could deal with 4.993 clients per second, after reaching the scenarios with 10 clients or more. Resin Hessian could deal with 4.807 clients per second in those same scenarios. JBossWS dealt with 0.943 clients per second in every scenario. Codehaus XFire seemed to work most efficiently around the scenario of 5 clients, dealing about 2.892 clients per second.

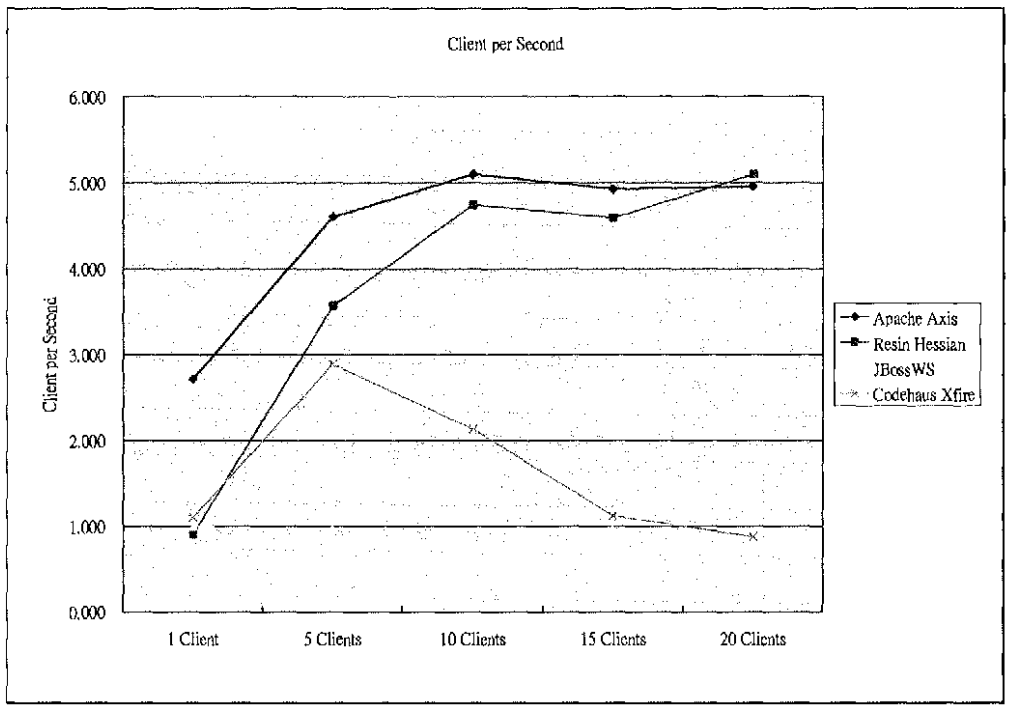

Figure 12: Throughput in Client Scenarios

6.1.2 Data Size Scenarios

The average response time for the five scenarios based on different data sizes is shown in Figure 13.

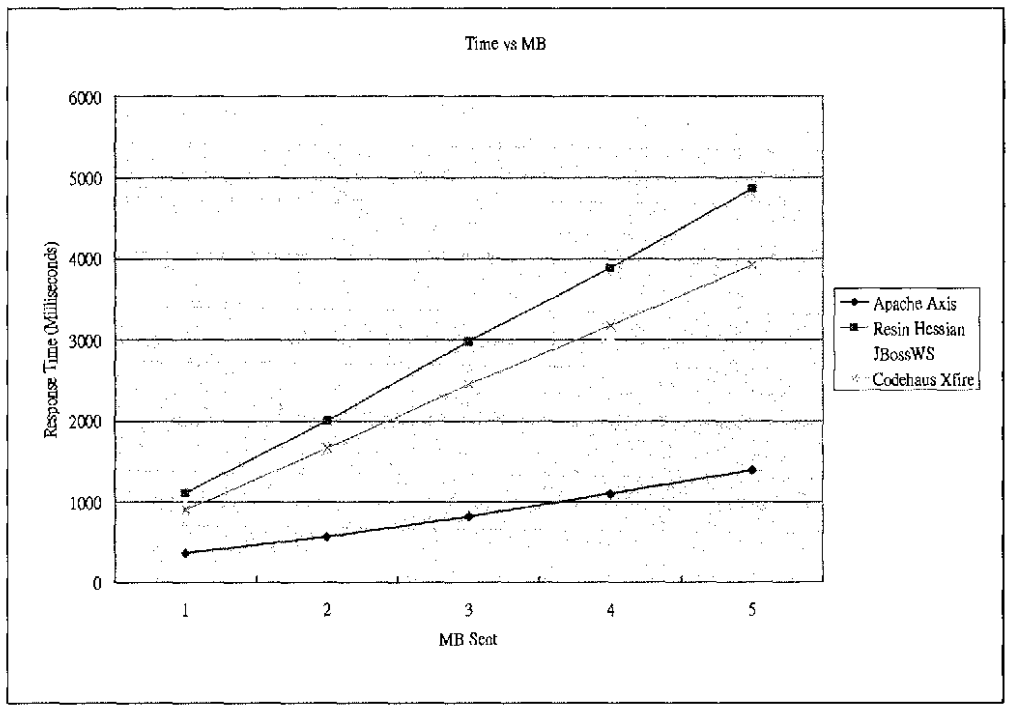

Figure 13: Latency in Data Size Scenarios

Figure 14 shows the most efficient data size scenario for each framework. All frameworks reached their best performance when data files of 2 MB or more were sent. Apache Axis processed an average of 3.617 MB/s, JBossWS an average of 1.287 MB/s, Codehaus XFire an average of 1.240 MB/s, and Resin Hessian an average of 1.017 MB/s.

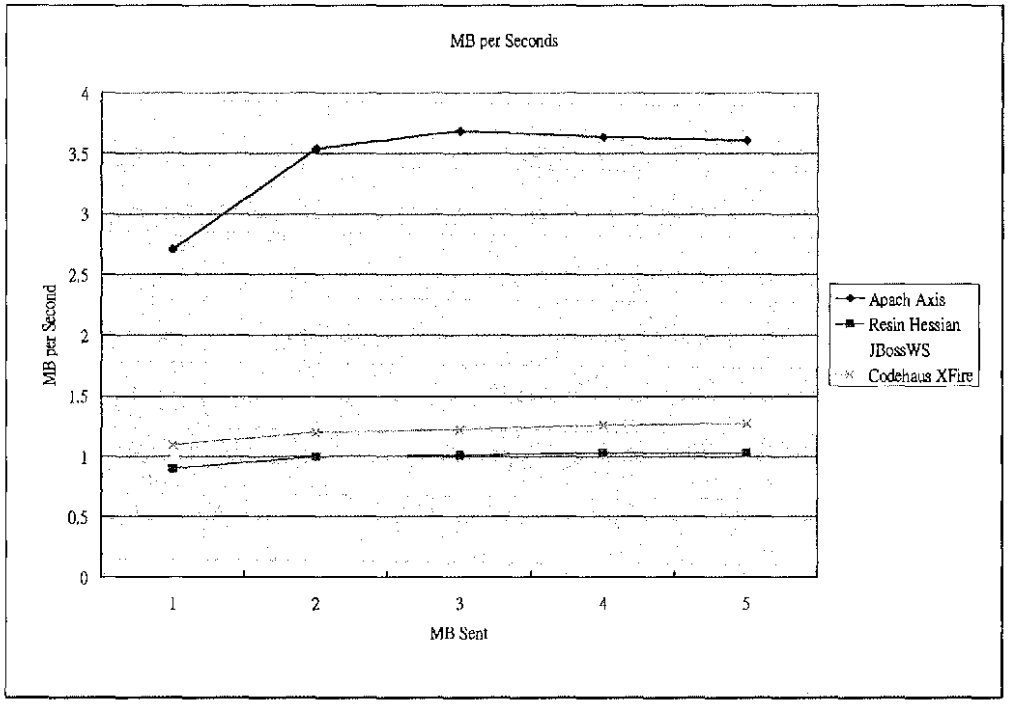

Figure 14: Throughput in Data Size Scenarios

From the graphs, it appears that Apache Axis had the best performance in all scenarios. To confirm this, further analysis was done.

6.2 Analyses

Obviously, response time depends on the choice of framework, the quantity of data transferred, and the number of clients invoking tasks from the web service. Thus, these three factors might significantly impact response times. We used the GLM model to determine if the interactions of the three factors were also significant. For interactions that were not significant, we determined we would use Tukey's method to do multiple

comparisons to see which framework had better performances in all cases and which had worse. For interactions that were significant, we would analyze the results case by case.

6.2.1 Client Scenarios

First, analyzing the results from the client scenarios, we used the SAS system to determine the significance of each factor. Figure 15 shows the results.

Source	Pr > F
clients	<.0001
framework	<.0001
clients*framework	<.0001

Figure 15: Factor Significance SAS in Client Scenarios

The "Pr > F" value, in Figure 15, shows whether the results were significant. If the value was lower than 0.05, that meant the factor was significant; if the value was greater than or equal to 0.05, then the factor would not be considered significant. In this case, not only were the number of clients and choice of framework significant factors, but the interaction between them was also significant. This means if one of the frameworks was significantly faster in some scenarios, it would not necessarily be faster in other scenarios. So, SAS cannot directly compare all frameworks in all scenarios.

A pair-wise comparison from the GLM procedure was used to compare the frameworks in different scenarios. The first framework was compared to the second in the first scenario, then the first to third, first to fourth, second to third, second to fourth, and third to fourth. So, there were six comparisons in each scenario. Figure 16 shows the results of these comparisons.

Parameter	Estimate	Standard Error	t Value	Pr > |t|
f1-f2 at c1	-742.0000	601.720767	-1.23	0.2183
f1-f3 at c1	-633.8500	601.720767	-1.05	0.2928
f1-f4 at c1	-538.1500	601.720767	-0.89	0.3717
f2-f3 at c1	108.1500	601.720767	0.18	0.8575
f2-f4 at c1	203.8500	601.720767	0.34	0.7350
f3-f4 at c1	95.7000	601.720767	0.16	0.8737
f1-f3 at c2	-4053.2000	601.720767	-6.74	<.0001
f1-f4 at c2	-642.0500	601.720767	-1.07	0.2866
f2-f3 at c2	-3737.1500	601.720767	-6.21	<.0001
f2-f4 at c2	-326.0000	601.720767	-0.54	0.5883
f3-f4 at c2	3411.1500	601.720767	5.67	<.0001
f1-f2 at c3	-148.4500	601.720767	-0.25	0.8053
f1-f3 at c3	-9738.4500	601.720767	-16.18	<.0001
f1-f4 at c3	-2727.7500	601.720767	-4.53	<.0001
f2-f3 at c3	-9590.0000	601.720767	-15.94	<.0001
f2-f4 at c3	-2579.3000	601.720767	-4.29	<.0001
f3-f4 at c3	7010.7000	601.720767	11.65	<.0001
f1-f2 at c4	-223.7500	601.720767	-0.37	0.7102
f1-f3 at c4	-12565.7000	601.720767	-20.88	<.0001
f1-f4 at c4	-10305.2000	601.720767	-17.13	<.0001
f2-f3 at c4	-12341.9500	601.720767	-20.51	<.0001
f2-f4 at c4	-10081.4500	601.720767	-16.75	<.0001
f3-f4 at c4	2260.5000	601.720767	3.76	0.0002
f1-f2 at c5	110.4000	601.720767	0.18	0.8545
f1-f3 at c5	-17435.2500	601.720767	-28.98	<.0001
f1-f4 at c5	-18729.2500	601.720767	-31.13	<.0001
f2-f3 at c5	-17545.6500	601.720767	-29.16	<.0001
f2-f4 at c5	-18839.6500	601.720767	-31.31	<.0001
f3-f4 at c5	-1294.0000	601.720767	-2.15	0.0321

Figure 16: Pair-Wise Comparison Results from SAS in Client Scenarios

Looking at the "estimate" in the first row, which is the comparison between the first framework (f1) and the second framework (f2), f1 is faster than f2 by -742 milliseconds. But the "Pr > | t |" value shows this difference is not significant, because the value is higher than 0.05. By integrating all the "Pr > | t |" values together, to see which frameworks were significantly faster, a better view of the performances was created. Table 1 shows the integration results by separating the frameworks into groups.

	1 Client	5 Clients	10 Clients	15 Clients	20 Clients
Apache Axis	A	A	A	A	A
Resin Hessian	A	A	A	A	A
JBossWS	A	B	C	C	C
Codehaus XFire	A	A	B	B	B

Table 1: Response Time Comparison for Client Scenarios

Table 1 should be read one scenario at a time, i.e., when looking at the 1 client scenario, ignore the data in the 5 client, 10 client, 15 client, and 20 client scenarios. Groups labeled with lower alphabetic characters had lower response times, which meant better performance. In the 1 client scenario, all frameworks were in group A, meaning they all had approximately the same performance in this scenario. In the 5 client scenario, JBossWS was in group B while the others were in group A. This means, in this scenario, JBossWS had worse performance than the others, and the difference was significant,

while the others performed about the same. In the last three scenarios, Apache Axis and Resin Hessian were faster than Codehaus XFire, and Codehaus XFire was faster than JBossWS.

Although from the SAS analysis results, the better performance of frameworks is a case by case matter, as the number of clients increase to 10 or more, Apache Axis and Resin Hessian performed better than Codehaus XFire, and Codehaus XFire better than JBossWS.

6.2.2 Data Size Scenarios

The process of analyzing performance based on data size was the same as the process used for analyzing it based on client amount. First, the interaction between data size and choice of framework was determined. The results are as shown in Figure 17.

Source	Pr > F
mb	<.0001
framework	<.0001
mb*framework	<.0001

Figure 17: Factor Significance in Data Size Scenarios

It turned out that the interaction between data size and choice of framework was also significant. Therefore, the same pair-wise comparison procedure was used. The results are given in Figure 18.

Parameter	Estimate	Error	t Value	Pr > |t|
f1-f2 at c1	-742.00000	30.8873620	-24.02	<.0001
f1-f3 at c1	-633.85000	30.8873620	-20.52	<.0001
f1-f4 at c1	-538.15000	30.8873620	-17.42	<.0001
f2-f3 at c1	108.15000	30.8873620	3.50	0.0005
f2-f4 at c1	203.85000	30.8873620	6.60	<.0001
f3-f4 at c1	95.70000	30.8873620	3.10	0.0021
f1-f2 at c2	-1435.75000	30.8873620	-46.48	<.0001
f1-f3 at c2	-1108.20000	30.8873620	-35.88	<.0001
f1-f4 at c2	-1099.95000	30.8873620	-35.61	<.0001
f2-f3 at c2	327.55000	30.8873620	10.60	<.0001
f2-f4 at c2	335.80000	30.8873620	10.87	<.0001
f3-f4 at c2	8.25000	30.8873620	0.27	0.7895
f1-f2 at c3	-2163.10000	30.8873620	-70.03	<.0001
f1-f3 at c3	-1501.85000	30.8873620	-48.62	<.0001
f1-f4 at c3	-1638.90000	30.8873620	-53.06	<.0001
f2-f3 at c3	661.25000	30.8873620	21.41	<.0001
f2-f4 at c3	524.20000	30.8873620	16.97	<.0001
f3-f4 at c3	-137.05000	30.8873620	-4.44	<.0001
f1-f2 at c4	-2780.05000	30.8873620	-90.01	<.0001
f1-f3 at c4	-1937.70000	30.8873620	-62.73	<.0001
f1-f4 at c4	-2075.20000	30.8873620	-67.19	<.0001
f2-f3 at c4	842.35000	30.8873620	27.27	<.0001
f2-f4 at c4	704.85000	30.8873620	22.82	<.0001
f3-f4 at c4	-137.50000	30.8873620	-4.45	<.0001
f1-f2 at c5	-3473.55000	30.8873620	-112.46	<.0001
f1-f3 at c5	-2341.00000	30.8873620	-75.79	<.0001
f1-f4 at c5	-2537.55000	30.8873620	-82.15	<.0001
f2-f3 at c5	1132.55000	30.8873620	36.67	<.0001
f2-f4 at c5	936.00000	30.8873620	30.30	<.0001
f3-f4 at c5	-196.55000	30.8873620	-6.36	<.0001

Figure 18: Pair-Wise Comparison Results from SAS in Data Size Scenarios

Table 2 shows frameworks separated into groups based on the integration of their performance results.

	1 MB	2 MB	3 MB	4 MB	5 MB
Apache Axis	A	A	A	A	A
Resin Hessian	D	C	D	D	D
JBossWS	C	B	B	B	B
Codehaus XFire	B	B	C	C	C

Table 2: Response Time Comparison for Data Size Scenarios

When sending 1 MB of data, Apache Axis was better than Codehaus XFire, which was better than JBossWS, which was in turn better than Resin Hessian. Although each scenario was a different case, as the data size increased, Apache Axis was the best, and Resin Hessian was the worst. The only differences were, in the second scenario, the performances of Codehaus XFire and JBossWS were equivalent, and, in the last three scenarios, JBossWS was faster than Codehaus XFire.

6.2.3 Others

Other metrics such as memory usage, CPU usage, and SLOC were also tested in this project. Table 3 shows memory and CPU used on the web service application created, using each framework.

	Memory Usage	CPU Usage
Apache Axis	13%	0%
Resin Hessian	8.7%	0%
JBossWS	16%	0%
Codehaus XFire	13%	0%

Table 3: Memory and CPU Usage of Four Frameworks

Since the web service applications created, using the four frameworks, barely used any CPU at all, CPU usage was not a useful factor. Comparing the memory usages, Resin Hessian used the least, approximately half of that was used by JBossWS. Apache Axis and Codehaus XFire used an intermediate amount of memory.

The SLOC of web services created, using each framework, are as shown in Table 4.

SLOC	Server Side	Client Side	Total
Apache Axis	64	120	184
Resin Hessian	70	85	155
JBossWS	94	127	221
Codehaus XFire	48	128	176

Table 4: SLOC of Application of Four Frameworks

JBossWS required the most lines of code and Resin Hessian required the least. The web service application used to test the frameworks had only one trivial function, so it required few lines of code. But, if these frameworks were to be used to create large real-world applications, the 42% difference of SLOC between JBossWS and Hessian could mean many, many more lines, which would greatly increase the effort, time, and errors for an application development.

Chapter 7

CONCLUSION

For web applications that require communication through the network between different computer platforms, web service would be a good choice, since it is designed based on a platform-independent language – XML. Instead of developing web services from scratch, using existing frameworks can greatly increase productivity and lessen the time and effort that developers spend on learning the details of web services.

From the test results of this project, Apache Axis had the best performance overall. When processing with small amounts of data, Hessian performed just as well as Apache Axis. In contrast, it had the poorest performance of the four frameworks when processing larger amounts of data. However, Hessian required the least amount of code and used the least memory and CPU capacity. Thus, for developing a small application with small amounts of data being processed, such as is the case with mobile devices, Hessian would be a viable choice, due to its high performance and low price. If developing a big application that processes a large amount of data, Apache Axis appears to be a better solution. The benefit of JBossWS is it is more compatible with other JBoss projects or applications using JBoss Application Server.

REFERENCES

Print Publications:

[Ching01]
Ching, A., and A. Wagner, "Understanding Performance Testing," Technical Report, Microsoft Developer Network, (February, 2001), pp. 52-55.

[Curbera02]
Curbera, F., M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," IEEE Internet Computing, (March, 2002), pp. 86-93.

[Curbera03]
Curbera F., R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The Next Step in Web Services," Communications of the ACM, (October, 2003), pp. 29-34.

[Decker00]
Decker, S., S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks, "The Semantic Web: The Roles of XML and RDF," IEEE Internet Computing, (September-October, 2000), pp. 63-73.

[Hendler01]
Hendler, J., "Agents and the Semantic Web," IEEE Intelligent Systems, (March-April, 2001), pp. 30-37.

[McIlraith01]
Sheila A. McIlraith, T. C. Son, and H. Zeng , "Semantic Web Services," IEEE Intelligent Systems, (March-April, 2001), pp. 46-53.

[Narayanan02]
Narayanan, S., and S. A. McIlraith, "Simulation, Verification and Automated Composition of Web Services," International World Wide Web Conference, (2002), pp. 77-88.

[Yang02]
Yang, J., and M. P. Papazoglou, "Web Component: A Substrate for Web Service Reuse and Composition," Lecture Notes in Computer Science, 2348 (2002), pp. 21-36.

Electronic Sources:

[GLM06]
General Linear Model, http://www.socialresearchmethods.net/kb/genlin.php, last revision October 10, 2006, last accessed November 20, 2007.

[HBWSP06]
Hessian Binary Web Service Protocol, http://hessian.caucho.com/, last revision 2006, last accessed November 20, 2007.

[JBossWS07]
JBossWS, http://jbws.dyndns.org/mediawiki/index.php?title=JBossWS, last revision November 10, 2007, last accessed November 20, 2007.

[SAS07]
SAS, http://www.sas.com/, last revision 2007, last accessed November 20, 2007.

[WSA06]
Web Services – Axis, http://ws.apache.org/axis/, last revision April 22, 2006, last accessed November 20, 2007.

[XFire07]
Codehaus XFire, http://xfire.codehaus.org/, last revision May 3, 2007, last accessed November 20, 2007.

APPENDIX A

APACHE AXIS SERVER CODE: SENDMB

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                       *
*  Program name: sendMB.java                                            *

package mine;

public class sendMB
{
      public String request(int num)
      {
            byte[] oneMB = new byte[1048576];
            for (int i = 0; i<oneMB.length; i++)
                  oneMB[i]=(byte)'1';
            String theString = new String(oneMB);
            for (int i = 0; i<num-1; i++)
                  theString += new String(oneMB);
            return theString;
      }
}
```

APPENDIX B

APACHE AXIS SERVER CODE: CPUMEM

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                       *
*  Program name: CPUmem.java                                            *

package mine;

import java.lang.*;
import java.io.*;
import java.util.*;

public class CPUmem
{
      public String percentage()
      {
            String memUsage = null;
            String cpuUsage = null;

            try
            {

                  Runtime rt = Runtime.getRuntime();
                  Process proc = rt.exec("ps u -e");

                  InputStream inputstream = proc.getInputStream();
                  InputStreamReader inputstreamreader = new
InputStreamReader(inputstream);
                  BufferedReader bufferedreader = new
BufferedReader(inputstreamreader);

                  String line;
                  for (int i=0;(line = bufferedreader.readLine())!=
null;i++)
                  {
                        String spliting[] = line.split(" ");
                        if (spliting[0].equals("tomcat"))
                        {
                              int counter = 0;
                              for (int j=0; j<spliting.length; j++)
                              {
                                    if (!spliting[j].equals(""))
                                    {
                                          counter++;
                                          if (counter==3)
                                                cpuUsage=spliting[j];
                                          else if (counter==4)
                                          {
                                                memUsage=spliting[j];
```

```
                                                    break;
                                                }
                                            }
                                        }
                                        break;
                                    }
        }
        //System.out.println("CPU Usage: "+cpuUsage);
        //System.out.println("Memory Usage: "+memUsage);
        return cpuUsage+" "+memUsage;
              }
              catch(Exception e)
              {
                    e.printStackTrace();
              }
    return cpuUsage+" "+memUsage;
        }
}
```

APPENDIX C

APACHE AXIS CLIENT CODE: CLIENT

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                         *
*  Program name: client.java                                              *

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;

import javax.xml.namespace.QName;
import javax.xml.rpc.ParameterMode;
import java.text.DecimalFormat;

import java.io.*;
import java.util.*;

public class client
{
      static DecimalFormat decimal = new DecimalFormat(".000");

      public static void main(String [] args)
      {
            try
            {
                  Options options = new Options(args);

                  String endpointURL = options.getURL();
                  String MBToSend;
                  String numOfClients;

                  args = options.getRemainingArgs();
                  if ((args == null) || (args.length < 2))
                  {
                      MBToSend = "1";
                      numOfClients = "1";
                  }
                  else
                  {
                      MBToSend = args[0];
                      numOfClients = args[1];
                  }
                  int MB = Integer.parseInt(MBToSend);
                  int num = Integer.parseInt(numOfClients);

                  simClients sim = new simClients(MB,endpointURL);
                  //threading to simulate multiple users
                  for (int i=0; i<num; i++)
```

```
                        new simClients(MB,endpointURL).start();
                    try
                    {
                        while (simClients.counter<num)
                        {
                            Thread.sleep(1000);
                        }
                    }
                    catch(Exception e)
                    {
                        e.printStackTrace();
                    }

                    long response_time = simClients.total_time/num;
                    float usageCPU = simClients.total_CPU/num;
                    float usageMem = simClients.total_mem/num;
                    System.out.println("Average Response Time =
"+response_time+" milliseconds");
                    System.out.println("Average CPU Usage = "+usageCPU+"
%");
                    System.out.println("Average Memory Usage =
"+usageMem+" %");
                    System.out.println("Total CPU change =
"+simClients.total_CPU_change+" %");
                    System.out.println("Average Memory Change =
"+simClients.total_mem_change+" %");
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
        }
}

class simClients extends Thread
{
        int MB = 0;
        String endpointURL = new String();

        long start_time = 0;
        long end_time = 0;
        long process_time = 0;
        static long total_time = 0;
        static float total_CPU = 0;
        static float total_mem = 0;
        static float total_CPU_change = 0;
        static float total_mem_change = 0;
        static int counter = 0;
        static ArrayList each_time = new ArrayList();

        public simClients(int mb, String url)
        {
                MB = mb;
                endpointURL = url;
        }
```

```
	public void run()
	{
		try
		{
			start_time = System.currentTimeMillis();


			Service service = new Service();
			Call call    = (Call) service.createCall();

			call.setTargetEndpointAddress( new
java.net.URL(endpointURL+"CPUmem") );
			call.setOperationName( new QName("http://CPUmem",
"percentage") );

	call.setReturnType( org.apache.axis.encoding.XMLType.XSD_STRING
);

			String startUsage = (String) call.invoke( new
Object[] {} );
			String splitStartUsage[] = startUsage.split(" ");

			service = new Service();
			call = (Call) service.createCall();

			call.setTargetEndpointAddress( new
java.net.URL(endpointURL+"sendMB") );
			call.setOperationName( new QName("http://sendMB",
"request") );
			call.addParameter( "arg1", XMLType.XSD_INT,
ParameterMode.IN);

	call.setReturnType( org.apache.axis.encoding.XMLType.XSD_STRING
);

			start_time = System.currentTimeMillis();
			String ret = (String) call.invoke( new Object[]
{MB} );
			end_time = System.currentTimeMillis();

			service = new Service();
			call = (Call) service.createCall();

			call.setTargetEndpointAddress( new
java.net.URL(endpointURL+"CPUmem") );
			call.setOperationName( new QName("http://CPUmem",
"percentage") );

	call.setReturnType( org.apache.axis.encoding.XMLType.XSD_STRING
);

			String usage = (String) call.invoke( new Object[]
{} );
			String splitUsage[] = usage.split(" ");
```

```
                    total_CPU_change += Float.parseFloat(splitUsage[0])-
Float.parseFloat(splitStartUsage[0]);
                    total_CPU += Float.parseFloat(splitUsage[0]);
                    total_mem_change += Float.parseFloat(splitUsage[1])-
Float.parseFloat(splitStartUsage[1]);
                    total_mem += Float.parseFloat(splitUsage[1]);

                    process_time = end_time - start_time;
                    total_time += process_time;
                    counter++;
              }
              catch(Exception e)
              {
                    e.printStackTrace();
              }
        }
}
```

APPENDIX D

RESIN HESSIAN SERVER CODE: SENDMB

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                        *
*  Program name: sendMB.java                                             *

package resinProject;

public class sendMB implements sendAPI
{
      public String request(int num)
      {
            byte[] oneMB = new byte[1048576];
            for (int i = 0; i<oneMB.length; i++)
                  oneMB[i]=(byte)'1';
            String theString = new String(oneMB);
            for (int i = 0; i<num-1; i++)
                  theString += new String(oneMB);
            return theString;
      }
}
```

APPENDIX E

RESIN HESSIAN SERVER CODE: CPUMEM

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                       *
*  Program name: CPUmem.java                                            *

package resinProject;

import java.lang.*;
import java.io.*;
import java.util.*;

public class CPUmem
{
      public String percentage()
      {
            String memUsage = null;
            String cpuUsage = null;

            try
            {

                  Runtime rt = Runtime.getRuntime();
                  Process proc = rt.exec("ps u -e");

                  InputStream inputstream = proc.getInputStream();
                  InputStreamReader inputstreamreader = new
InputStreamReader(inputstream);
                  BufferedReader bufferedreader = new
BufferedReader(inputstreamreader);

                  String line;
                  for (int i=0;(line = bufferedreader.readLine())!=
null;i++)
                  {
                        String spliting[] = line.split(" ");
                        if (spliting[0].equals("resin"))
                        {
                              int counter = 0;
                              for (int j=0; j<spliting.length; j++)
                              {
                                    if (!spliting[j].equals(""))
                                    {
                                          counter++;
                                          if (counter==3)
                                                cpuUsage=spliting[j];
                                          else if (counter==4)
                                          {
                                                memUsage=spliting[j];
```

```
                                                    break;
                                                }
                                            }
                                        }
                                        break;
                                    }
        }
        return cpuUsage+" "+memUsage;
              }
              catch(Exception e)
              {
                    e.printStackTrace();
              }
    return cpuUsage+" "+memUsage;
        }
}
```

APPENDIX F

RESIN HESSIAN CLIENT CODE: CLIENT

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                       *
*  Program name: client.java                                            *

package resinProject;

import com.caucho.hessian.client.HessianProxyFactory;

import java.text.DecimalFormat;

import java.io.*;
import java.util.*;

public class client
{
      static DecimalFormat decimal = new DecimalFormat(".000");

      public static void main(String [] args)
      {
            try
            {
                  String endpointURL;
                  String MBToSend;
                  String numOfClients;

                  if ((args == null) || (args.length < 3))
                  {
                        MBToSend = "1";
                        numOfClients = "1";
                        endpointURL =
"http://callisto.ccec.unf.edu:8088/n00168553";
                  }
                  else
                  {
                        endpointURL = args[0];
                        MBToSend = args[1];
                        numOfClients = args[2];
                  }
                  int MB = Integer.parseInt(MBToSend);
                  int num = Integer.parseInt(numOfClients);

                  simClients sim = new simClients(MB,endpointURL);
                  //threading to simulate multiple users
                  for (int i=0; i<num; i++)
                        new simClients(MB,endpointURL).start();
                  try
                  {
```

```
                        while (simClients.counter<num)
                        {
                              Thread.sleep(1000);
                        }
                  }
                  catch(Exception e)
                  {
                        e.printStackTrace();
                  }

                  long response_time = simClients.total_time/num;
                  float usageCPU = simClients.total_CPU/num;
                  float usageMem = simClients.total_mem/num;
                  System.out.println("Average Response Time =
"+response_time+" milliseconds");
                  System.out.println("Average CPU Usage = "+usageCPU+"
%");
                  System.out.println("Average Memory Usage =
"+usageMem+" %");
                  System.out.println("Total CPU change =
"+simClients.total_CPU_change+" %");
                  System.out.println("Average Memory Change =
"+simClients.total_mem_change+" %");
            }
            catch (Exception e)
            {
                  e.printStackTrace();
            }
      }
}

class simClients extends Thread
{
      int MB = 0;
      String endpointURL = new String();

      long start_time = 0;
      long end_time = 0;
      long process_time = 0;
      static long total_time = 0;
      static float total_CPU = 0;
      static float total_mem = 0;
      static float total_CPU_change = 0;
      static float total_mem_change = 0;
      static int counter = 0;
      static ArrayList each_time = new ArrayList();

      public simClients(int mb, String url)
      {
            MB = mb;
            endpointURL = url;
      }

      public void run()
      {
            try
```

```
            {
                  HessianProxyFactory factory = new
HessianProxyFactory();
                  HessianProxyFactory factory2 = new
HessianProxyFactory();

      CPUmemAPI CPUmem = (CPUmemAPI) factory.create(CPUmemAPI.class,
endpointURL+"/CPUmem");
      sendAPI send = (sendAPI) factory2.create(sendAPI.class,
endpointURL+"/sendMB");

                  String startUsage = CPUmem.percentage();
                  String splitStartUsage[] = startUsage.split(" ");

                  start_time = System.currentTimeMillis();
                  String ret = send.request(MB);
                  end_time = System.currentTimeMillis();

                  String usage = CPUmem.percentage();
                  String splitUsage[] = usage.split(" ");

                  total_CPU_change += Float.parseFloat(splitUsage[0])-
Float.parseFloat(splitStartUsage[0]);
                  total_CPU += Float.parseFloat(splitUsage[0]);
                  total_mem_change += Float.parseFloat(splitUsage[1])-
Float.parseFloat(splitStartUsage[1]);
                  total_mem += Float.parseFloat(splitUsage[1]);

                  process_time = end_time - start_time;
                  total_time += process_time;
                  counter++;
            }
            catch(Exception e)
            {
                  e.printStackTrace();
            }
      }
}
```

APPENDIX G

JBOSSWS SERVER CODE: SENDMBBEAN

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                       *
*  Program name: sendMBBean.java                                        *

package jboss.project;

import javax.ejb.Remote;
import javax.ejb.Stateless;
import javax.jws.WebService;

@Stateless
@WebService(endpointInterface = "jboss.project.sendMB")
@Remote(sendMB.class)
public class sendMBBean {
      public String request(int num) {
            byte[] oneMB = new byte[1048576];
            for (int i = 0; i<oneMB.length; i++)
                  oneMB[i]=(byte)'1';
            String theString = new String(oneMB);
            for (int i = 0; i<num-1; i++)
                  theString += new String(oneMB);
            return theString;
      }
}
```

APPENDIX H

JBOSSWS SERVER CODE: CPUMEMBEAN

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                       *
*  Program name: CPUmemBean.java                                        *

package jboss.project;

import java.lang.*;
import java.io.*;
import java.util.*;

import javax.ejb.Remote;
import javax.ejb.Stateless;
import javax.jws.WebService;

@Stateless
@WebService(endpointInterface = "jboss.project.CPUmem")
@Remote(CPUmem.class)
public class CPUmemBean {
      public String percentage() {
            String memUsage = null;
            String cpuUsage = null;

            try
            {

                  Runtime rt = Runtime.getRuntime();
                  Process proc = rt.exec("ps u -e");

                  InputStream inputstream = proc.getInputStream();
                  InputStreamReader inputstreamreader = new
InputStreamReader(inputstream);
                  BufferedReader bufferedreader = new
BufferedReader(inputstreamreader);

                  String line;
                  for (int i=0;(line = bufferedreader.readLine())!=
null;i++)
                  {
                        String spliting[] = line.split(" ");
                        if (spliting[0].equals("jboss"))
                        {
                              int counter = 0;
                              for (int j=0; j<spliting.length; j++)
                              {
                                    if (!spliting[j].equals(""))
                                    {
                                          counter++;
```

```
                                        if (counter==3)
                                              cpuUsage=spliting[j];
                                        else if (counter==4)
                                        {
                                              memUsage=spliting[j];
                                              break;
                                        }
                                    }
                                }
                                break;
                            }
                    }
                    return cpuUsage+" "+memUsage;
                          }
                          catch(Exception e)
                          {
                                e.printStackTrace();
                          }
              return cpuUsage+" "+memUsage;
                    }
}
```

APPENDIX I

JBOSSWS CLIENT CODE: CLIENT

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                       *
*  Program name: Client.java                                            *

package jboss.project;

import java.io.*;
import java.util.*;

import java.net.URL;

import javax.xml.namespace.QName;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;

public class Client {
      public static void main(String[] args) throws Exception
      {
            try
            {
                  String endpointURL;
                  String MBToSend;
                  String numOfClients;

                  if ((args == null) || (args.length < 3))
                  {
                        MBToSend = "1";
                        numOfClients = "1";
                        endpointURL =
"http://callisto.ccec.unf.edu:18080";
                  }
                  else
                  {
                        endpointURL = args[0];
                        MBToSend = args[1];
                        numOfClients = args[2];
                  }
                  int MB = Integer.parseInt(MBToSend);
                  int num = Integer.parseInt(numOfClients);

                  simClients sim = new simClients(MB,endpointURL);
                  //threading to simulate multiple users
                  for (int i=0; i<num; i++)
                        new simClients(MB,endpointURL).start();

                  while (simClients.counter<num)
                  {
```

```
                        Thread.sleep(1000);
                    }

                    long response_time = simClients.total_time/num;
                    float usageCPU = simClients.total_CPU/num;
                    float usageMem = simClients.total_mem/num;
                    System.out.println("Average Response Time =
"+response_time+" milliseconds");
                    System.out.println("Average CPU Usage = "+usageCPU+"
%");
                    System.out.println("Average Memory Usage =
"+usageMem+" %");
                    System.out.println("Total CPU change =
"+simClients.total_CPU_change+" %");
                    System.out.println("Average Memory Change =
"+simClients.total_mem_change+" %");

            }
            catch (Exception e)
            {
                    e.printStackTrace();
            }
      }
)

class simClients extends Thread
{
      int MB = 0;
      String endpointURL = new String();

      long start_time = 0;
      long end_time = 0;
      long process_time = 0;
      static long total_time = 0;
      static float total_CPU = 0;
      static float total_mem = 0;
      static float total_CPU_change = 0;
      static float total_mem_change = 0;
      static int counter = 0;
      static ArrayList each_time = new ArrayList();

      public simClients(int mb, String url)
      {
            MB = mb;
            endpointURL = url;
      }

      public void run()
      {
            try
            {
                    URL url = new
URL(endpointURL+"/sendMBBeanService/sendMBBean?wsdl");
                    URL url2 = new
URL(endpointURL+"/CPUmemBeanService/CPUmemBean?wsdl");
```

```
                    QName qname = new
QName("http://project.jboss/","sendMBBeanService");
                    QName qname2 = new
QName("http://project.jboss/","CPUmemBeanService");

                    ServiceFactory factory =
ServiceFactory.newInstance();
                    ServiceFactory factory2 =
ServiceFactory.newInstance();
                    Service remote = factory.createService(url, qname);
                    Service remote2 = factory2.createService(url2,
qname2);

                   sendMB proxy = (sendMB) remote.getPort(sendMB.class);
                    CPUmem proxy2 = (CPUmem)
remote2.getPort(CPUmem.class);

                    String startUsage = proxy2.percentage();
                    String splitStartUsage[] = startUsage.split(" ");

                    start_time = System.currentTimeMillis();
                    String ret = proxy.request(MB);
                    end_time = System.currentTimeMillis();

                    String usage = proxy2.percentage();
                    String splitUsage[] = usage.split(" ");

                    total_CPU_change += Float.parseFloat(splitUsage[0])-
Float.parseFloat(splitStartUsage[0]);
                    total_CPU += Float.parseFloat(splitUsage[0]);
                    total_mem_change += Float.parseFloat(splitUsage[1])-
Float.parseFloat(splitStartUsage[1]);
                    total_mem += Float.parseFloat(splitUsage[1]);

                    process_time = end_time - start_time;
                    total_time += process_time;
                    counter++;
            }
            catch(Exception e)
            {
                    e.printStackTrace();
            }
      }
}
```

APPENDIX J

XFIRE SERVER CODE: SENDMBIMPL

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                        *
*  Program name: sendMBImpl.java                                         *

package xfire.project;

/** XFire WebServices implementation class.
 */
public class SendMBImpl implements SendMB
{
      public String request(int num)
      {
            byte[] oneMB = new byte[1048576];
            for (int i = 0; i<oneMB.length; i++)
                  oneMB[i]=(byte)'1';
            String theString = new String(oneMB);
            for (int i = 0; i<num-1; i++)
                  theString += new String(oneMB);
            return theString;
      }
}
```

APPENDIX K

XFIRE SERVER CODE: CPUMEMIMPL

```
*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *
*                                                                       *
*  Program name: CPUmemImpl.java                                        *

package xfire.project;

import java.lang.*;
import java.io.*;
import java.util.*;

/** XFire WebServices implementation class.
 */
public class CPUmemImpl implements CPUmem
{
      public String percentage()
      {
            String memUsage = null;
            String cpuUsage = null;

            try
            {

                  Runtime rt = Runtime.getRuntime();
                  Process proc = rt.exec("ps u -e");

                  InputStream inputstream = proc.getInputStream();
                  InputStreamReader inputstreamreader = new
InputStreamReader(inputstream);
                  BufferedReader bufferedreader = new
BufferedReader(inputstreamreader);

                  String line;
                  for (int i=0;(line = bufferedreader.readLine())!=
null;i++)
                  {
                        String spliting[] = line.split(" ");
                        if (spliting[0].equals("tomcat"))
                        {
                              int counter = 0;
                              for (int j=0; j<spliting.length; j++)
                              {
                                    if (!spliting[j].equals(""))
                                    {
                                          counter++;
                                          if (counter==3)
                                                cpuUsage=spliting[j];
                                          else if (counter==4)
```

```
                                                {
                                                      memUsage=spliting[j];
                                                      break;
                                                }
                                          }
                                    }
                                    break;
                              }
      }
      return cpuUsage+" "+memUsage;
            }
            catch(Exception e)
            {
                  e.printStackTrace();
            }
    return cpuUsage+" "+memUsage;
      }
}
```

APPENDIX L

XFIRE CLIENT CODE: CLIENT

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                       *
*  Program name: Client.java                                            *

import xfire.project.*;

import java.net.MalformedURLException;
import java.text.DecimalFormat;
import java.text.NumberFormat;

import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;
import org.codehaus.xfire.XFire;
import org.codehaus.xfire.XFireFactory;
import org.apache.log4j.Logger;

import java.io.*;
import java.util.*;

public class Client
{

      public static void main(String args[])
      {
            try {
                  String endpointURL;
                  String MBToSend;
                  String numOfClients;

                  if ((args == null) || (args.length < 3))
                  {
                        MBToSend = "1";
                        numOfClients = "1";
                        endpointURL =
"http://callisto.ccec.unf.edu:8080/n00168553/services";
                  }
                  else
                  {
                        endpointURL = args[0];
                        MBToSend = args[1];
                        numOfClients = args[2];
                  }
                  int MB = Integer.parseInt(MBToSend);
                  int num = Integer.parseInt(numOfClients);

                  simClients sim = new simClients(MB,endpointURL);
```

```
                    //threading to simulate multiple users
                    for (int i=0; i<num; i++)
                          new simClients(MB,endpointURL).start();
                    try
                    {
                          while (simClients.counter<num)
                          {
                                Thread.sleep(1000);
                          }
                    }
                    catch(Exception e)
                    {
                          e.printStackTrace();
                    }

                    long response_time = simClients.total_time/num;
                    float usageCPU = simClients.total_CPU/num;
                    float usageMem = simClients.total_mem/num;
                    System.out.println("Average Response Time =
"+response_time+" milliseconds");
                    System.out.println("Average CPU Usage = "+usageCPU+"
%");
                    System.out.println("Average Memory Usage =
"+usageMem+" %");
                    System.out.println("Total CPU change =
"+simClients.total_CPU_change+" %");
                    System.out.println("Average Memory Change =
"+simClients.total_mem_change+" %");

              } catch (Exception e){
                    System.out.println("EXCEPTION: main(): " +
e.toString());
              }
        }
}

class simClients extends Thread
{
        private static Logger log = Logger.getLogger(Client.class);

        int MB = 0;
        String endpointURL = new String();

        long start_time = 0;
        long end_time = 0;
        long process_time = 0;
        static long total_time = 0;
        static float total_CPU = 0;
        static float total_mem = 0;
        static float total_CPU_change = 0;
        static float total_mem_change = 0;
        static int counter = 0;
        static ArrayList each_time = new ArrayList();

        public simClients(int mb, String url)
        {
```

```
            MB = mb;
            endpointURL = url;
      }

      public void run()
      {
            try
            {
                  String startUsage = getCPUmem(endpointURL);
                  String splitStartUsage[] = startUsage.split(" ");

                  start_time = System.currentTimeMillis();
                  String ret = sendMB(MB,endpointURL);
                  end_time = System.currentTimeMillis();

                  String usage = getCPUmem(endpointURL);
                  String splitUsage[] = usage.split(" ");

                  total_CPU_change += Float.parseFloat(splitUsage[0])-
Float.parseFloat(splitStartUsage[0]);
                  total_CPU += Float.parseFloat(splitUsage[0]);
                  total_mem_change += Float.parseFloat(splitUsage[1])-
Float.parseFloat(splitStartUsage[1]);
                  total_mem += Float.parseFloat(splitUsage[1]);

                  process_time = end_time - start_time;
                  total_time += process_time;
                  counter++;
            }
            catch(Exception e)
            {
                  e.printStackTrace();
            }
      }

      /* call the web service
      *
      */
      public String sendMB(int MB,String endpointURL)
            throws MalformedURLException, Exception{

            //create a metadata of the service
        Service serviceModel = new
ObjectServiceFactory().create(SendMB.class);
        //System.out.println("callSoapServiceLocal(): got service
model." );

        //create a proxy for the deployed service
            XFire xfire = XFireFactory.newInstance().getXFire();
        XFireProxyFactory factory = new XFireProxyFactory(xfire);
        //String serviceUrl = "xfire.local://Banking" ;
        String serviceUrl = endpointURL+"/SendMB";
        SendMB client = null;
        try {
                  client = (SendMB) factory.create(serviceModel,
serviceUrl);
```

```
        } catch (MalformedURLException e) {
            log.error("simClients.sendMB(): EXCEPTION: " +
e.toString());
        }

        //invoke the service
        String serviceResponse = "";
        try {
            serviceResponse = client.request(MB);
       } catch (Exception e){
                log.error("simClients.sendMB(): EXCEPTION: " +
e.toString());
                serviceResponse = e.toString();
        }
            log.debug("simClients.sendMB(): status=" +
serviceResponse);

            //return the response
            return serviceResponse;
      }
      public String getCPUmem(String endpointURL)
            throws MalformedURLException, Exception{

            //create a metadata of the service
        Service serviceModel = new
ObjectServiceFactory().create(CPUmem.class);
        //System.out.println("callSoapServiceLocal(): got service
model." );

        //create a proxy for the deployed service
            XFire xfire = XFireFactory.newInstance().getXFire();
        XFireProxyFactory factory = new XFireProxyFactory(xfire);
        //String serviceUrl = "xfire.local://Banking" ;
        String serviceUrl = endpointURL+"/CPUmem";
        CPUmem client = null;
        try {
                client = (CPUmem) factory.create(serviceModel,
serviceUrl);
        } catch (MalformedURLException e) {
            log.error("simClients.getCPUmem(): EXCEPTION: " +
e.toString());
        }

        //invoke the service
        String serviceResponse = "";
        try {
            serviceResponse = client.percentage();
       } catch (Exception e){
                log.error("simClients.getCPUmem(): EXCEPTION: " +
e.toString());
                serviceResponse = e.toString();
        }
            log.debug("simClients.getCPUmem(): status=" +
serviceResponse);

            //return the response
```

```
            return serviceResponse;
        }
    }
```

VITA

Je-Loon Yang has a Bachelor of Engineering degree from Chung Yuan Christian University in Information and Computer Engineering, 2005, and expects to receive a Master of Science degree in Computer and Information Sciences from the University of North Florida, December 2007. Dr. Sanjay Ahuja of the University of North Florida is serving as Je-Loon's project director. Je-Loon is currently employed as a teacher at the Jacksonville Chinese-American Cultural Association (JCCA) Chinese School in Jacksonville, Florida, and has been with the association for 9 months.

Je-Loon has ongoing interests in database administration and web application development, and has extensive experience with MySQL and Apache web server. Je-Loon has extensive programming experience in C, Java, and Ruby. Je-Loon's academic work has also included the use of SQL, XML, and HTML. Je-Loon is fluent in Mandarin Chinese and enjoys playing basketball.