EFFICIENCY OF SOAP VERSUS JMS

Dr. Roger Eggen
Department of Computer and Information Sciences
University of North Florida
Jacksonville, Florida, USA

Mr. Suresh Sunku
Department of Computer and Information Sciences
University of North Florida
Jacksonville, Florida, USA

Abstract

Parallel application programmers and software engineers have a variety of paradigms at their disposal. Some of the tools include traditional environments such as Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). Tools that are more modern involve the World Wide Web, including Simple Object Access Protocol (SOAP) [1], Java Messaging Service (JMS) [2], Common Object Request Broker Architecture (CORBA), and JavaSpaces. Java's object oriented approach proves to be a desirable platform for developing parallel distributed applications. This paper describes the benefits and expected efficiencies of using SOAP based communication compared to JMS.

Key Words

SOAP, JMS, Distributed Parallel Processing, WEB

1 Introduction

Computer scientists, along with software engineers, are enjoying a remarkable period of increased processing speeds and declining prices. Symmetric multiprocessors of up to 3 gigahertz are affordable and common. These new machines present unique challenges to the computing professional to develop software that adequately takes advantage of these computing systems. Clusters of such workstations provide a massively parallel computing environment. Wide varieties of applications challenge the software engineer to choose an environment that best solves the current challenge. While techniques and opportunities exist to create sophisticated programs, there is an increasing supply of challenging problems requiring ever faster and more capable computers.

The development of software tools to take advantage of fast, new processors traditionally lags behind the hardware production and development. However, new software programming tools have been implemented in an attempt to take advantage of the remote environments that exists. Most notably, the object-oriented paradigm through Java is recognized as a major component of WEB based distributed programming. The Java Development Environment (JDE) [1] includes several features that facilitate the production of stable, robust code for parallel processing. Threads provide an efficient and effective paradigm for utilizing tightly coupled systems. JMS [2,3] and SOAP [4] provide convenient tools for utilizing distributed systems over the World Wide Web, each with its own set of advantages and disadvantages.

2 Hardware

The hardware for this study consists of a cluster of homogeneous workstations all running RedHat Linux v7.2. The machines are all Intel based PCs consisting of single 500 MHz processors connected by 100 megabit fast Ethernet.

3 Software

The software is Java (TM) 2 Runtime Environment, Standard Edition (build 1.3.1) [5] available free from Sun. In order to keep the variables in performance evaluation low, the Java language environment is used with AXIS (an apache-soap project) implementation of SOAP and J2EE , a JMS API implementation. The application should be common to many problems, thus sorting was chosen. The application is well understood and therefore, introduces no variables into the evaluation.

4 JMS – Java Messaging Service

Messaging is a method of communication between software components or applications. [6] A messaging system is a peer-to-peer facility: a messaging client can send messages to, and receive messages from, any other client. Each client connects to a messaging agent that provides facilities for creating, sending, receiving, and reading messages without specific knowledge of each individual client.

In contrast, during the point-to-point communication, each client has a direct connection

to a specific remote processing agent. There is no consideration for workload or resources available at the remote machine. The client invokes the desired workers and simply waits for results from each. Figure 4 describes the basic communication between the workers and clients. Essentially, the client divides the task into subtasks equal to the number of workers, distributes a subtask to a worker through a thread, and waits for each worker to return its results.

JMS messaging enables distributed communication that is loosely coupled. A component sends a message to a destination, and the recipient can retrieve the message from the destination. However, the sender and the receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver, nor does the receiver need to know anything about the sender.

The sender and the receiver need to know only what message format and what destination to use. In this respect, messaging differs from tightly coupled technologies, such as Remote Method Invocation (RMI)[3] or message passing interface (MPI)[4], which require an application to know a remote application's methods and both the sender and receiver must be active and ready to communicate.

5 SOAP

The XML syntax of a SOAP message is uncomplicated. A SOAP message consists of an envelope containing: 1) an optional header containing zero or more header entries (sometimes ambiguously referred to as headers), 2) a body containing zero or more body entries, and 3) zero or more additional, non-standard elements.

The only body entry defined by SOAP is a “SOAP fault” which is used for reporting errors. Some of the XML elements of a SOAP message define namespaces, each in terms of a URL and a local name, and encoding styles; a standard one is defined by SOAP.

Header entries may be tagged with the following optional SOAP attributes: 1) an actor which specifies the intended recipient of the header entry in terms of a URL, and 2) an indication that specifies whether or not the intended recipient of the header entry is required to process the header entry. The SOAP message model is shown in Figure 1.

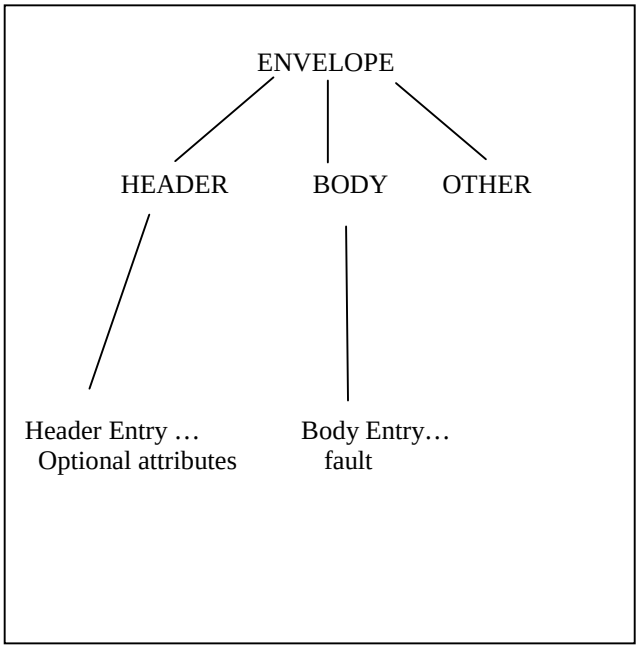

Figure 1
Soap Message Model

6 Distributed Processing Technologies

Figure 2 provides a review of various distributed processing technologies. The figure is provided to place SOAP and JMS in proper perspective and show the capabilities of each technology.[7,8,9]

Feature/Protocol	SOAP	Messaging Service(JMS)*	DCOM	RMI	CORBA
Format	XML(readable)	Binary	Binary	Binary	Binary
platform	Independent	Independent	Mostly Windows	Independent	Independent
Interface Specification	WSDL (Readable)	None	IDL(nor readable)	None	IDL(Not readable)
Programming Language	Independent	Independent	Independent **	JAVA	Independent
Ease Of Use	Easy	Easy***	Complex	Easy	Complex
Firewall Friendly	Yes	No	No	No	No****
Static Type Checking	No	No	Yes	Yes	Yes
Reliabilty, Security & Performance	Left to vendor	Left to vendor	Yes	Yes	Yes
Object Oriented	No	No	Yes	Yes	Yes

Figure 2
Comparison of Various Distributed Computing Protocols &Technologies

* Messaging service is not a protocol but it is a very

** Not implemented in JAVA

*** JMS is easy to learn but implementing a messaging product across multiple systems is complex.

**** HTTP tunneling is required.

7 Basics of SOAP

SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP is a fully formed and valid XML document as wire protocol or packet layout language and currently uses HTTP, but other transport mechanisms such as SMTP are under consideration. A vendor-agnostic open technology base supports loosely coupled applications. SOAP is a text-based protocol, and is therefore, easy to learn, debug, and implement. The protocol has the ability to transport through firewalls and is used in a Web services paradigm. [10]

The basics of SOAP communication are shown through the simple example in Figure 3.

```
Public interface Hello
{
      public String SayHelloTo(String
name);
}
<?xml version="1.0"?>
<Hello>
<sayHelloTo>
      <name>John</name>
</sayHelloTo>
</Hello>
```

Figure 3
SOAP Hello World

The corresponding response to the communication in Figure 3 is provided by Figure 4.

```
<?xml version="1.0"?>
<Hello>
<sayHelloToResponse>
      <message>Hello John, How are
you?</message>
</sayHelloToResponse>
</Hello>
```

Figure 4
SOAP Response

8 Basics of JMS

A JMS application is composed of the following parts:

- Provider

A JMS provider is a messaging system that implements the JMS interfaces and provides administrative and control features. The J2EE platform at release 1.3 includes a JMS provider.

- Client

JMS clients are the programs or components, written in the JavaTM programming language, that produce and consume messages.

- Messages

Messages are the objects that communicate information between JMS clients. Administered objects are preconfigured JMS objects created by an administrator for the use of clients. The two kinds of administered objects are destinations and connection factories.

Native clients are programs that use a messaging product's native client API instead of the JMS API. An application first created before the JMS API became available and subsequently modified is likely to include both JMS and native clients.

Administrative tools allow you to bind destinations and connection factories into a Java Naming and Directory InterfaceTM (JNDI) API namespace. A JMS client can then look up the administered objects in the namespace and then establish a logical connection to the same objects through the JMS provider.

9 The Research

The goal of the study is to compare performance of distributed computing using SOAP against the messaging protocol. Although both these technologies have distinct advantages over one another, these technologies are very viable solutions for many distributed computing environments.

The first application uses SOAP to distribute data to workers, each of whom process the data, and upon completion, return the results to the client. The client generates data sets, each of which are divided among N workers. Each worker sorts the data using an n-squared algorithm. The goal of this application is to affect the total time with both demanding execution in addition to the communication requirements. The total time of distribution, communication, and execution is recorded. Several size data sets are used as shown in Figure 7. Figure 5 is the conceptual view of the application.

A corresponding application within the API was developed using JMS. Again, several data sets were generated by a client and communicated to workers. Figure 6 shows the conceptual view of this application.

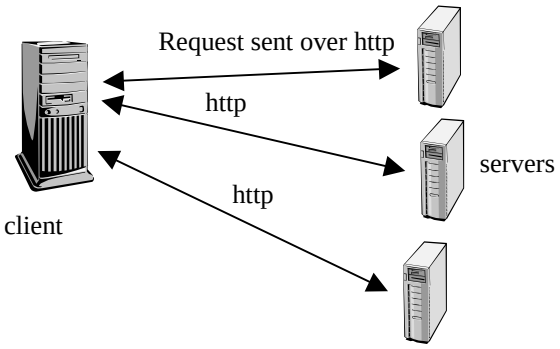

Figure 5
SOAP Communication

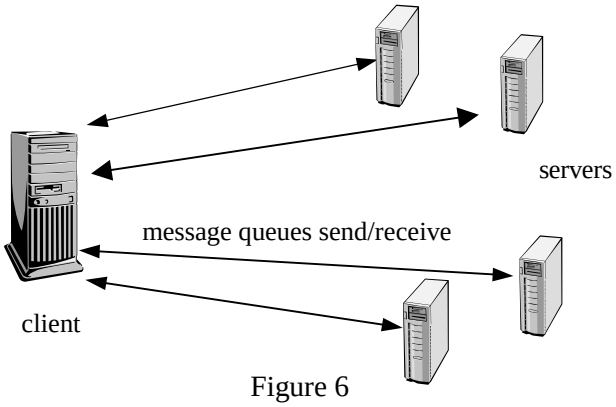

Figure 6
JMS Communication

Figure 5 shows the client sending messages to each of the servers. The client requests services from the servers by placing each request in a message queue. The servers are performing a sort similar to what was done in the SOAP environment.

10 The Results

To evaluate software systems for performing distributed parallel processing, the programs used for the testing should be real applications. For this reason, a sorting routine has been chosen. Since sorting is ubiquitous in many applications, it represents a problem with real world applicability and is well understood. We know there are no subtleties effecting performance in the algorithm. The sort routine chosen is a simple insertion sort that has both average and worst case performance in $O(n^2)$. This algorithm was chosen to measure not only communication times, but also significant execution demands at the servers.

Two applications were implemented in a consistent manner utilizing the features provided by SOAP and JMS. The programs are implemented in a consistent manner, to the degree allowed by the API.

The research evaluates the performance of SOAP and JMS using 1, 3, and 5 servers over data ranges of 500, 5,000, 50,000, and 100,000 integers. The data are distributed so that each server has the same amount of data. The servers do all the work while the client only distributes and receives data. All tests were executed under exactly the same conditions for each of the communication protocols during a time when the load on the dedicated network and servers was at a minimum. The table in Figure 7 summarizes the results.

Figures 8 and 9 show the benefits of distributed processing in JMS and SOAP. Clearly, parallelism is desired as significant speedup is realized.

Figure 10 displays the significant performance difference realized between the flexibility and robust environment provided by SOAP and that provided by JMS. In contrast, however, Figure 11 indicates that the rate of change between JMS and SOAP is relatively similar.

	1 SERVERS		3 SERVERS		5 SERVERS	
Time in MilliSeconds						
DATA	JMS	SOAP	JMS	SOAP	JMS	SOAP
500	2872	3816	2345	7723	2128	3922
5000	8278	13799	2429	11845	2932	11420
50000	658199	179513	76008	96071	29690	86691
100000	2627119	401105	295689	203780	108581	1177286

Figure 7
Execution Times for JMS and SOAP

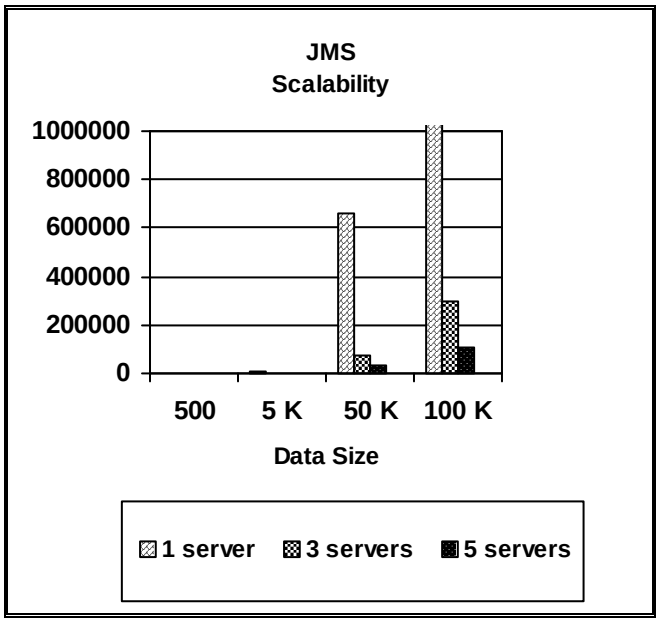

Figure 8
JMS times

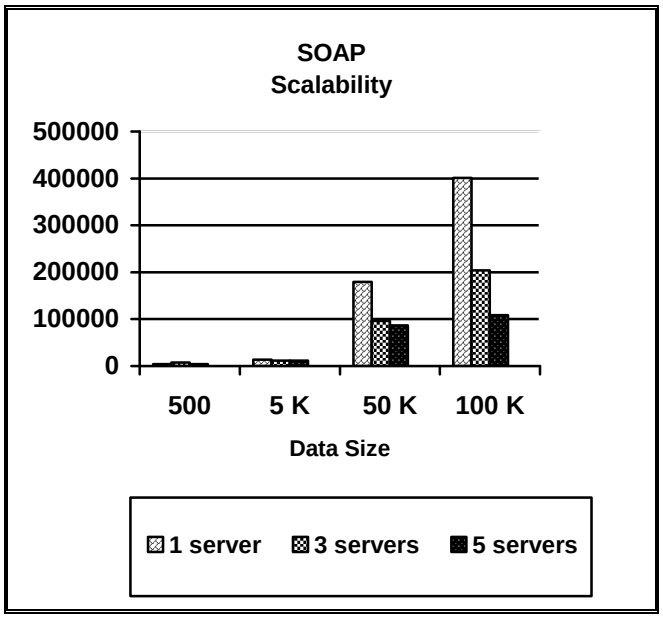

Figure 9
SOAP times

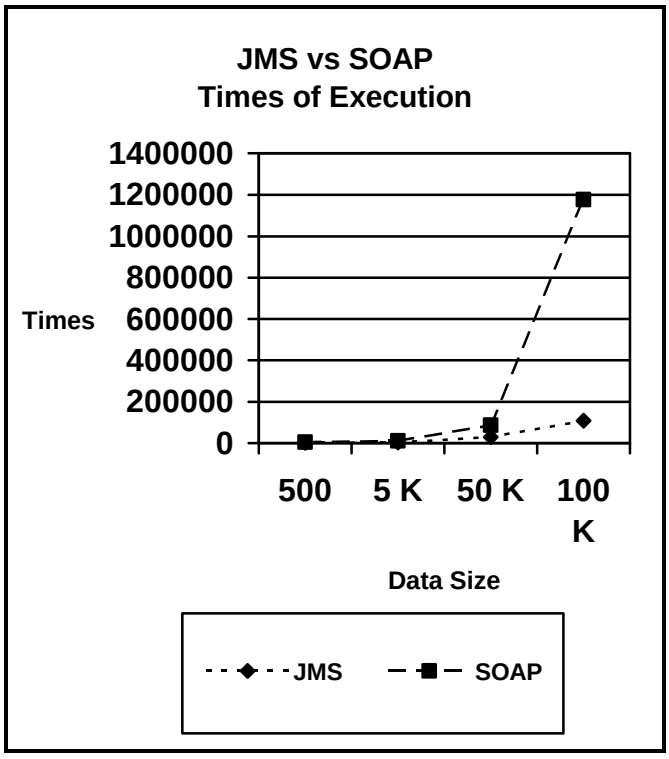

Figure 10
Execution times for JMS and SOAP

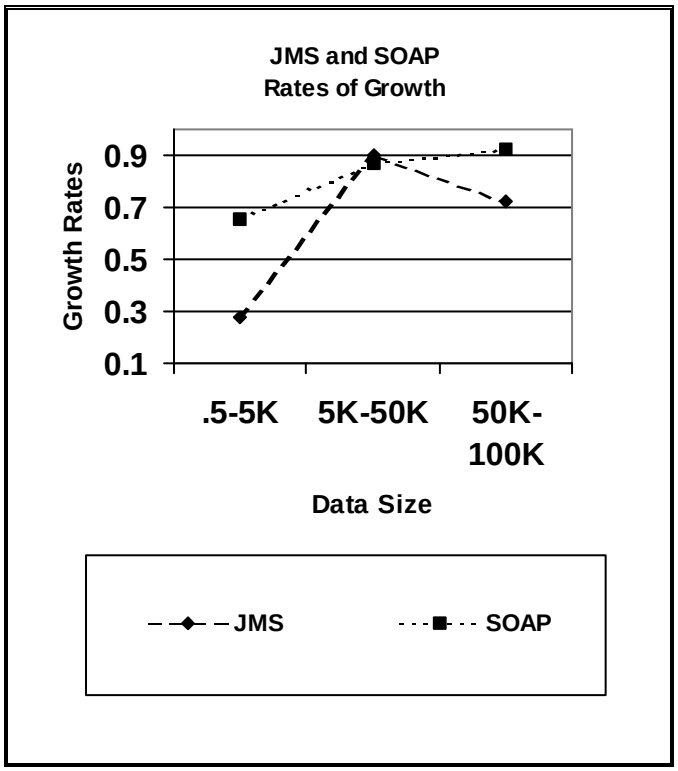

Figure 11
Rates of Growth

11 Summary and Conclusions

There are several views one can take from this data. First, we observe SOAP and JMS grow at nearly the same rate as indicated in Figure 9 and both technologies scale essentially the similarly as shown in Figures 6 and 7. Secondly, the overhead associated with SOAP requires more time for communication. In past studies of comparable middleware, communication dominated all work. [11] However, there is not a major difference between these two and the flexibility provided by SOAP may justify its additional time. The nature of the SOAP environment, which provides a wide degree of flexibility, comes with some cost. SOAP executes at nearly half the efficiency of JMS. This is not particularly significant until large data sizes are incurred. The time of communication for SOAP is nearly double that of JMS until large data sizes are used. The rates of growth (Figure 9) are nearly the same between the two technologies. Figure 9 shows comparable growth as data sizes grow by an order of magnitude and doubles, as shown in the move from 50K to 100K amounts of data.

Figure 5 deserves comment. Within the SOAP environment, using three servers, takes an inappropriate amount of time. With these data sizes, one should not be using three servers. There is insufficient amount of data to over come the cost of establishing communication. With one server, there is little communication and each server does more work, providing efficient results. With five servers, each server has “very little” work to do, thus reasonable performance is achieved. The application developer must weigh the communication cost in comparison with the degree of processing required to determine the optimal scalability.

The benefits of Object Oriented programming can be and probably should be realized when doing parallel distributed processing. Portability and flexibility continue to benefit application development in the parallel arena just as it does in the sequential program development. Both SOAP and JMS take advantage of the object oriented programming paradigm. For parallel distributed processing, all other factors being equal, JMS is slightly more efficient.

12 Future Research

JavaSpaces provide a Linda-like environment. Efficiency considerations of JavaSpaces in comparison with the above techniques should be

studied. Lite for Perl provides a SOAP environment offering some of the benefits realized by using Java and SOAP, JMS and JavaSpaces. Efficiencies of these environments warrant study.

Jini also provides an object-oriented environment for parallel distributed processing. A comparison between the Jini environment with SOAP and JMS would be interesting. JavaSpaces is implemented using Jini as the foundation. Determining the cost incurred by JavaSpaces as opposed to Jini would be an interesting study.

13 References

[1] http://java.sun.com

[2] http://java.sun.com/products/jms/

[3] S. Halter and S. Munroe, *Enterprise Java Performance* (Englewood Cliffs, New Jersey: Prentice Hall, 2001).

[4] http://xml.apache.org

[5] http://java.sun.com

[6] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers* (Englewood Cliffs, New Jersey: Prentice Hall, 1999).

[7] E. Harold, *Java Network Programming* (Sebastopol, California: O'Reilly Publishers, 1997).

[8] D. Lea, *Concurrent Programming in Java, Second Edition, Design Principles and Patterns* (Reading, Massachusetts: Addison Wesley Publishers, 2000).

[9] J. Farley, *Java Distributed Computing* (Sebastopol, California: O'Reilly Publishers, 1998).

[10] Hortsmann, Cay and Cornell, Gary, *Core Java Volume II Advanced Features*, (Reading, California: Prentice Hall, 2000).

[11] R. Eggen and M. Eggen, Efficiency of Distributed Parallel Processing using Java RMI, Sockets, and CORBA, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 2001, 888-893.