2003

Comparison of JavaSpace and CORBA Technologies

Anjani Kumar Jha

Comparison of JavaSpaces and CORBA Technologies

By

Anjani Kumar Jha

A Professional Option Project submitted to the Department of Computer and Information Sciences in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

December, 2003

The Professional Option Project "Comparison of JavaSpaces and CORBA Technologies" submitted by Anjani Kumar Jha 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 Dr. Sanjay Ahuja Project Director	12/5/03
Signature Deleted Dr. Charles N. Winton Graduate Director	12/5/03
Signature Deleted Dr. Judith L. Solano Department Chairperson	12/5/03

ACKNOWLEDGEMENT

I wish to express my gratitude to Dr. Sanjay Ahuja for his invaluable guidance and support throughout the project. Thanks are especially due to Dr. Bill Wilson of the Mathematics department, for providing me with his invaluable guidance and time on statistical methods to evaluate the test results. My lovely wife Rashmi and my little daughter Aditi provided me their constant support and encouragement while I pursued graduation. I also wish to thank my parents who motivate me every single moment of my life. Additionally, I would like to thank all those people who have helped me come up in life.

CONTENTS

FIGURES

TABLES

ABSTRACT

With computer industry increasingly moving towards network-centric systems, particularly the Internet, competing technologies that help design and develop such systems are fast emerging in the marketplace. The fundamental characteristics of a networked environment are heterogeneity, partial failure, latency and difficulty of "gluing together" multiple, independent processes into a robust, scalable application. JavaSpaces, a shared memory paradigm, provides high-level coordination mechanism for Java easing the burden of creating distributed systems. Large class of distributed problems can be approached using Javaspaces' simple framework. JavaSpaces allows processes to communicate even if each was wholly ignorant of the others. CORBA on the other hand is a standard developed by OMG that allows communication between objects written in different programming languages. It provides common message passing mechanism for interchanging data and discovering services. The purpose of this graduate project was to compare JavaSpaces and CORBA technologies by developing an Insertion Sort and comparing their response times. Javaspaces outpaced CORBA in terms of response time. These technologies make the implementation of distributed algorithms reasonably fault tolerant and highly scalable.

Chapter 1

INTRODUCTION

Client/server and multi-tier models operating within a single business enterprise have given way to an Internet/Web environment where services are provided by nodes scattered over a far-flung network. Next generation of network interaction is emerging that place unprecedented demands upon existing network technologies and architectures. For example, participants in one network will need to directly access and use the services provided by participants in another network. It is in this network environment - one of mind-numbing complexity driven by geometric increases in scale, rate of change, and multiplicity of participant interactions that technologies such as JavaSpaces and CORBA present competing options for software architects and distributed systems designers multiple and competing options and opportunities.

Distributed systems are hard to build. They require careful thinking about problems that do not occur in local computation. The fundamental characteristics of a networked environment such as partial failure, latency, and heterogeneity and the difficulty of "gluing together" multiple, independent processes into a robust, scalable application present the programmer with many challenges that don't arise when designing and building desktop applications. JavaSpaces technology is a simple, expressive, and powerful tool that eases the burden of creating distributed applications. Processes are loosely coupled- coupled communicating and synchronizing their activities using a persistent object store called a space, rather than through direct communication. [Arnold99]. Another technology that allows communication between objects that are written in different programming languages is Common Object Request Broker Architecture (CORBA). CORBA is an open, vendor-independent architecture and infrastructure for distributed object technology. CORBA standards define a common message passing mechanism for interchanging data and discovering services. It is widely used today as the basis for many mission-critical software applications. Objects do not talk directly to each other, they always use an object request broker (or ORB) to find out information and activating any requested services. CORBA technology uses an Interface Definition Language (or IDL) to specify the signatures of the messages and the types of the data different objects can send and understand [CapeSc02]. These technologies introduce new paradigm for developing distributed applications that are loosely coupled, dynamically and naturally scalable and fault tolerant.

For evaluating JavaSpaces and CORBA technologies both quantitatively and non-quantitatively, we have chosen a distributed, parallel application that can help understand the performance of the two technologies under various load conditions. We have implemented a parallel application that sorts a large array of positive integers of increasing sizes by partitioning the sort space into smaller components (smaller arrays) and dropping each such smaller "job" into the shared memory space and then each worker app which was free would pick up the job, do the sorting, drop off the result back into the shared memory space, and then the main thread would put back the individually sorted jobs into the proper overall order. On another dimension, we also increase the number of workers or processors to measure the performance of the applications developed in JavaSpaces and CORBA under these varying and increasing load conditions. The hardware platforms for both implementations are identical.

Chapter 2

JAVASPACES AND CORBA TECHNOLOGIES

2.1.1 JavaSpaces - A New Distributed Computing Model

Building distributed applications with conventional network tools usually entails passing messages between processes or invoking methods on remote objects. In JavaSpaces applications, in contrast, processes don't communicate directly, but instead coordinate their activities by exchanging objects through a *space,* or shared memory [Artima02]. JavaSpaces is a specification developed by SUN Microsystems that presents a model of interaction between (mostly) Java applications. Applications seek to exchange information in an asynchronous but transactional-secure manner and can use a space to coordinate the exchange.

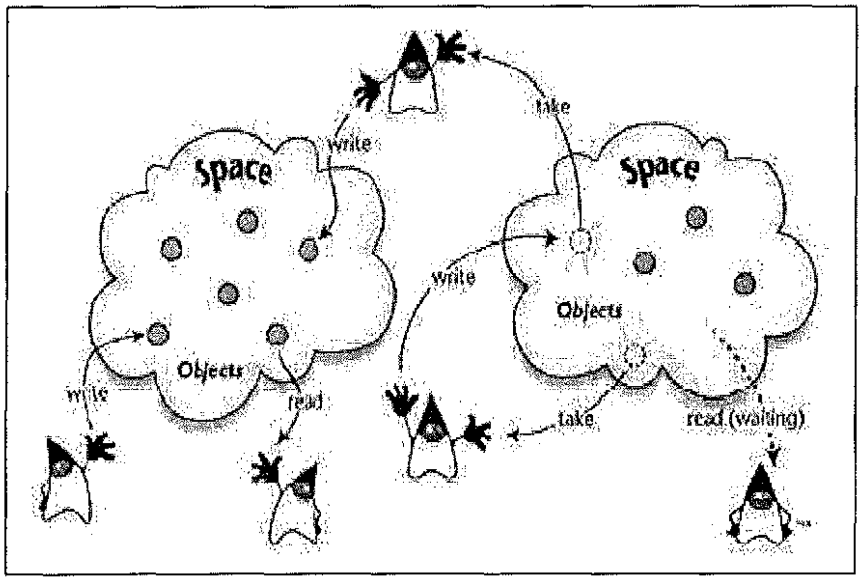


Figure 1: Flow of Objects between JavaSpaces

Figure 1 depicts several applications (the Duke images) interacting with two spaces. Each application can write objects (called Entries) to a space, read objects from a space, and take objects from a space (take means read + delete). In addition, applications may express interest in special entries arriving at a space by registering for notifications. The JavaSpaces API is very simple and elegant, and it provides software developers with a simple and effective tool to solve coordination problems in distributed systems, especially areas like parallel processing and distributed persistence. The developer can design the solution as a flow of objects rather than a traditional request/reply message based scenario. Combined with the fact that JavaSpaces is a Jini service, thus inheriting the dynamic nature of Jini, JavaSpaces is a killer model for programming highly dynamic distributed applications.

The JavaSpaces API consists of four main method types:

· Write() - writes an entry to a space.

· Read() - reads an entry from a space.

· Take() - reads an entry and deletes it from a space.

· Notify()- registers interest in entries arriving at a space.

In addition, the API enables JavaSpaces clients (applications) to provide optimization hints to the Space implementation (the method snapshot ()).

This minimal set of APIs reduces the learning curve of developers and encourages them to adopt the technology quickly. JavaSpaces enable full use of transactions, leveraging the default semantic of Jini Distributed Transactions model. This enables developers to build transactional-secure distributed applications using JavaSpaces as a coordination mechanism. The APIs themselves provide non-blocking versions, where a read() or take() operation may take a maximum timeout to wait before returning to the caller. This is very important for applications that cannot permit themselves to block for a long time or in the case that the space itself is in some kind of a deadlock. JavaSpaces also make extensive use of Jini leases, as it mandates that entries in the space be leased and thus, expire at a certain time unless renewed by a client. This prevents out-of-date entries, and saves the need for manual cleanup administration work [Arnold99].

2.1.2 GigaSpaces Platform

GigaSpaces Technologies has built an industrial-strength JavaSpaces implementation. This implementation is called "the GigaSpaces platform", or "GigaSpaces" in short. We selected GigaSpaces because it is freely available for evaluation. GigaSpaces is a 100% conforming and a 100% pure Java implementation of the JavaSpaces specification. Moreover, GigaSpaces blends naturally with Suns' implementation of the Jini API.

The application accesses the space API through a space proxy, which is embedded in the application JVM. This proxy is usually obtained by a lookup in a directory service, like a Jini Lookup service or a JNDI name space. The space proxy communicates with the server-side part of the space, which holds most of the logic and data of the space. The space itself may be an in-memory space or a persistent space. An in-memory space holds all its data in virtual memory. This results in fast access. However, memory spaces are bounded by the amount of virtual memory in the system, and are vulnerable to server crashes. A persistent space uses a DBMS backend to persist its data, while still caching some of the data in memory. Persistent spaces do not lose data as a result of server reboots/crashes and can hold a large amount of data. The server-side part of the space is shared among all applications that refer to the same logical space. This is how different applications can share and exchange information through the space. A GigaSpaces Container is a service that can contain and manage several spaces in one JVM. Spaces in the same container share resources in order to reduce resource consumption. The container is also responsible of registering spaces to directory services in the environment. A GigaSpaces Server can launch several services, like HTTP Service, Transaction Service, Lookup Service and GigaSpaces Container is one physical JVM. This is a single point of configuration for launching several services in a single physical process [Giga02].

2.2.1 CORBA Technology

The Common Object Request Broker Architecture (CORBA) is a standard for transparent communication between applications objects. [OMG03] Object Management Group (OMG) developed the CORBA standards, which is a non-profit industry consortium. It allows a distributed, heterogeneous collection of objects to inter-operate. Part of CORBA standard is the Interface Definition Language (IDL), which is an implementation-independent language for describing the interface of remote objects. Corba offers greater portability in that it isn't tied to one language, and as such, can integrate with legacy systems of the past written in older languages, as well as future languages that include support for CORBA.

CORBA applications are composed of objects, individual units of running software that combine functionality and data. There could be many instances of an object of a single type or only one instance. For each object type, we define an interface in OMG IDL. The interface is the syntax part of the contract that the server object offers to the clients that invoke it. Any client that wants to invoke an operation on the object must use this IDL interface to specify the operation it wants to perform, and to marshal the arguments that it sends. When the invocation reaches the target object, the same interface definition is used there to unmarshal the arguments so that the object can perform the requested operation with them. The interface definition is then used to marshal the results for their trip back, and to unmarshal them when they reach their destination. The IDL interface definition is independent of programming language, but maps to all of the popular programming languages via OMG standards: OMG has standardized mappings from IDL to several popular languages like C++, Java, COBOL, Python, etc. This separation of interface from implementation, enabled by OMG IDL, is the essence of CORBA - how it enables interoperability, with all of the transparencies we have mentioned. The interface to each object is defined very strictly. In contrast, the implementation of an object - its running code, and its data -

is hidden from the rest of the system (that is, encapsulated) behind a boundary that the client may not cross. Clients access objects only through their advertised interface, invoking only those operations that the object exposes through its IDL interface, with only those parameters (input and output) that are included in the invocation.

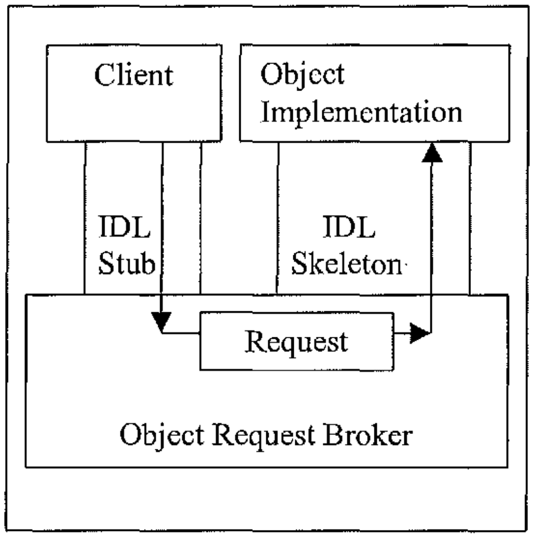


Figure 2: A request passing from client to object implementation

Figure 2 shows how everything fits together, at least within a single process: Compile the IDL into client stubs and object skeletons, and write the object and a client for it. Stubs and skeletons serve as proxies for clients and servers, respectively [OMG03]. Because IDL defines interfaces so strictly, the stub on the client side has no trouble meshing perfectly with the skeleton on the server side, even if the two are compiled into different programming languages, or even running on different ORBs from different vendors. In order to invoke the remote object instance, the client first obtains its object reference using Trader service or naming service. The client knows the type of object it's invoking and the client stub and object skeleton are generated from the same IDL. Although the ORB can tell from the object reference that the target object is remote, the client can not.

2.2.2 ORBACUS

Orbacus is a mature CORBA product that has been deployed around the world in mission critical systems in the telecommunications, finance, government, defense, aerospace and transportation industries. Orbacus is 'CORBA 2.5 compliant' and is designed for rapid development, deployment and support in the language of our choice C++ or Java; its small footprint allows it to be easily embedded into memory constrained applications [Orbacus03]. We chose ORBACUS for CORBA evaluation, as it is freely available for evaluation is an industry grade CORBA product.

Chapter 3

PROJECT DESCRIPTION

3.1 Overview

In this project, we implemented a distributed, parallel Insertion sort application because in our view such an algorithm significantly exercises the CPU computationally. The Insertion sort algorithm has a complexity of O (n^2). In the worst case scenario the algorithm may have demands for computing powers that can be truly met through a distributed and parallel application. Our application sorts a very large array of positive integers by partitioning the sort space into smaller components (smaller arrays) and dropping each such smaller "job" into the shared memory space and then each worker application which was free picked up the job, perform the sorting work, drop off the result back into the shared memory space, and then the main thread put back the individually sorted jobs into the proper overall order. The performance was measured by increasing the number of processor/worker or server as well as increasing the problem size by increasing the size of the array that needed sorting. We have also decided in our implementation to run one worker/server per node. Implementing the same application using JavaSpaces and CORBA allowed comparison of performance, ease of development, ease of maintenance, and portability across platforms between the two technologies.

3.2 Hardware

The hardware for this project 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.

3.3 Software

The software for the project consists of Java™ 2 Runtime environment, Standard Edition version 1.3.1. We used Java language for coding for the entire application to keep variables in performance evaluation to a minimum. We used GigaSpaces3.0 an implementation of JavaSpaces and ORBACUS4.1.2 an implementation of CORBA.

Chapter 4

TESTING METHODOLOGY

4.1 Testing method

Performance testing was implemented by recording the response time of each sort work performed using JavaSpaces and Corba applications. We increased the number of workers from one worker to multiple workers deployed to perform the same sort work. Later we doubled the size of the data for sorting. With this increased size of work, we again recorded the response time to sort this work using one worker and then changing the number of worker from one to two, four and then eight.

In case of Corba, the same methodologies described in the above paragraph was employed however in this case we were using servers that were performing sort work and passing the results back to the client which will then measure the response time and display the sorted data and response time. We plotted several graphs and recorded our inferences.

In addition we have also used statistical methods to evaluate our response time data and used the model to conclude our results from a statistical approach.

Chapter 5

RESULTS

5.1 Testing

We ran a series of executions for both the architectures by changing parameters for each run. We used 8K, 16K, 32K and 64K integers, which were randomly generated and used 1, 2, 4 and 8 workers/servers. The data are distributed so as each server has access to same amount of data. The servers do all the work while the client only distributes and collects data. All the executions were ran under similar conditions for both the technologies. We ran our measurements when the load on the network and servers was at a minimum.

The table below summarizes the observed data:

JavaSpaces Number of workers (Response time in ms)				
Input size	Ts(P=1)	Tp(P=2)	Tp(P=4)	Tp(P=8)
8K	4636	3726	3451	3573
16K	10744	6701	4898	4465
32K	34223	17529	10459	7488
64K	128508	47488	20003	12056

Table 1: Response time for JavaSpaces

CORBA Number of servers (Response time in ms)				
Input size	Ts(P=1)	Tp(P=2)	Tp(P=4)	Tp(P=8)
8K	7947	6438	5941	6399
16K	14747	8839	7395	7263
32K	39599	18816	11097	9282
64K	139199	66365	35280	20119

Table 2: Response time for CORBA

Note:

Ts: Response time when one worker was deployed to perform sort work

Tp: Response time when more than 1 worker was deployed to perform sort work

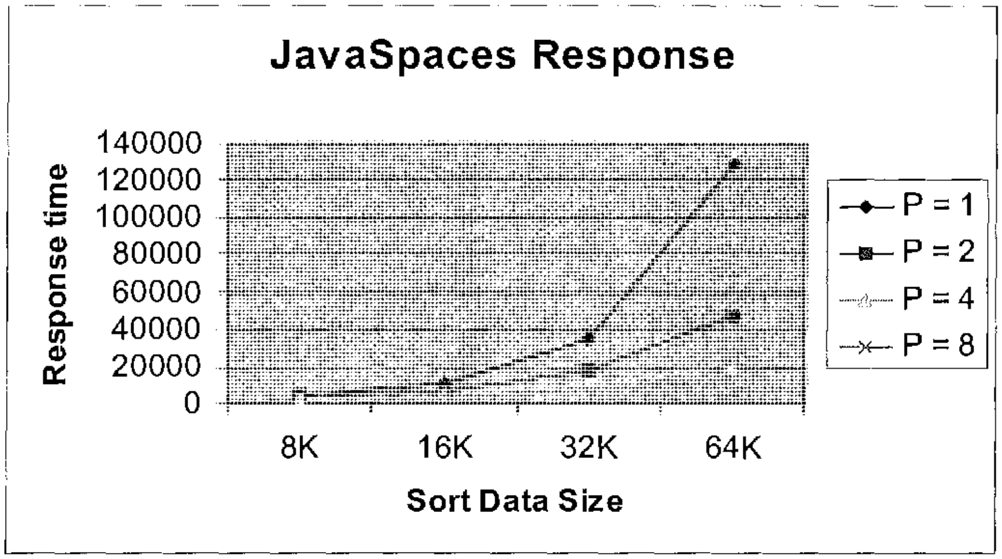


Figure 3: JavaSpaces Response with varying processors and varying data size

We also plotted graphs representing the measured response times with the data from Table 1 and Table 2.

The above graph (Figure 3) is a plot of response time with increasing sort work and number of workers for JavaSpaces implementation.

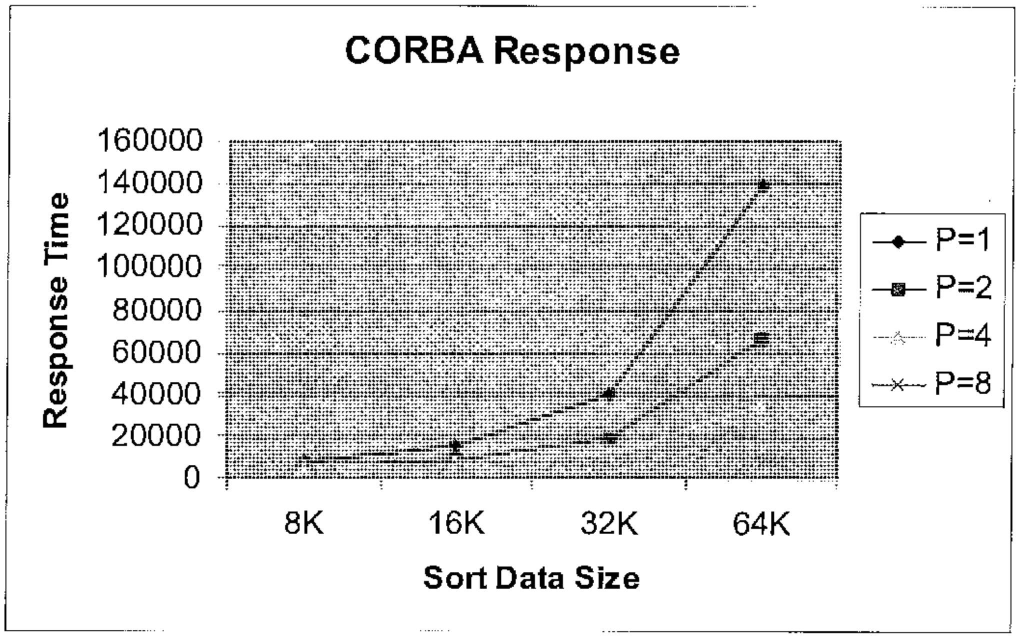


Figure 4: Corba response with varying processors and varying data size

The above figure is a plot of response time with increasing sort work and number of servers for Corba implementation.

5.2 Speed-up

For any parallel process, Speed-up is an important measured. It is defined as a ratio of time taken to process the same amount of work sequentially to time taken to process it in parallel. We have calculated and plotted graphs for speed-up in the following section.

The tables below show the speed-up for the observed data:

JavaSpaces Number of workers				
Input size	P=1	P=2	P=4	P=8
8K	1	1.2441	1.3433	1.2975
16K	1	1.6034	2.1935	2.4066
32K	1	1.9524	3.2721	4.5702
64K	1	2.7061	6.4245	10.6594

Table 3: Speed-up for JavaSpaces

CORBA Number of servers				
Input size	P=1	P=2	P=4	P=8
8K	1	1.2344	1.3377	1.2419
16K	1	1.6684	1.9943	2.0303
32K	1	2.1046	3.5684	4.2664
64K	1	2.0975	3.9456	6.9188

Table 4:Speed-up for CORBA

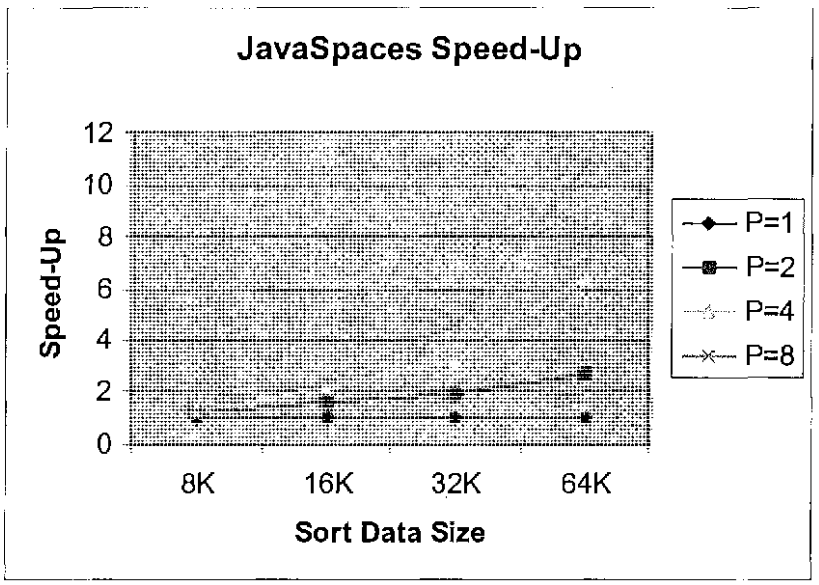


Figure 5: JavaSpaces speed-up

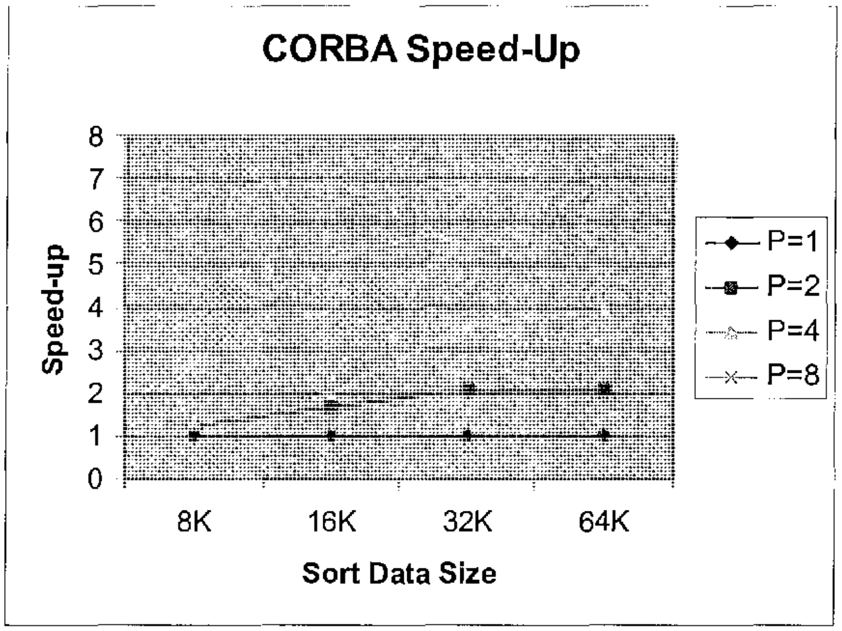


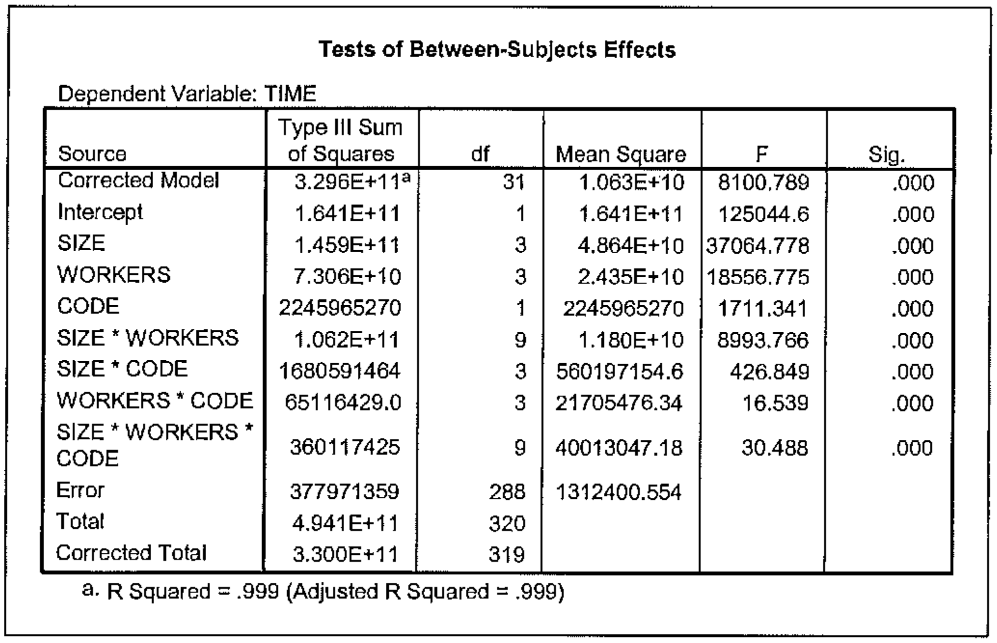
Figure 6: Corba speed-up

Comparing figure 5 and 6, we derive that we have a better speed-up when processing large amount of sort data. We also observe that we have better speed-up in JavaSpaces.

Chapter 6

STATISTICAL EVALUATION

6.1 Statistical Evaluation of test results



Tests of Between-Subjects Effects

Dependent Variable: TIME

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	3.296E+11[a]	31	1.063E+10	8100.789	.000
Intercept	1.641E+11	1	1.641E+11	125044.6	.000
SIZE	1.459E+11	3	4.864E+10	37064.778	.000
WORKERS	7.306E+10	3	2.435E+10	18556.775	.000
CODE	2245965270	1	2245965270	1711.341	.000
SIZE * WORKERS	1.062E+11	9	1.180E+10	8993.766	.000
SIZE * CODE	1680591464	3	560197154.6	426.849	.000
WORKERS * CODE	65116429.0	3	21705476.34	16.539	.000
SIZE * WORKERS * CODE	360117425	9	40013047.18	30.488	.000
Error	377971359	288	1312400.554		
Total	4.941E+11	320			
Corrected Total	3.300E+11	319			

a. R Squared = .999 (Adjusted R Squared = .999)

Figure 7: Tests of Between-Subjects Effects

Figure 7 represents tests of between subject effects. The last column represents statistical significance. This table shows that all the terms and all the interactions are statistically significant. That is, the probability that the differences found are due to chance alone are listed as .000 (rounded to three decimals they all are zero) [Mario99].

To determine the nature of these interactions, means plots are given where each pair of means is compared at the 0.05 level (That is, that differences are due to chance only 5% of the time).

From Figure 8 below, we observe that for each data size, CORBA takes significantly longer than JavaSpaces. The difference is the same for all data sizes.

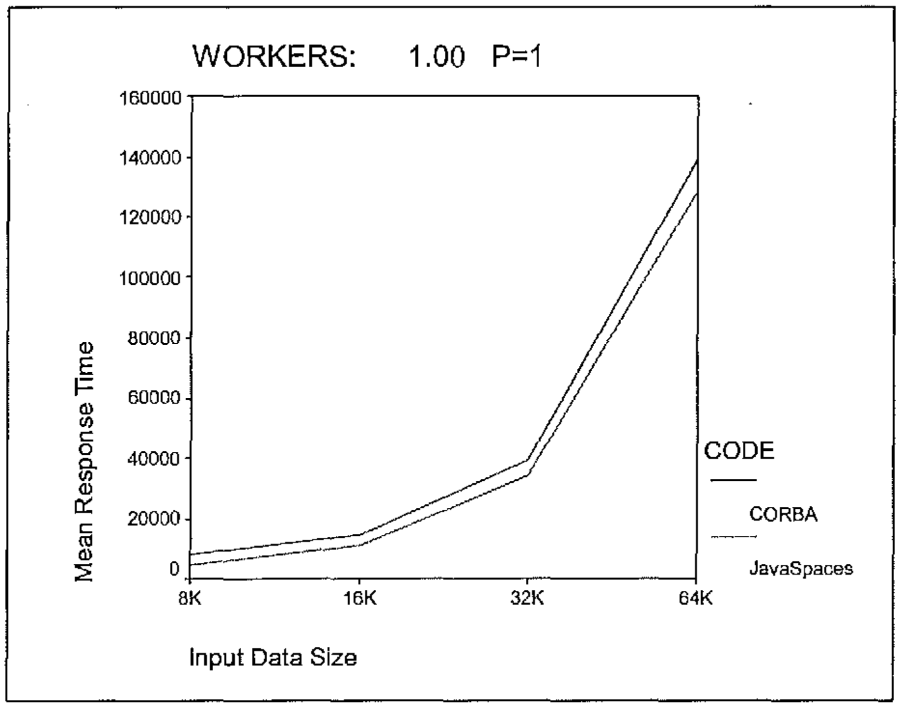


Figure 8: Mean response time for P=1 for JavaSpaces and CORBA

From Figure 9 below, we observe when we employed two workers, Corba is significantly higher in response time than JavaSpaces for all but input data size of 32K, where there is no significant difference. The difference is higher in data size 64K.

We have similar observation as above when we have four workers. CORBA is significantly higher in response time than JavaSpaces in all data sizes except 32K, where there is no difference. The difference is higher in data size of 64K.

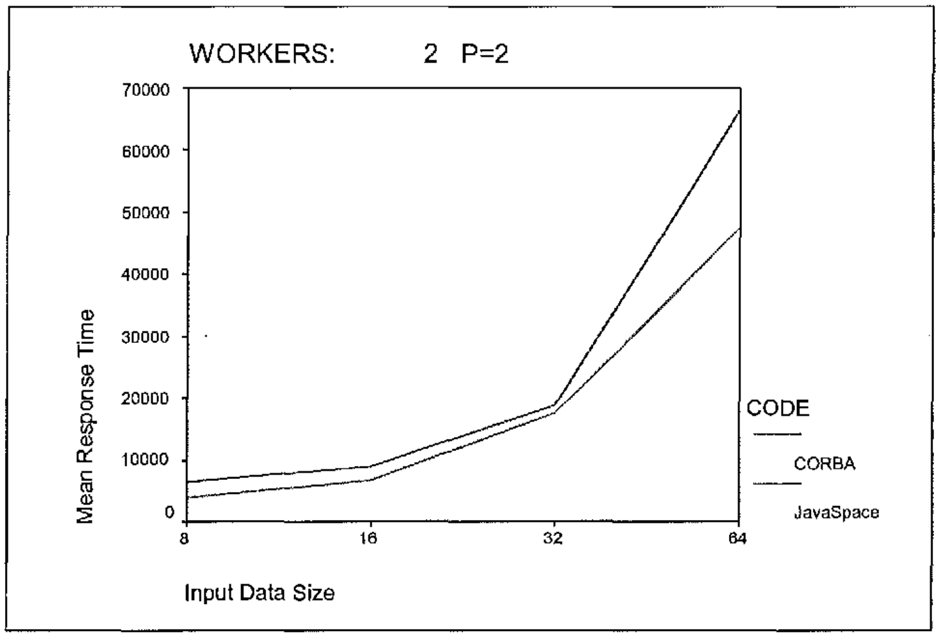


Figure 9: Mean response time for P=2 for JavaSpaces and CORBA

For eight workers CORBA is significantly higher in response time for all data sizes. The difference is higher in data sets of 64K.

Chapter 7

CONCLUSIONS

JavaSpaces consistently outperformed CORBA in terms of response time on both the parameters - size of the problem and number of processors deployed to work as workers/servers. We can conclude from the observed data that distributed parallel algorithm of master-worker pattern may be able to perform more efficiently when developed using JavaSpaces platform. CORBA is language neutral and thousands of sites rely on CORBA for enterprise, Internet, and other computing. Both CORBA and JavaSpaces architectures provide tremendous benefits in terms of fault-tolerance and scalability. In terms of ease of use and implementation of the two technologies, implementation of JavaSpaces was easier than CORBA. GigaSpaces platform already provides most of the implementation details and from an application programmer's perspective, there are only five commands to learn. We did face some challenges in implementing JavaSpaces due to its increased security considerations that is in-built within the JavaSpaces and its underlying Jini technologies and GigaSpaces platform. JavaSpaces does have the limitation that it can be only implemented on Java platform supporting Jini architecture. In comparison, implementation of CORBA platform is harder due to much detailed standards that developers must adhere.

In statistical analysis, the model we employed provided better insight and we observed that all the terms and all the interactions are statistically significant between the response times of the technologies.

The work carried in this project can bc extended and evaluated in the fields of on demand computing also known as Grid computing. This study can also be extended in evaluating service-oriented architectures where these technologies are the underlying technology infrastructure.

REFERENCES

[Arnold99]
Freeman, E., Hupfer, S., Ken Arnold, "JavaSpaces Principles, Patterns, and Practice", Addison Wesley, 1999, pp. 4-16.

[CapeSc02]
An introduction to implementing Web Services using CORBA servers
http://www.capescience.com/resources/

[Artima02]
Designing Distributed Systems
http://www.artima.com/jini/

[Giga02]
GigaSpaces Platform – White Paper
http://www.gigaspaces.com/download/GigaSpacesWhitePaper.pdf

[OMG03]
OMG CORBA/IIOP specifications
http://www.omg.org/technology/documents/formal/

[Orbacus03]
White Paper – Orbacus October 2003
http://www.orbacus.com/support/new_site/pdf/OrbacusWP.pdf

[Mario99]
Triola, Mario F., "Essentials of Statistics", Addison Wesley, 1999, pp. 4-16.

VITA

Anjani K. Jha has a Bachelor of Engineering degree from Birla Institute of Technology, India in Computer Science, 1991 and expects to receive his Master of Science in Computer and Information Sciences from the University of North Florida in December 2003. Dr. Sanjay Ahuja of University of North Florida is serving as Anjani's project Director. Anjani is currently employed as a Principal Consultant at Keane, Inc. and works at its client site, CSX Technology for the past six years. Prior to that Anjani worked in India as a Systems Analyst in Visakhaptanam Steel Plant. Anjani has over 12 years of industry experience in Information Sciences.

Anjani has diverse interests that include Data Base design and Administration, distributed systems design and development, project management. Anjani is married and lives with his wife and their three-year-old daughter in Jacksonville.