2002

Performance Evaluation and Comparison of CORBA Implementations for the Java Platform

Irina K. Grant

University of North Florida

PERFORMANCE EVALUATION AND COMPARISON OF CORBA IMPLEMENTATIONS FOR THE JAVA PLATFORM

by

Irina K. Grant

A graduate 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

April, 2002

The graduate project " Performance Evaluation and Comparison of CORBA Implementations for the Java Platform " submitted by Irina K. Grant in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by the graduate project committee: Date

Signature deleted

4/29/02

Dr. Roger Eggen
Project Director

Signature deleted

5/2/02

Dr. Judith L. Solano
Chairperson of the Department

Signature deleted

5/6/02

Dr. Charles N. Winton
Graduate Director

ACKNOWLEDGMENT

I would like to thank my project director, Dr. Roger Eggen, for his support and guidance of the project.

I would like to thank Dr. Sanjay Ahuja who brought to my attention the Distributed Systems through his lectures and instruction.

I would like to thank my husband and my mother for their encouragement and support.

CONTENTS

FIGURES

TABLES

ABSTRACT

Middleware is a software layer between the applications, services and the operating system that provides an abstraction to the application programmer. It masks the heterogeneous nature of the network and provides such services as remote calls, naming service, transaction process abilities, and security services. Common Object Request Broker Architecture (CORBA) is a middleware design that is implemented through the use of Object Request Broker (ORB), which is a software component that allows communication between the remote objects and applications that use them in a distributed environment. CORBA applications can run on almost any platform, operating system, and support different languages. There are many types of distributed object middleware on the market such as Sun's Java 2 ORB, Inprises's VisiBroker for Java, IONA's ORBacus for Java, and IONA's Orbix 2000 for Java. Because of these various products, it is difficult to select the product that will provide the specific requirements for one's application. The goal of this project is to evaluate the above-mentioned implementations of the CORBA standards and, additionally, CORBA was compared to LAM/MPI for efficiency. The results of this project should provide developers and novices studying distributed systems the necessary data to evaluate and select the most efficient CORBA product to meet their specific design requirements, and provide a methodology for further evaluation.

Chapter 1

INTRODUCTION

Accomplishing complex computational tasks efficiently in today's computer environment requires using either distributed systems or parallel processing. Due to the increased number of different types of computer networks and the growth within these networks, one of which is the Internet, completing these tasks requires selecting the most expeditious implementation of the various CORBA products or LAM/MPI. With a great variety of the CORBA middleware products available, research should be done to choose a product that will complement the application. In this project, four middleware products will be evaluated quantitatively and qualitatively. To make a quantitative comparison, the performance obtained in each case will be measured by response time during client/server communication. To make a qualitative comparison, each distributed object middleware is compared from a programmer's criteria. Additionally, CORBA is compared to LAM/MPI for efficiency purposes to provide a baseline. The results of this project will be helpful for developers by establishing a basis for testing other CORBA implementations. This project also will be helpful for students or those who would like to learn about different distributed object middleware such as Java 2 ORB, VisiBroker for Java, ORBacus for Java, and Orbix 2000 for Java.

Chapter 2

CORBA

CORBA is a standard architecture for distributed object systems. It allows a heterogeneous collection of objects to interoperate [Coulouris01]. The Object Management Group (OMG), established in 1989 with eight original members, is a 760-plus-member organization whose charter is to provide a common architectural framework for object-oriented applications based on widely available interface specifications [Rosenberger98]. The OMG achieves its goals with the establishment of the Object Management Architecture (OMA), of which CORBA is a part [Rosenberger98]. This set of standards delivers the common architectural framework on which applications are built [Rosenberger98]. The OMA consists of the ORB function, object services, common facilities, domain interfaces, and application objects [Rosenberger98]. CORBA's role in OMA is to implement the ORB function [Rosenberger98].

The major components of the CORBA architecture are IDL, ORB, CORBA Services, and IIOP. Interface Definition Language (IDL) is an implementation-independent language that provides a way of describing interfaces of the remote objects in CORBA. ORB provides the interactions between remote objects and the applications that use them. CORBA has a wide array of services. Naming Service is one of the standard services in CORBA that allows remote clients to locate remote objects on the network. The Internet

Inter-ORB Protocol (IIOP) handles the low-level communication between processes in a CORBA context [Flanagan99].

Since this project evaluates the results of various CORBA middleware, it is important to understand how distributed systems operate. Distributed systems allow hardware or software components located at networked computers to communicate by passing messages. Some of these characteristics are as follows:

- Allows multiple components in a system to run concurrently, such as multiple servers handling multiple client requests at the same time [Coulouris01].
- Absence of a global clock meaning that programs cooperate not by any shared idea of time, but by passing messages. Because there are limits to the accuracy with which computers in a network can synchronize their clocks, thus, there is no single global notion of time [Coulouris01].
- Ability to allow independent failure of components, meaning that one of the servers can be taken down while the other servers are still effectively functioning [Coulouris01].

The main reason for constructing and using distributed systems is the ability to share various resources such as hardware and software components. The hardware components can be printers and disks, and the software components can be files, databases and different data objects. Since these components may have many differences, the distributed system must be capable accepting new components, handling an increasing

number of users, absorbing software and hardware failures, and making the number of servers in the system transparent to the end user. In order to accomplish these differences, middleware becomes an important component in a distributed system.

Chapter 3

LAM / MPI

LAM (Local Area Multicomputer) is an MPI programming environment and development system for heterogeneous computers on a network [Lam1]. With LAM, a dedicated cluster or an existing network computing infrastructure can act as one parallel computer solving one problem [Lam1].

MPI (Message Passing Interface) is a library, that specifies the names, calling sequences, and results of subroutines to be called from Fortran programs, the functions to be called from C programs, and the classes and methods that make up the MPI C++ library [Gropp99]. The programs that are written in Fortran, C, and C++ are compiled with ordinary compilers and linked with the MPI library. MPI is the message-passing model, where processes communicate via messages and have separate address spaces. Communication occurs when a portion of one process's address space is copied into another process's address space [Gropp99]. This operation is cooperative and occurs only when the first process executes a send operation and the second process executes a receive operation [Gropp99].

3.1 LAM History

LAM was originally developed at the Ohio Supercomputer Center [Lam2]. Since then, the original members of the LAM Team moved to other jobs, and LAM became an orphaned project [Lam2]. The Laboratory for Scientific Computing (LSC) at the University of Notre Dame, headed by Dr. Andrew Lumsdaine, adopted LAM and hosted it in its servers [Lam2]. Soon after LAM was adopted by Notre Dame, version 6.2b was released, which contained some unreleased work from the original LAM Team, and a few contributions from the new Notre Dame LAM Team. Version 6.2b proved to be stable and robust in a variety of Unix environments [Lam2]. Development on LAM has continued by the Notre Dame LAM Team; the release of 6.3.2 included several new debugging features for user MPI programs, new environmental controls, and a variety of bug fixes from the original 6.2b release [Lam2]. Debugging parallel programs, which has been problematic in the past, is now much easier due to relaxations in LAM's process model [Lam2]. In the fall of 2001, Dr. Lumsdaine and the LSC moved to Indiana University. The LSC was renamed the Open System Laboratory (OSL) and all LSC projects, including LAM, moved from Notre Dame to Indiana with Dr. Lumsdaine [Lam2].

LAM/MPI is intended to be an open implementation of MPI. Since version 6.5.5, LAM/MPI has been licensed under the familiar revised BSD license. LAM 6.5.6 is now

the officially supported version of LAM. All prior versions are neither available nor supported [Lam2].

In this project, a LAM version 6.4-a3 was used that is identical to LAM 6.3.2 except that it includes support for Interoperable MPI (IMPI) [Lam3]. This is an alpha release [Lam3].

Chapter 4

Project Description

The purpose of this project is to determine the most efficient CORBA implementation. Comparison of Orbix 2000, VisiBroker, ORBacus, and Java 2 ORB, is the focus of this project. For baseline purposes, the application is also implemented using LAM/MPI.

4.1 Characteristics of the cluster

This application was developed on a Beowulf cluster. Applications for four ORBs tested ran on Vega, and Node1 through Node 9 machines. Name service and Oribx services (some of them were accepted by default during Orbix configuration) ran on Vega machine. Server applications ran on Node 2 through Node 9. Client application ran on Node 1 machine. All the Node machines run Red Hat Linux 7.1 version, with 128 megabytes of RAM, that have a single processor of 450 MHz. Vega runs also Red Hat Linux 7.1 version, and consists of four tightly-coupled CPUs each of which is 500 MHz, with 1.5 gigabytes of RAM. LAM/MPI application ran on Node1 through Node 9. It was booted from Node1 machine. The following versions of software used by this project are:

- Orbix 2000 1.2
- VisiBroker 4.5.1
- ORBacus 4.0.4

- Java 2 ORB 1.3.1
- LAM/MPI 6.4-a3

For the location of the files on CD refer to Appendix A.

4.2 The Nature of the Application

The practical nature of this project can be demonstrated in the following example. You have already an existing network in your company, but a new project development requires additional use of a mainframe. One of the options is to buy it, investing much needed capital, or to use the existing network and develop a distributed application. What can be the possible solution? One of the advantages of distributed systems is that they can have more computing power than a mainframe in terms of speed. In terms of economics, computers linked together can give a better price/performance ratio than mainframes. Extensibility and incremental growth is another advantage of distributed systems, as it is possible to gradually scale up (in terms of processing power and functionality) by adding more sources (both hardware and software). This can be done without disruption of the system. So the solution can be to use the client/server model for computation that is depicted on Figure 1. This project measures the results of implementing one such solution.

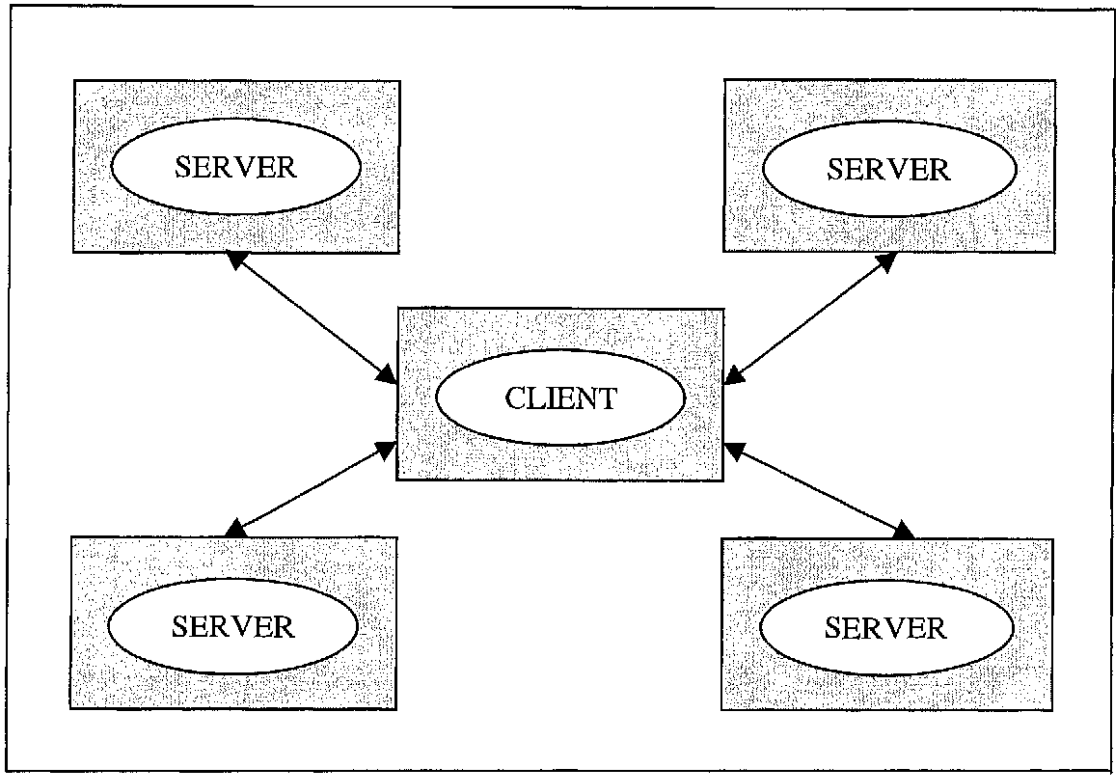


Figure 1: One Client/ Four Servers model used in this project

The applications for CORBA implementations were written in Java. The application for LAM/MPI implementation was written using C language. For this project, the number of servers specified was 1, 2, 4, or 8. Array size specified was 0, 10000, 20000, 40000, or 80000. Operations implemented were reply, search, or sort. Reply operation was implemented to measure strictly communication time. The number of steps in this operation is a constant. By using this operation, there was "null" work done on the server side, and to ensure two-way communication, server sends a string back at the client's request. For the search operation, a sequential search was implemented. The execution time for this algorithm is O(n). The input for search operation was an array of sorted

numbers generated by the client application where search was done to find the last element of the array initially created by the client. This type of operation would be considered as a medium load on the server. For the sort operation, an insertion sort was implemented. The execution time for sorting is O(n) when initial array is sorted, that is, only one comparison is made on each pass. The execution time is O(n^2) when the array is sorted in reverse order. The input for sort operation was an array of random numbers that was also generated by the client application. This type of operation would be considered as a heavy load on the server. In this project, for all CORBA implementations and LAM/MPI, the efficiency was measured by timing the communication process and the work on the server side. Therefore, for the sort operation, the merge of the sorted arrays is irrelevant to our study. The timer used for CORBA implementations was provided by System class that is located in java.lang package. The timer used for LAM/MPI implementation is provided by MPI routine, MPI_Wtime().

4.3 Client Application

Client application obtains the following arguments from the command-line such as number of servers, array size, and operation. Also, depending on the CORBA implementations or LAM/MPI, other arguments were added to the command line. The following characters were supplied for the command-line to specify the particular operation: 'n' is for reply operation, 's' is for searching, and 't' is for sorting. For reply

operation the size of the array will be zero. For the search and sort operations, the size of the array will be 10000, 20000, 40000, or 80000. When all arguments are obtained from the command line then the array is generated by the client application. For the reply operation there is no array generated. For the search operation a sorted array in ascending order is created. For the sort operation an array of random numbers is generated. Then the data size was divided evenly by the number of servers. When all the necessary data is obtained by the client application, a request is sent to the server. The client application is multi-threaded, so the array was distributed concurrently among all the servers. If the number of servers is one, then the entire array will be passed to the server application. The goal of the client application is to measure the communication process and the work that is done on the server side. As a result, the output of the client application is the time in seconds. Instructions for compiling and running client application are contained in Appendix A.

4.4 Server Application

The goal of the server application is to satisfy the client request. Depending on parameters supplied by the client, server application performs a particular task. For the reply operation, there is no data sent by the client application, and the server sends a reply string back to the client. For the search operation, the server application receives an array and a search key. The key is the last element of the array that was generated initially by

the client that will be found in only one server. If the element was found, the server sends an index of the array that contains that element to the client. Otherwise, negative one is returned. For the sort operation, the server receives an array, sorts it implementing insertion sort, and sends a sorted array back to the client. Instructions for compiling and running server application are contained in Appendix A.

4.5 CORBA Services

CORBA Naming Service was used to allow the client to locate remote objects on the network. Every CORBA implementation tested establishes its environment allowing communication to occur. Prior to running the server application, the Naming Service has to be started, which is then followed by the running of the client application. Each of the CORBA implementations has its own way of invoking Naming Service. For the Orbix 2000, additional services will be running with the Naming Service as the background processes due to the Orbix 2000 configuration. In this project for simplicity, a Korn Shell script was written to start Service(s) for each of the CORBA applications. To run this script, type at the prompt run.ksh. To stop Naming Service or Orbix 2000 services, use dd.ksh script.

Chapter 5

TESTING

Performance for each distributed parallel application was measured by response time in seconds. The client application creates a thread for each server. Each thread is initialized sequentially. The timer starts immediately before thread initializations, and stops when the last thread completes the task. That means that the client receives a response from the server, which in turn completes either searching, sorting, or simply sends a reply string back to the client. The response time is the difference between stop and start time.

The experiments performed were: one client/one server; one client/two servers; one client/four servers; and one client/eight servers. Then for each of these experiments, tasks evaluated were search, sort, and reply operations. The array size was 10K, 20K, 40K, and 80K for searching and sorting. Tests were performed on the various combinations ten times each in order to obtain an average operation processing time. The total number of tests performed was 1800. For each of the 5 products (four CORBA implementations and LAM/MPI), there were 10 reply tests for each of the 4 server combinations, 40 search tests for each of the 4 server combinations, and 40 sort test for each of the 4 server combinations.

Chapter 6

RESULTS

In this section quantitative and qualitative comparison will be provided for all the implementations of CORBA and LAM/MPI. A quantitative comparison is based on the performance of each middleware product and MPI that was obtained by measuring response time during client/server communication. A qualitative comparison is made based upon ease of environment configuration, application time development, ease of use, and difference in architecture and services for CORBA implementations.

General Observations:

- There is a clear ranking of products from best to worst that does not change across the various applications for all operations. Java 2 ORB performs closer to LAM/MPI than the other products. Orbix 2000 performs significantly worse than LAM/MPI for reply and search operations.
- For the sort operation, there is a marked improvement in performance as the number of servers is increased.
- All the products are significantly worse than LAM/MPI for all operations other than sort for 4 and 8 servers.
- The variance between VisiBroker and ORBacus is fairly insignificant across all operations.

- For the search operation, as servers are added the performance levels out for all products regardless of array size.
- For graphs of results see Appendix B.
- For comparison of services and features of the CORBA implementations, see Table 1 [McKeller01].

6.1 Orbix 2000

6.1.1 Quantitative Comparison

For reply and search operations, Orbix 2000 performs significantly worse than the other CORBA implementations for medium or light workloads. However, as the workload is increased, as for example, in the sort operation where the array size is increased, the percentage of variance versus the other CORBA implementations is reduced. Therefore, Orbix 2000 is more efficient as workload and servers are added. Across all operations, Orbix 2000 is the weakest performer versus the other CORBA implementations and LAM/MPI.

6.1.2 Qualitative Comparison

Orbix 2000 is a commercially available CORBA implementation from Iona Technologies. It complies with CORBA 2.3 specifications. The Orbix 2000 allows development and deployment of large-scale enterprise applications in C++ and Java. Orbix 2000 provides a unique code generation toolkit named "genie" that generates a complete client-server application automatically by using an IDL file. Developing an application by using "genie" allows the programmer to concentrate on the business application logic, thereby saving time in the overall development process. "Genie" can also be used for debugging purposes. For example, it can be used for an auto-generated server to debug a client application. Orbix 2000 provides support for single and multi-threaded applications. It also allows spanning a large number of hosts across a network and can be extended with new hosts, therefore, is highly scalable. Orbix 2000 provides several useful services, such as naming service, object transaction service, and event service. Orbix 2000 supports POA (Portable Object Adapter) as do VisiBroker and ORBacus. POA provides portability on the server side that acts as an intermediary between the object implementation and ORB. It provides such fundamental services as object creation, servant registration, and request dispatching. While developing an application with Orbix 2000 was fairly easy, its configuration was extremely cumbersome. For this application, file-based domain configuration was used [Orbix2].

6.2 VisiBroker

6.2.1 Quantitative Comparison

For reply operation, Visibroker performs better than Orbix, but Java 2 ORB outperforms it significantly. Adding additional processors does not improve response time. As more servers are added, the response time increases due to the communication overhead. For the search operation, there are no significant results obtained when the number of servers and array sizes are increased. For the sort operation, one client/one server, and one client/two servers, VisiBroker performance is tied with all other CORBA implementations. Overall, for sort operation, response time is significantly improved as the number of servers is increased. This means if sufficient work has to be done on the server side, it is worthwhile to add additional processors.

6.2.2 Qualitative Comparison

VisiBroker is another commercially available CORBA implementation Inprise product from Borland Software Corp., and is CORBA 2.3 compliant. Its features include location service, naming service, and event service. Additional tools are "Smart Agent" and the "Object Activation Daemon". Smart Agent (osagent) is a dynamic, distributed directory service that locates the implementation requested by the client. Multiple Smart Agents on

the network can cooperate in order to provide load balancing at a high availability for the client to access object implementations. The Object Activation Daemon can be used to automatically start the implementation upon client request. VisiBroker provides native support for single and multi-threading management. It also provides the interceptor feature, as do Orbix 2000 and ORBacus, which provides a framework to add ORB behavior such as security, transactions, or logging. Although, BOA (Basic Object Adapter) is being depreciated, VisiBroker 4.0 will still support BOA functionality [Visi]. VisiBroker provides a feature known as Quality of Services that "allows to define properties that influence how the connection is made" [Visi]. Another feature of VisiBroker, and also Orbix 2000 and ORBacus, is a Dyn-Any interface that allows the dynamic creation of basic and constructed data types at run time. VisiBroker also offers IR (Interface Repository) that contains information about CORBA objects and enables clients to learn about or update interface descriptions at runtime [Visi]. Excellent documentation, including clear and understandable examples, greatly assisted the development of applications using VisiBroker.

6.3 ORBacus

6.3.1 Quantitative Comparison

For all servers combinations, all operations, and array sizes, ORBacus performance is slightly better than VisiBroker, but follows the same pattern of performance when additional servers are added.

6.3.2 Qualitative Comparison

ORBacus is a CORBA compliant product for C++ and Java, and is free for non-commercial use. There was no complicated configuration. One of the ORBacus features is Implementation Repository (IMR) that "provides support for indirect binding for persistent object references" [ORBacus]. The advantage of this indirect binding is that it loosens the coupling between client and server. Therefore, changing location of the server will not affect the client. IMR also can activate server using Object Activator Daemon (OAD). CORBA Properties Service permits you to annotate an object with extra attributes (called properties) that were not defined by the object's IDL Interface [ORBacus]. Properties can represent any values because they can make use of CORBA Any Data type [ORBacus]. ORBacus has Time Service that provides operations related to time and intervals. ORBacus has Event Service, which allows some applications to exchange information without explicitly knowing about one another. Since a server is

sometimes not aware of the nature and number of clients that are interested in the data the server has to offer, a special mechanism is required that provides decoupled data transfer between servers and clients. ORBacus provides Trading Service that is similar to Naming. Both of them are needed to locate objects. In Trading Service, an object is not published by name, so a server advertises an object based on the kind of services provided by the object. A client locates of object of interest by asking the Trading Service to find all objects that provide a particular service. A CORBA Interface Repository (IFR) is essential for applications using dynamic futures of CORBA, such as Dynamic Invocation Interface and DynAny. The IRF holds IDL type definitions and can be queried and traversed by applications.

6.4 Java 2 ORB

6.4.1 Quantitative Comparison

For Java 2 ORB, when there is no work to be done on the server side, for instance, reply operation, adding servers does not improve the results. For the sort operation, adding another processor improves the results significantly. In the case where the experiment was performed on sort operation with eight servers and array size of 80K, Java 2 ORB outperformed LAM/MPI. This result can be explained because Java applications for all

CORBA implementations are multithreaded. Therefore, data is sent to each of the server concurrently. LAM/MPI is not multithreaded. According to MPI documentation, MPI library can be called only in single thread. Thus, in LAM/MPI application data is sent to all the servers sequentially. Also, MPI uses send and receive functions to pass messages. Function send_MPI() is blocking. For example, in the case with eight servers, the last server cannot sort data until all the other servers receive it. This is what creates additional time. Compared to all other CORBA implementations, such as Orbix 2000, VisiBroker, and ORBacus, across all operations, servers combinations and array size, this is the most efficient product that exhibits the best performance.

6.4.2. Qualitative Comparison

Java 2 ORB is free software that is CORBA 2.3 compliant and was very easy to use to develop applications. There is sufficient documentation available through web sites and books. Java 2 ORB is language independent, that is, any exported CORBA objects can be implemented in different languages, such as C, C++, or Ada. One of the disadvantages is that compared to other CORBA implementations, applications that are developed with Java 2 ORB can be written only in Java. The other of the disadvantages is that it lacks many services available in other CORBA products, namely, event service, trading service, object transaction service, and location service. The only service that is provided by Java 2 ORB is Naming Service. Another disadvantage is that the current version of

Java 2 ORB does not support POA as do other CORBA implementations. Another disadvantage is that Java 2 ORB is the only ORB that does not support single thread option. Despite all these disadvantages, this is a great tool with which to learn and develop simple distributed applications.

6.5 LAM/MPI

6.5.1 Quantitative Comparison

LAM/MPI application was used as an indicator by looking at which CORBA implementations can be analyzed, and their performance will be evaluated based on LAM/MPI results. For all combinations of the servers, across all the operations, and regardless size of the array, results obtained for LAM/MPI demonstrate the best performance. With the exception of Java 2 ORB the variance was fairly significant versus the CORBA implementations.

6.5.2 Qualitative Comparison

LAM is a freeware implementation of the Message Passing Interface standard [Lam4]. The MPI standard is the de facto industry standard for parallel applications [Lam4]. LAM

tends to be "cluster friendly" by using small daemons to effect fast process control, application startup/shutdown, etc [Lam4]. LAM/MPI was easy to learn and also to develop the application, which is a monolithic, where client and server are implemented in one program. The problem with such an application is that it is difficult to debug, and as the complexity of the program increases, it becomes cumbersome and difficult to maintain. Another issue is that LAM is not thread - safe, but it can be used in multi-threaded applications. Languages used with MPI/LAM are Fortran, C, and C++.

It was easy to set environment for this application. To obtain all the results for LAM/MPI, this application can be invoked from one machine therefore it was easy to run. There are also many web sites and books published that provide sufficient information about implementations of LAM/MPI.

Feature	Orbix 2000	VisiBroker	ORBacus	Java 2ORB	LAM/MPI
IDL supports code generation for multiple languages	Yes	Yes	Yes	No	N/A
Single & Multi threading support	Yes	Yes	Yes	Multi-thread only	Single thread only
POA support	Yes	Yes	Yes	Not in current version	N/A
Policy support	Yes	Yes	Yes	No	N/A
Dyn-Any	Yes	Yes	Yes	No	N/A
Interface Repository	Yes	Yes	Yes	No	N/A
Implementation Repository with Object Activation Daemon	Yes	Yes	Yes	No	N/A
Naming Service	Yes	Yes	Yes	Yes	N/A
Event Services	Yes	Yes	Yes	No	N/A
Location Service	Yes	Yes	No	No	N/A
Transaction Service	Yes	No	No	No	N/A
Code Generator	Yes	No	No	No	N/A
Time to develop	20	9	10	25hrs	15hrs
Time to configure	35hrs	0	4hrs	0	2hrs
Ease of implementation	Not as easy	Easy	Easy	Easy	Easy

Table 1: Comparison of Features and Services of CORBA Implementations

Chapter 7

CONCLUSIONS

In conclusion, selection of an appropriate middleware product is largely a function of the work to be performed. For a light workload, a product that is easy to use might be preferable to one that might be slightly faster. For a heavy workload, faster is definitely better, even if it requires more time during the development process. This project produced test results which were very consistent across varying ranges of operations and workload in that the four products tested maintained their relative ranking throughout all operations, and indicated that Java 2 Orb was the best implementation available to satisfy a variety of requirements.

REFERENCES

[Colouris01]
Colouris, George, J. Dollimore, and T. Kindberg, Distributed Systems Concepts and Design, Addison-Wesley, 2001

[Flanagan99]
Flanagan, D., et al, Java Interprise in a Nutshell, O'Reilly & Associates, Inc., 101 Morris Street, Seabastopol, CA, 1999

[Gropp99]
Gropp, W., Lusk, E., and Skjellum,A., Using MPI, the MIT Press Cambridge, Massachusetts, London, England, 1999

[Java1]
Java™ IDL, main page http://java.sun.com/j2se/1.3/docs/guide/idl/index.html

[Java2]
Java™ IDL FAQ, http://java.sun.com/j2se/1.3/docs/guide/idl/jidlFAQ.html

[Java3]
Java™ IDL, about idlj compiler, http://java.sun.com/products/jdk/idl/index.html

[Lam1]
LAM / MPI Parallel Computing, main page http://www.lam-mpi.org

[Lam2]
The History of LAM/MPI, http://www.lam-mpi.org/history.php

[Lam3]
LAM / MPI Parallel Computing, LAM versions, http://hpc.snu.ac.kr/document/mpi/lam/

[Lam4]
LAM / MPI Parallel Computing, LAM FAQ: LAM terms and definitions, http://www.lam-mpi.org/faq/category1.php3

[Lam5]
LAM / MPI Parallel Computing, One-Step Tutorial: Getting started with LAM, http://lam-mpi.org/tutorials/one-step/lam.php

[Langsam96]
Langsam, Y., Augenstein, M.J., Tenenbaum A.M., Data Structure Using C and C++, Prentice Hall, Upper Saddle River, NJ, 1996

[McKeller01]
McKeller, Michelle, L., CORBA: A Quantitative and Qualitative Comparison of Industrial Strength, Commercial CORBA ORBs For The Java Platform, University of North Florida, Jacksonville, 2001

[ORBacus]
ORBacus for C++ and Java, version 4.0.4 (comes with software distribution)

[Orbix]
Orbix 2000 Programmer's Guide
http://www.iona.com/docs/orbix2000/1.2/pguide_java/pdf/pguide_java.pdf

[Orbix2]
Orbix 2000 Administrator's Guide, Configuring a File-Based Domain, ch3,
http://www.iona.com/docs/orbix2000/1.2/admin/pdf/admin.pdf

[Orbix3]
Orbix 2000 Manuals, http://www.iona.com/docs/orbix2000/1.2/index.html

[Rosenberger98]
Rosenberger, Jeremy L., Teach Yourself CORBA in 14 Days, Sams Publishing, 1998.

[Visi]
VisiBroker documentation version 4.5
http://www.borland.com/techpubs/books/vbj/vbj45/programmers-guide/vbj45programmers-guide.pdf

APPENDIX A

Directory Structure for Project CD, Compiling and Running Applications

Project CD contains the following five folders: Documentation, Executables, Full_Package, Results_xls, and Source_code.

1. Documentation
This folder contains two files: presentation Power Point slides and write-up in pdf format.

2. Executables
This folder contains scripts to run all the applications and executables. Applications used in this folder can be run by using run.ksh and run_all.ksh scripts. Note the permission for these scripts has to be set to 755 to execute them. Scripts that were used to compile applications were removed from this folder.

3. Full_Package
This folder contains the complete project that includes all source codes and executables with all the scripts necessary to compile and run these applications. Also, additional text files with instructions are provided for each application. Zip file for this package is included.

To zip files or folders:
$> tar -cvf newfile.tar myfile
$> gzip newfile.tar

To unzip files or folders:
$> gzip -vd newfile.tar.gz
$> tar -xvf mewfile.tar

4. Results_xls
This folder contains two files in xls format that detail results for this project.

5. Source_code folder
This folder contains the source code for the five applications: Orbix 2000 (Orbix2000 folder), VisiBroker (Vbroker folder), ORBacus (ORBacus folder), Java 2 ORB (Java_IDL folder), and LAM/MPI (Lam folder).

Each of these folders contains the following directories:
home_dir_env that contains the .bash file from the home directory where all the environment should be set for all the ORBs and LAM.

node1_client directory contains java files for the multithreaded client application, idl and results files.

node2_server (all these server nodes contain files for the server application and idl file)
node3_server
node4_server
node5_server
node6_server
node7_server
node8_server
node9_server

vega_name_serv (this is the directory from which naming service runs and stops)

read file (gives you the instruction on how to compile and run a particular application)

For VisiBroker, ORBacus, and Java 2 ORB:
First, to compile your application you need to set environment, then go to the name service directory, start the name service by running run.ksh script. Next go to the servers directories and run run.ksh script that compiles and starts the servers. Finally go to the client directory and use run_all.ksh script to compile and run the client application.

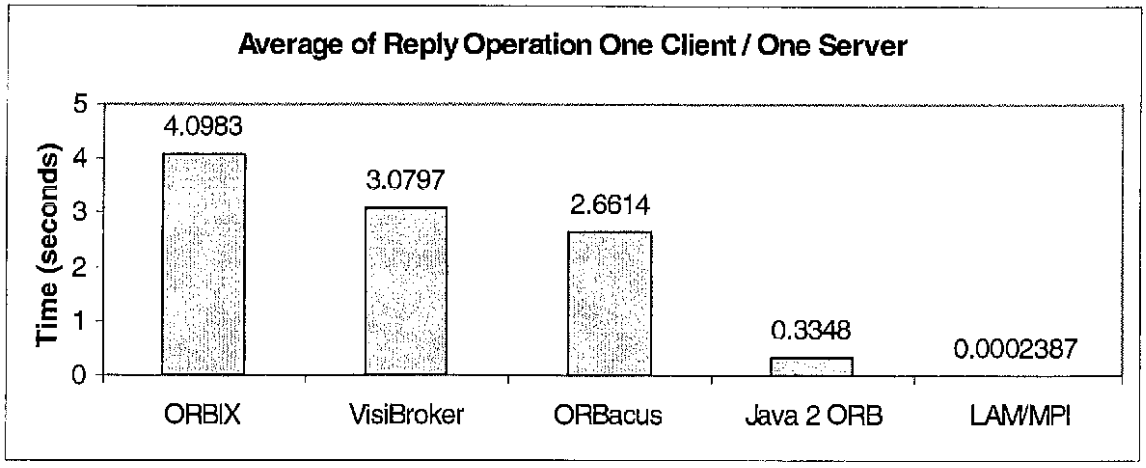
For Orbix 2000:
To configure Orbix, file-based configuration was used, see [Orbix2] reference. The domain name was igrant3000. It was created on Vega machine. Also, .bashrc file contains some settings for the application. File in home_dir_env shows how the additional settings are done. Orbix application has different structure compared to the rest of the ORBs. Prior to running server and client applications, Orbix services should be started on Vega machine from the D:\Burn_this_CD\Source_code\GRAD_PROJ_scode\Orbix2000\vega_name_serv directory. To run Orbix services, type run.ksh from the command-line. Client and server applications were generated by using Orbix code generator. Folder node1_client contains several xml files. The xml file name uses the following format: build_NumberOfServers_ArraySize_Operation.xml. These xml files are very important. The initial file generated was build.xml. This file contains the structure where domain name for the Orbix was specified, name for the executable (java file name that contains main function for client and server), and command-line arguments. This file is used by ant utility that comes with Orbix2000. Scripts such as run1.ksh, run2.ksh, run4.ksh and run8.ksh set environment for that utility to run, see file ant_env.sh. Orbix client application can be built only on Vega machine. To build client application, type at the prompt vega_build.ksh from the node1_client directory on Vega machine. To run client application run run_all.ksh script from Node 1 machine. Directory NoPackage in the node1_client contains source code for client application. The following directories contain server applications: node2_server, node3_server, node4_server, node5_server,

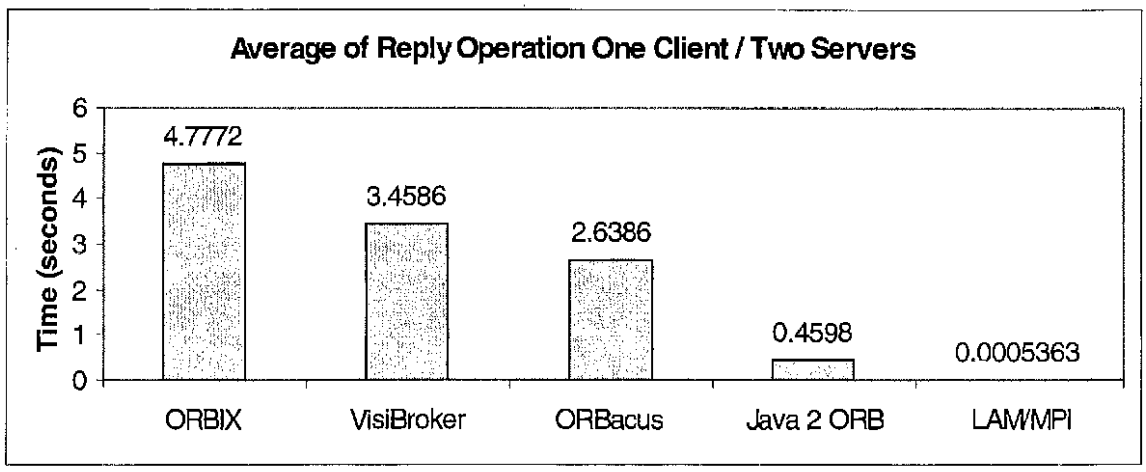
node6_server, node7_server, node8_server, and node9_server. Servers directories have the same structure as client except that they contain only one build.xml file and directory NoPackage contains source code for the server application. All the servers applications have to be built also on Vega machine using vega_build.ksh script from the mentioned above servers directories. Then go to the specified Node machine and run run.ksh script. For example, if directory name is node2_server. You should login on Node 2 machine, go to the appropriate directory, in this case it will be D:\Burn_this_CD\Source_code\GRAD_PROJ_scode\Orbix2000\node2_server, and run run.ksh script. File Readme_Orbix.txt provides additional information on how to run Orbix application.

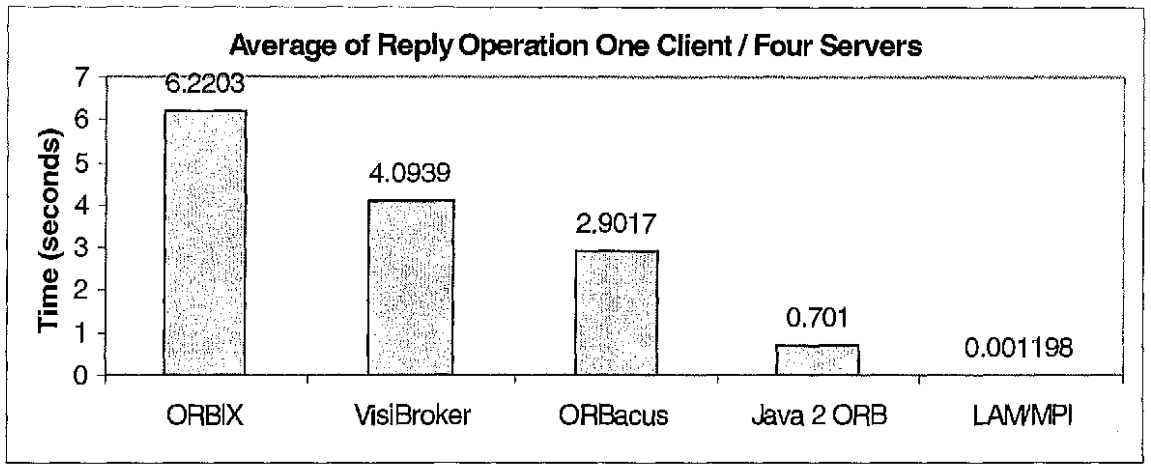
For LAM/MPI:
home_dir_env is a directory that contains two files. One of them is a copy of .bashrc file from your home directory where Lam environment is set. The other file is home_.hosts which is the contents of the .rhost file that should be also in your home directory. The following files such as, lamhosts1, lamhosts2, lamhosts4, and lamhosts8 contain the address of the node(s) that will be booted. File services.c is a source code for Lam program. To compile this source code, type 'make' at the prompt. Makefile was created to make this application. To obtain all the results run this application using run_all.ksh script. Files Readme.txt and notes_scipt.txt provide documentation and examples on how to compile and run this application.

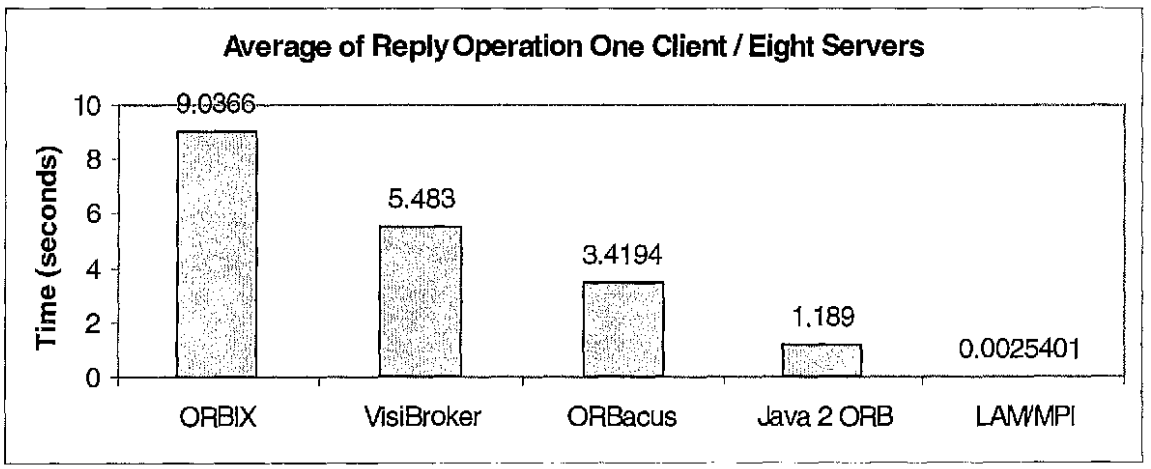
APPENDIX B

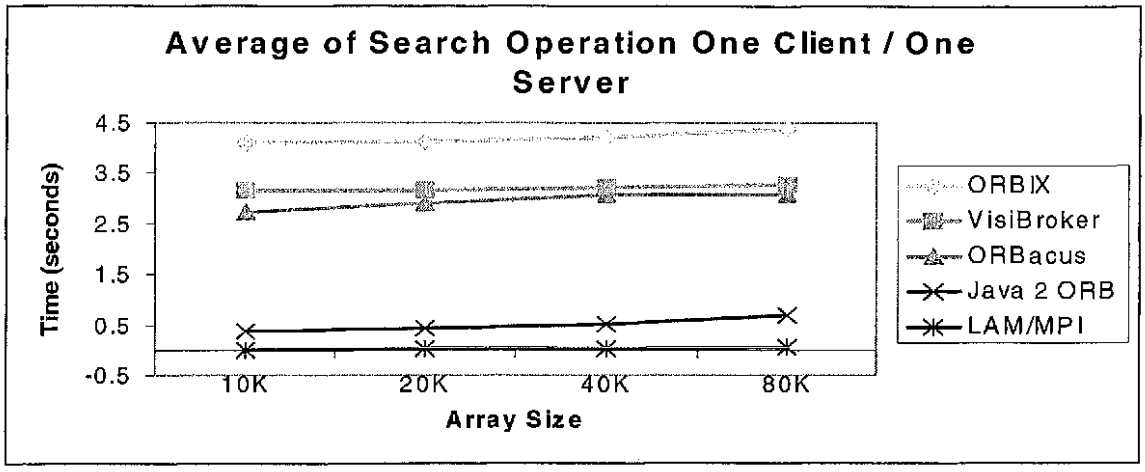
GRAPHS

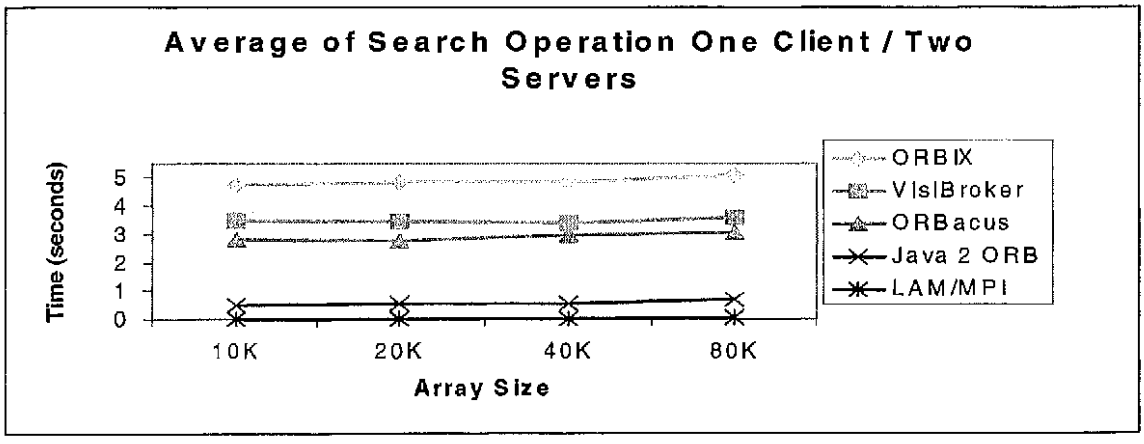


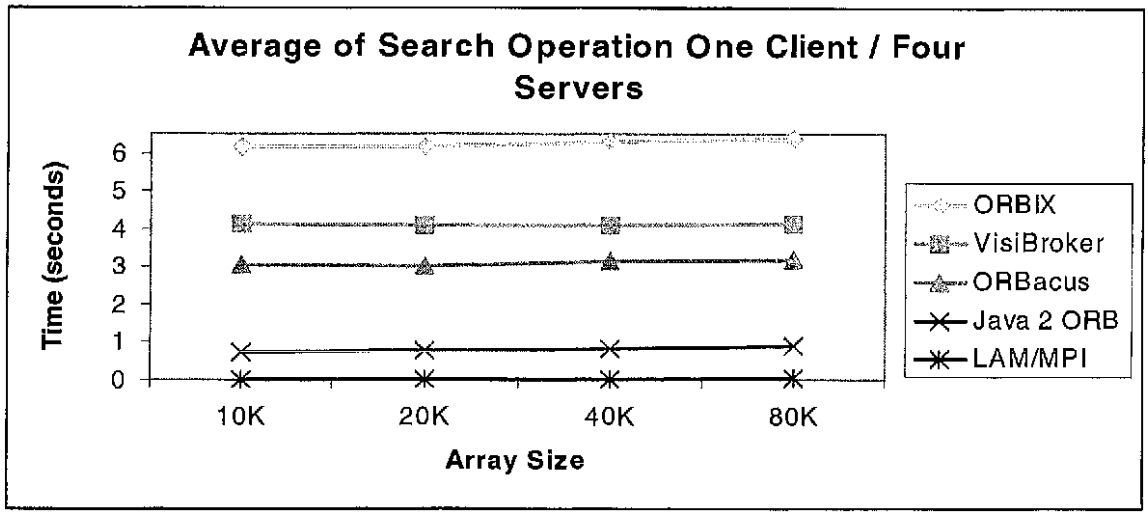


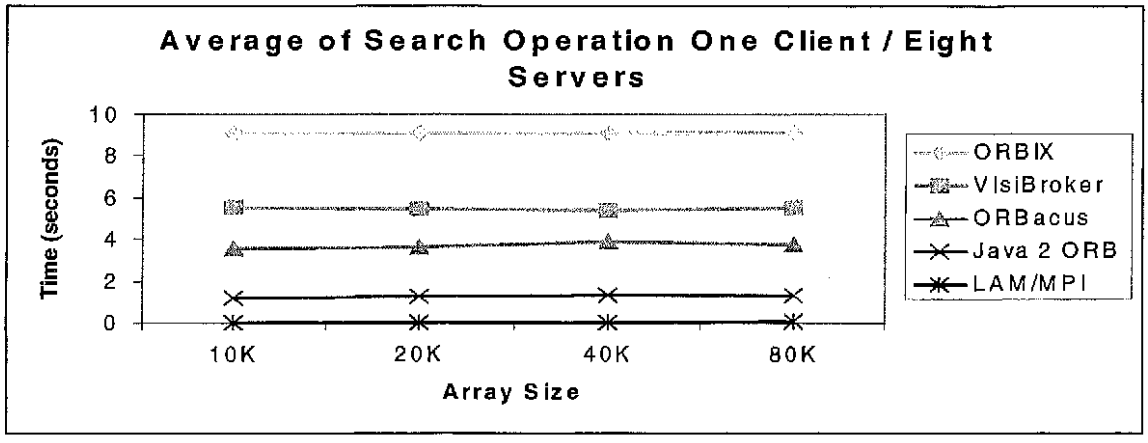

Average of Reply Operation One Client / Four Servers
Time (seconds)
7
6
5
4
3
2
1
0
6.2203
4.0939
2.9017
0.701
0.001198
ORBIX
VisiBroker
ORBacus
Java 2 ORB
LAM/MPI

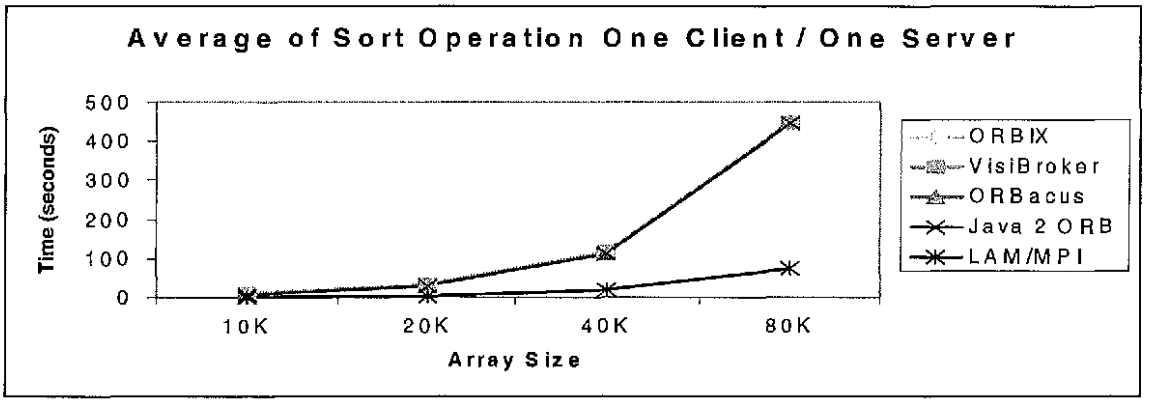

Average of Reply Operation One Client / Eight Servers
Time (seconds)
10
8
6
4
2
0
9.0366
5.483
3.4194
1.189
0.0025401
ORBIX
VisiBroker
ORBacus
Java 2 ORB
LAM/MPI

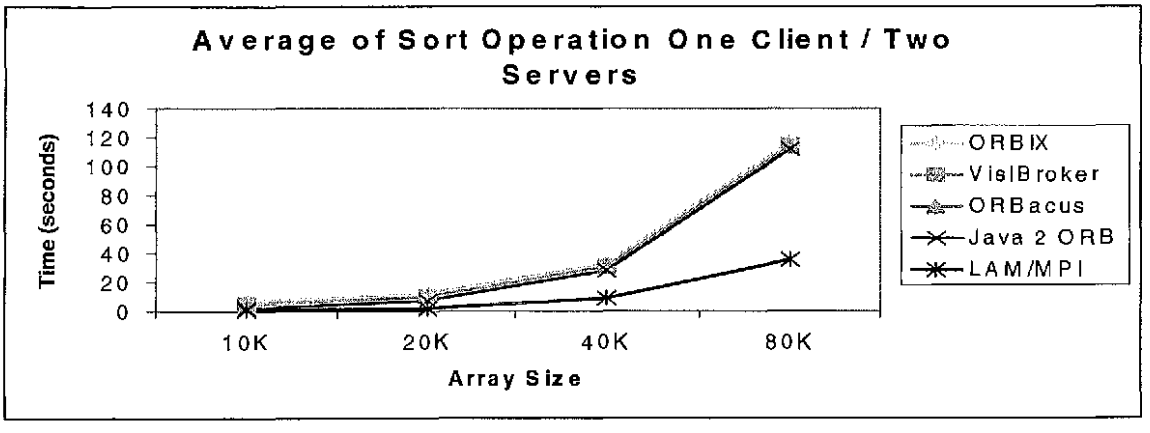

Average of Search Operation One Client / One Server
Time (seconds)
4.5
3.5
2.5
1.5
0.5
-0.5
10K
20K
40K
80K
Array Size
ORBIX
VisiBroker
ORBacus
Java 2 ORB
LAM/MPI

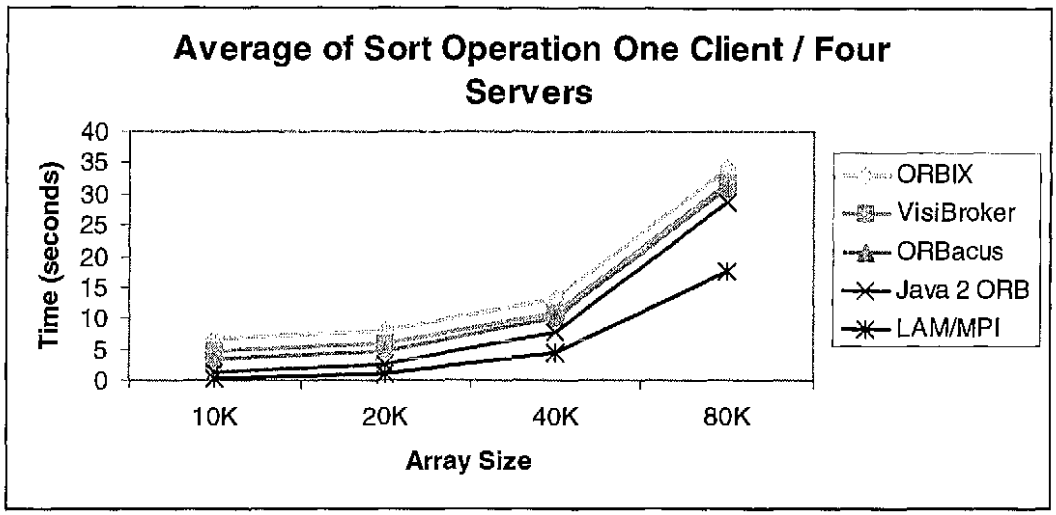

Average of Search Operation One Client / Two Servers
Time (seconds)
5
4
3
2
1
0
10K
20K
40K
80K
Array Size
ORBIX
VisiBroker
ORBacus
Java 2 ORB
LAM/MPI

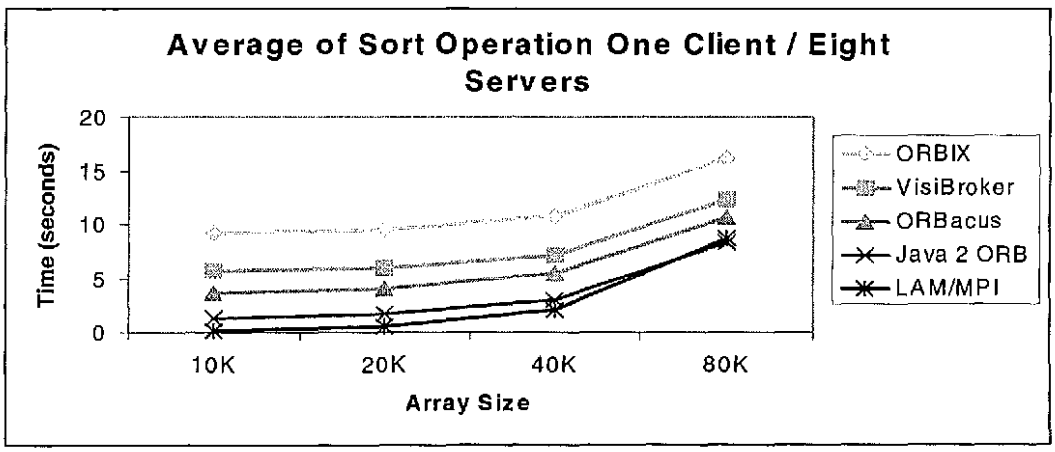

Average of Search Operation One Client / Four Servers
Time (seconds)
6
5
4
3
2
1
0
10K
20K
40K
80K
Array Size
ORBIX
VisiBroker
ORBacus
Java 2 ORB
LAM/MPI


Average of Search Operation One Client / Eight Servers
Time (seconds)
10
8
6
4
2
0
10K
20K
40K
80K
Array Size
ORBIX
VisiBroker
ORBacus
Java 2 ORB
LAM/MPI


Average of Sort Operation One Client / One Server
Time (seconds)
500
400
300
200
100
0
10K
20K
40K
80K
Array Size
ORBIX
VisiBroker
ORBacus
Java 2 ORB
LAM/MPI


Average of Sort Operation One Client / Two Servers
Time (seconds)
140
120
100
80
60
40
20
0
10K
20K
40K
80K
Array Size
ORBIX
VisiBroker
ORBacus
Java 2 ORB
LAM/MPI


Average of Sort Operation One Client / Four Servers
Time (seconds)
40
35
30
25
20
15
10
5
0
10K
20K
40K
80K
Array Size
ORBIX
VisiBroker
ORBacus
Java 2 ORB
LAM/MPI


Average of Sort Operation One Client / Eight Servers
Time (seconds)
20
15
10
5
0
10K
20K
40K
80K
Array Size
ORBIX
VisiBroker
ORBacus
Java 2 ORB
LAM/MPI

VITA

Irina K. Grant has a Bachelor of Science degree in Computer Engineering from the Institute of Atomic Power Engineering in Obninsk, Russia in 1991 and is pursuing a Masters of Science degree in Computer and Information Science from the University of North Florida. Dr. Roger Eggen is Irina's graduate project advisor. While attending the University of North Florida, Irina has served internships at IDS as a Programmer I, at ECI Telecom as an Associate Software Engineer, and at CitiStreet as a Developer. Prior to immigrating to the United States, Irina worked as a C Programmer at Nets Informer Company in Odessa, Ukraine, and at the Institute of Physics and Seismology in Obninsk, Russia. Irina has experience in the programming languages of C, C++, Java, Perl, SQL, TCL, PHP, and embedded SQL, and Korn Shell script. Irina's interests include her family, tennis, and the performing arts.