

**A GRAPHICAL USER INTERFACE TO MONITOR  
AND MANAGE THE DDAS SYSTEM  
PERFORMANCE**

By

**SEONG SEOL HONG**

Bachelor of Science  
Hallym University  
Chunchon, Korea  
1994

Master of Engineering  
Soongsil University  
Seoul, Korea  
1996

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
**MASTER OF SCIENCE**  
December, 1999

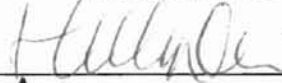
**A GRAPHICAL USER INTERFACE TO MONITOR  
AND MANAGE THE DDAS SYSTEM  
PERFORMANCE**

Thesis Approved:

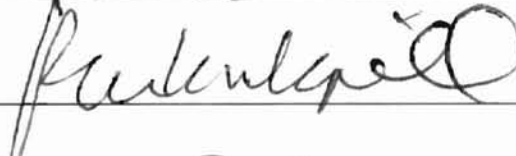


---

Thesis Adviser



---



---



---

Wayne B. Powell  
Dean of the Graduate College

## ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my thesis advisor, Dr. K. M. George for his supervision, constructive guidance, encouragement, and friendship during this study. My sincere appreciation extends to my other committee members Dr. H. K. Dai and Dr. N. Park, whose guidance, assistance and friendship are also invaluable.

I wish to express my sincere gratitude to my father in Korea and my mother in Heaven for their endless love, encouragement and support. My special appreciation extends to my wife, Nam-Youn, for her strong love and understanding. Thanks go to my lovely family for their encouragement and faith. I would also like to thank to Dr. K. Lee who has been my advisor in Korea for his academic support.

Finally, I would like to thank the Department of Computer Science during two and one-half years of study.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION .....	1
II. LITERATURE REVIEW .....	4
2.1 Client-Server Model .....	4
2.1.1 Distributed Computing Environment (DCE).....	7
2.1.2 Remote Procedure Call (RPC).....	10
2.2 Mobile Agents .....	12
2.2.1 Telescript .....	14
2.2.2 Agent Tcl.....	14
2.2.3 Agents for Remote Access (Ara).....	15
2.3 Middleware.....	16
2.4 Common Object Request Broker Adapter (CORBA).....	17
2.5 Condor .....	19
2.6 Piranha .....	20
III. PREVIOUS WORK .....	22
3.1 The DDAS System.....	22
3.2 Drawbacks of the DDAS System .....	23
IV. ENHANCED DDAS SYSTEM.....	25
4.1 The Monitoring Function .....	25



4.2 The Dynamic Network.....	27
4.3 The User Interface for Application Programs .....	27
 V. IMPLEMENTATION OF THE EDDAS SYSTEM.....	 29
5.1 Implementation of the Monitoring Function .....	30
5.2 Construction of the Dynamic Network.....	32
5.3 User Interface .....	36
 VI. PERFORMANCE EVALUATION.....	 38
6.1 System Environment.....	38
6.2 Distributed Quicksort Algorithm.....	40
6.3 System Performance .....	45
 VII. CONCLUSIONS .....	 56
 REFERENCES .....	 57
 APPENDIXES	
APPENDIX A. EDDAS SYSTEM SOURCE CODE IN JAVA .....	62
APPENDIX B. SYSTEM LOAD INFORMATION .....	86
APPENDIX C. ABBREVIATIONS AND ACRONYMS IN ALPHABETICAL ORDER.....	89

## LIST OF TABLES

Table	Page
Table 4.1 Responsibilities of each window .....	26
Table 6.1 Host environment.....	39
Table 6.2 Command set of the EDDAS system .....	40
Table 6.3 Generated files after execution of the quicksort algorithm .....	44

## LIST OF FIGURES

Figure	Page
Figure 2.1 Configuration of client-server model .....	5
Figure 2.2 Client-server model.....	6
Figure 2.3 DCE components.....	9
Figure 2.4 Remote Procedure Call.....	12
Figure 2.5 CORBA architecture .....	19
Figure 3.1 Basic operations of the DDAS system .....	23
Figure 4.1 Interface file for a quicksort algorithm .....	28
Figure 5.1 Task flow for the EDDAS system .....	30
Figure 5.2 Monitoring of setup processes .....	31
Figure 5.3 Monitoring of the message-passing .....	32
Figure 5.4 CPU usage: connecting .....	34
Figure 5.5 Degree of CPU usage.....	34
Figure 5.6 Initial status of the system configuration file.....	35
Figure 5.7 Status of the system configuration file with machines.....	36
Figure 5.8 User interface .....	37
Figure 6.1 Flow chart for intermediate hosts .....	41
Figure 6.2 Flow chart for final host.....	42
Figure 6.3 Processes of distributed computation .....	45

Figure 6.4 Test 1: CPU states.....	47
Figure 6.5 Test 1: Setup time and working time (sec.).....	48
Figure 6.6 Test 2: CPU states.....	49
Figure 6.7 Test 2: Setup time and working time (sec.).....	50
Figure 6.8 Test 3: CPU states.....	51
Figure 6.9 Test 3: Setup time and working time (sec.).....	52
Figure 6.10 Test 4: CPU states.....	53
Figure 6.11 Test 4: Setup time and working time (sec.).....	54
Figure 6.12 Working time (sec.).....	55

## CHAPTER I. INTRODUCTION

The computer industry has continuously evolved over the past few decades. Although the computer industry is young compared to other industries, it has made rapid improvement in a short time. The merging of computers and communications has had a profound influence on the way computer systems are organized. These systems are called computer networks. A computer network means the collection of interconnected autonomous computers.

This evolution experienced two major advances in the process of development. They are, namely, the production of powerful microprocessors and the invention of high-speed computer networks. They have contributed the computer systems composed of large number of processors and connected over high-speed networks. These systems are called distributed systems. The need for exchanging information among different areas accelerates the rapid deployment of distributed systems. The sharing of rare resources is another motivation for using distributed systems [1].

Distributed systems have been very popular and have been considered as a major issue during the past two decades. In a distributed system, there are several interacting and running processes in different hosts for exchanging information. That is, distributed systems can be defined as a collection of independent computers interconnected over networks. The applications of distributed systems are called distributed computing that is opposed to centralized process computing or single process computing. The major goal of

distributed computing is to reduce the processing time of tasks by distributing execution among multiple processors.

Compared with centralized computing, distributed computing has a lot of advantages. First of all, distributed computation is less expensive to operate intensive computation tasks that have been run on supercomputers. The second is that distributed computing yields higher performance by using high-speed networks and dividing computation. The third is that computation can be migrated from a host with high workload to a host with low workload. This advantage causes more resource utilization and shorter execution time. Distributed computing, moreover, provides sharing expensive resources such as hardware device and data file.

In general, distributed computation systems are classified as distributed computation tools and distributed computation applications. Distributed Computing Environment (DCE), Remote Procedure Call (RPC) and Common Object Request Broker Adapter (CORBA) provide basic models for transferring structured data over networks as programming tools [13]. Based on these tools, client-server model and mobile agents have been applied for distributed computation applications.

Security is one of the most important concerns in the distributed system. Sending data, receiving results and accessing resources are performed on open communication port. Distributed systems, therefore, have to support security of environment [16].

The main goal of this thesis is to enhance the Dynamic Distributed Agents Server (DDAS) system and to develop an interface between the enhanced DDAS system and applications. The enhanced system is called the Enhanced DDAS (EDDAS). The EDDAS system constructs a dynamic network. That is, a user is able to add or remove any host by using the EDDAS system. In the EDDAS system, a user executes application program through the Interface file. The Interface file is in the middle between the EDDAS system and an application program. Java object oriented programming language is used to implement the EDDAS system. Java object oriented programming language provides various communication methods and object message-passing methods [17].

The rest of this thesis is organized as follows. Chapter II provides the literature review related to distributed systems such as client-server model and mobile agents. Chapter III presents the previous work related to the DDAS system. Chapter IV and Chapter V contain the design and implementation of the EDDAS system. Chapter VI discusses the performance of the EDDAS system with application programs. Finally, Chapter VII gives the conclusions and the future work.

## CHAPTER II. LITERATURE REVIEW

The use of distributed systems is one of the most important events of computer technology, and is constantly evolving during the last decade. The reason for this development is the availability of lower cost and more powerful hardware and the improvement in higher execution speed that can be achieved by use of parallelism [2]. With a distributed system, it is possible to concurrently execute a variety of distributed applications.

A distributed system can be defined as a collection of processors interconnected by a communication network. The major goal for using distributed systems is to share resources among a lot of different sites. The distributed systems have become better alternative than other architectures for developing new distributed applications. The distributed systems can achieve better performance, higher system reliability and lower cost. But there are several security problems that have been a major issue in distributed systems. Many distributed applications have tried to resolve these security problems using better algorithms and architectures [16]. This chapter describes distributed computation applications and tools.

### 2.1 Client-Server Model

The client-server model was first used in the 1980s and actually accepted in the late 1980s [36]. This model is extensively used in modern software



products. The use of client-server model ensures higher execution speed, possibility to divide the computing load into other machines and possibility to develop many user interfaces referring to different computer systems. The client-server model is a versatile, message-based and modular infrastructure and is intended to improve usability, flexibility, interoperability and scalability as compared to centralized, mainframe, time sharing computing [36]. The most common model of modern client-server model, the traditional software and hardware configuration of client-server model is shown in Figure 2.1.

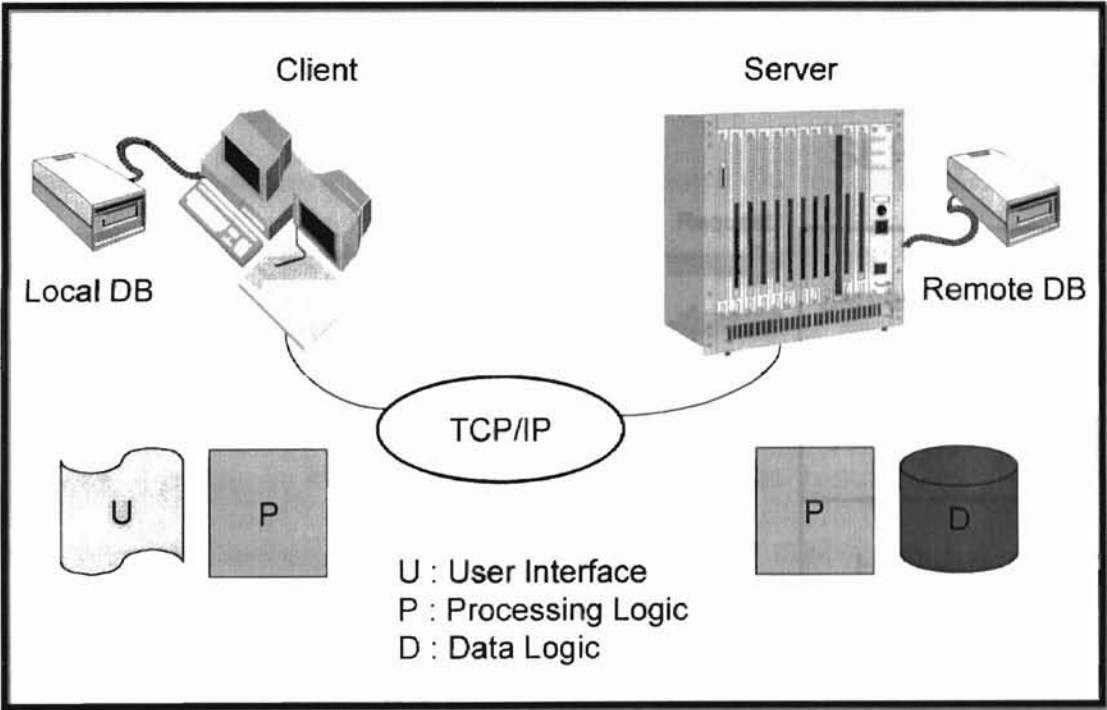


Figure 2.1 Configuration of client-server model (adopted from [35])

Many distributed systems use a client-server model. Client-server model has always been an implementation of a distributed environment. Client-server systems contain two main parts: a client part and a server part. The client part

runs on one computer and the server part runs on another computer. That is, they are distributed. However, some client-server systems are not distributed. In these systems, client and server are separate processes running on the same computer.

As shown in Figure 2.2, the client part makes request for some service and the server part makes response by providing that service. The client might be referred to as a master and the server may be referred to as a slave. In the traditional client-server model, client and server have fixed relationship.

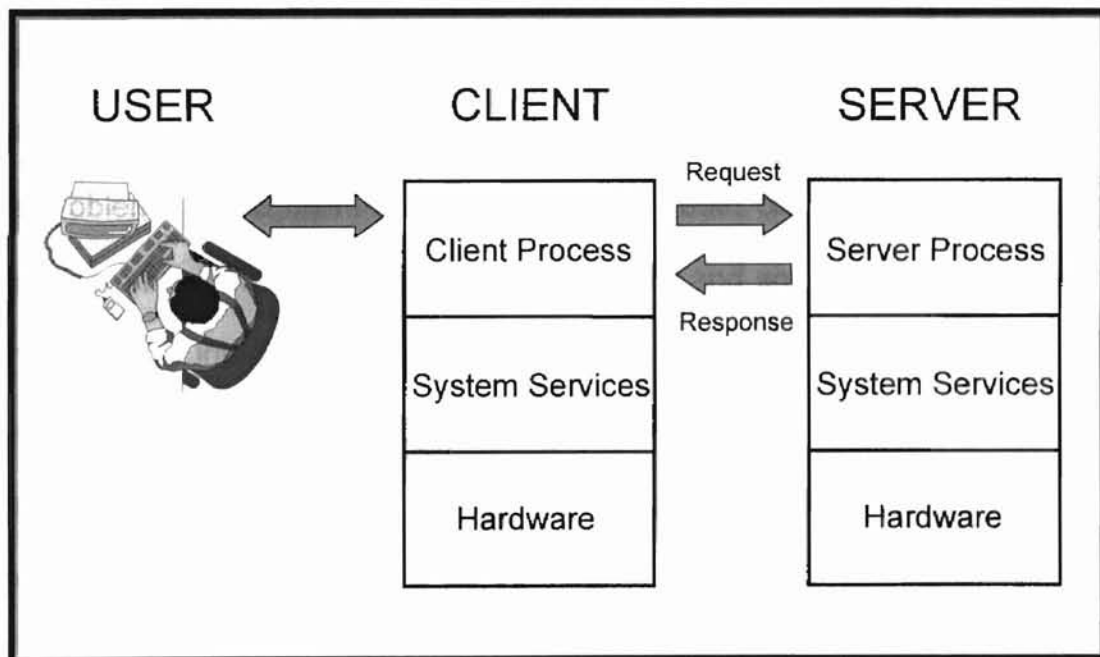


Figure 2.2 Client-server model (adopted from [30])

The major functions of the client are to perform the presentation functions and user's interactions and to execute any client-resident business logic. Although the server is a working machine in the client-server model, the client

has enough power to process the applications and handle the presentation requirements. The server is a working machine that supports multi-user access for services and data. There are many kinds of servers: computation, communication, application and database. Both client and server are expected to execute in a homogeneous system with the exact location of the server known to the client [1].

The client-server model provides several benefits such as longer system life cycle, higher execution speed, lower cost and better software reusability [4]. This model can make it easier for user to access information, to develop application and to manage a distributed computing system. It is easy to construct the client-server model with a software mechanism because this model is constructed on hardware independent environment. On the other hand, one of the problems of the client-server model is that both client and server have the same communication interfaces. When a server and some clients upgrade their functions and interfaces, but some clients do not upgrade their functions and interfaces, it produces serious communication problems in multi-user distributed computation application. In this model, therefore, the maintenance of software version is very important [30].

### **2.1.1 Distributed Computing Environment (DCE)**

A distributed application is composed of separate parts that are executed on different nodes of the network. They cooperate in order to achieve a common

goal. An infrastructure should make the complexity of distributed processing transparent as much as possible. This infrastructure is required to integrate a wide range of computer systems. The Open Software Foundation (OSF) presented an infrastructure called Distributed Computing Environment (DCE) [23]. DCE is a collection of integrated software components that are added to a computer operating system. DCE provides support to build and run distributed applications in heterogeneous environments.

A typical arrangement of DCE components and services is shown in Figure 2.3. The core services of DCE are security, time, threads, directory and the remote procedure call. Figure 2.3 represents the interrelationships among the DCE components [20].

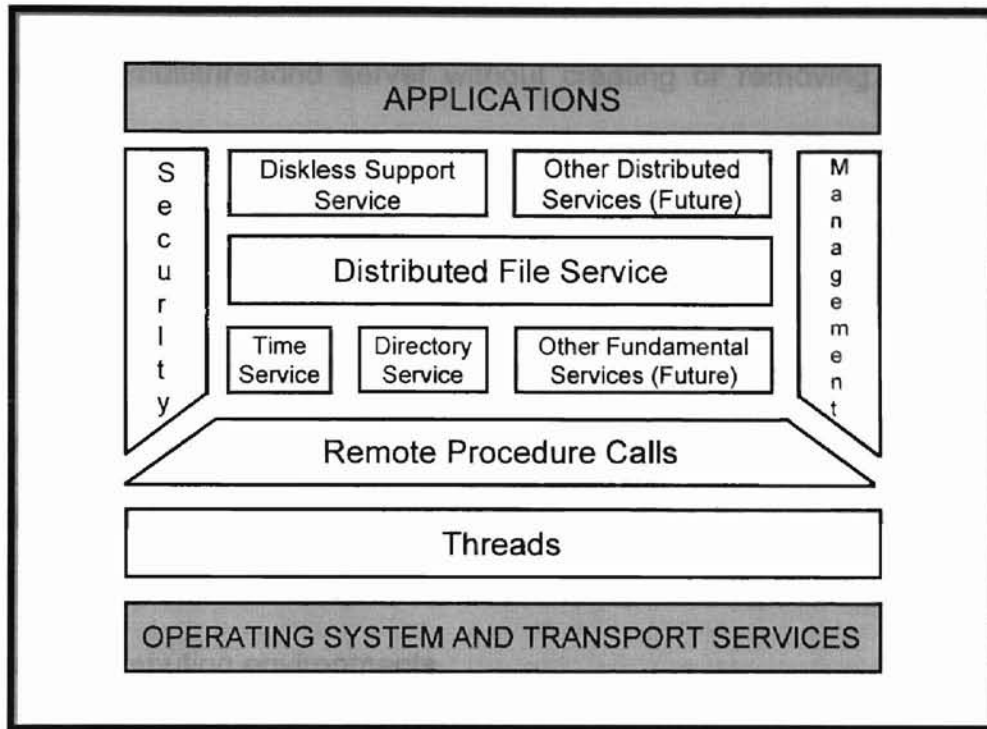


Figure 2.3 DCE components (adopted from [20])

The primary goal of DCE is to overcome the obstacle of heterogeneous transport protocols, operating systems and computer architectures. DCE can be described as the set of software tools and services that make it much easier to develop and operate distributed computing applications. For the primary goal of DCE, DCE supplies services that can be found in other computer network environments, but packs these services so as to make them easier to use. For example, DCE RPC provides a communication method between software modules running on different systems that is simpler to code than older methods [27]. Second, DCE supplies new capabilities. The DCE Security Service supplies a reliable way that decides if a user is allowed to perform a certain action [27]. Third, DCE integrates software components to make them more valuable

components. The DCE RPC uses threads in such a way that a developer can implement a multithreaded server without creating or removing a thread [27]. Finally, DCE supports interoperability and portability. DCE can provide the developer with capability that hides differences among the various hardware, software and network components [27].

DCE provides a set of integrated services that work across multiple systems and remain independent of any single system. DCE makes the distributed computing environment by depending on its own design principles. DCE has been standardized for developing client-server applications in distributed computing environments.

### **2.1.2 Remote Procedure Call (RPC)**

RPC implemented by Sun in 1988 was a major milestone in the area of RPC [31]. The set of RPC tools provided by Sun was designed for performance on a Unix system. RPC made it possible that same tools can be used regardless of the program languages. RPC provides a simple way for the programmer to handle network communication details in client-server applications [15]. The network programs using RPC become simpler tasks.

RPC is the most important and the simplest way to implement the client-server applications. For client-server programming, a message can be used for the interaction between client and server. However the message passing may be prone to error. To ease programming of client-server model, RPC programming

model was developed [33]. RPC removes the task of low level message passing by hiding the complex details of message passing from the user. RPC provides the client with the ability to call a procedure on a remote server, as if the procedure is local to the client.

RPC makes it easier to implement clients and servers. The application developers only have to deal with the simple logic flow of the relationship between subroutines and their callers. This is exorbitant simplification over other communications for building distributed applications [31]. In addition, RPC provides transparency related to the use of the network. Because it hides most of the communications related to use of the network, the application developers can ignore the network and focus on tasks related to the application itself. RPC transparency means that the application developers are hidden from the details of the network type, operating system and computer hardware. This supports the goals of portability and interoperability.

Figure 2.4 shows the relationship between application code and RPC communication during a remote procedure call. In client application code, a remote procedure call looks like a local procedure call because it is a call to client stub code [7]. The client stub communicates with the server stub using the RPC runtime library. The server's RPC runtime library receives the remote procedure call and sends client information to the server stub. The server stub code invokes the remote procedure in the server application code. The server application code is executed in the server address space. When the server finishes executing the remote procedure, the server stub code sends a reply to the client stub code by

using the RPC runtime library. Finally, the client stub code returns output to the client application code.

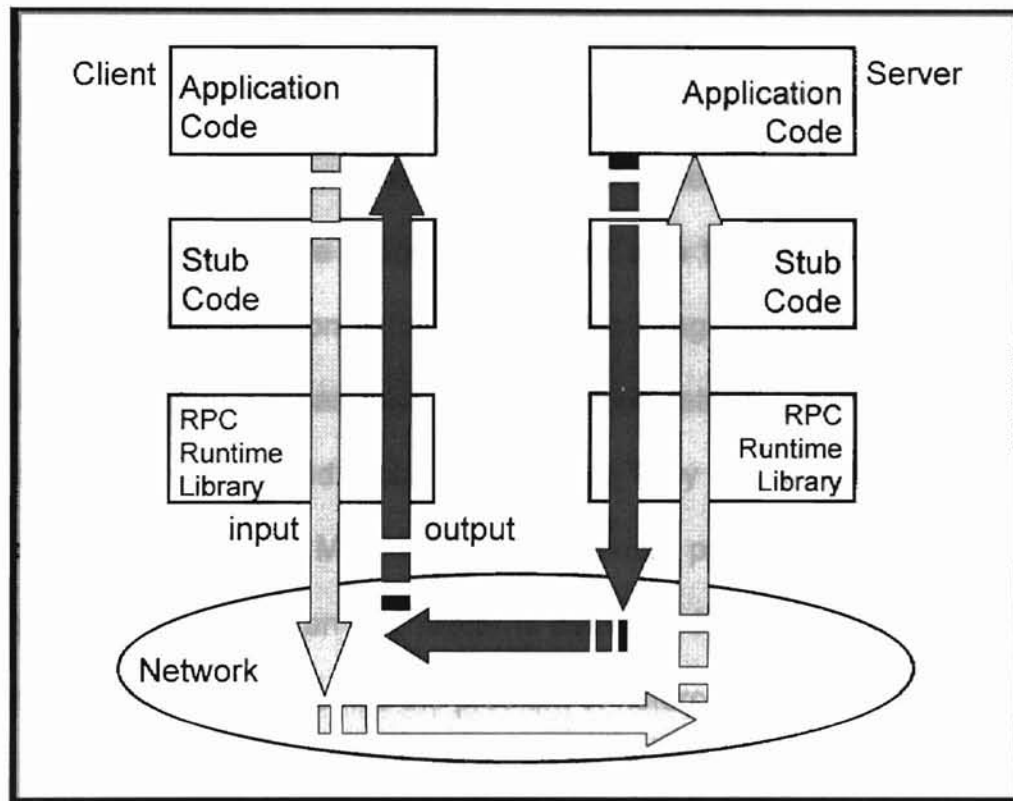


Figure 2.4 Remote Procedure Call (adopted from [15])

RPC is the heart of the heterogeneous OSF DCE [15]. Clients and servers in DCE applications interact through RPC. Many of the DCE features used by the applications, including directory and security service, is used through RPC. The application developers must use DCE RPC to develop distributed applications with DCE.

## 2.2 Mobile Agents



With the recent explosive development of mobile computers and computer networks, the need for communicating information among people at anyplace and anytime has been increasing [38]. Mobile agents offer a new method for the development of applications in distributed systems. Different architectures for their implementations have been proposed. They are a new approach to the architecture and implementation of distributed systems.

Mobile agents are programs that are able to travel through a computer network to fulfill a task on behalf a user. Mobile agents provide effective remote executions in many areas such as information retrieval, network management, electronic commerce and mobile computing. They are an efficient paradigm for distributed applications. Mobile agents are mostly programmed as an interpreted language. This is why an interpreter is available for a wide range of computer systems. Also this overcomes the problem of heterogeneity.

There are two reasons that can justify the use of the mobile agents instead of client-server model. The first is that the size of input data and output data of mobile agents is smaller than that of client-server. The second is that mobile agents can minimize the communication traffic in the distributed system [8]. Using mobile agents in distributed systems has several advantages [10].

- (1) Mobile agent is able to autonomously achieve the goal.
- (2) The resources that can be used by mobile agent can change dynamically in distributed systems.
- (3) Mobile agent uses the network for their migrations during short period.

(4) The architecture of mobile agent is very scalable and reliable.

(5) Mobile agent can coexist with client-server model.

Unfortunately, mobile agents involve some serious security problems for the host and the agent. Because the insertion of foreign code into a local host is at the core of the mobile agents model, security is the most important concern of mobile agents. The detection of hostile agents from authorized agents is difficult because mobile agents use open network systems. There are some solutions to cover security problems by using safe languages such as Safe-TCL, Safe-Python or Java [38]. To solve security problems, several mobile agents systems have been developed.

### **2.2.1 Telescript**

Telescript, developed by General Magic, Inc., is the first system to bring network agents into the public view. Telescript presents the problems of security, mobility and transaction in the marketplace. Telescript is the foundation of new shopping model using online transaction. Telescript is currently used for network management, active e-mail, electronic commerce and business process management [14].

### **2.2.2 Agent Tcl**

Agent Tcl, developed by Robert S. Gray and colleagues at the Dartmouth College Computer Science Department, is a powerful Internet agent system. Agent Tcl runs on UNIX workstation and allows the rapid development of complex agents. Agent Tcl is based on the Tool Command Language (Tcl) developed at Sun Microsystems. Tcl is a high-level scripting language that is both powerful and easy to learn. Tcl was designed to allow programmers to tie together various applications and utilities on UNIX machines. Agent Tcl has simple architecture, communication among agents and docking system that makes an agent jump off a partially connected computer [24]. Agent Tcl is an effective platform for Internet agents and small or medium sized applications. An example Agent Tcl is an agent that travels around a network to look for who is working on each machine and sends this information back to the agent's home.

### **2.2.3 Agents for Remote Access (Ara)**

Mobile agents should be able to move easily without interfering with their execution and utilizing many programming languages and programs [18]. They are independent of the operating systems of the participating machines. This is the basic idea of the Agents for Remote Access (Ara) system. To do this, the system supports facilities for the specific requirements of mobile agents in real application. The Ara system develops a software platform for agents that can move freely when they decide to. An example Ara based on agents also is an

agent that travels around to Web site to look for information and sends it back to the user.

### **2.3 Middleware**

The need to support development of distributed applications using appropriate layer has grown. Its aim is to give transparency to application developers [20]. This software layer is called middleware. Middleware resides in the middle between the application and the operating system. Middleware has been introduced to facilitate the implementation of client-server system in a heterogeneous environment. Middleware provides a unified programming model to application developers and mask out problems of heterogeneity and distribution [6].

Middleware specifies interfaces for distributed operation among computing system, user interaction, system programming and network management. Middleware is able to develop multiple implementations that are available on multiple computing systems. Application developers and system integrators can use middleware. Middleware provides them with the information as the format that they need.

Middleware represents the diverse needs of distributed computing environments. In particular, the connectivity and interoperability are issues of middleware. Middleware includes Distributed Component Object Model (formerly

called Network OLE) of Microsoft, DCE of OSF, RPC of Sun, various standards of the CCITT such as X.400, X.500 and CORBA of OMG [34].

## **2.4 Common Object Request Broker Adapter (CORBA)**

Currently, the integration of object-oriented architecture and distributed system becomes a new issue in the area of distributed computing system. It is called distributed object computing. It is used in many successful cases. Security is still one of the important problems to be solved. Object Management Group (OMG), a consortium of computer hardware and software vendors, creates the object model and the secure architecture for distributed object computing [22]. OMG is developing interface standards for distributed object computing known as Common Object Request Broker Architecture (CORBA).

CORBA is a specification of interfaces and protocols for a distributed computing application. CORBA is based on the object-oriented architecture. The language independence of CORBA supports objects written in different languages to communicate with one another. All object interactions are routed through Object Request Broker (ORB) that communicates through the industry standard Internet Inter-ORB Protocol (IIOP). CORBA provides the ORB that is a standard communication hub through which distributed objects and their clients can interact. Using the ORB, an object and its client can reside in the same process or in different processes in which case they can execute on different hosts connected by a network. ORB provides the software to communicate

requests from client to object and responses from object to client [12]. Also, CORBA supports the Interface Definition Language (IDL). IDL is a declarative language that can be used to specify the operation. IDL maps to various languages like C, C++, Smalltalk and OLE.

CORBA has two major aims. The first is that it makes it easier to implement new applications that place components on different hosts over the network or use different programming languages. The second is that it is able to make open applications that are used as components of large systems. Each application consists of components. The integration is supported by allowing other applications to communicate with these components. Figure 2.5 shows CORBA components.

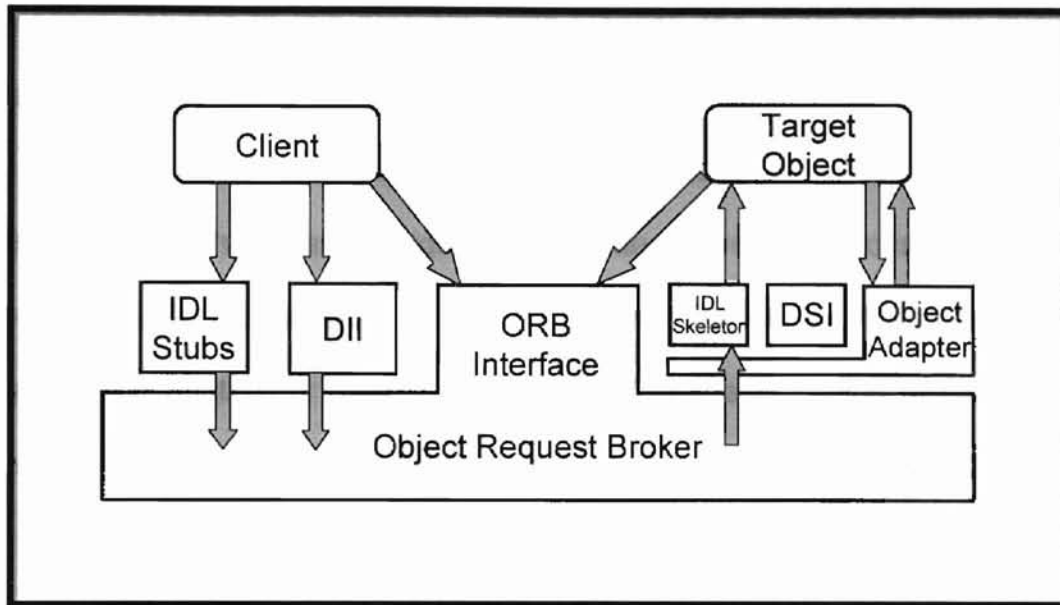


Figure 2.5 CORBA architecture (adopted from [3])

CORBA is a powerful set of tools for developing multi-user distributed applications. CORBA is widely recognized a middleware standard for building distributed, heterogeneous and object-oriented applications. CORBA provides the best technical solution for integrating object-oriented architecture and distributed system [3]. It is robust, interoperable, heterogeneous, multi-platform and multi-vendor supported.

## 2.5 Condor

Workstations are powerful machines executing million of instructions each second. In most cases, the demands of the users are much smaller than the capacity of the workstation. However, the users can face the problem that the

capacity of workstations is too small to meet their demands [26]. Condor system provides a solution to this problem. With Condor, the users take any available capacity that they can access and that can support their needs.

Condor was designed at the University of Wisconsin – Madison. Condor is a distributed batch system designed to provide convenient access to unutilized workstations. Condor supports users who suffer from the limitations of computing on a single workstation. The basic mechanism of the operation of Condor is to determine when a workstation is not in use by its owner. If a workstation is not in use by its owner, the workstation should become a part of available machines. This is supported by measuring both the CPU load and the time since the last keyboard or mouse action. The default for considering a machine as idle state is when the average CPU load measured by UNIX is less than 0.3, and the keyboard or mouse movement is not detected for at least 15 minutes. The workstation owners can adjust these parameters [26].

The networks with workstations have increased in great numbers in recent years. These networks represent powerful computing environments. Using Condor, users can expand their capacity to that of the entire network.

## **2.6 Piranha**

Most parallel processing is static. These programs execute on fixed processors throughout a computation. Adaptive parallelism provides dynamic processors. That is, the number of processors may vary according to availability.



Adaptive parallelism is based on the process model. Processes are dynamically reallocated among free nodes. When a node becomes unavailable, its processes move elsewhere. Piranha is adaptively parallel.

Piranha was developed at Yale University. Piranha gathers idle network nodes and uses them for executing parallel computations. In Piranha, only the number of workers can change while the computation proceeds. Whenever a node becomes idle, the system makes a new worker process on that node. Piranha has task descriptors that are movable and remappable computation units. The task descriptor supports strong heterogeneity [9].

Piranha provides a mechanism for solving large problems by using parallel programs on idle workstations.

## CHAPTER III. PREVIOUS WORK

### 3.1 The DDAS System

To achieve better security and more effective management of distributed computing, Dynamic Distributed Agents Server (DDAS) system is proposed by [21]. The DDAS system is one of the distributed computing applications based on the TCP/IP network environment. The DDAS system provides user with a dynamic network. In this network, nodes can be added or removed as necessary. The DDAS system provides better security because only an authorized user can access a dynamic network.

The DDAS system consists of a DDAS and a Manager. The DDAS uses a specific communication port to construct a dynamic network. After the construction of a dynamic network, the Manager distributes processes, creates processes, runs processes or deletes processes through DDAS. The Manager serves as an interface between the DDAS and a distributed application program. The application program communicates with the Manager through a message-passing method. Jobs are distributed to the nodes over dynamic network, and results are returned to the main node when all nodes complete the jobs.

Figure 3.1 shows the basic operations of the DDAS system. If there is a job in a certain host, the DDAS system constructs a dynamic network. The Manager checks available nodes and sends DDAS components to these nodes. The Manager distributes application program through a dynamic network. After

the computation of application program, the DDAS system in the main host disconnects the dynamic system and removes nodes occupied by the DDAS system in the remote host [21].

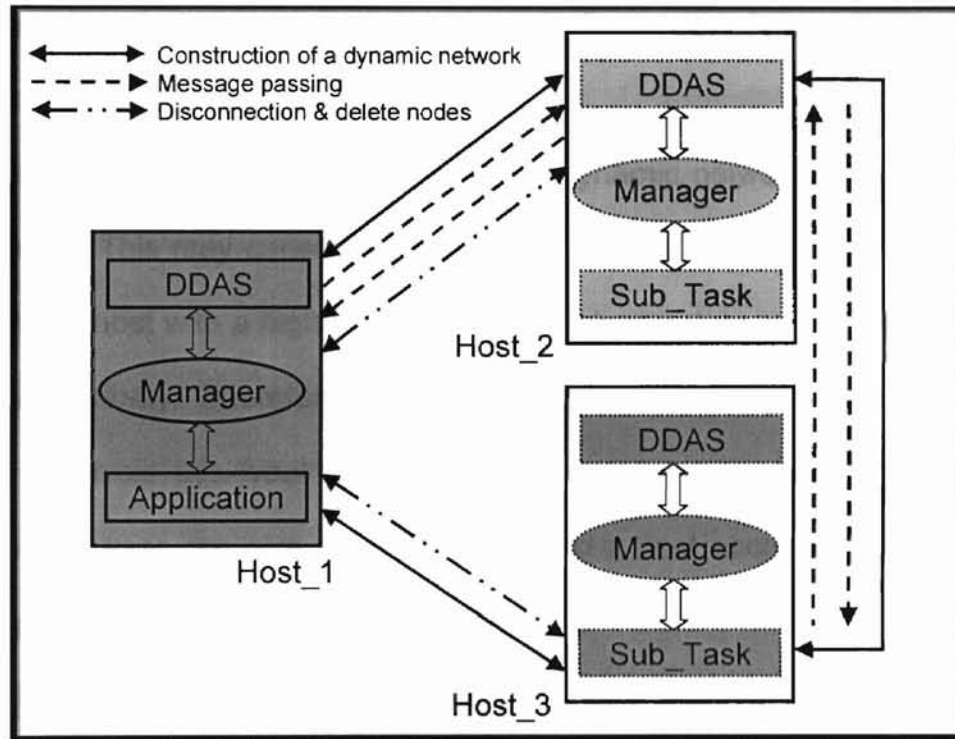


Figure 3.1 Basic operations of the DDAS system (adopted from [21])

The security problem that is one of the major problems in open network system can be reduced because the dynamic network consists of only authorized users and accessible machines.

### 3.2 Drawbacks of the DDAS System

When a distributed computing process is running and one of the hosts with the DDAS system is slow or down, the DDAS system wastes lots of time to wait for the result of computation. A solution to deal with this drawback is to implement monitoring functions over the network. A monitoring function can notify machine status to the user. It may also inform user of hosts that completed their work.

When the DDAS system constructs a dynamic network, each host has a fixed priority. This may cause that a busy host has a higher priority than an idle host. A busy host with a high priority may have larger data to compute than host with lower priority. Therefore, this can decrease the performance of DDAS system. To avoid this weakness, it needs to measure system load degree of each host. A user can give a proper priority to a host according to the system load degree.

The implementation of these functions can improve the performance of DDAS system.

## CHAPTER IV. ENHANCED DDAS SYSTEM

For more effective performance of the DDAS system, the DDAS system is modified and is equipped with a Graphical User Interface (GUI) to monitor and to manage performance. The DDAS system with the addition of these new features is called the Enhanced DDAS (EDDAS) system. A distributed computing system like the DDAS system requires a monitoring function to view status of all machines that participate in communication and a dynamic network to utilize an idle host and to support better security. The EDDAS system satisfies these functions. Also, the EDDAS system can change the priority of remote hosts. A host priority decides the data size to be computed. Therefore, the EDDAS system can adjust the system load of remote hosts. These functions make the performance of the EDDAS system better.

### 4.1 The Monitoring Function

As observed in the previous chapter, a distributed computing process wastes a long time to wait for the result of computation when the host with a distributed computing process is too slow or down. To detect this problem, it needs to have access to monitoring functions. A monitoring function can notify a user of the status of all hosts that participate in communication. A monitoring function for EDDAS system is developed in this research. It is composed of several windows: Manager window to manage all hosts, CPU Usage window to

check the degree of CPU usage on remote hosts, Select Remote Hosts window to select remote hosts that a user wants to use, Remote Host window to understand status of remote hosts and Application Program window to execute an application program. Table 4.1 presents responsibilities of each window.

Window	Responsibilities
Manager window	<ol style="list-style-type: none"> <li>1. Main window</li> <li>2. Manage all hosts that participate in communication</li> <li>3. Commands</li> </ol>
CPU Usage window	<ol style="list-style-type: none"> <li>1. Measure the degree of CPU usage on remote hosts</li> <li>2. UNIX shell programming</li> </ol>
Select Remote Hosts window	<ol style="list-style-type: none"> <li>1. Show available remote hosts</li> <li>2. Select remote hosts that a user wants to use</li> </ol>
Remote Host window	<ol style="list-style-type: none"> <li>1. Represent the status of remote hosts</li> <li>2. Communicate with the Manager window</li> </ol>
Application Program window	<ol style="list-style-type: none"> <li>1. Execute an application program</li> <li>2. Notify Interface, computing algorithm and data</li> <li>3. Use Telnet and FTP</li> </ol>

Table 4.1 Responsibilities of each window

The main goal of using GUI is to develop monitoring function. A user can obtain all processing information of the system from monitoring function using GUI.

## 4.2 The Dynamic Network

When an application program invokes the EDDAS system, this system constructs a dynamic network. Dynamic network construction is adopted from the DDAS system. For establishing a dynamic network, the system checks the system configuration file as one of the configuration setup of this system. The system configuration file named `vns.data` has the IP addresses of the remote hosts that are chosen by a user, the user ID and the user password. By reading the system configuration file, the system investigates if a user is an authorized user in the accessible remote hosts. This means that the dynamic network consists of the only authenticated remote hosts. This makes the EDDAS system a highly secure network using an open network system.

The dynamic network means that a user can add or delete any host as necessary. In EDDAS system, a user can select any host that a user wants to use. Also, a user can change a host priority according to the degree of CPU usage. Both the IP addresses and the priorities of selected hosts are written into the system configuration file. The host selected first by a user has the highest priority. The host with a high priority has larger data than the host with a low priority. Therefore, a user can adjust the system load of remote hosts.

The EDDAS system performs these operations in its own dynamic network environment.

## 4.3 The User Interface for Application Programs

There are many different application programs. These application programs have different formats. A user must build a common format to execute different application programs. For creating a common format, the Interface file is made by user. The Interface file is in the middle between the EDDAS system and an application program. The Interface file has the proper format for the EDDAS system and an application program. The EDDAS system calls an application program through the Interface file. The called application program is performed by communicating with a main host and remote hosts. Figure 4.1 shows the Interface file for a quicksort algorithm.

```

.....
/*                               */
/*                               */
/*                               */
/*                               */
/*                               */
.....
/*                               */
/*                               */
/*                               */
/*                               */
/*                               */
.....

import java.io.*;

class Interface {
    static public void main(String args[]) {
        Quicksort q = new Quicksort();

        DDASControl.readSetupFile();

        if (DDASControl.nextMachine.equals("final")) {
            q.qsort(1, "input.dat");
            DDASControl.sendData(DDASControl.nextMachine, "output2.dat", "input.dat");
            DDASControl.messageCommunication(DDASControl.nextMachine);
            q.qsort(2, "output1.dat");
        }

        else {
            q.qsort(2, "input.dat");
            DDASControl.sendData(DDASControl.prevMachine, "output3.dat", "return.dat");
            DDASControl.msgSend = "return";
            DDASControl.messageSend(DDASControl.prevMachine);
        }
    }
}

```

Figure 4.1 Interface file for a quicksort algorithm



## CHAPTER V. IMPLEMENTATION OF THE EDDAS SYSTEM

The EDDAS system has a monitoring function to view status of all hosts that participate in inter-networking and a dynamic network to utilize an idle host. The EDDAS system also is easy to use user interface for execution of an application program. These functions are implemented in Java programming language. Java programming language provides various communication methods and object message-passing methods. Java programming language provides support for networking. The networking support encapsulates networking classes for better security. Also, the Java bytecode allows solving both the security and the portability problems. Figure 5.1 describes the task flow for the EDDAS system.

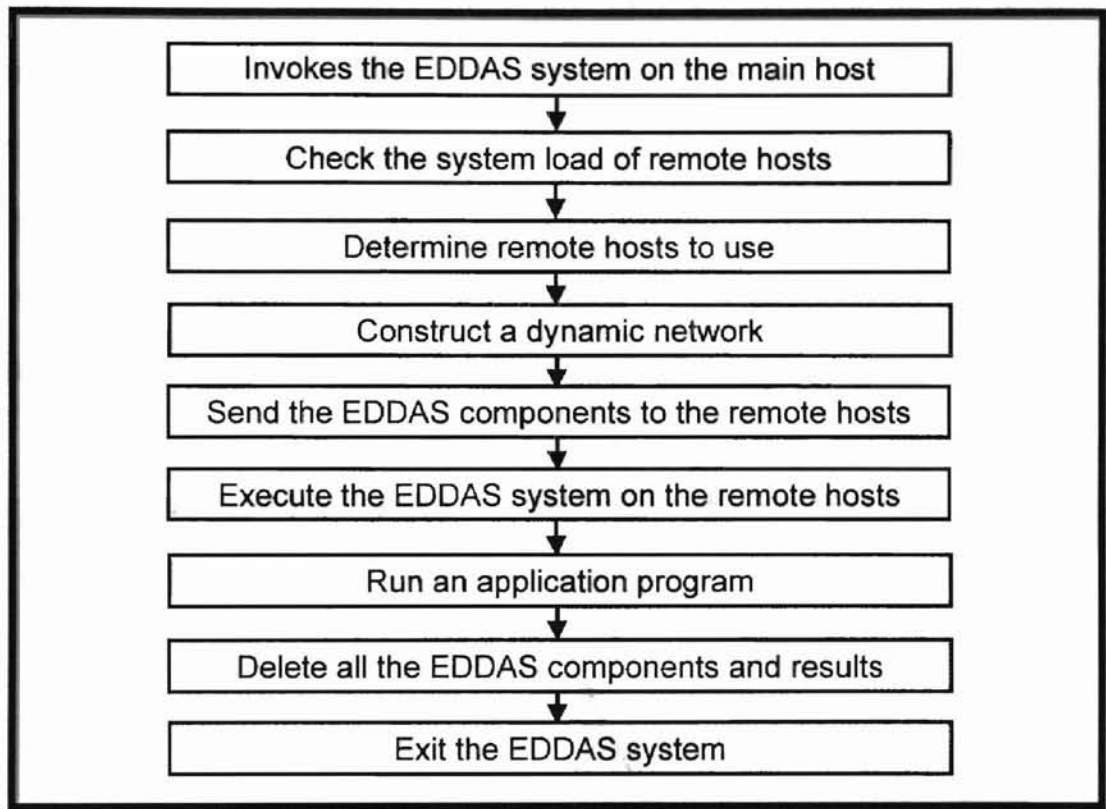


Figure 5.1 Task flow for the EDDAS system

### 5.1 Implementation of the Monitoring Function

The monitoring function of the EDDAS system provides a user with the status of all hosts with the EDDAS components. The Manager window is used for the monitoring of a main host, and the Remote Host window is used for the monitoring of remote hosts.

For communication among hosts, the EDDAS system has two major methods, `connectSocket()` and `openSocket()`, and two major classes, `Sender()` and `Receiver()`. The `connectSocket()` method tries to connect to other hosts, and the `openSocket()` method waits for connecting the hosts. These methods are the

basic functions for the dynamic network, and they are executed on the specific communication port, port number 7777. The connectSocket() and openSocket() are in a main host. Using these methods, the main host sets up the remote hosts. After the remote hosts are set up, they display their current status in the Remote Host window. Figure 5.2 shows the monitoring of setup processes of all hosts that take part in the communication.



Figure 5.2 Monitoring of setup processes

The Sender() and Receiver() classes execute all communications in the system. These classes provide message-passing method for the communication among hosts. With these classes, the EDDAS system distributes system

components, monitors the current status of the remote hosts and executes application algorithms. After the message-passing among hosts is completed, the main host and each remote host inform a user of the its current status. Figure 5.3 presents the monitoring of the message-passing among hosts.

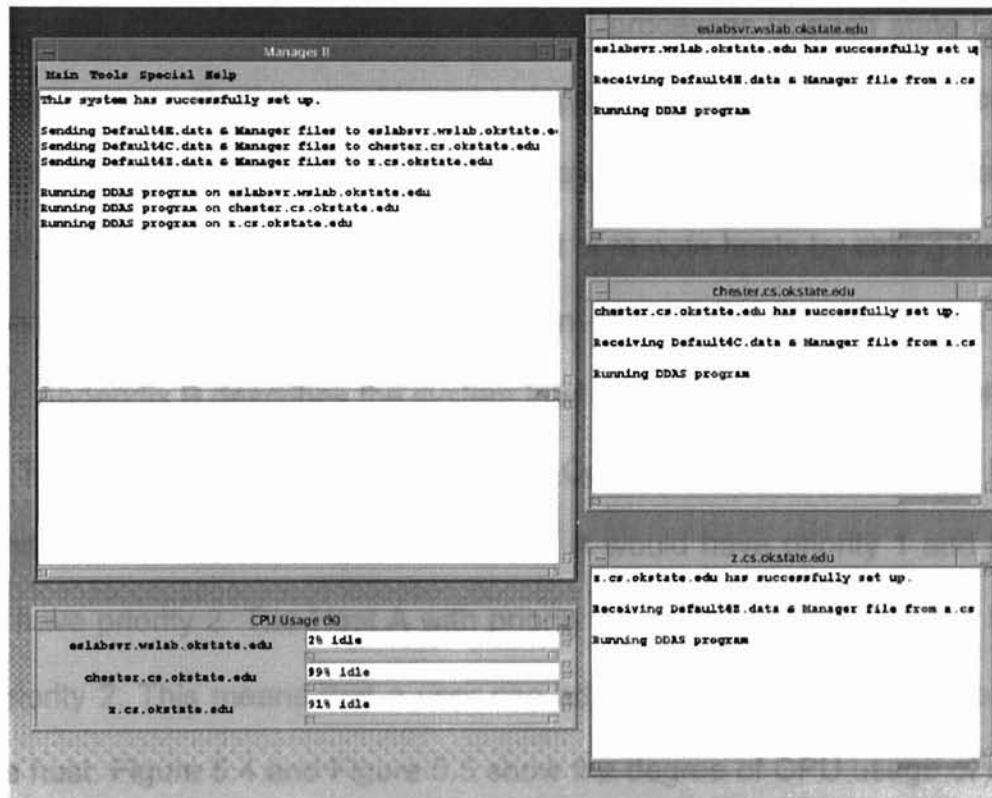


Figure 5.3 Monitoring of the message-passing

## 5.2 Construction of the Dynamic Network

The EDDAS system establishes its own dynamic network for the inter-networking among hosts. Initially, when the EDDAS system is invoked, the system investigates the system load of remote hosts. The system load of remote

hosts is measured by using Telnet and UNIX command TOP. The UNIX command TOP displays the running processes on the system. CPU percentage is used to rank the running processes. The TOP shows general information about the state of the system including the last process ID assigned to a process, the three load averages, the current time and the number of existing processes. It also includes information about the number of processes in each state such as sleeping, running, starting, zombies and stopped and about the load percentage spent in each of the processor states such as user, kernel, IOwait and swap. The EDDAS system investigates the system load of remote hosts by calling the UNIX command TOP, and a user determines an idle remote host and that host's priority. Appendix B describes the system load information through the TOP.

The priority is determined by a user. Consider that a user selects Host A first and selects Host B second. Then, Host A would have priority 1 and Host B would have priority 2. The Host A with priority 1 has larger data than the Host B with priority 2. This means that a user can adjust the data size that is sent to a remote host. Figure 5.4 and Figure 5.5 show the degree of CPU usage of remote hosts.

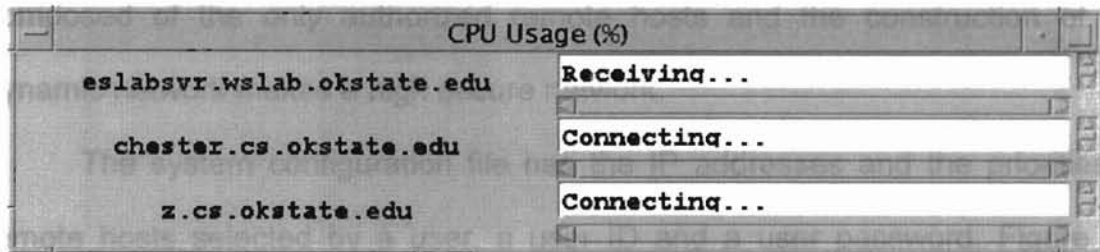


Figure 5.4 CPU usage: connecting

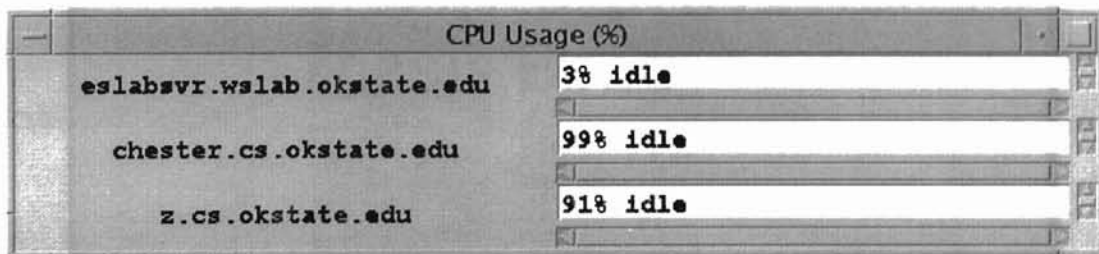


Figure 5.5 Degree of CPU usage

After a user selects the remote hosts, the EDDAS system checks by reading the system configuration file on the main host whether the user is a valid user in the remote hosts or not. If the user is an authorized user in the remote hosts, the user has read-, write-, and execute-privilege. The dynamic network is composed of the only authorized remote hosts and the construction of the dynamic network makes a high secure network.

The system configuration file has the IP addresses and the priorities of remote hosts selected by a user, a user ID and a user password. Figure 5.6 shows the initial status of the system configuration file and Figure 5.7 shows the status of the system configuration file after remote hosts are chosen.

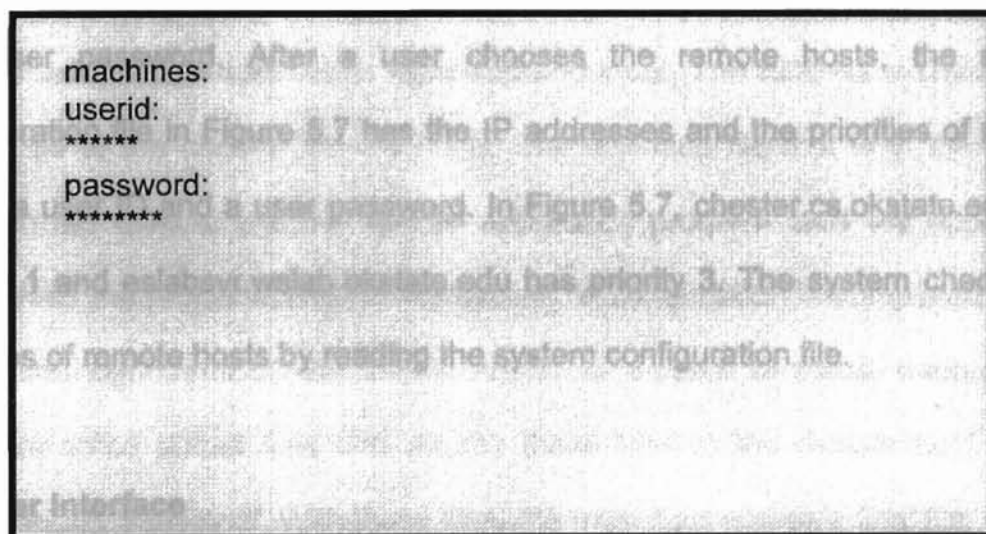


Figure 5.6 Initial status of the system configuration file

```
machines:
chester.cs.okstate.edu
z.cs.okstate.edu
eslabsvr.wslab.okstate.edu
userid:
*****
password:
*****
```

Figure 5.7 Status of the system configuration file with machines

In Figure 5.6, the initial system configuration file has only one user ID and one user password. After a user chooses the remote hosts, the system configuration file in Figure 5.7 has the IP addresses and the priorities of remote hosts, a user ID and a user password. In Figure 5.7, chester.cs.okstate.edu has priority 1 and eslabsvr.wslab.okstate.edu has priority 3. The system checks the priorities of remote hosts by reading the system configuration file.

### 5.3 User Interface

An application algorithm takes advantage of the EDDAS system since the system distributes the application algorithm to the remote hosts. The EDDAS system has easy to use user interface shown Figure 5.8.



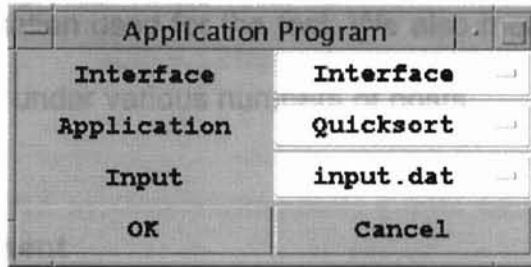


Figure 5.8 User interface

For execution of an application program, the user informs the system of the Interface file, an application algorithm and input. The EDDAS system sends these files to the chosen remote hosts. The Interface file that is in the middle between the EDDAS system and an application program calls the application program as the proper format for the system. A computation is completed by the application algorithms on the remote hosts. As a result of these, each remote host has some output files and returns these files to the remote host with a higher priority than itself. With these files, the main host makes a final file, named the result.dat.

## CHAPTER VI. PERFORMANCE EVALUATION

This chapter shows the host environment and the command set of a test system. We present an application program, network status and the process of an application computation used for the test. We also measure the performance of the EDDAS system under various numbers of hosts.

### 6.1 System Environment

We test the EDDAS system on four machines: a.cs.okstate.edu, z.cs.okstate.edu, chester.cs.okstate.edu and eslabsvr.wslab.okstate.edu. Three machines, a.cs.okstate.edu, z.cs.okstate.edu and chester.cs.okstate.edu, are the host computer system of Computer Science Department and one machine, eslabsvr.wslab.okstate.edu, is the host computer system of Electrical Engineering Department. The a.cs.okstate.edu is used as a main host. This means that a user must execute the EDDAS system from a.cs.okstate.edu.

For execution of an application program, the EDDAS system sends the Interface file and an application algorithm to the remote hosts that are selected by a user. The EDDAS system in the remote host executes the application algorithm and returns the resulting file to the remote host with a higher priority than itself. For example, the remote host with priority 3 returns the resulting file to the remote host with priority 2. Table 6.1 shows the IP address, OS name and Java Virtual Machine (JVM) version of each host used for this test.

Host Name	IP Address	OS	JVM Version
a.cs.okstate.edu	139.78.113.1	SunOS 5.7	1.2.1
z.cs.okstate.edu	139.78.113.110	SunOS 5.7	1.2.1
chester.cs.okstate.edu	139.78.113.102	SunOS 5.6	1.1.3
eslabsvr.wslab.okstate.edu	139.78.96.1	Solaris 2.5.1	1.1.1

Table 6.1 Host environment

When the EDDAS system is executed, a user can set up and check the system by using commands of the system. The EDDAS system provides a user with user control functions such as checking a system load, selecting remote hosts, creating a dynamic network, transferring files and running an application algorithm. Table 6.2 presents the commands and the their descriptions of the EDDAS system.

Oklahoma State University, Stillwater

	Command	Description
Main	Setup	Set up all hosts participating in inter-networking
	Connect	Create a dynamic network
	Close	Close the processes of the remote hosts
	Delete	Delete the EDDAS components on the remote hosts
	Exit	Exit the EDDAS program
Tools	Running	Check the running process on the remote hosts
	Busy	Check the processes of the remote hosts
	Status	Check the status of the remote hosts
Special	Information	Display the Information file
	App	Execute the Application Program window
Help	Help Contents	Display the help message of a command set
	About Manager	Display a system copyright
	About Version	Display a system version

Table 6.2 Command set of the EDDAS system

## 6.2 Distributed Quicksort Algorithm

A quicksort algorithm is used as an application program. Figure 6.1 and Figure 6.2 describes the flow chart of the distributed quicksort algorithm using the EDDAS system. Figure 6.1 shows the flow chart for intermediate hosts and Figure 6.2 shows the flow chart for final host. The EDDAS system and a

quicksort algorithm are in the main host, a.cs.okstate.edu. When a user informs the main host of the quicksort algorithm as an application program, the main host transfers the quicksort algorithm to the remote hosts selected by the user.

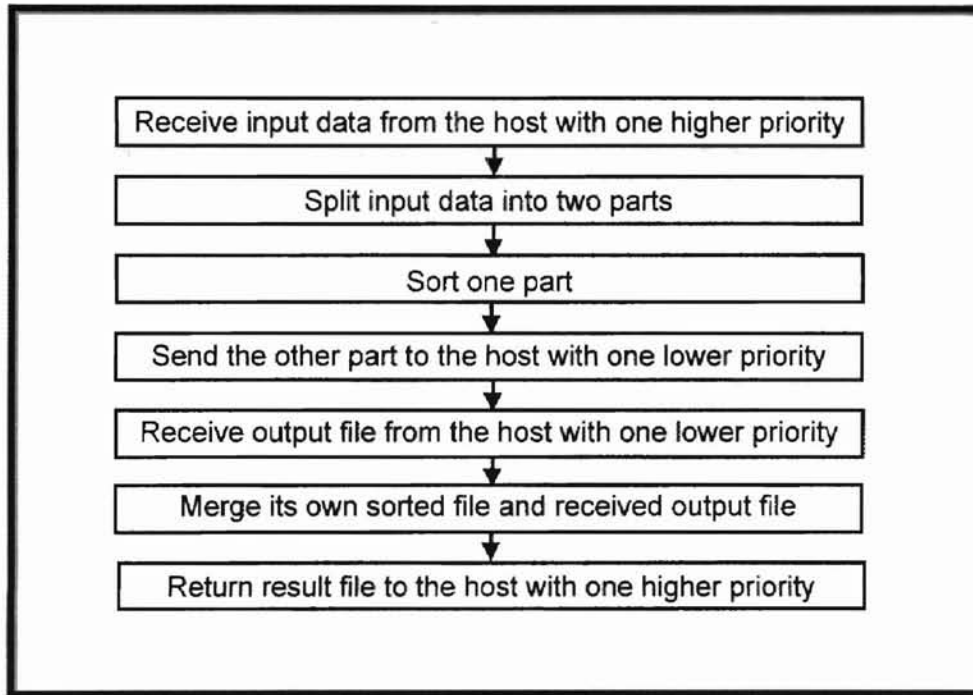


Figure 6.1 Flow chart for intermediate hosts

Oklahoma State University, Stillwater

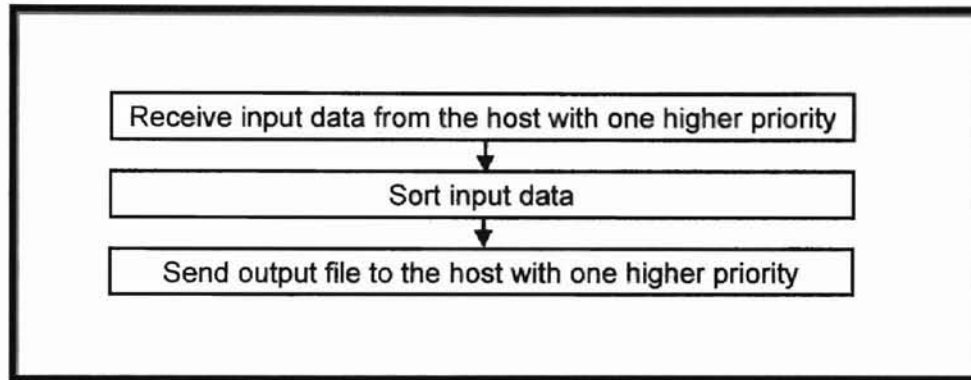


Figure 6.2 Flow chart for final host

The main host splits the input data for the application algorithm into two parts. The main host sorts one part with the quicksort algorithm and sends the other part to the remote host with priority 1. Like the main host, the remote host with priority 1 divides the input data received from the main host into two parts and performs the quicksort algorithm with one part. The other part is transferred to the remote host with priority 2. These processes are repeated up to the final remote host.

The input data that are split into two parts in the main host are named output1.dat and output2.dat. The output1.dat is sorted in the main host by the quicksort algorithm, and the sorted output1.dat is named output3.dat. The output2.dat is transferred to the remote host with priority 1. The remote host with

priority 1 receives output2.dat as input.dat. This remote host performs the same jobs as the main host. That is, the remote host with priority 1 divides input.dat into output1.dat and output2.dat and executes the quicksort algorithm with output1.dat. The sorted output1.dat is named output3.dat. The output2.dat of this remote host is sent to the remote host with priority 2. These executions are performed up to the final remote host. The final remote host receives output2.dat as input.dat, sorts input.dat by using the quicksort algorithm and generates output3.dat.

The output3.dat in the final remote host is returned as return.dat to the remote host with one higher priority. The remote host with one higher priority merges both its own output3.dat and return.dat and generates result.dat. This result.dat is transferred as return.dat to the next remote host. These processes are repeated up to the main host. The main host merges both its own output3.dat and return.dat and generates result.dat. This result.dat is the computation result. Table 6.3 shows the generated files in each host after execution of the quicksort algorithm, and Figure 6.3 describes the processes of distributed computation. Table 6.3 and Figure 6.3 suppose that four hosts participate in inter-networking and their priorities are as follows:

- (1) a.cs.okstate.edu: Main host
- (2) z.cs.okstate.edu: Priority 1
- (3) chester.cs.okstate.edu: Priority 2
- (4) eslabsvr.wslab.okstate.edu: Priority 3

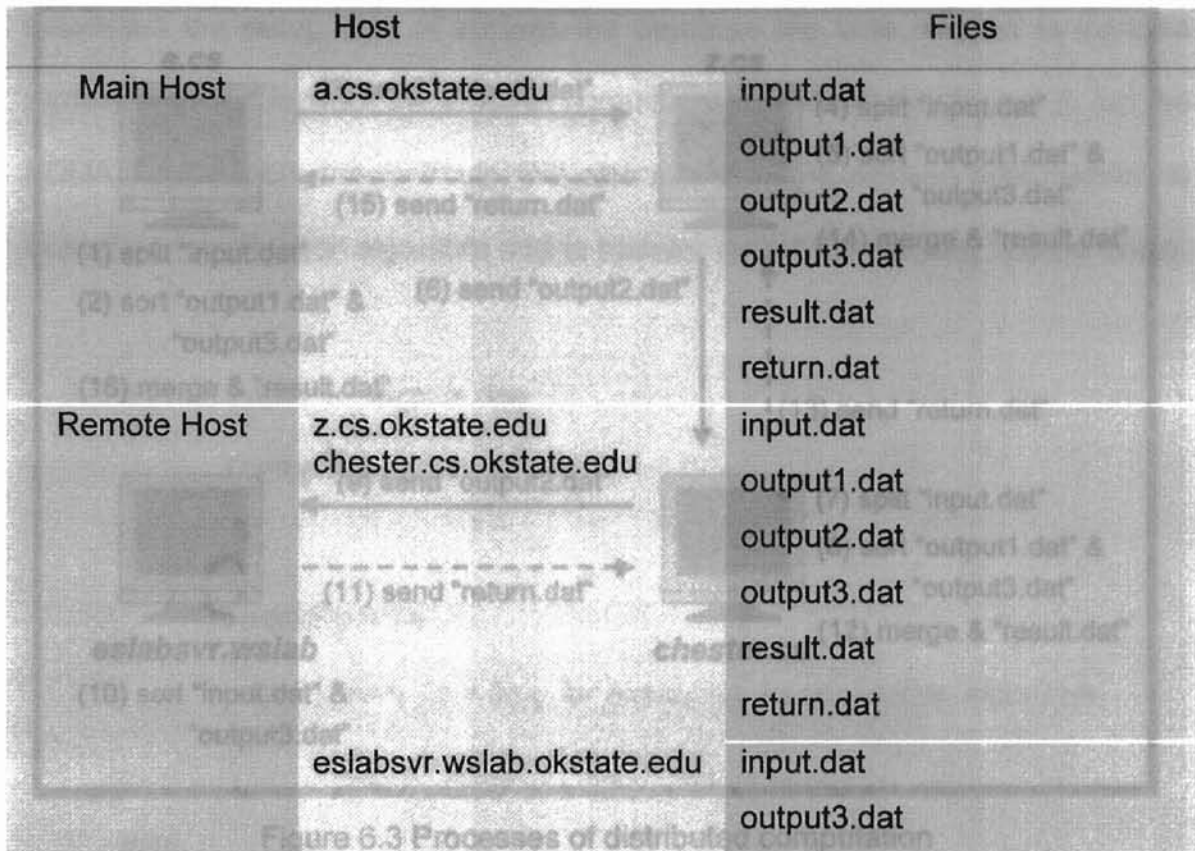


Table 6.3 Generated files after execution of the quicksort algorithm

Oklahoma State University



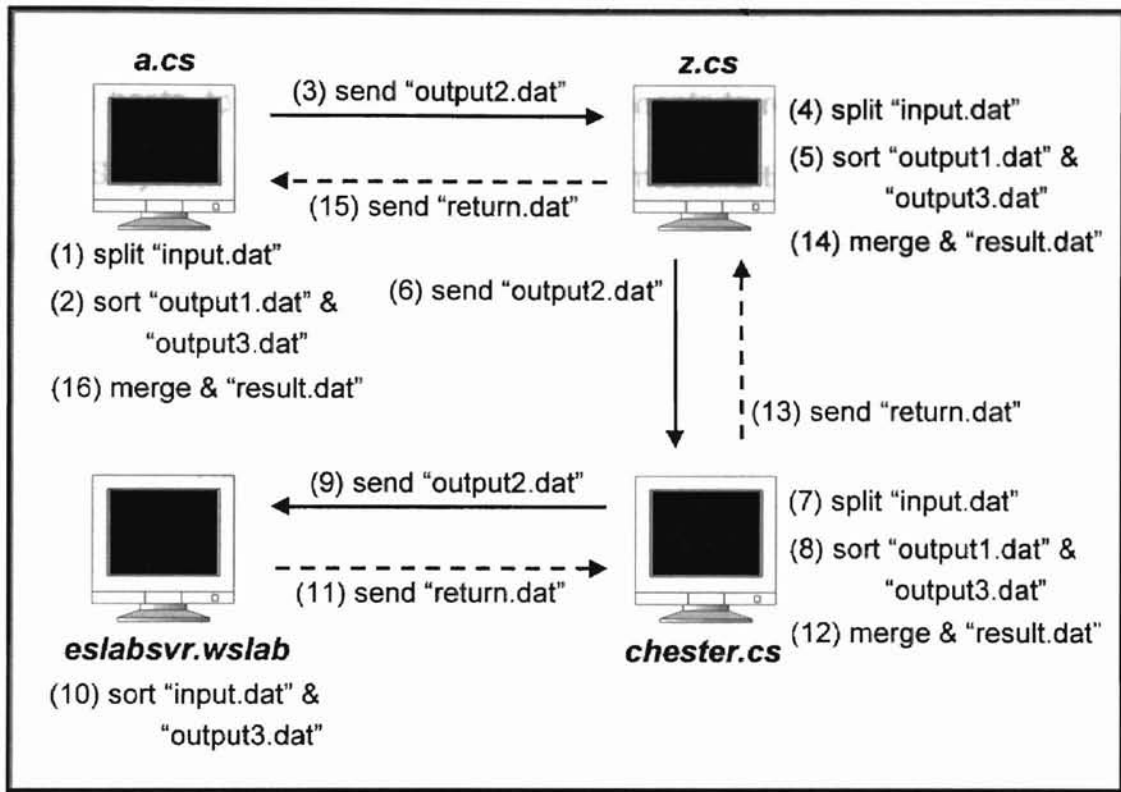


Figure 6.3 Processes of distributed computation

### 6.3 System Performance

We measure the performance of the EDDAS system using several remote hosts. All remote hosts run in the UNIX environment: OS type is SUN Solaris and machine type is SPARC station. The dynamic network of the EDDAS system is created on the specific communication port, port number 7777.

This test consists of setup time, working time and different data sizes. The setup time is the average elapsed time needed to set up remote hosts so that remote hosts are ready to run an application algorithm. Also, the working time is the average elapsed time needed to execute an application algorithm. To

Oklahoma State University

determine the setup time of system, we measure the time needed to initialize remote hosts, to forward the EDDAS components to remote hosts and to run the EDDAS system on remote hosts. We also measure the working time needed to execute an application algorithm and to transfer output files among remote hosts.

*if the number of hosts is one*

$$T_{setup} = 0$$

$$T_{working} = T_e$$

*where  $T_e$  = time for executing an application algorithm*

*else*

$$T_{setup} = T_i + T_s + T_r$$

*where  $T_i$  = time for initializing all remote hosts*

*$T_s$  = time for sending the EDDAS components to all  
remote hosts*

*$T_r$  = time for running the EDDAS system on all remote  
hosts*

$$T_{working} = T_{end} - T_{start}$$

*where  $T_{end}$  = end time of executing an application  
algorithm*

*$T_{start}$  = start time of executing an application*

Oklahoma State University

*algotirhm*

Four inputs are used to control the workload of the remote hosts. This test is performed with different data sizes: 5000, 10000, 20000 and 30000. In this test, a quicksort algorithm is used to illustrate the performance of the EDDAS system. Four tests are made. Results of these tests are shown in Figure 6.4 ~ Figure 6.12.

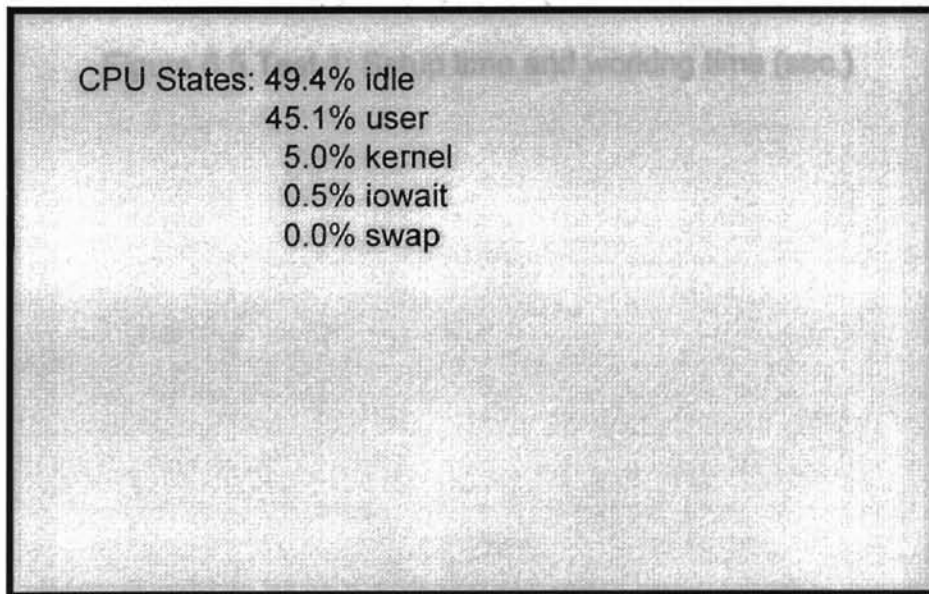


Figure 6.4 Test 1: CPU states

Alabama State University

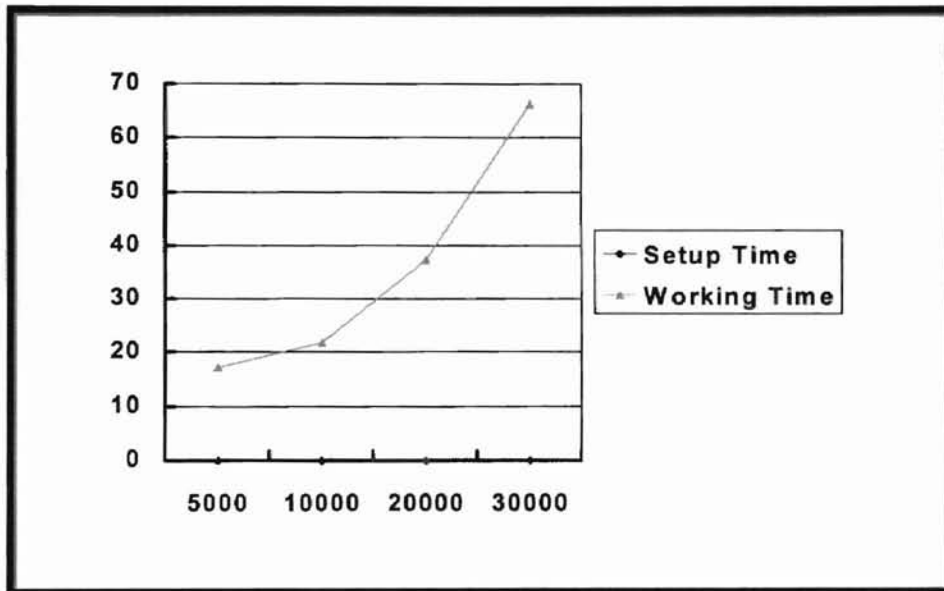


Figure 6.5 Test 1: Setup time and working time (sec.)

Alshammari, G. et al. (2011)

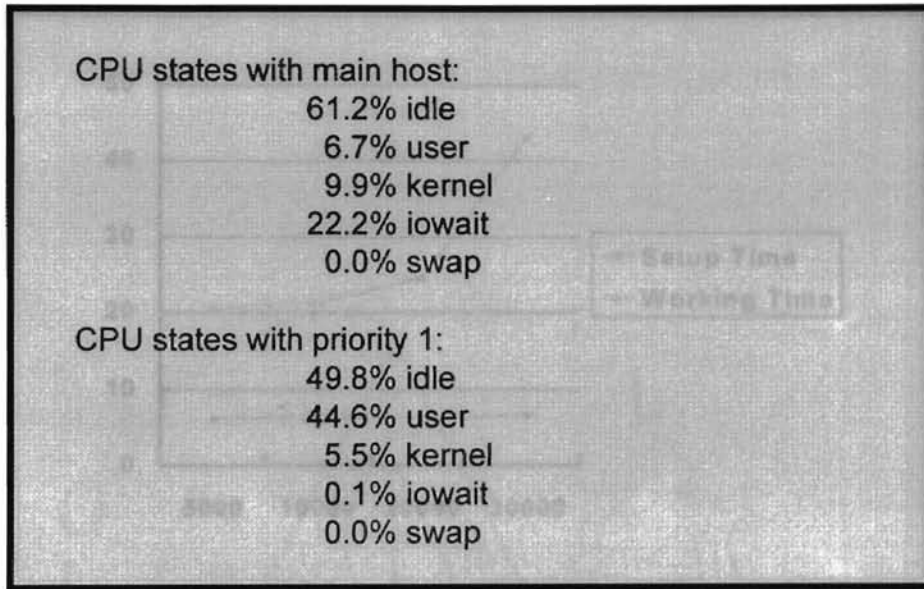


Figure 6.6 Test 2: CPU states

Mohammed Ghalib

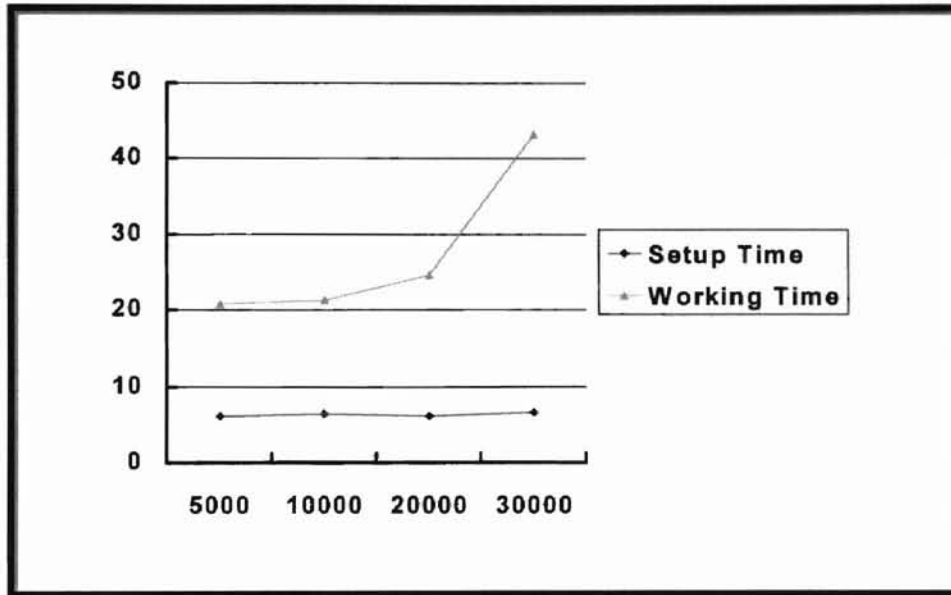


Figure 6.7 Test 2: Setup time and working time (sec.)

Alpham 2011-11-11 10:00

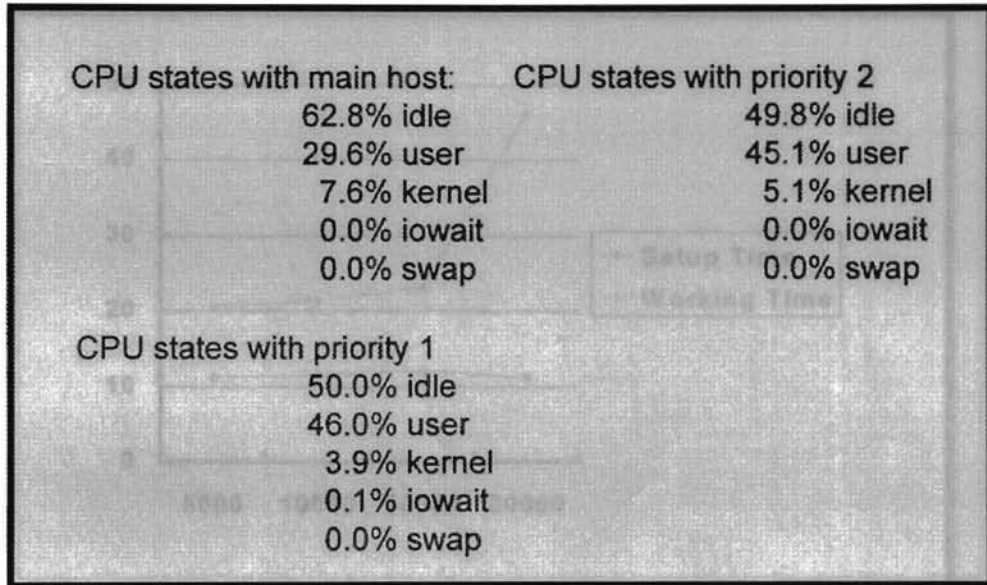


Figure 6.8 Test 3: CPU states

Alpham Oct 11 11:00

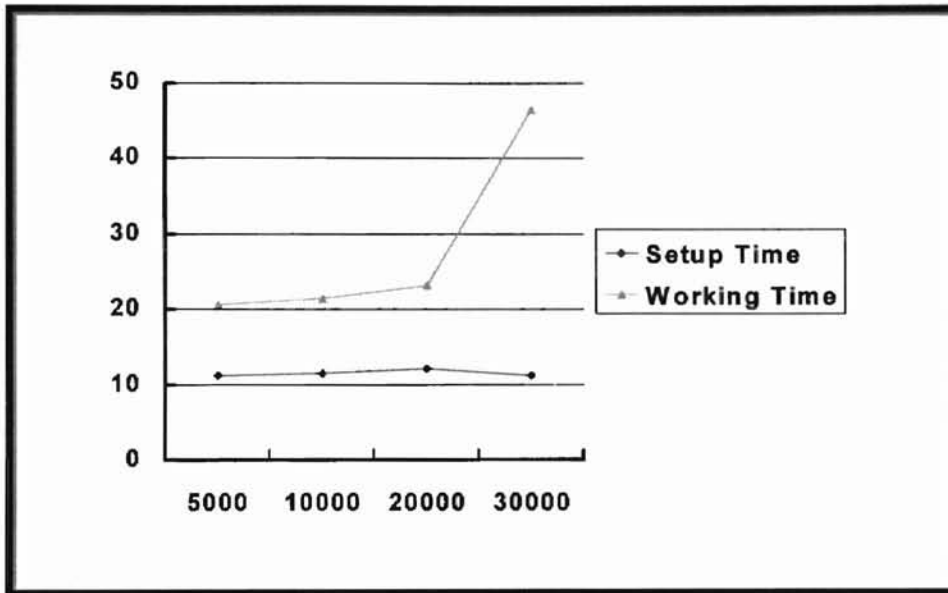


Figure 6.9 Test 3: Setup time and working time (sec.)

Alpham 2022



CPU states with main host		CPU states with priority 2	
	62.8% idle		49.3% idle
	21.8% user		45.5% user
	10.1% kernel		5.2% kernel
	5.3% iowait		0.0% iowait
	0.0% swap		0.0% swap
CPU states with priority 1		CPU states with priority 3	
	66.1% idle		49.2% idle
	30.4% user		46.1% user
	3.5% kernel		4.6% kernel
	0.0% iowait		0.1% iowait
	0.0% swap		0.0% swap

Figure 6.10 Test 4: CPU states

Mpham - 0111111111

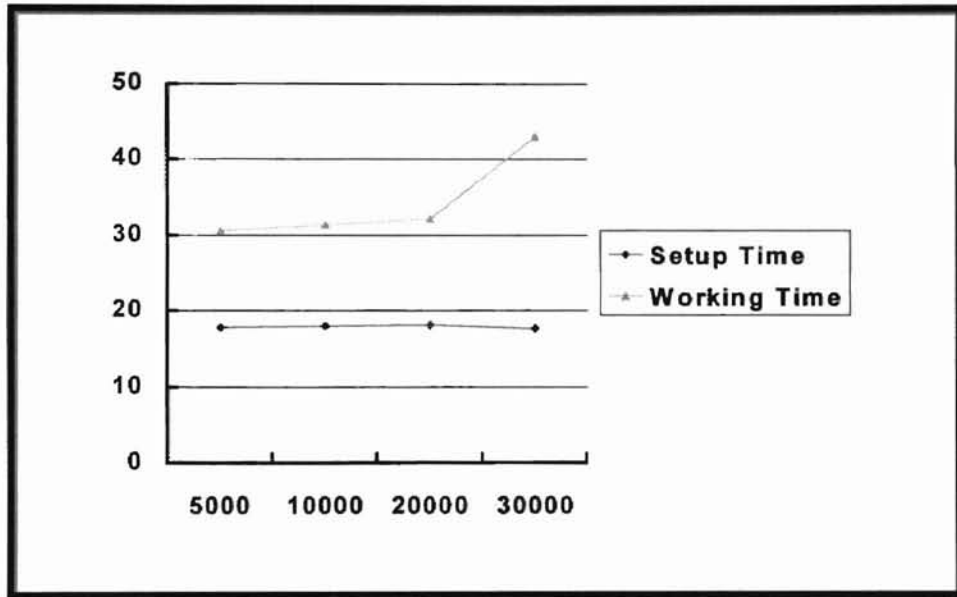


Figure 6.11 Test 4: Setup time and working time (sec.)

Handwritten text: Nishant - 01/11/20

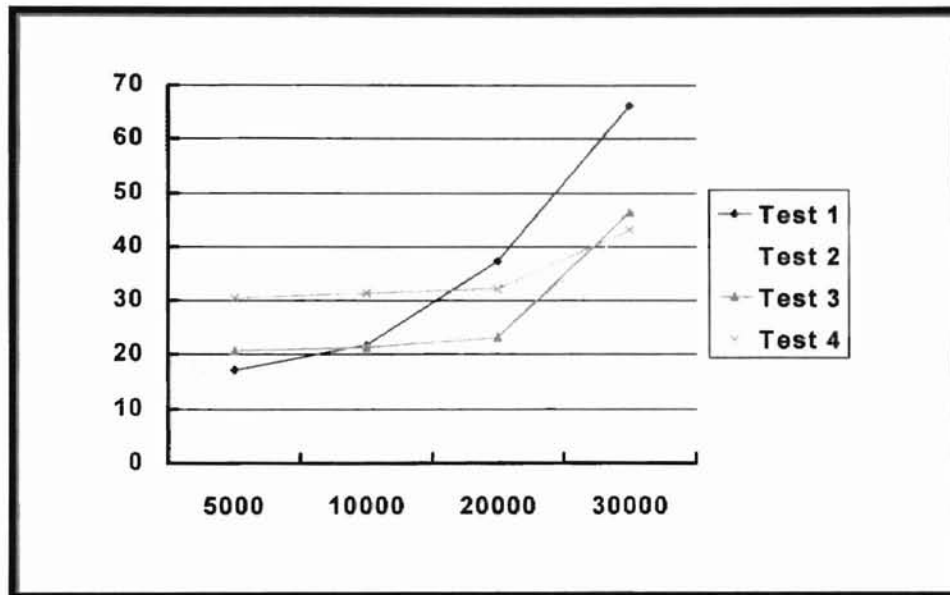


Figure 6.12 Working time (sec.)

Results shown in Figure 6.4 ~ Figure 6.12 indicate that the performance of using one-host is best in data size 5000. However, the performance of using four-host is best in data size 30000. We can conclude that the EDDAS system supports effective system performance and management in large data size.

## CHAPTER VII. CONCLUSIONS

When a user invokes the EDDAS system, the system investigates the system load of remote hosts. According to the system load of remote hosts, the user determines the priority of remote hosts, and the system constructs a dynamic network. Unlike other distributed computation applications, the EDDAS system does not require a predefined network. The EDDAS system creates its own dynamic network by authenticating the remote hosts. It means that the dynamic network consists of the only authorized remote hosts. An application program is performed by using message-passing method for the EDDAS system. For execution of the application program, the system uses an easy user interface. Using this user interface, a user informs the system of the Interface, algorithm and input for the application program. After the application program is completed, the system disconnects the dynamic network and removes the processes and the components for the EDDAS system in the remote hosts. These processes are monitored by GUI. The monitoring function provides a user with the status of all hosts that take part in inter-networking. During the running time, the EDDAS system supports more effective system management and better security for the distributed system than the DDAS system.

## REFERENCES

- [1] Istabrak Abdul-Fatah and Shikharesh Majumdar, Performance Comparison of Architectures for Client-Server Interactions in CORBA, *Distributed Computing Systems*, IEEE Computer Society, pp. 2-11, 1998.
- [2] T. K. Apostolopoulos and K. C. Pramataris, A Client-Server Architecture for Distributed Problem Solving, *Proceedings of IEEE Singapore International Conference*, pp. 513-517, 1995.
- [3] Sean Baker, *CORBA Distributed Objects Using Orbix*, Addison-Wesley, 1997.
- [4] Regis J. Bates, *Hands-On Client/Server Internetworking*, McGraw-Hill, Inc., 1998.
- [5] M. Bertocco, F. Ferraris, C. Offelli and M. Parvis, A Client-Server Architecture for Distributed Measurement Systems, *Instrumentation and Measurement Technology Conference*, IEEE, Vol. 1, pp. 67-72, 1998.
- [6] Gordon S. Blair, Geoff Coulson, Nigel Davies, Philippe Robin and Tom Fitzpatrick, Adaptive Middleware for Mobile Multimedia Applications, *Network and Operating System Support for Digital Audio and Video*, IEEE, pp. 245-254, 1997.
- [7] John Bloomer, *Power Programming with RPC*, O'Reilly & Associates, Inc., 1992.
- [8] L. M. Camarinha-Matos and Walter Vieira, Mobile Agents and Remote Operation, *Intelligent Engineering System*, IEEE, pp. 463-468, 1997.

- [9] Nicholas Carriero, Eric Freeman, David Gelernter and David Kaminsky, Adaptive Parallelism and Piranha, *Computer*, Vol. 28, Issue. 1, pp. 40-49, IEEE, 1995.
- [10] Manfred Dalmeijer, Eric Rietjens, Dieter Hammer, Ad Aerts and Michiel Soede, A Reliable Mobile Agents Architecture, *Object-Oriented Real-Time Distributed Computing*, IEEE, pp. 64-72, 1998.
- [11] A. A. El-Zoghabi, M. A. Ismail, S. N. T. Shen and E. A. Korany, A Client-Server Computing Model for Heterogeneous Distributed knowledge Management, *Southeastcon '93, Proceedings*, IEEE, 1993.
- [12] Eric Evans and Daniel Rogers, Using Java Applets and CORBA for Multi-User Distributed Applications, *IEEE Internet Computing*, Vol. 1, Issue. 3, pp. 43-55, 1997.
- [13] Jim Farley, *Java Distributed Computing*, O'Reilly & Associates, Inc., 1998.
- [14] Robert Gray, David Kotz, Saurab Nog, Daniela Rus and George Cybenko, Mobile Agents: The Next Generation in Distributed Computing, *Parallel Algorithms/Architecture Synthesis*, IEEE Computer Society, pp. 8-24, 1997.
- [15] David Gunter, *Client/Server Programming with RPC and DEC*, QUE, 1995.
- [16] P. W. Halliden, Security for Distributed Applications, *Security and Detection*, pp. 156-160, IEE, 1995.
- [17] Elliotte Rusty Harold, *Java Network Programming*, O'Reilly & Associates, Inc., 1997.

- [18] Craig Hunt, *TCP/IP Network Administration*, O'Reilly & Associates, Inc., 1993.
- [19] Jerry R. Jackson and Alan L. McClellan, *JAVA by Example*, SunSoft Press, 1997.
- [20] Raman Khanna, *Integrating Personal Computers in a Distributed Client-Server Environment*, Prentice-Hall, Inc., 1995.
- [21] Kwan-Sung Kim, *DDAS: Design and Implementation of a Kernel Application for Adaptive Distributed Computation*, Oklahoma State University, 1999.
- [22] Kyeongbeom Kim, Youngkyun Kim, Youngkee Song and Soran Ine, A Software Platform for Secure Applications based on CORBA, *Distributed Computing Systems*, IEEE Computer Society, pp. 22-27, 1997.
- [23] Charles Knouse, *Practical DCE Programming*, Prentice-Hall, Inc., 1996.
- [24] David Kotz, Robert Gray, Saurab Nog, Daniela Rus, Sumit Chawla and George Cybenko, Agent Tcl: Targeting the Needs of Mobile Computers, *IEEE Internet Computing*, Vol. 1, Issue. 4, pp. 58-67, 1997.
- [25] John Lewis and William Loftus, *JAVA Software Solutions*, Addison-Wesley, 1998.
- [26] Mike Litzkow and Miron Livny, Experience with the Condor Distributed Batch System, *Experimental Distributed Systems*, IEEE, 1990.
- [27] Harold W. Lockhart, *OSF DCE*, McGraw-Hill, Inc., 1994.
- [28] David Medinets, *Unix Shell Programming Tools*, McGraw-Hill, Inc., 1999.

- [29] Patrick Naughton and Herbert Schildt, *The Complete Reference JAVA 1.1*, McGraw-Hill, 1998.
- [30] Paul E. Renaud, *Introduction to Client/Server Systems*, John Wiley & Sons, Inc., 1993.
- [31] C. Sashidhar and S. M. Shatz, Design and Implementation Issues for Supporting Callback Procedures in RPC-Based Distributed Software, *Computer Software and Application Conference*, IEEE Computer Society, pp. 460-466, 1997.
- [32] Jeffrey D. Schank, *Novell's Guide to Client-Server Applications and Architecture*, Novell Press, 1994.
- [33] Alexander Schill, Christian Mittasch, Otto Spaniol and Claudia Popien, *Distributed Platforms*, Chapman & Hall, 1996.
- [34] Alan R. Simon and Tom Wheeler, *Open Client/Server Computing and Middleware*, AP Professional, 1995.
- [35] Seungwoo Son, Injoong Yoon and Changkap Kim, A Component-Based Client/Server Application Development Environment Using Java, *Technology of Object-Oriented Language*, pp. 168-179, 1998.
- [36] Antonella Di Stefano, Lucia Lo Bello and Corrado Santoro, A Distributed Heterogeneous Database System Based on Mobile Agents, *Seventh IEEE International Workshops*, pp. 223-228, 1998.
- [37] Andrew S. Tanenbaum, *Computer Networks*, Prentice-Hall, Inc., 1996.



- [38] Hartmut Vogler, Thomas Kunkelmann and Marie-Louise Moschgath, Distributed Transaction Processing as a Reliability Concept for Mobile Agents, *Distributed Computing System*, IEEE, pp. 59-64, 1997.

## APPENDIX A

### EDDAS SYSTEM SOURCE CODE IN JAVA

1110

```

/*****/
/*                                     */
/*           Manager.java             */
/*                                     */
/* Functions:                          */
/* 1. This program manages DDAS system. */
/* 2. This program executes each command. */
/*                                     */
/*****/

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

class Manager extends DDAS {
    public static final int MAX_MACHINE = 100;
    static final int ENROLLED_CMD_NO = 11;
    static final int MAX_ARGS = 5;

    public static String[] machineNames = new String[MAX_MACHINE];

    static String userID = null;
    static String passWord = null;

    static String inputCommand = null;
    public static int machineNumber = 0; // available machine counts
    static StringTokenizer st;

    static String[] enrolledCommands = {"?", "busy?", "delete",
                                        "establish", "execute", "exit",
                                        "kill", "running?", "send",
                                        "set", "status"};

    public static boolean debug = true;
    static Quicksort q = new Quicksort();

    public static void manager() {
        initialSystem();
    }

    static void initialSystem() {
        machineNumber = readSetupFile();
    }
}

```

```

}

static void inputCheck() {
    int commandNumber = 0;

    try{
        BufferedReader stdin =
            new BufferedReader(new InputStreamReader(System.in));
        inputCommand = stdin.readLine();

        st = new StringTokenizer(inputCommand);
        String[] arguments = new String[MAX_ARGS];

        int saveBufCnt = st.countTokens();

        for (int i = 0; i < saveBufCnt; i++)
            arguments[i] = st.nextToken();

        commandNumber = makeCommandToInt(arguments[0]);
        commandService(commandNumber, arguments, saveBufCnt);
    } catch (IOException e) {
        System.err.println(e);
    }
}

//input command service Method

public static void commandService(int exeNo, String[] argsOther, int argNo) {
    String sendMessage = null;

    switch(exeNo) {
        // ? help service
        case 0:
            if (argNo == 1) {
                for (int i = 0; i < ENROLLED_CMD_NO; i++)
                    System.out.println(enrolledCommands[i]);
            }
            else
                helpService(argsOther);
            break;

        // busy? check service
        case 1:
            sendMessage = "busy?";
    }
}

```

```

makeSendMessage (sendMessage);

if (argNo == 1)
    messageCommunication();
else if (argNo == 2)
    messageCommunication(argsOther[1]);
else
    displayErrorMessage(0);
break;

// delete file service
case 2:
    deleteAllFiles();
    break;

// establish service
case 3:
    if (argNo < 2) {
        sendFiles();
        executeFiles();
    }
    else
        displayErrorMessage(0);

    break;

// execute file service
case 4:
    q.qsort(1, "input.dat");

    sendData(machineNames[0], "output2.dat", "input.dat");

    sendMessage = "execute";
    makeSendMessage (sendMessage);
    messageCommunication(machineNames[0]);

//execute quicsort ...
q.qsort(2, "output1.dat");

openSocket();
while(nextFlag == 0)
    pause(PAUSE);

q.concatFile();
break;

```

```

// exit service
case 5:
    System.exit(0);
    break;

// kill process service
case 6:
    sendMessage = "kill";
    makeSendMessage (sendMessage);

    if (argNo == 1)
        messageCommunication();
    else if (argNo == 2)
        messageCommunication(argsOther[1]);
    else
        displayErrorMessage(0);
    break;

// running? service
case 7:
    sendMessage = "running?";
    makeSendMessage (sendMessage);

    if (argNo == 1)
        messageCommunication();
    else if (argNo == 2)
        messageCommunication(argsOther[1]);
    else
        displayErrorMessage(0);
    break;

// send service
case 8:
    if (argNo == 1) {
        sendMessage = "send";
        makeSendMessage (sendMessage);
        messageCommunication();
    }
    else if (argNo == 2) {
        sendMessage = "send " + argsOther[1];
        makeSendMessage (sendMessage);
        messageCommunication();
    }
    else
        displayErrorMessage(0);
    break;

```

```

    // set environment service
case 9:
    if (argNo == 1)
        displayEnvironment();
    else if (argNo == 3)
        setEnvironment(argsOther);
    else
        displayErrorMessage(0);
    break;

    // status service
case 10:
    sendMessage = "status";
    makeSendMessage (sendMessage);

    if (argNo == 1)
        messageCommunication();
    else if (argNo == 2)
        messageCommunication(argsOther[1]);
    else
        displayErrorMessage(0);
    break;

default:
    break;
}
}

```

```

static void messageCommunication() {
    try {
        FileReader fr = new FileReader("vns.data");
        BufferedReader br = new BufferedReader(fr);
        String s;

        br.readLine();

        for (int i = 0; i < machineNumber; i++) {
            Server = machineNames[i];

            connectSocket();
            while(nextFlag == 1)
                pause(PAUSE);

            openSocket();
        }
    }
}

```

```

        while(nextFlag == 0)
            pause(PAUSE);

        s = br.readLine();
    }
    fr.close();
} catch (IOException e) {
    System.err.println(e);
}
}

static void messageCommunication(String machine) {
    Server = machine;

    connectSocket();
    while(nextFlag == 1)
        pause(PAUSE);

    openSocket();
    while(nextFlag == 0)
        pause(PAUSE);
}

static void makeSendMessage(String sendMessage) {
    msgSend = sendMessage;
}

/* Set Environment (commands and arguments)
 * set
 * set changeld    userid
 * set changePasswd password
 * set addMachine  machineName
 * set deleteMachine machineName
 */

static void setEnvironment(String[] argsOther) {
    if(argsOther[1].equals("changeld")) {
        userID = argsOther[2];
    }
    else if(argsOther[1].equals("changePasswd")) {
        passWord = argsOther[2];
    }
    else if(argsOther[1].equals("addMachine")) {

```



```

        machineNames[machineNumber] = argsOther[2];
        machineNumber++;
    }
    else if(argsOther[1].equals("deleteMachine")) {
        machineNames[machineNumber] = argsOther[2];
        machineNumber--;
    }
    else {
        displayErrorMessage(2);
    }
}

```

```

static void displayEnvironment() {
    System.out.println("Machines:");

    for (int i=0 ; i < machineNumber; i++)
        System.out.println(" " + machineNames[i]);

    System.out.println("UserID: " + userID);
    System.out.println("Password: " + passWord);
}

```

```

static void helpService(String[] argsOther) {
    int serNo;

    serNo = makeCommandToInt (argsOther[1]);

    switch(serNo) {
        // ?
        case 0:
            System.out.println(" ? Commands display service.");
            break;

            // busy?
        case 1:
            System.out.println(" busy? Check running program in other
machines.");
            System.out.println();
            System.out.println(" busy? --- Display all conected machine");
            System.out.println(" busy? machine_name --- Display input machine");
            break;

            // delete
        case 2:

```

```

        System.out.println(" delete  Delete all files about DDAS.");
        System.out.println();
        System.out.println(" delete          --- Deleted all connected
machine");
        System.out.println(" delete machine_name  --- Deleted input machine");
        break;

        // establish
    case 3:
        System.out.println(" establish  Establish dynamic virtual network.");
        break;

        // execute
    case 4:
        System.out.println(" execute  Execute program.");
        System.out.println();
        System.out.println(" execute program_name          --- execute input
program");
        System.out.println(" execute program_name machine_name");
        break;

        // exit
    case 5:
        System.out.println(" exit  Exit DDAS program.");
        break;

        // kill
    case 6:
        System.out.println(" kill  Kill the process of other machines.");
        System.out.println();
        System.out.println(" kill machine_name");
        break;

        // running?
    case 7:
        System.out.println(" running?  Check running process");
        System.out.println();
        System.out.println(" running? machine_name");
        break;

        // send
    case 8:
        System.out.println(" send  Send file to other machine.");
        System.out.println();
        System.out.println(" send program");
        System.out.println(" send program machine_name");

```

```

        break;

        // set
    case 9:
        System.out.println(" set  Set the configuration of virtual network.");
        System.out.println();
        System.out.println(" set changeld userid");
        System.out.println(" set changePasswd password");
        System.out.println(" set addMachine machine_name");
        System.out.println(" set deleteMachine machine_name");
        break;

        // status
    case 10:
        System.out.println(" status  Check the status of other machines.");
        System.out.println();
        System.out.println(" status machine_name");
        break;

    default:
        break;
    }
}

```

```

static int makeCommandToInt(String inputCommand) {
    int commandNo = 0;

    for(commandNo = 0; commandNo < ENROLLED_CMD_NO;
commandNo++) {
        if(enrolledCommands[commandNo].equals(inputCommand))
            return commandNo;
    }
    return commandNo;
}

```

```

static int readSetupFile() {
    String readData = null;
    String MACHINE = "machines:";
    String USERID = "userid:";
    String PASSWORD = "password:";

    int index = 0;

    try {

```

```

//read from Virtual Network Setup data file
BufferedReader in =
    new BufferedReader(new FileReader(new File("vns.data")));

while ((readData = in.readLine()) != null) {
    if(readData.equals(PASSWORD))
        passWord = in.readLine();
    else if(readData.equals(USERID))
        userID = in.readLine();
    else if(readData.equals(MACHINE)) {
        machineNames[index] = in.readLine();
        index++;
    }
    else {
        machineNames[index] = readData;
        index++;
    }
}
in.close();
} catch (FileNotFoundException e) {
    System.err.println("vns.data file does not exist !!!");
} catch (IOException e) {
    System.err.println(e);
}
}
return index;
}

```

```

public static void sendFiles() {
    String commands = null;
    String fileName[] = new String[machineNumber];

    HostFrame hf = new HostFrame();

    if (hf.state[0] == 1 && hf.state[1] == 0 && hf.state[2] == 0)
        fileName[0] = "Default4E.data";

    if (hf.state[0] == 0 && hf.state[1] == 1 && hf.state[2] == 0)
        fileName[0] = "Default4C.data";

    if (hf.state[0] == 0 && hf.state[1] == 0 && hf.state[2] == 1)
        fileName[0] = "Default4Z.data";

    if (hf.state[0] == 1 && hf.state[1] == 1 && hf.state[2] == 0) {
        try {
            FileReader fr = new FileReader("vns.data");

```

```

BufferedReader br = new BufferedReader(fr);
String s[] = new String[2];

br.readLine();
s[0] = br.readLine();
s[1] = br.readLine();

if (s[0].equals("eslabsvr.wslab.okstate.edu") &&
    s[1].equals("chester.cs.okstate.edu")) {
    fileName[0] = "Default4E.data";
    fileName[1] = "Default4C.data";
}

if (s[0].equals("chester.cs.okstate.edu") &&
    s[1].equals("eslabsvr.wslab.okstate.edu")) {
    fileName[0] = "Default4C.data";
    fileName[1] = "Default4E.data";
}
} catch (IOException e) {
    System.err.println(e);
}
}

if (hf.state[0] == 1 && hf.state[1] == 0 && hf.state[2] == 1) {
    try {
        FileReader fr = new FileReader("vns.data");
        BufferedReader br = new BufferedReader(fr);
        String s[] = new String[2];

        br.readLine();
        s[0] = br.readLine();
        s[1] = br.readLine();

        if (s[0].equals("eslabsvr.wslab.okstate.edu") &&
            s[1].equals("z.cs.okstate.edu")) {
            fileName[0] = "Default4E.data";
            fileName[1] = "Default4Z.data";
        }

        if (s[0].equals("z.cs.okstate.edu") &&
            s[1].equals("eslabsvr.wslab.okstate.edu")) {
            fileName[0] = "Default4Z.data";
            fileName[1] = "Default4E.data";
        }
    } catch (IOException e) {
        System.err.println(e);
    }
}
}

```

```

    }
}

if (hf.state[0] == 0 && hf.state[1] == 1 && hf.state[2] == 1) {
    try {
        FileReader fr = new FileReader("vns.data");
        BufferedReader br = new BufferedReader(fr);
        String s[] = new String[2];

        br.readLine();
        s[0] = br.readLine();
        s[1] = br.readLine();

        if (s[0].equals("chester.cs.okstate.edu") &&
            s[1].equals("z.cs.okstate.edu")) {
            fileName[0] = "Default4C.data";
            fileName[1] = "Default4Z.data";
        }

        if (s[0].equals("z.cs.okstate.edu") &&
            s[1].equals("chester.cs.okstate.edu")) {
            fileName[0] = "Default4Z.data";
            fileName[1] = "Default4C.data";
        }
    } catch (IOException e) {
        System.err.println(e);
    }
}

if (hf.state[0] == 1 && hf.state[1] == 1 && hf.state[2] == 1) {
    try {
        FileReader fr = new FileReader("vns.data");
        BufferedReader br = new BufferedReader(fr);
        String s[] = new String[3];

        br.readLine();
        s[0] = br.readLine();
        s[1] = br.readLine();
        s[2] = br.readLine();

        if (s[0].equals("eslabsvr.wslab.okstate.edu")) {
            if (s[1].equals("chester.cs.okstate.edu") &&
                s[2].equals("z.cs.okstate.edu")) {
                fileName[0] = "Default4E.data";
                fileName[1] = "Default4C.data";
                fileName[2] = "Default4Z.data";
            }
        }
    }
}

```

```

    }

    if (s[1].equals("z.cs.okstate.edu") &&
        s[2].equals("chester.cs.okstate.edu")) {
        fileName[0] = "Default4E.data";
        fileName[1] = "Default4Z.data";
        fileName[2] = "Default4C.data";
    }
}

if (s[0].equals("chester.cs.okstate.edu")) {
    if (s[1].equals("eslabsvr.wslab.okstate.edu") &&
        s[2].equals("z.cs.okstate.edu")) {
        fileName[0] = "Default4C.data";
        fileName[1] = "Default4E.data";
        fileName[2] = "Default4Z.data";
    }

    if (s[1].equals("z.cs.okstate.edu") &&
        s[2].equals("eslabsvr.wslab.okstate.edu")) {
        fileName[0] = "Default4C.data";
        fileName[1] = "Default4Z.data";
        fileName[2] = "Default4E.data";
    }
}

if (s[0].equals("z.cs.okstate.edu")) {
    if (s[1].equals("eslabsvr.wslab.okstate.edu") &&
        s[2].equals("chester.cs.okstate.edu")) {
        fileName[0] = "Default4Z.data";
        fileName[1] = "Default4E.data";
        fileName[2] = "Default4C.data";
    }

    if (s[1].equals("chester.cs.okstate.edu") &&
        s[2].equals("eslabsvr.wslab.okstate.edu")) {
        fileName[0] = "Default4Z.data";
        fileName[1] = "Default4C.data";
        fileName[2] = "Default4E.data";
    }
}
} catch (IOException e) {
    System.err.println(e);
}
}

```

```

try {
    FileReader fr = new FileReader("vns.data");
    BufferedReader br = new BufferedReader(fr);
    String s;

    br.readLine();

    for (int i = 0; i < machineNumber; i++) {
        commands = "mftp" + " " + machineNames[i] + " " + fileName[i];
        Process pc = Runtime.getRuntime().exec(commands);
        pc.waitFor();

        s = br.readLine();
    }
    fr.close();
} catch (IOException e) {
    System.err.println(e);
} catch (InterruptedException e) {
    System.out.println("");
}
}

```

```

private static void sendData(String machineNames, String fileName,
                             String desFilename) {
    String commands = null;

    try {
        commands = "ftpdata" + " " + machineNames + " " + fileName
            + " " + desFilename;
        Process pc = Runtime.getRuntime().exec(commands);
        pc.waitFor();
    } catch (IOException e) {
        System.err.println(e);
    } catch (InterruptedException e) {
        System.out.println("");
    }
}

```

```

public static void executeFiles() {
    try {
        FileReader fr = new FileReader("vns.data");
        BufferedReader br = new BufferedReader(fr);
        String s;

```



```

br.readLine();

for (int i = 0; i < machineNumber; i++) {
    String host = machineNames[i];
    String msg = "/home/hseong>>";
    String msgZ = "/z/hseong>>";
    String msgMore = "/home/hseong/java/DDAS>>";
    String msgMoreZ = "/z/hseong/java/DDAS>>";
    String command = "nohup java Manager\r";
    String runMsg = ">>> Running DDAS System";

    s = br.readLine();

    int hostN = 0;

    if (host.equals("z.cs.okstate.edu"))
        hostN = 1;

    switch(hostN) {
    case 0:
        ExecuteFile ef0 = new ExecuteFile(host, msg, command,
msgMore, runMsg);
        break;
    case 1:
        ExecuteFile ef1 = new ExecuteFile(host, msgZ, command,
msgMoreZ, runMsg);
        break;
    default:
        break;
    }
}
fr.close();
} catch (IOException e) {
    System.err.println(e);
}
}

public static void deleteAllFiles() {
    try {
        FileReader fr = new FileReader("vns.data");
        BufferedReader br = new BufferedReader(fr);
        String s;

        br.readLine();

```

```

for (int i = 0; i < machineNumber; i++) {
    String host = machineNames[i];
    String msg = "/home/hseong>>";
    String msgZ = "/z/hseong>>";
    String msgMore = "/home/hseong/java/DDAS>>";
    String msgMoreZ = "/z/hseong/java/DDAS>>";
    String command = "rm -r *\r";
    String runMsg = ">>> Running DDAS System";

    s = br.readLine();

    int hostN = 0;

    if (host.equals("z.cs.okstate.edu"))
        hostN = 1;

    switch(hostN) {
    case 1:
        DeleteFile df0 = new DeleteFile(host, msgZ, command,
msgMoreZ);
        break;
    default:
        DeleteFile df1 = new DeleteFile(host, msg, command, msgMore);
        break;
    }
    }
    fr.close();
} catch (IOException e) {
    System.err.println(e);
}
}

```

```

static void displayErrorMessage (int errNumber) {
    switch (errNumber) {
    case 0:
        System.out.println("Your input arguments are wrong. Try again!");
        break;

    case 1:
        System.out.println(" ");
        break;

    case 2:
        System.out.println ("Set arguments are wrong. Try again");
        break;
    }
}

```

```
    default:  
        break;  
    }  
}
```

```

/*****
/*
/*          MakeDefaultTwo.java          */
/*
/* Functions:                             */
/* 1. This program is for using two remote hosts.      */
/* 2. After reading "vns.data" file, this program makes default files */
/*    for selected remotes hosts.                    */
/* 3. This program makes two default files.           */
/*
/*****

```

```
import java.io.*;
```

```
public class MakeDefaultTwo {
```

```
    public static void makedefaulttwo() {
        String machineTemp[] = new String[2];
```

```
        try {
            FileReader fr = new FileReader("vns.data");
            BufferedReader br = new BufferedReader(fr);
```

```
            br.readLine();
            machineTemp[0] = br.readLine();
            machineTemp[1] = br.readLine();
        } catch (IOException e) {
            System.err.println(e);
        }
    }
```

```
        try {
            if (machineTemp[0].equals("eslabsvr.wslab.okstate.edu")) {
                FileWriter ee = new FileWriter("Default4E.data");
                ee.write("machines:" + "\n");
                ee.write("a.cs.okstate.edu" + "\n");
                ee.write(machineTemp[0] + "\n");
                ee.write(machineTemp[1] + "\n");
                ee.write("userid:" + "\n");
                ee.write("*****" + "\n");
                ee.write("password:" + "\n");
                ee.write("*****");
                ee.flush();
                ee.close();
            }
        }
    }
}
```

```

if (machineTemp[1].equals("chester.cs.okstate.edu")) {
    FileWriter c = new FileWriter("Default4C.data");
    c.write("machines:" + "\n");
    c.write(machineTemp[0] + "\n");
    c.write(machineTemp[1] + "\n");
    c.write("final" + "\n");
    c.write("userid:" + "\n");
    c.write("*****" + "\n");
    c.write("password:" + "\n");
    c.write("*****");
    c.flush();
    c.close();
}

else if (machineTemp[1].equals("z.cs.okstate.edu")) {
    FileWriter z = new FileWriter("Default4Z.data");
    z.write("machines:" + "\n");
    z.write(machineTemp[0] + "\n");
    z.write(machineTemp[1] + "\n");
    z.write("final" + "\n");
    z.write("userid:" + "\n");
    z.write("*****" + "\n");
    z.write("password:" + "\n");
    z.write("*****");
    z.flush();
    z.close();
}
}

if (machineTemp[0].equals("chester.cs.okstate.edu")) {
    FileWriter c = new FileWriter("Default4C.data");
    c.write("machines:" + "\n");
    c.write("a.cs.okstate.edu" + "\n");
    c.write(machineTemp[0] + "\n");
    c.write(machineTemp[1] + "\n");
    c.write("userid:" + "\n");
    c.write("*****" + "\n");
    c.write("password:" + "\n");
    c.write("*****");
    c.flush();
    c.close();

    if (machineTemp[1].equals("eslabsvr.wslab.okstate.edu")) {
        FileWriter ee = new FileWriter("Default4E.data");
        ee.write("machines:" + "\n");
        ee.write(machineTemp[0] + "\n");
    }
}

```

```

        ee.write(machineTemp[1] + "\n");
        ee.write("final" + "\n");
        ee.write("userid:" + "\n");
        ee.write("*****" + "\n");
        ee.write("password:" + "\n");
        ee.write("*****");
        ee.flush();
        ee.close();
    }

    else if (machineTemp[1].equals("z.cs.okstate.edu")) {
        FileWriter z = new FileWriter("Default4Z.data");
        z.write("machines:" + "\n");
        z.write(machineTemp[0] + "\n");
        z.write(machineTemp[1] + "\n");
        z.write("final" + "\n");
        z.write("userid:" + "\n");
        z.write("*****" + "\n");
        z.write("password:" + "\n");
        z.write("*****");
        z.flush();
        z.close();
    }
}

if (machineTemp[0].equals("z.cs.okstate.edu")) {
    FileWriter z = new FileWriter("Default4Z.data");
    z.write("machines:" + "\n");
    z.write("a.cs.okstate.edu" + "\n");
    z.write(machineTemp[0] + "\n");
    z.write(machineTemp[1] + "\n");
    z.write("userid:" + "\n");
    z.write("*****" + "\n");
    z.write("password:" + "\n");
    z.write("*****");
    z.flush();
    z.close();

    if (machineTemp[1].equals("eslabsvr.wslab.okstate.edu")) {
        FileWriter ee = new FileWriter("Default4E.data");
        ee.write("machines:" + "\n");
        ee.write(machineTemp[0] + "\n");
        ee.write(machineTemp[1] + "\n");
        ee.write("final" + "\n");
        ee.write("userid:" + "\n");
        ee.write("*****" + "\n");
    }
}

```

```

        ee.write("password:" + "\n");
        ee.write("*****");
        ee.flush();
        ee.close();
    }

    else if (machineTemp[1].equals("chester.cs.okstate.edu")) {
        FileWriter c = new FileWriter("Default4C.data");
        c.write("machines:" + "\n");
        c.write(machineTemp[0] + "\n");
        c.write(machineTemp[1] + "\n");
        c.write("final" + "\n");
        c.write("userid:" + "\n");
        c.write("*****" + "\n");
        c.write("password:" + "\n");
        c.write("*****");
        c.flush();
        c.close();
    }
}
} catch (IOException e) {
    System.err.println(e);
}
}
}

```

```

/*****/
/*                                          */
/*          App.java                      */
/*                                          */
/* Functions:                             */
/* 1. This program is for an application algorithm. */
/* 2. A quicksort algorithm is used as an application algorithm. */
/* 3. Steps:                               */
/*   3.1 Execute the quicksort algorithm   */
/*   3.2 Divide input into two data files  */
/*   3.3 Send the data file to the next node */
/*   3.4 A distributed computation job is complete */
/*      Initial data: input.dat           */
/*      Sorted data: result.dat           */
/*                                          */
/*****/

```

```

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

```

```

class App extends DDAS {
    static final int MAX_MACHINE = 100;
    static final int ENROLLED_CMD_NO = 11;
    static final int MAX_ARGS = 5;

    static String[] machineNames = new String[MAX_MACHINE];

    static String userID = null;
    static String passWord = null;

    static String inputCommand = null;
    static int machineNumber = 0; // available machine counts
    static StringTokenizer st;

    private static boolean debug = true;

    static Manager ddas = new Manager();

    static String[] arguments = new String[3];

    public static void main(String[] strg) throws Exception {
        ddas.initialSystem();
    }
}

```



```

String sendMessage = null;

arguments[0] = "establish";
ddas.commandService(3, arguments, 1);

arguments[0] = "execute";
ddas.commandService(4, arguments, 1);
}

static void messageCommunication() {
    Server = "a.cs.okstate.edu";

    connectSocket();
    while (nextFlag == 1)
        pause(PAUSE);

    openSocket();
    while(nextFlag == 0)
        pause(PAUSE);
}

static void messageCommunication(String machine) {
    Server = machine;

    connectSocket();
    while(nextFlag == 1)
        pause(PAUSE);

    openSocket();
    while(nextFlag == 0)
        pause(PAUSE);
}

static void makeSendMessage(String sendMessage) {
    msgSend = sendMessage;
}
}

```

## **APPENDIX B**

### **SYSTEM LOAD INFORMATION**

load averages: 1.50, 1.59, 1.56 14:17:37

542 processes: 529 sleeping, 4 zombie, 7 stopped, 2 on cpu

CPU states: 44.0% idle, 34.0% user, 21.6% kernel, 0.4% iowait, 0.0% swap

Memory: 512M real, 20M free, 396M swap in use, 3092M swap free

PID	USERNAME	THR	PRI	NICE	SIZE	RES	STATE	TIME	CPU	COMMAND
26646	wdongch	1	10	0	23M	4400K	cpu6	99:02	46.58%	.netscape.bin
5054	bhat	1	58	0	24M	12M	sleep	5:41	2.26%	.netscape.bin
16219	fujino	1	21	0	6376K	5448K	sleep	0:03	1.19%	gs
5820	hseong	1	58	0	7096K	5184K	sleep	0:13	1.15%	dtterm
16200	zxiaoze	1	58	0	6832K	5168K	sleep	0:01	0.46%	emacs-20.3
16472	hseong	1	18	0	1904K	1160K	cpu7	0:00	0.31%	top
16195	fujino	1	58	0	4664K	4016K	sleep	0:00	0.15%	xdvi.bin
16380	reubinl	1	58	0	6760K	5016K	sleep	0:00	0.15%	emacs-20.3
16384	wanq	1	58	0	1416K	1072K	sleep	0:00	0.13%	csch
11165	cherukr	1	58	0	6624K	5616K	sleep	0:01	0.12%	dtpad
5545	wdongch	1	58	0	6864K	5888K	sleep	0:22	0.10%	dtpad
16220	samad	1	58	0	1832K	1296K	sleep	0:00	0.06%	vi
10233	cherukr	8	58	0	9488K	8096K	sleep	0:04	0.05%	dtwm
5834	hseong	1	48	0	2576K	2024K	sleep	0:01	0.05%	tcsh
14506	cs3423	1	59	0	6992K	5304K	sleep	0:00	0.03%	dtterm
16470	cs3423	1	58	0	1832K	1288K	sleep	0:00	0.03%	vi
5536	wdongch	8	58	0	8472K	6544K	sleep	0:03	0.03%	dtwm
3981	root	1	49	0	1744K	568K	sleep	0:10	0.02%	rpc.rstatd
11904	cs2113a	1	59	0	7000K	5424K	sleep	0:02	0.02%	dtterm
29388	sherryr	1	48	0	1768K	1192K	sleep	0:00	0.02%	ksh
226	root	1	58	0	2056K	640K	sleep	1:09	0.01%	inetd
16079	fujino	1	58	0	7224K	6176K	sleep	0:01	0.01%	emacs-20.3
5052	bhat	1	59	0	6696K	4824K	sleep	0:01	0.01%	sdtperfmeter
12127	xiaohon	1	48	0	1760K	1272K	sleep	0:00	0.01%	ksh
16276	trotman	1	48	0	1760K	1272K	sleep	0:00	0.01%	ksh
525	root	1	58	0	1000K	464K	sleep	1:01	0.01%	utmpd
5584	wdongch	1	58	0	6760K	4816K	sleep	0:01	0.01%	dtterm
13103	elockha	1	17	14	3256K	1976K	sleep	0:00	0.01%	xlock
16381	root	1	38	0	1752K	1352K	sleep	0:00	0.01%	in.telnetd
16361	cassk	1	56	0	1832K	1288K	sleep	0:00	0.01%	vi

10047 stolhan	1	58	0	1832K	1288K	sleep	0:00	0.01%	vi
15201 lzhilan	1	59	0	6696K	5184K	sleep	0:00	0.01%	sdtperfmeter
1 root	1	58	0	752K	184K	sleep	1:37	0.01%	init
772 root	1	58	0	2072K	952K	sleep	0:47	0.01%	nmbd
11898 cs2113a	8	58	0	8488K	7296K	sleep	0:03	0.01%	dtwm
11922 cs2113a	1	58	0	1760K	1272K	sleep	0:00	0.01%	ksh
16454 trotman	1	58	0	1368K	1128K	sleep	0:00	0.01%	pico
14488 cs3423	1	59	0	6696K	5184K	sleep	0:00	0.01%	sdtperfmeter
16308 cassk	1	45	0	1760K	1272K	sleep	0:00	0.00%	ksh
11908 cs2113a	1	58	0	6696K	5120K	sleep	0:00	0.00%	sdtperfmeter
16141 gunawan	1	58	0	4632K	3352K	sleep	0:00	0.00%	pine
15545 wangk	1	58	0	1760K	1272K	sleep	0:00	0.00%	ksh
777 root	1	58	0	9240K	6360K	sleep	5:40	0.00%	xfx
775 root	1	0	0	1848K	816K	sleep	3:02	0.00%	sshd1
281 root	10	53	0	4464K	3304K	sleep	2:25	0.00%	nscd
253 root	17	58	0	3424K	1584K	sleep	1:25	0.00%	syslogd
514 zxing	1	59	0	28M	2008K	sleep	1:13	0.00%	.netscape.bin
243 root	5	59	0	4696K	992K	sleep	0:56	0.00%	automountd
598 root	1	58	0	4768K	2344K	sleep	0:40	0.00%	dtlogin
23393 root	4	59	0	1904K	592K	sleep	0:34	0.00%	in.rarpd

PID: The process ID

USERNAME: The name of the process's owner

PRI: The current priority of the process

NICE: The nice amount (in the range -20 to 20)

SIZE: The total size of the process (text, data and stack)

RES: The current amount of resident memory

STATE: The current state (one of sleep, wait, run, idl, zomb or stop)

TIME: The number of system and user CPU seconds that the process has used

CPU: The raw percentage

COMMAND: The name of the command that the process is currently running

**APPENDIX C**

**ABBREVIATIONS AND ACRONYMS  
IN ALPHABETICAL ORDER**

## ABBREVIATIONS AND ACRONYMS

ARA	Agents for Remote Access
CCITT	Consultative Committee on International Telephone and Telegraph
CORBA	Common Object Request Broker Adapter (OMG)
DCE	Distributed Computing Environment (The Open Group)
DDAS	Dynamic Distributed Agents Server
EDDAS	Enhanced Dynamic Distributed Agents Server
FTP	File Transfer Protocol
GUI	Graphical User Interface
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronic Engineers
IIOB	Internet Inter-ORB Protocol
IP	Internet Protocol
JVM	Java Virtual Machine
OMG	Object Management Group
ORB	Object Request Broker
OSF	Open Software Foundation, now The Open Group
RPC	Remote Procedure Call
TCL	Tool Command Language
TCP	Transmission Control Protocol

2

## VITA

Seong Seol Hong  
Candidate for the Degree of  
Master of Science

Thesis: A GRAPHICAL USER INTERFACE TO MONITOR AND MANAGE THE  
DDAS SYSTEM PERFORMANCE

Major Field: Computer Science

### Biographical:

Personal Data: Born in Jinhae, Korea, On July 16, 1969, the son of Ke-Pyo and Eul-Soon Jung Hong.

Education: Graduated from Chunchon High School, Chunchon, Korea in February 1988; received Bachelor of Science degree in Computer Science from Hallym University, Chunchon, Korea in February 1994; received Master of Engineering degree in Computer Science from Soongsil University, Seoul, Korea in February 1996. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December 1999.

Experience: Employed by Soongsil University, Department of Computer Science as an undergraduate teaching assistant; Soongsil University, Department of Computer Science, May 1995 to December 1995.