

**DDAS: DESIGN AND IMPLEMENTATION OF  
A KERNEL APPLICATION FOR ADAPTIVE  
DISTRIBUTED COMPUTATION**

By

**KWAN-SUNG KIM**

Bachelor of Engineering

Kyunghee University

Seoul, Korea

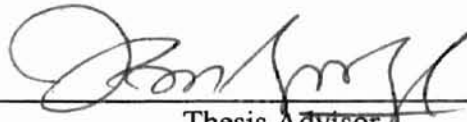
1991

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**  
July, 1999

**DDAS: DESIGN AND IMPLEMENTATION OF  
A KERNEL APPLICATION FOR ADAPTIVE  
DISTRIBUTED COMPUTATION**

I would like to express my appreciation to my thesis advisor, Dr. C. M. George, for his inspiration, guidance, and patience. I would also like to thank my family and friends for their support and encouragement. I would like to thank the following individuals for their assistance and support during the course of this project: [Faded names and text]

Thesis Approved:

  
\_\_\_\_\_  
Thesis Advisor

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_  
Dean of the Graduate College

## **ACKNOWLEDGMENTS**

I would like to express my sincere appreciation to my thesis advisor, Dr. K. M. George for his inspiration, guidance, friendship and patience. I would also like to express my respectful thanks to Dr. John P. Chandler and Dr. H. K. Dai for serving on my graduate committee and for providing valuable suggestions and ideas.

I sincerely thank my family and friends with all my heart.

**TABLE OF CONTENTS**

Chapter ..... 51  
Chapter Page

I. INTRODUCTION..... 1

II. LITERATURE REVIEW ..... 4

2.1. Distributed Computation System ..... 4

2.2. Distributed Computation Tools..... 7

2.2.1. RPC ..... 7

2.2.2. DCE..... 9

2.2.3. CORBA ..... 10

2.2.4. Java RMI ..... 12

2.3. Distributed Computation Applications ..... 14

2.3.1. Client-Server Model..... 15

2.3.2. Mobile Agents ..... 17

III. DDAS SYSTEM ..... 22

3.1 The DDAS System Configuration..... 22

3.2 The Basic Operations of the DDAS System..... 24

IV. IMPLEMENTATION OF THE DDAS SYSTEM..... 27

4.1. Implementation of DDAS and Manager..... 27

4.2. Construction of a Dynamic Network..... 31

4.3. Inter-DDAS Communication ..... 33

4.4. Node Disconnection and Clean up ..... 34

V. EXECUTION OF DISTRIBUTED COMPUTATION JOBS..... 35

VI. TEST AND DISCUSSION ..... 39

6.1. Test Environment ..... 39

6.2. Example..... 40

6.3. Discussion ..... 46



VII. CONCLUSIONS AND FUTURE WORK .....	49
7.1. Conclusions .....	49
7.2. Future Work .....	49

LIST OF TABLES

REFERENCES .....	51
APPENDIX	
APPENDIX A. ABBREVIATIONS AND ACRONYMS .....	55
APPENDIX B. PARTIAL DDAS SYSTEM SOURCE CODE .....	58

## LIST OF TABLES

Table	Page
1. Communication components and method of communication.....	23
2. Responsibilities of each communication component.....	23
3. System characteristics .....	39
4. Input command set of the DDAS system .....	41
5. Files after construction of the dynamic network .....	42
6. Files after finishing data transfer to the next nodes.....	43
7. Files after execution of an application program.....	45
8. Comparisons of DDAS to related systems .....	47
9. Problems of client-server and mobile agents and solution in DDAS system.....	48

**LIST OF FIGURES**

Figure	Page
1. LAN and WAN connections .....	5
2. TCP/IP Layer and OSI model Layer.....	6
3. Remote procedure call between client and server .....	8
4. OSF DCE architecture.....	10
5. One of OMG CORBA reference models .....	12
6. Java RMI Layers.....	13
7. Client-server model.....	16
8. Operations of general mobile agents system.....	18
9. Basic configuration of the DDAS system .....	24
10. Basic operations of the DDAS system .....	25
11. Functions of the DDAS methods .....	28
12. Functions of the DDAS classes.....	29
13. Methods of the Manager class.....	30
14. Flow for construction of a dynamic network .....	32
15. Computation steps of an application program.....	36
16. A sample diagram and pseudo-codes for a distributed job .....	37
17. Flow chart for the distributed quick sort application using the DDAS system .....	42



## I. INTRODUCTION

Distributed systems and distributed computing have been very popular terms for the last two decades in computer science. A distributed system is a collection of independent computers interconnected by a communication network, and its applications are called distributed computing as opposed to centralized or single-process computing. The major goal of distributed computing is to reduce the processing time of tasks by distributing job execution throughout the system to multiple processors [31].

There are many advantages of distributed computing over centralized computing. Distributed computation provides a less expensive method of executing computation intensive tasks previously run on supercomputers [44]. Distributed computation also yields higher performance. Through high-speed network interconnection and dividing computation, the processing speed is higher for large problems than using a single processor. Moreover, it allows sharing of expensive resources such as data files, color printers and specialized hardware devices. A user can run programs on many different machines and he/she can share varied peripherals that are parts of other computers.

With these many advantages, distributed computing applications have been developed over the years. RPC (Remote Procedure Call) [24], DCE (Distributed Computing Environment) [28], CORBA (Common Object Request Broker Adapter) [18] and Java RMI (Remote Method Calls) [29] provide models for transferring structured data through networks as programming tools [18]. Based on these tools, Client-Server model [7] and Mobile Agents [6, 23] have been applied to the distributed computing applications [34].

Unfortunately, there exist security problems in highly efficient distributed computation. Distributed computing applications face two-fold security problems. Transmitting data, receiving results and accessing other computers' resources are accomplished by open communication ports on computer devices. Therefore, distributed programming tools provide their own security functions and distributed computing applications have to deal with security environments of a computer system [25]. Furthermore, the systems need to be concerned with their own security and integrity.

In this research, we propose DDAS (Dynamic Distributed Agents Server) system that includes the advantages of distributed computing and supports secure and reliable communication between nodes. The proposed system is designed to support adaptive run-time distributed computation processing. The DDAS system implements TCP/IP (Transmission Control Protocol/Internet Protocol) [19] network protocol for low level network connection. Java object-oriented programming language [22] is used in the construction of the higher level network system and message passing because it provides various communication methods and object message passing methods [12, 26, 27]. DDAS system can be used to build a virtual network. A virtual network system is comprised of several domains. The virtual network system is a dynamic network since new domains may be added or removed. In order to address the two-fold security problem, user accounts must be authorized on all domains.

The DDAS itself is a distributed system. Any node in the system may add another node. It also has the capability to distribute processes, create processes, run processes, kill processes and delete processes.

The remainder of this thesis is organized as follows:

- Chapter 2: Describes the basic environment of the distributed computation system and provides an overview of distributed computation tools and applications.
- Chapter 3: Discusses thesis objectives and describes the basic configuration and operations of proposed system, DDAS: Dynamic Distributed Agents Server.
- Chapter 4: Discusses the implementation specific design of the DDAS system architecture.
- Chapter 5: Describes the execution of distributed computation jobs using DDAS system.
- Chapter 6: Discusses the test evaluation of DDAS system with a distributed application program
- Chapter 7: Presents a summary of the thesis and presents future work.
- Appendix A: Describes abbreviations and acronyms of the thesis.
- Appendix B: Includes parts of DDAS system source code.

## II. LITERATURE REVIEW

In this chapter, we will describe the basic environment of the distributed computation system and provide an overview of some distributed computation tools and applications.

### 2.1. Distributed Computation System

Distributed computation is executed in a collection of interconnected processors that are connected by communication networks. LAN (Local Area Network) emerged in the early 1970's as a high-speed network for combining main frames, mini computers, desktop workstations and peripherals within a special area [32]. LANs are convenient for installation and management, and are very reliable [35]. LANs connected to other LANs have been developed to form WAN (Wide Area Network) [4]. Figure 1 shows the network connection of LAN and WAN. Furthermore, all different types of networks are connected into the Internet. The Internet is developed based on the TCP/IP protocol suite [8].

In 1983, the DOD (Department Of Defense) adopted the TCP/IP protocol suite as a standard. Since then the TCP/IP protocol has expanded very fast. The DOD's ARPANET (Advanced Research Projects Agency NETWORK), the world's first packet switching network, is the genesis of the Internet [43].



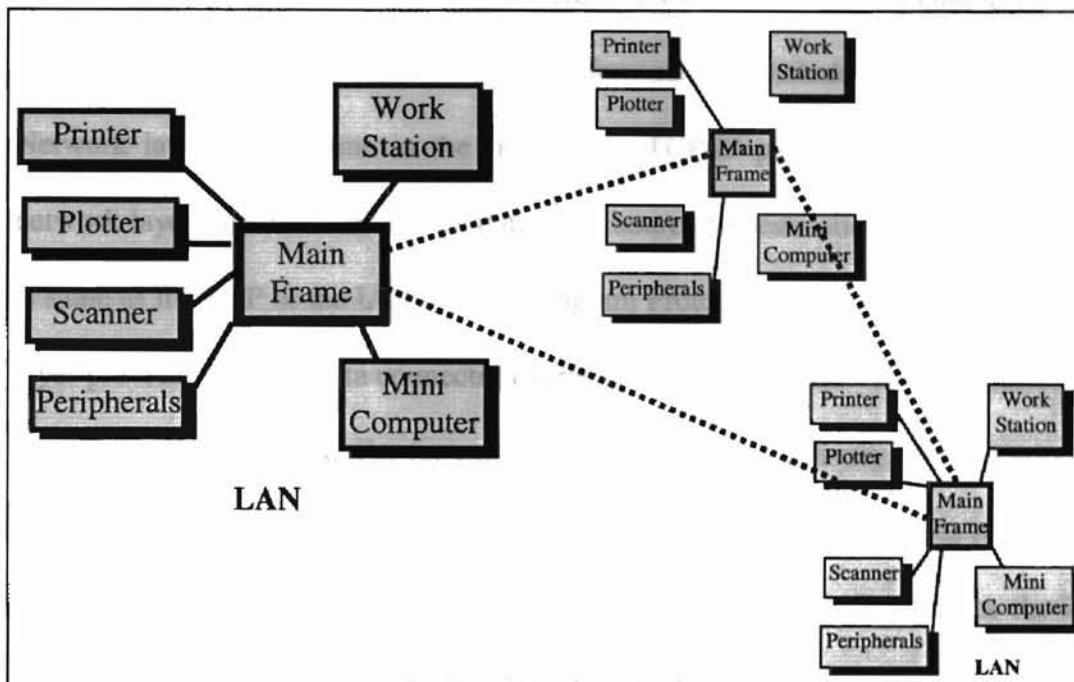


Figure 1. LAN and WAN connections.

The TCP/IP protocols were designed to be independent of host hardware or operating system. In communication between different networks, the OSI (Open Systems Interconnection) model was developed by ISO (International Organization for Standardization) as another standard data communication model. Even though the OSI model is used by very few organizations for network communication, it provides stable hardware independent design. Therefore, both of them, TCP/IP protocol and the OSI model, are used in distributed network.

Figure 2 shows the TCP/IP layers and OSI model layers. TCP/IP protocol and OSI model consist of several layers and each layer has the following behavior: The physical layer is an interface between the other layers and the hardware devices. In this layer, the transmitted data are represented as bits signal. The Data Link layer provides

reliable data transfer across the network from LANs to WANs. In Data Link layer, the transmitted data are represented by frame formats that include physical address. The Network layer is the same as the IP layer in TCP/IP protocol. This layer performs network layer functions and routes data from source to destination. The Transport layer is same as the TCP or the UDP (User Datagram Protocol) in TCP/IP protocol [19]. This layer provides reliable data connection services to applications. The transferred data are enclosed by header and trailer for ensuring communication delivery. The Session layer is the interface between the users and network. It manages two processes to establish and to control. The Presentation layer determines data syntax. It provides the interface between the application and the required service. The application layer provides required software applications such as file transfer, terminal access, electronic mail, news and so on [43].

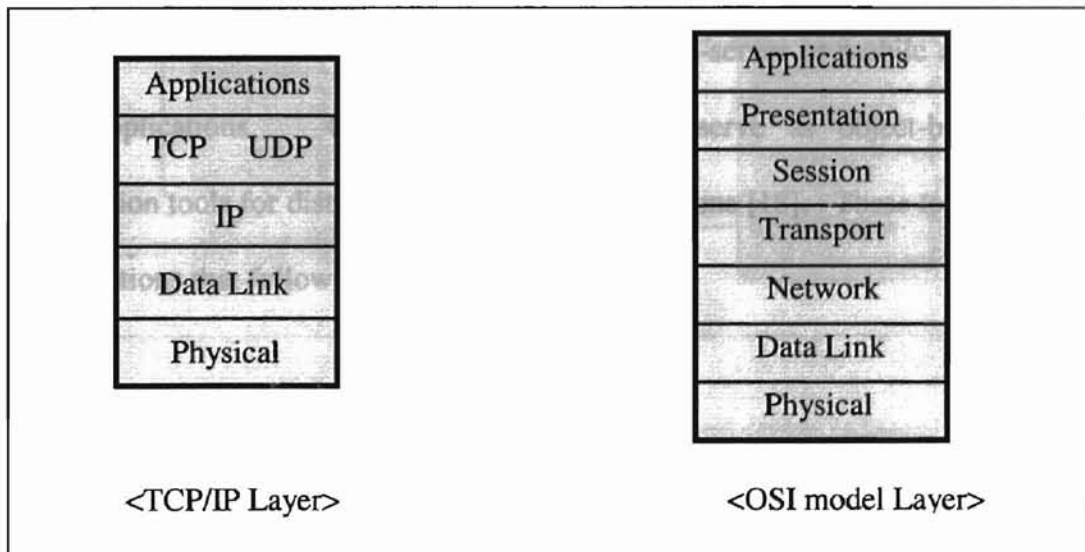


Figure 2. TCP/IP Layer and OSI model Layer (adopted from [19]).

Since TCP/IP protocol and OSI model are the basic protocols in a distributed network environment [40], the TCP/IP protocol and the OSI model are applied to many computing tools and applications in distributed computation systems.

## **2.2. Distributed Computation Tools**

RPC and DCE were developed as low-level data communication APIs (Application Programming Interfaces) and protocols. APIs play an important role in message passing during distributed computing. The message passing API provides a common interface that allows users to send messages and files from within their applications without having to exit the application. One of the problems is to select from the variety of messaging APIs that are supported by the different network components from different manufacturers. Therefore, many tools, such as RPC, DCE, CORBA and Java RMI, have been developed to provide a hardware free environment.

RPC and DCE are used as the basis of client-server or mobile agents models in many applications. CORBA and Java RMI serve as object-based distributed computation tools for distributed computing applications [18]. These tools are described in the sections that follow.

### **2.2.1. RPC**

RPC (Remote Procedure Call) was developed by Sun Microsystems in the mid-1980s. An implementation of RPC was developed in the University of California at Berkeley. Later, the Sun Microsystems implemented the standard RPC in 1988 [42].

RPC is a special type of IPC (Inter-Process Communication) that is provided for concurrent processes or threads to communicate with each other. RPC tool of Sun Microsystems was developed for operating on the Unix system and this tool is designed so that the network programmers can use any language for developing their application programs [42]. Nowadays the RPC is a widely used method for developing client-server applications.

As an extension of the procedure call that executes applications on different hosts, RPC is composed of synchronous requests and responses. An RPC begins with a request to a remote host and completes after receiving the results from the remote procedure. Figure 3 illustrates the remote calling RPC process.

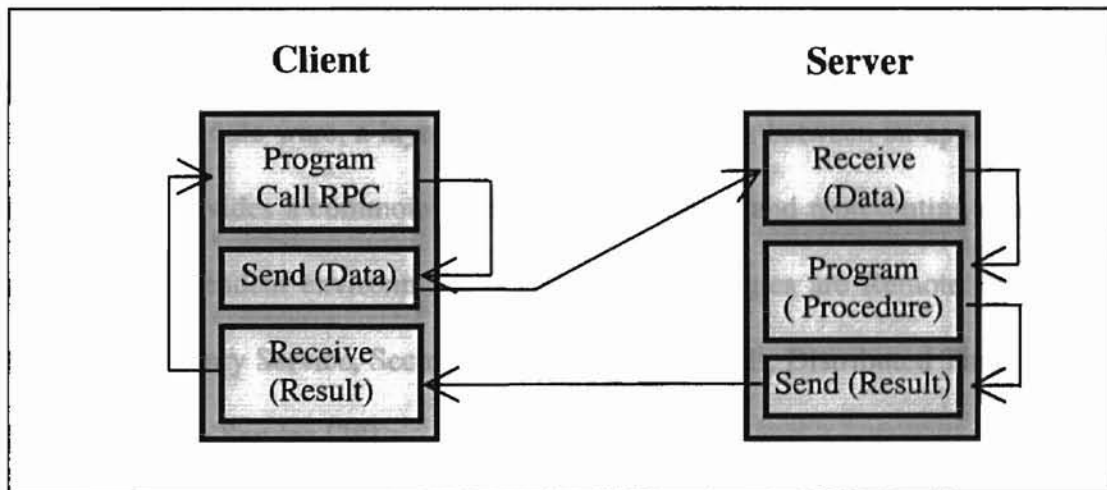


Figure 3. Remote procedure call between client and server (adopted from [24]).

If a running program in the client side wants to call remote process in the server side, the sender in the client transfers data to the server. The receiver in the server accepts data and executes a server program. The result of execution transfers to the

receiver of the client side through the server side's sender. When this return value is transferred to the program of the client, its task is done.

Since RPCs are the major communication tools of most client-server models, most middle-ware such as DCE (Distributed Computing Environment), message queuing and network SQL (Structured Query Language) are built on RPCs.

### 2.2.2. DCE

The OSF (Open Software Foundation) which was founded in 1988 [47] describes DCE (Distributed Computing Environment). It is called middle-ware [5] or enabling technology. With members consisting of system vendors, software vendors, end-users, government agencies, research centers and university communities, the OSF evaluates distributed software technologies.

As a middle-ware, a layer of software that resides between an application and the network, it provides a common interface for describing and representing data types in a machine independent environment. The key technologies are Remote Procedure Call Service, Directory Service, Security Service, DCE threads, Distributed Time Service and Distributed File Service [38].

Figure 4 describes the DCE architecture. The figure shows the functions of each element of a configuration. RPC distributes application execution. Distributed Directory Service provides a single naming model throughout the distributed environment. DCE threads control the flow of information within applications. Distributed time service synchronizes all clocks on a network. Distributed File System gives users access to remote files, regardless of their geographical location. And

Security Service provides a secure means of communication that ensures both data integrity and privacy by preventing unauthorized access to the distributed environment.

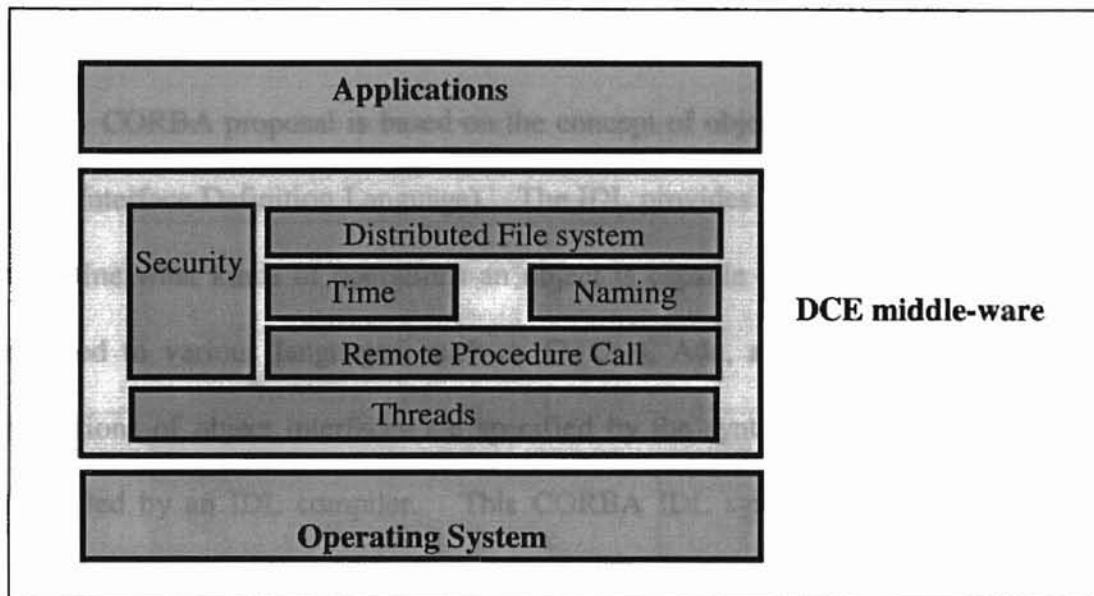


Figure 4. OSF DCE architecture (adopted from [38]).

Since DCE provides tools and services to the application developer through communicable RPC, it has been standardized for developing client-server applications in distributed computation environment.

### 2.2.3. CORBA

CORBA (Common Object Request Broker Adapter) is another middle-ware platform in distributed computing environment. As a middle-ware, it is different from DCE in that CORBA uses an object-oriented distributed model whereas DCE utilizes a procedure-oriented distributed model [1].

CORBA was developed in 1991 by the members of the OMG (Object Management Group) and their corporate members and sponsors as a distributed object standard. CORBA expands the benefits of reusability and modularity across a distributed computing system [37].

CORBA proposal is based on the concept of object interface that is described in IDL (Interface Definition Language). The IDL provides a platform independent method to define what kinds of operations an object is capable of performing. IDL has been mapped to various languages, such as C, C++, Ada, and Java. The attributes and operations of object interfaces are specified by the syntax of CORBA IDL when it is compiled by an IDL compiler. This CORBA IDL syntax is used for specifying the attributes and operations of object interfaces.

CORBA framework for distributing objects consists of an ORB (Object Request Broker), methods for specifying the interfaces and a binary protocol for communication between ORBs called IIOP (Internet Inter-ORB Protocol).

Figure 5 describes an OMG reference model that is a technical reference model. The ORB plays an important role in CORBA for distributing objects. It provides clients and servers of distributed objects the ability to send and receive requests to each other. It also provides naming service, which lets clients lookup object by name and security service and makes possible secure inter-object communication. All objects communicate to ORB through interfaces. CORBA “services objects” use fundamental interfaces. CORBA “facilities objects” reuse and extend CORBA “services interfaces”. CORBA “domains objects” reuse and extend CORBA services and CORBA “facilities interfaces”.

“Application objects” reuse all of standard categories and may add custom extensions [37].

CORBA has one problem in its execution environment. CORBA is installed on the hosts for executing. When software needs updating, the user client software does not replace new functions and new interfaces, which generate complicated problems in multi-user distributed applications. But this problem has been solved with the evolution of WWW (World Wide Web) technology [16].

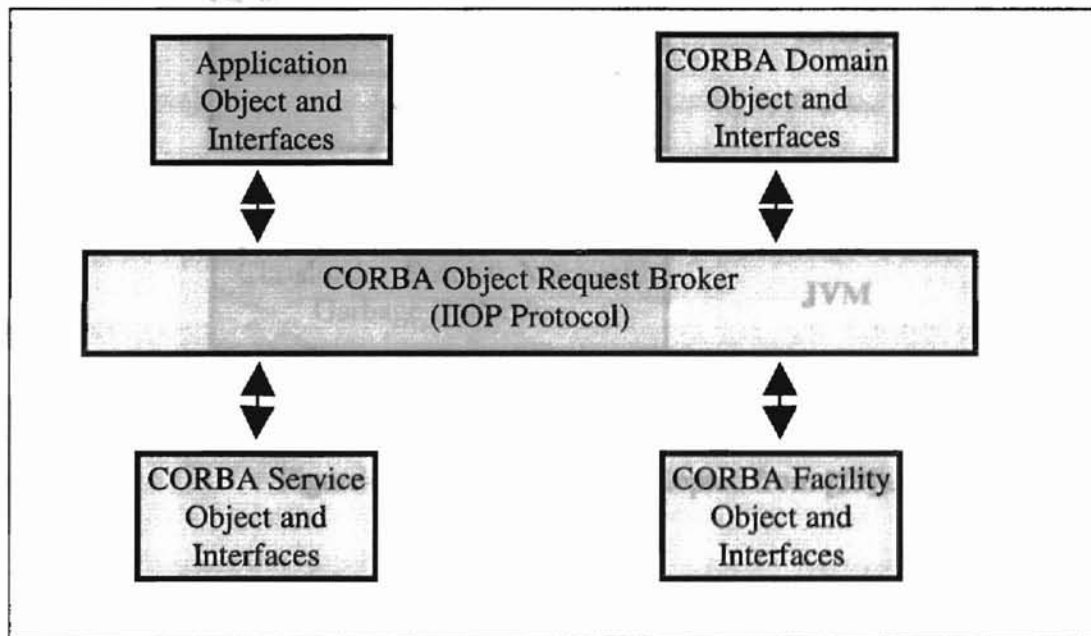


Figure 5. One of OMG CORBA reference models (adopted from [37]).

#### 2.2.4. Java RMI

Java RMI (Remote Method Invocation) has been a part of core Java API since JDK (Java Development Kit) version 1.1. Similar to CORBA's IOP, RMI has object communication tools to allow access to remote objects and includes data serialization,



remote class loading and socket manipulation [3]. As shown in Figure 6, Java RMI consists of several layers and exists between applications and JVM (Java Virtual Machine) [48].

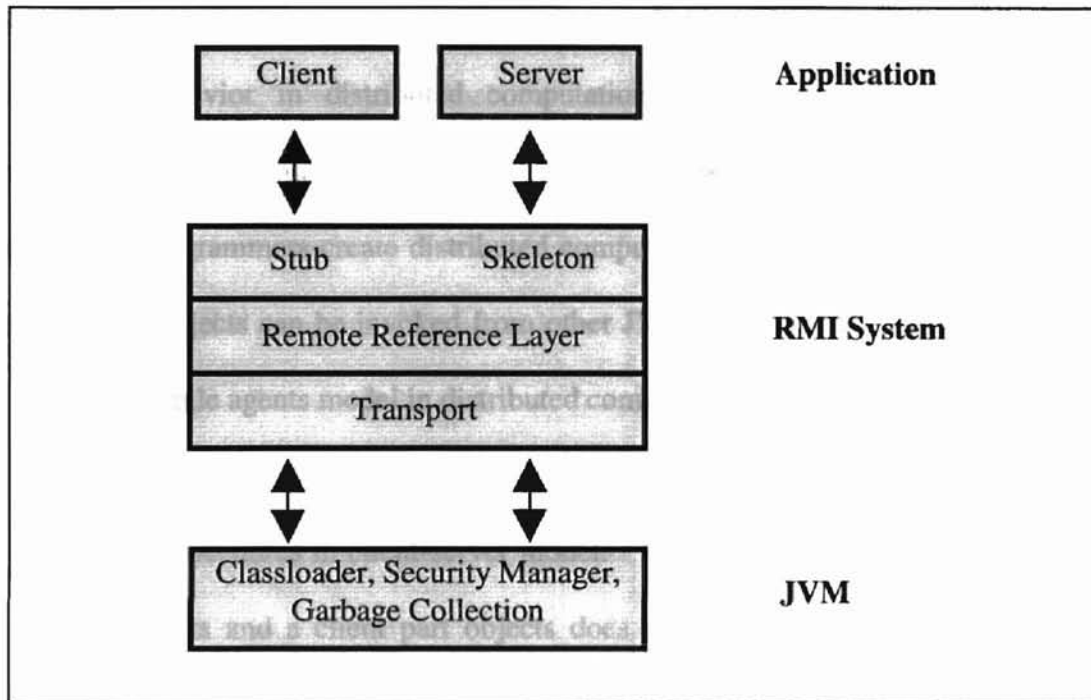


Figure 6. Java RMI Layers (adopted from [29]).

The application layer exists on the top of the RMI System and the layers of RMI system consist of three parts: Client side stubs and server side skeletons, remote reference layer, and transport layer. The stub/skeleton layer is the interface between the application and RMI system. A stub is the client side proxy for remote object and a skeleton is the server side entity for remote object. The remote reference layer is responsible for low-level transport interface, which is independent of the stubs and

skeletons. The transport layer is responsible for connection setup and management, and keeping and dispatching remote objects [29].

Java RMI makes it possible to invoke a method from one JVM to another JVM [36] and it is a distributed technology that extends the pure Java object model to the network. RMI allows moving behavior between clients and servers. Its ability to move intelligent behavior in distributed computation applications is distinguished from traditional RPC tools in that RMI serves as object-based distributed tools [29]. With Java RMI, programmers create distributed computing applications in which methods of remote Java objects can be invoked from other JVMs. Java RMI supports both client-server and mobile agents model in distributed computing applications.

But Java RMI has the same problem as CORBA executing distributed computation procedures in client-server model. If a server part objects are upgraded to new components and a client part objects does not use new functions and interfaces, serious communication problems may be generated in multi-user distributed computation applications [3]. This problem has been solved with introducing mobile agents paradigm.

### **2.3. Distributed Computation Applications**

Since distributed computing tools provide more reliable and faster distributed computation environment to the distributed computation applications such as Client-Server model and Mobile Agents, distributed computation applications have been developed rapidly over the years.

A distributed computing application, existing on top of several communication layers, consists of several elements. It includes communication layers. The low-level

communication layers provide connection to other computers for transmitting and receiving data packets and the high-level communication layers provide communication protocols. A distributed computing application makes use of processes, threads, objects and agents. The processes, created by a programming language, are the programs compiled into an executable form and running in the computer system. Some operating systems provide multiple threads that run independently from each other and they do multiple jobs concurrently. The objects that are groups of related data can be accessed by other threads in a running process. A significant functional element of a distributed application is called an agent. The agent is a recent approach in distributed computation, and it executes distributed jobs throughout the multiple processors [18].

Security problem has been a major issue in distributed computation and many distributed applications have tried to implement the solutions with algorithms and architectures. The security problems of distributed system are described in distributed applications that follow.

### **2.3.1. Client-Server Model**

Client-server distributed computation model has been very popular and its technologies have been expanded rapidly in heterogeneous communication system [24].

There are many forms of client-server models because of the various environments of server and client. All Internet services are provided by server and accessed by a client. A server is a program running on a computer that delivers a specific service and a client is a program that communicates with the server and requests specific services. Generally in the client-server model, the server is passive and waiting for client

requests. As seen in Figure 7, the client part requests to the server part and receives the answer from the server. And the server part receives the request from the client and responds to the client [39].

The client-server technology provides several benefits such as lower cost, higher productivity, longer system life cycle and better software reusability. It also can make it easier for users to access information, to use and to develop applications, and to manage a distributed computing system. Since the client-server model can be constructed in a hardware independent environment, it is easier to build a client-server model with a software mechanism.

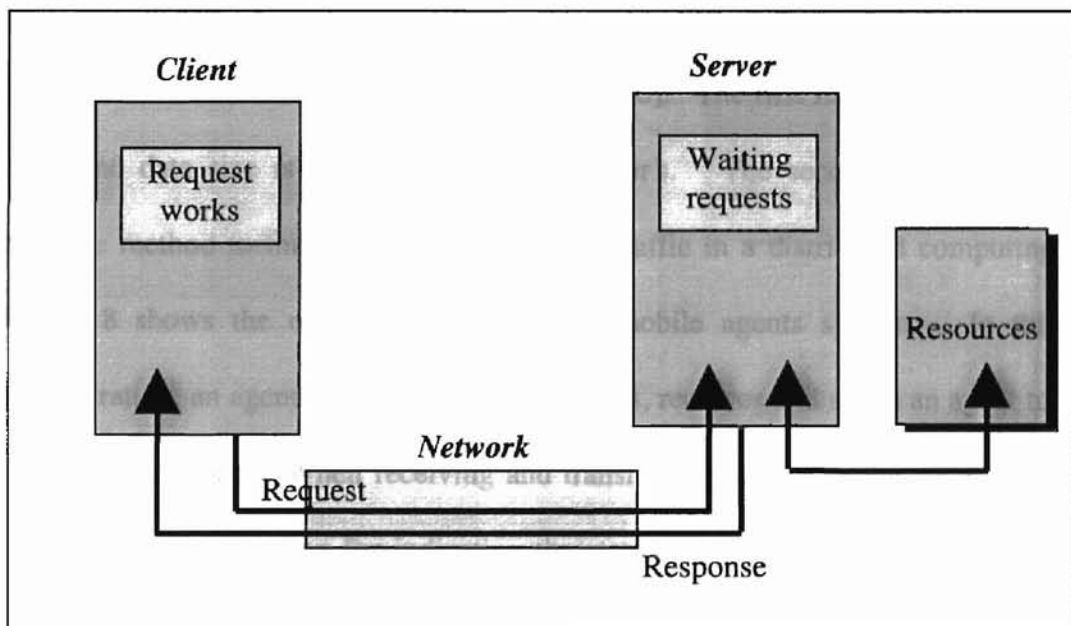


Figure 7. Client-server model (adopted from [7]).

On the other hand, one of the problems of the client-server model is that the same communication interfaces are required in both sides. When the server side software and some client side software upgrade their functions and interfaces, and some client side

software does not upgrade with new contents, it produces serious communication problems in multi-user distributed computation applications. Therefore, maintenance of software versions is very important in client-server model.

### **2.3.2. Mobile Agents**

As a relatively recent approach in distributed computation, mobile agents are able to travel through a network and provide efficient remote execution in many areas such as information retrieval, network management, workflow, mobile computing and telecommunication [46].

Camarinha-Matos and Vieira indicated two advantages of mobile agents compared to client-server model in their paper [10]. The first is that the running program code and data size is smaller than client-server's. The second is that it provides a possible method to minimize communication traffic in a distributed computing system. Figure 8 shows the operation of a general mobile agents system. In this system configuration, an agent server, exists in each host, receives and sends an agent to the next node of a network. When receiving and transmitting an agent, the agent server uses a monitor that is operating for indicating the status of network flow. The agent moves from host to host carrying its internal state and works its tasks in each host and then returns to its home host when its job is done.

There are several advantages using mobile agents in distributed computing system. Mobile agent architecture is scalable and reliable because there are no central components. The resources and services can be changed dynamically by an agent that has the ability for achieving the goal autonomously. During its migration, a mobile agent

travels using short network paths, and mobile agents can coexist with client-server model applications. While the agents act on a remote site, the system executes other task [13].

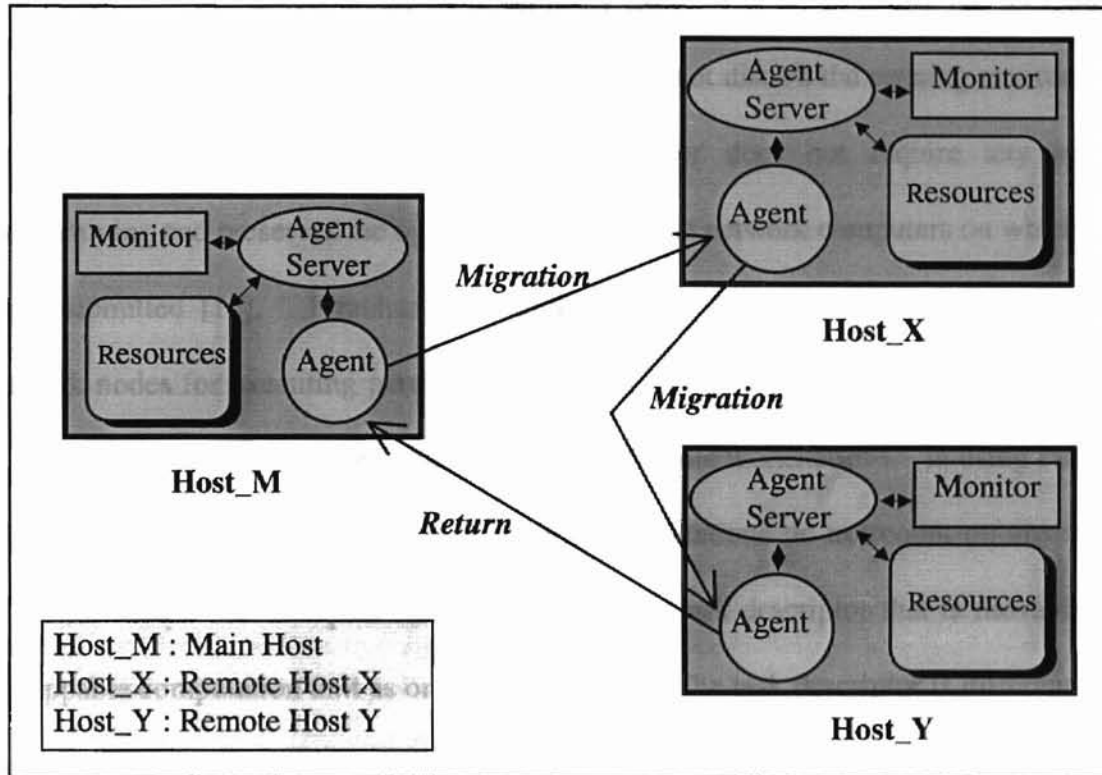


Figure 8. Operations of general mobile agents system.

These advantages have been found in several application systems. In the beginning, these systems have emerged as an adaptive execution of parallel computations such as Condor [17] and Piranha [11]. These systems are running in LAN environment that is comprised of standard UNIX workstations. For adaptive parallelism, executing over a set of dynamically changing processors, these systems utilize idle-state workstations. Condor is a software package developed at the University of Wisconsin - Madison. It is a distributed batch system for automatic location and allocation of idle

machines. Condor has several functions such as automatic location of idle machine and migration of processes. These processes are executed without any modifications to the UNIX kernel programs. Condor is responsible for locating the required resources, making the job to use resources, monitoring its execution and notifying the user on its progress by a batch job. This batch processing does not disturb the running environment of programs provided by workstations. Condor does not require any special programming and preserves the operating system of the network computers on which jobs were submitted [15]. Piranha, developed at Yale University, gathers and uses idle network nodes for executing parallel computations. Piranha provides a mechanism for solving large problems through parallel programs on idle workstations. In using Piranha, programmers do not create processes and their applications do not count on any active processes for parallel computations. Piranha has a task descriptor that is movable and remappable computation unit as one of advantages. The task descriptor is different from processes and supports heterogeneity. A process cannot be moved to other computer systems of different types, but the task descriptor provides an access function in all different types of machines. It is possible using tuplespace. Tuplespace is virtual shared, associative and object memory, accessible to all nodes in all workstations on a LAN. These task descriptors are stored in tuplespace in Piranha, and different nodes can access tuplespace for using a task descriptor [21].

As mobile agents appear in distributed computation, several systems have implemented those for adaptive distributed computing such as Telescript [45], Tacoma [30], Agent Tcl [23] and JVMS [9]. Telescript is an object-oriented programming language for utilizing mobile agents in network environments. Telescript enables

processes to migrate in different computers and Telescript's process model is autonomous process migration. Telescript agents are currently used for network management, active e-mail, electronic commerce and business process management [14]. Tacoma is an operating system support for mobile agents that migrate through a network. As a project, Tacoma dictates how mobile agents can be used to solve problems that have been generally handled by operating system [30]. Agent Tcl is a server that executes on each computer system. Agent Tcl has migration and communication services, security mechanisms and tracking tools such as docking systems that allow an agent to migrate on connected machines and return later. Agent Tcl has been used primarily in information-retrieval applications [33]. JVMS, a Java-Based Mobile actor System, was proposed by Legand Burge in his thesis [9]. JMAS allows a programmer to create mobile actors, initialize their behaviors, and send them messages using constructs provided by the JMAS Mobile Actor API (Application Programming Interface). Since JVMS makes locations of mobile actor visible to programmers, it ensures their explicit control over actor placement.

Mobile agents can be utilized in network management with their various advantages. First, using mobile agents can reduce the network bottleneck through distributing simplified codes instead of transferring large amounts of data. Second, mobile agents can be used to decentralize network management. Even though the main domain's configuration is changed, it does not need to pay attention to the other activities. In the same manner, mobile agents are used in dynamic changing of network management policies. Third, mobile agents are suitable for monitoring large numbers of protocol variables. They are also suitable for memorizing the behavior of network components



for a long time. Mobile agents are used for searching, filtering and collecting data. In addition, mobile agents are suitable for high-speed network management because it does not need to bring all data to the manager. Various usage of mobile agents in network management is well indicated in Sahai and Morin's paper that suggests a distributed and dynamic architecture for network management [41].

In spite of these advantages, mobile agents have serious security problems because mobile processing is generated in open network system. It is difficult to distinguish hostile agents from authorized agents. Many security problems in mobile agents have been disclosed and the methods of conquest of security problems have been suggested by many authors over the years [20, 46].

### **III. DDAS SYSTEM**

To achieve better security in distributed computing and more efficient management of a distributed computation system, DDAS (Dynamic Distributed Agents Server) system is proposed in this thesis. The DDAS system is one of the distributed computation applications based on TCP/IP network environment. It provides the user with the capability to use a dynamic network. The dynamic network is adaptive in the sense that nodes can be added or dropped as needed for a distributed computation in the DDAS system. The dynamic network systems consist of domains that can be accessed by a user with read-write-execute privileges. This is true if a user has an account in a computer. This provides better security because only an authorized user can use a dynamic network for distributed computation. It also supports adaptive network because nodes can be added or removed as necessary.

#### **3.1. The DDAS System Configuration**

The DDAS system consists of a DDAS and its manager. Table 1 describes the communication components and the method of communication and Table 2 indicates the responsibilities of each communication component. Figure 9 shows the basic configuration of the DDAS system. The DDAS exists as a program in UNIX system. It utilizes a specified communication port for construction of a dynamic network. After establishing a dynamic network system with dynamic distributed agents server, the manager in the user host executes to distribute processes, create processes, run processes, kill processes, and delete processes through DDAS for increasing the performance of distributed computing system. The manager is executed as an interface between DDAS

and a distributed job application program that has computation jobs. The application program communicates with the manager through a message-passing function in that the DDAS system and an application program are operated independently. After construction of a dynamic network, computing jobs are distributed to the nodes of the network adaptively. When the nodes complete computation, results are returned to the application program, and nodes are removed adaptively.

	DDAS	Manager	Application (Job)
DDAS	message-passing	method call	N/A
Manager	method call	N/A	method call
Application (Job)	N/A	method call	N/A

Table 1. Communication components and method of communication.

	Responsibilities
DDAS	<ol style="list-style-type: none"> <li>1. Construct a dynamic network</li> <li>2. Message-passing between nodes</li> <li>3. Communicate with the Manager</li> </ol>
Manager	<ol style="list-style-type: none"> <li>1. Initialize the DDAS system</li> <li>2. Manage DDAS while it is active</li> <li>3. Communicate with the application programs</li> </ol>
Application	<ol style="list-style-type: none"> <li>1. Invoke the Manager of the DDAS system</li> <li>2. Notify computing algorithm and data to the Manager</li> <li>3. Communicate with the DDAS system</li> </ol>

Table 2. Responsibilities of each communication component.

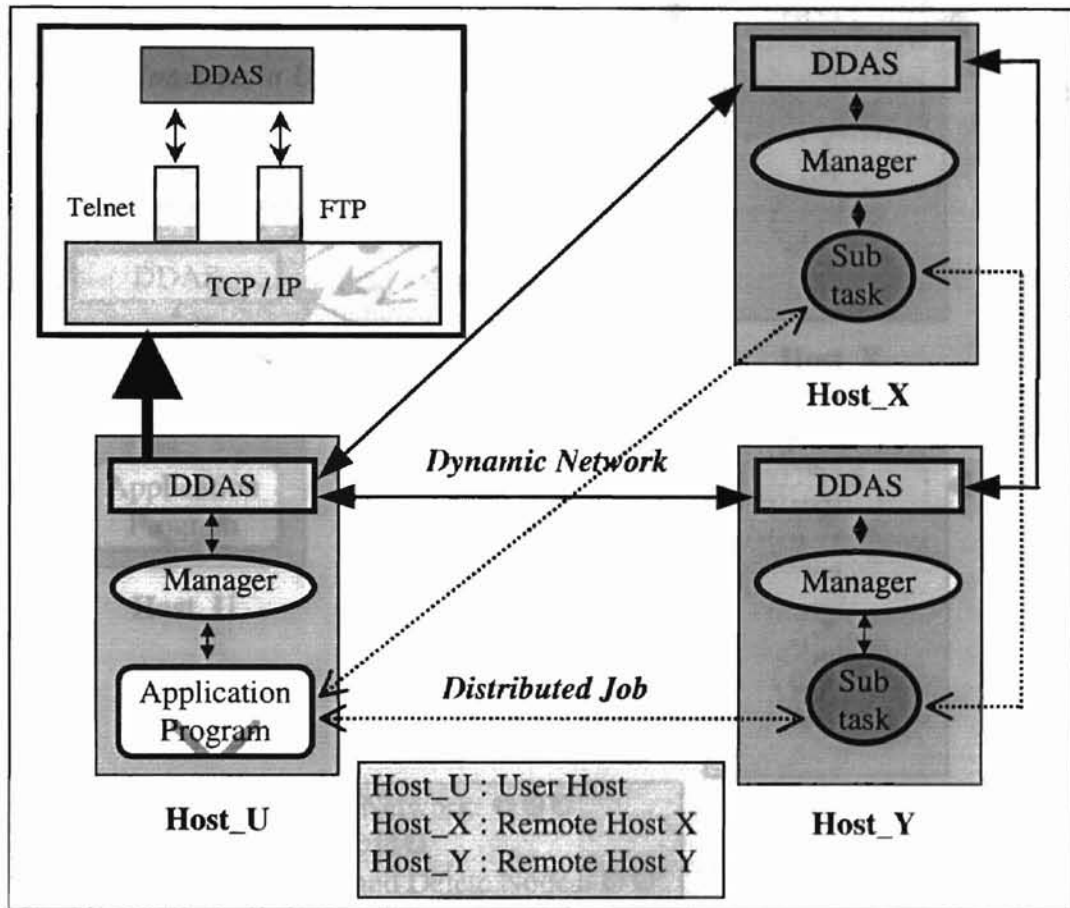


Figure 9. Basic configuration of the DDAS system.

### 3.2. The Basic Operations of the DDAS System

The basic operations of the DDAS system are described in Figure 10. If there are some jobs in a certain host, first of all, the proposed system constructs a dynamic network through the DDAS components. The manager checks the availability of other nodes, distributes DDAS components to those nodes, and initiates their execution.

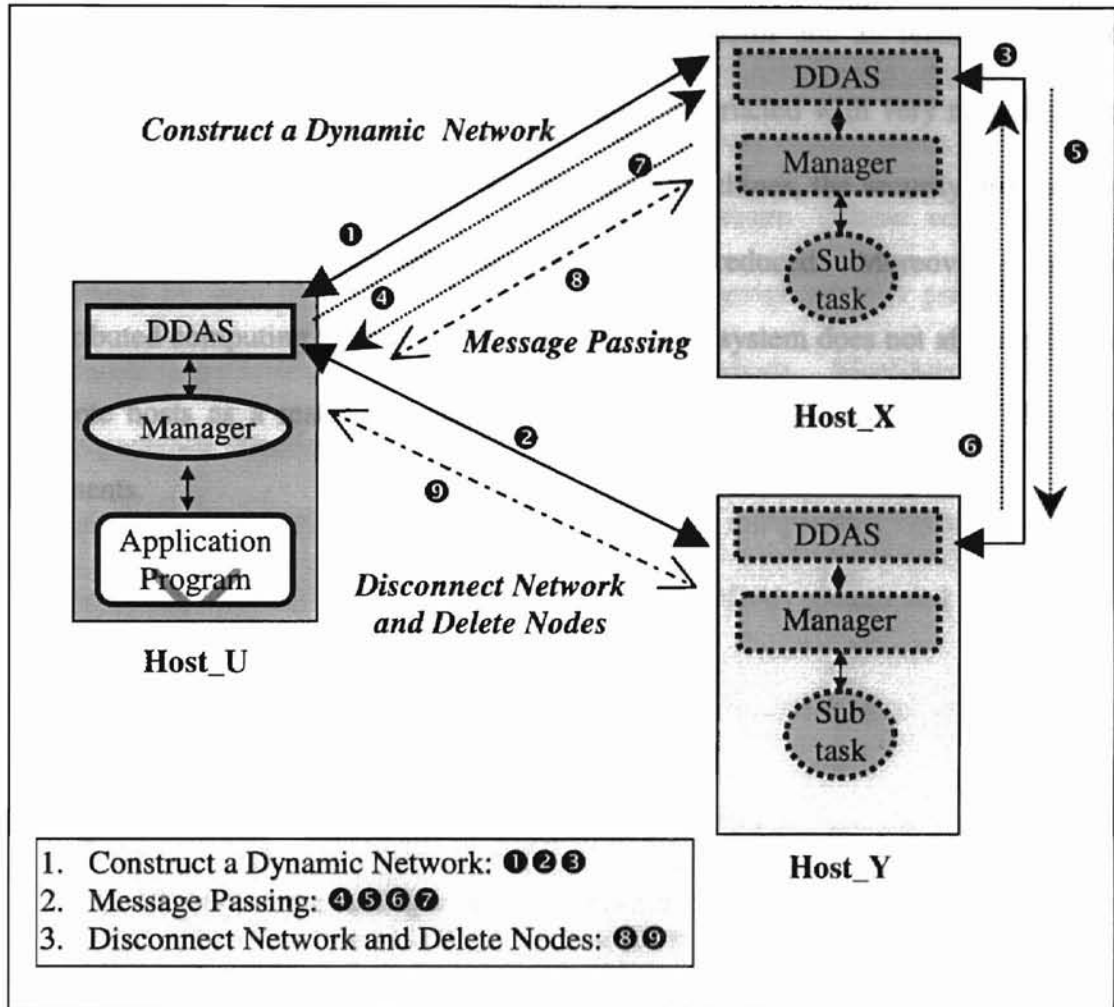


Figure 10. Basic operations of the DDAS system.

As a result of these processes, each DDAS, as a kernel program of a network, can communicate with each other. When the dynamic network construction is completed, message-passing is followed. Through communication with application programs, the manager distributes computing jobs and transfers the results to the application programs. After completion of computing jobs, the DDAS system in the user host disconnects the dynamic network and deletes all of remote nodes for releasing occupied components of

the DDAS systems in the remote hosts. It kills the running processes of the DDAS system in the remote hosts and deletes the DDAS systems and all related jobs in the remote machines. Since the dynamic network is constructed with very reliable remote hosts that consist of only authorized user accessible machines, the security problem, one of the major problems in open network system, can be reduced. Moreover, if there are no distributed computing jobs in a user host, the DDAS system does not affect efficiency of remote hosts as a result of killing processes and deleting occupied DDAS system components.

## IV. IMPLEMENTATION OF THE DDAS SYSTEM

For executing distributed computation, the DDAS system uses its own dynamic network. A dynamic network is built on TCP/IP network environment. The DDAS system consists of two major parts, DDAS and manager. These components are implemented by Java object-oriented programming language since it provides various communication methods and object message-passing methods. Moreover, it provides a data encapsulation function for better security. It also provides a open environment for component-based network computing with its platform independent bytecode standard. In this section, we describe the implementation of the DDAS system and how it can be used to build dynamic network.

### 4.1. Implementation of DDAS and Manager

The DDAS and the manager are two software components composing the system. The DDAS is responsible for construction of dynamic network, message passing between each DDAS, and communication with the manager. All of these functions are provided by the ddas class. The ddas class (ddas.java) consists of two major methods that are `openSocket()` and `connectSocket()`, and two classes, `Sender()` and `Receiver()`. Figure 11 and Figure 12 show these methods and classes. The `openSocket()` method executes a server function that waits for connecting other hosts on the specified port. The specified communication port is one of the available TCP/IP connection ports. The `connectSocket()` method tries to connect to the scheduled host. The scheduled host is similar to the server in the client-server model paradigm. The host is decided when the

DDAS system is invoked from an application program which has computation jobs. These two methods provide the basic functions for the construction of dynamic network. There are two kinds of Sender() and Receiver() classes. One pair of them is for message-passing between each DDAS and the other is for communication with the manager. With these message-passing functions in inter-DDAS, the DDAS system can distribute computation jobs, check the current status of the remote hosts, and kill the running DDAS processes of the remote hosts. Another communication function, between the DDAS and the manager, is accomplished by method call. The DDAS and the manager exist at the same location and the manager controls the DDAS component using method call.

```

public class DDAS extends Concurrent {
    protected static void openSocket() {
        EstablishRendezvous er = null;
        er = new EstablishRendezvous(PORT); // open server socket
        Rendezvous r = null;

        r = er.serverToClient();
        r.serverGetRequest(); // read the data from Client

        // Running Server
        new Receiver("Manager", r);
        r.serverMakeReply("connected"); // send reply to Client
        er.close();
    }

    // connect to the Server socket
    protected static void connectSocket() {
        Rendezvous r = null;

        // create a rendezvous to the Server object
        EstablishRendezvous er = null;
        er = new EstablishRendezvous(Server, PORT);
        r = er.clientToServer();
        // Send setup signal to Server and wait for reply
        r.clientMakeRequestAwaitReply("setup");
        // running Client
        new Sender("ManagerSender", r, er);
    }
}

```

Figure 11. Functions of the DDAS methods.



```

// Server side thread
class Receiver extends DDAS implements Runnable {
    private Rendezvous r = null;
    // construct
    public Receiver(String name, Rendezvous r) {
        this.r = r;
        new Thread(this).start();
    }
    // running a thread for receiving integer data
    public void run() {
        EstablishRendezvous er = null;
        r.serverMakeReply("received");
        r.close();
    }
}

// Client side thread
class Sender extends DDAS implements Runnable {
    private Rendezvous r = null;
    private EstablishRendezvous er = null;

    // construct
    public Sender(String name, Rendezvous r, EstablishRendezvous er)
    {
        this.er = er;
        this.r = r;
        new Thread(this).start();
    }
    // running a thread for transferring data
    public void run() {
        r.clientMakeRequestAwaitReply (msgSend);
        r.close();
        er.close();
    }
}

```

Figure 12. Functions of the DDAS classes.

The manager is responsible for initialization of the DDAS system, managing DDAS during computing works, and communication with the application programs that have computation jobs. The manager class (Manager.java) includes those functions. Figure 13 shows the methods of the manager class for managing the DDAS system. It also consists of several major methods that are inputCheck(), messageCommunication(), sendFiles() and executeFiles(). The inputCheck() method provides the user interface

functions in the DDAS system. Through this method and input commands, the DDAS system can be initialized for data communication and the user can check the DDAS system environment. The IP address of accessible remote hosts and running status of the remote DDAS systems are notified to the user using this method. The messageCommunication() method is for the inter-networking in the DDAS system and it invokes Sender() and Receiver() when there are some messages to transfer. The

```
class Manager extends DDAS {
    public static void main (String[] strg) throws Exception{
        initialSystem();
        while(true) {
            inputCheck();
        }
    }
    // read system configuration file and set-up the system
    static void initialSystem() {
        machineNumber = readSetupFile();
    }
    static void inputCheck() throws Exception{
        commandService(commandNumber, arguments, saveBufCnt);
    }
    public static void commandService(int exeNo, String[] argsOther,
    int argNo) {
        commandService(exeNo);
    }
    private static int checkReceivedData() {
        checkAndMakeData(opcodeNumber, receivedData, saveBufCnt);
    }
    // message passing method
    static void messageCommunication() {
        connectSocket();
        openSocket();
    }
    private static void sendFiles() {
        Process pc = Runtime.getRuntime().exec(commands);
    }
    private static void executeFiles() {
        ExecuteFile efl =
        new ExecuteFile(host, msg, command, msgMore, runMsg);
    }
}
```

Figure 13. Methods of the Manager class.

manager class provides two functions for accessing the `messageCommunication()` method. The `inputCheck()` method calls the `messageCommunication()` method with standard input, a console input operated by a user. An application can invoke this method with proper arguments, also. It makes possible for an application program to invoke a method-call function in the DDAS system. More details about these methods can be found in the following sections. The `sendFiles()` invokes the FTP (File Transfer Protocol) [40] function. The `executeFiles()` invokes the Telnet [40] function. These two methods play an important role in constructing a dynamic network of the DDAS system. The first method sends DDAS components, DDAS and manager to the remote machines and the other methods execute the DDAS system in the remote hosts.

#### **4.2. Construction of a Dynamic Network**

Since the DDAS system uses its own network for the inter-networking, when any application program invokes the DDAS system, the DDAS system constructs a dynamic network. The dynamic network only consists of the authorized remote hosts. This restriction on the construction of a dynamic network is for high security. Figure 14 describes the task flow for construction of a dynamic network.

Initially, an application program in a user host invokes the DDAS system for distributing computation. The manager of the DDAS system communicates with the application program using `messageCommunication()` method. The manager checks the accessible remote machines by reading a system configuration file.

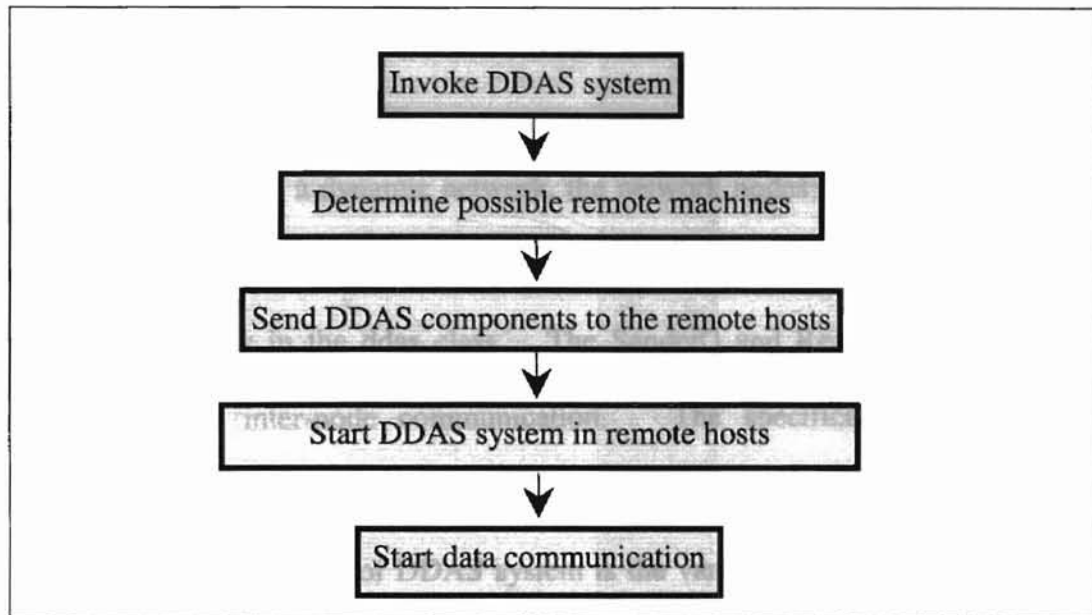


Figure 14. Flow for construction of a dynamic network.

Provided as one of the configuration setup of the DDAS system, the configuration file named `vns.data` has the IP addresses of the remote machines, the user IDs and the user passwords for accessing the remote hosts. After checking the accessible remote hosts, the manager sends DDAS components to the identified remote hosts. It is accomplished by FTP. After copying the DDAS system executable to the remote machines, the manager starts each DDAS system in remote hosts. This job is executed using Telnet that is another kernel application based on the TCP/IP. The DDAS system implements the Telnet function in the client side. Using Telnet function, the DDAS system can execute the remote DDAS systems in the remote hosts. Through all of these steps, the DDAS system now has its own network for distributing computation jobs.

### 4.3. Inter-DDAS Communication

After establishing a dynamic network, the network nodes can communicate with each other through DDAS. All of these data communications are executed by Sender() and Receiver() classes in the ddas class. The Sender() and Receiver() are message-passing classes for inter-node communication. The specification of commands implemented in DDAS system is given below in BNF (Backus-Naur Form). The start symbol of the BNF definition of DDAS system is the variable <command>. Message-passing scheme is used for sending commands from one machine to another.

<command> ::= <op-code> <blank> <machine-name>

<op-code> ::= <letter> | running? | status | execute | busy? | kill | delete | return

<machine-name> ::= <empty> | <letter> | <host-name>

<host-name> ::= <letter> | eslabsvr.wslab.okstate.edu | z.cs.okstate.edu |  
chester.cs.okstate.edu

<letter> ::= <letter> | . | ? | - | a | b | ..... | y | z

<empty> ::=

<blank> ::= any sequence of one or more blanks

A predefined port (port No. 7777) is used for communication between nodes in the network. The method of communication is similar to client-server model. One node makes a request for connection and the other one responds. The network has a hierarchy that is decided when the DDAS system is invoked by an application program. The highest level is the DDAS system of the user host. This is the node in which the

application starts. With inter-DDAS communication, a user can setup the network environment and can check current status of the dynamic network. The `setEnvironment()` command administrates the network environment that includes the IP addresses of accessible remote hosts and the user IDs and passwords. It also displays the current setup status by the user's input commands and the user can check the running status of the remote machines.

#### **4.4. Node Disconnection and Clean up**

When the application in the user host completes and/or the node is not performing any task related to the application, the DDAS system has the ability to disconnect and to remove the node from the dynamic network. This feature of DDAS supports efficient management of the dynamic network of DDAS system. The DDAS system does not affect the efficiency of the remote hosts because of the disconnection and the deletion of useless remote processes. The process of disconnection and deletion consists of several steps. First, the DDAS system deletes all the files created by the task except the DDAS components with the "delete" command issued from the user host. Next, for disconnecting the network, the "kill" command is dispatched to the remote hosts. With the "kill" command, DDAS terminates the running DDAS program in the remote host. At that time, the implemented telnet function is applied again for deleting the remote DDAS components. As a result of these jobs, there are no DDAS components or computation results in the remote machine.

## V. EXECUTION OF DISTRIBUTED COMPUTATION JOBS

Unlike other distributed computation applications where servers are pre-installed, the DDAS system starts the distributed jobs from the construction of its own network. The DDAS system establishes its own network, computes distributed jobs, and nullifies the established network when invoked by application programs. This feature ensures network security and efficient management of the system. In this section, we illustrate the computation processes of an application program.

Since the DDAS system can distribute the application's computing algorithm, several computation algorithms could take advantage of the DDAS system. Consider a large enough sorting job as an application program in the user host. The application program has its own sorting algorithm such as quick sort, merge sort, or shell sort [2]. In this example, let us assume that the algorithm is quick sort. The objective of this example is to illustrate the process. Speed-up of computation is ignored. When the DDAS system is invoked from a computation application with quick sort algorithm, the system constructs a dynamic network through DDASs. Figure 15 describes the computation steps for a sorting job. An application program invokes the DDAS system in the main host, Host<sub>N<sub>0</sub></sub>. The DDAS system of the Host<sub>N<sub>0</sub></sub> checks and calculates the required nodes from the lookup table and an applied algorithm. The lookup table is constructed from reading the environment file and the applied algorithm is the computing algorithm of the application program. Let  $n$  be the total number of nodes available for computation. DDAS system running in Host<sub>N<sub>0</sub></sub> sends its components to the other nodes. After that, the DDAS system running in Host<sub>N<sub>0</sub></sub> makes the DDAS systems

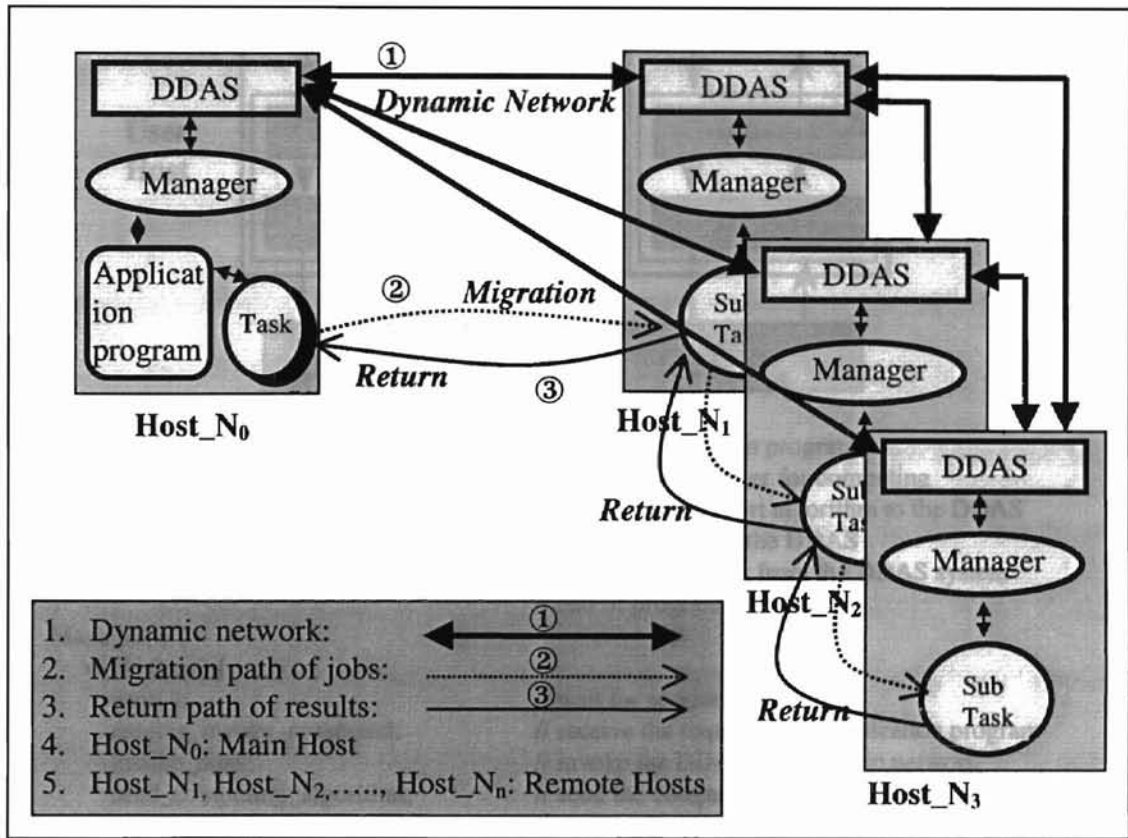


Figure 15. Computation steps of an application program.

executable for nodes Host<sub>N<sub>1</sub></sub> to Host<sub>N<sub>n</sub></sub>. As a result of these, all DDASs can communicate with each other. Figure 15 shows the constructed network of the DDAS system. After the construction of a dynamic network, computation jobs follow. Figure 16 describes the sample code for a distributed job. The quick sort divides the data based on a pivot element. Now, the half of data, smaller than the pivot and the other part, bigger than the pivot are separated into data files in Host<sub>N<sub>0</sub></sub>. One data file is kept in the main machine and the other file is transferred to the Host<sub>N<sub>1</sub></sub>. Now Host<sub>N<sub>0</sub></sub> and Host<sub>N<sub>1</sub></sub> repeat the same computation with the respective data files.



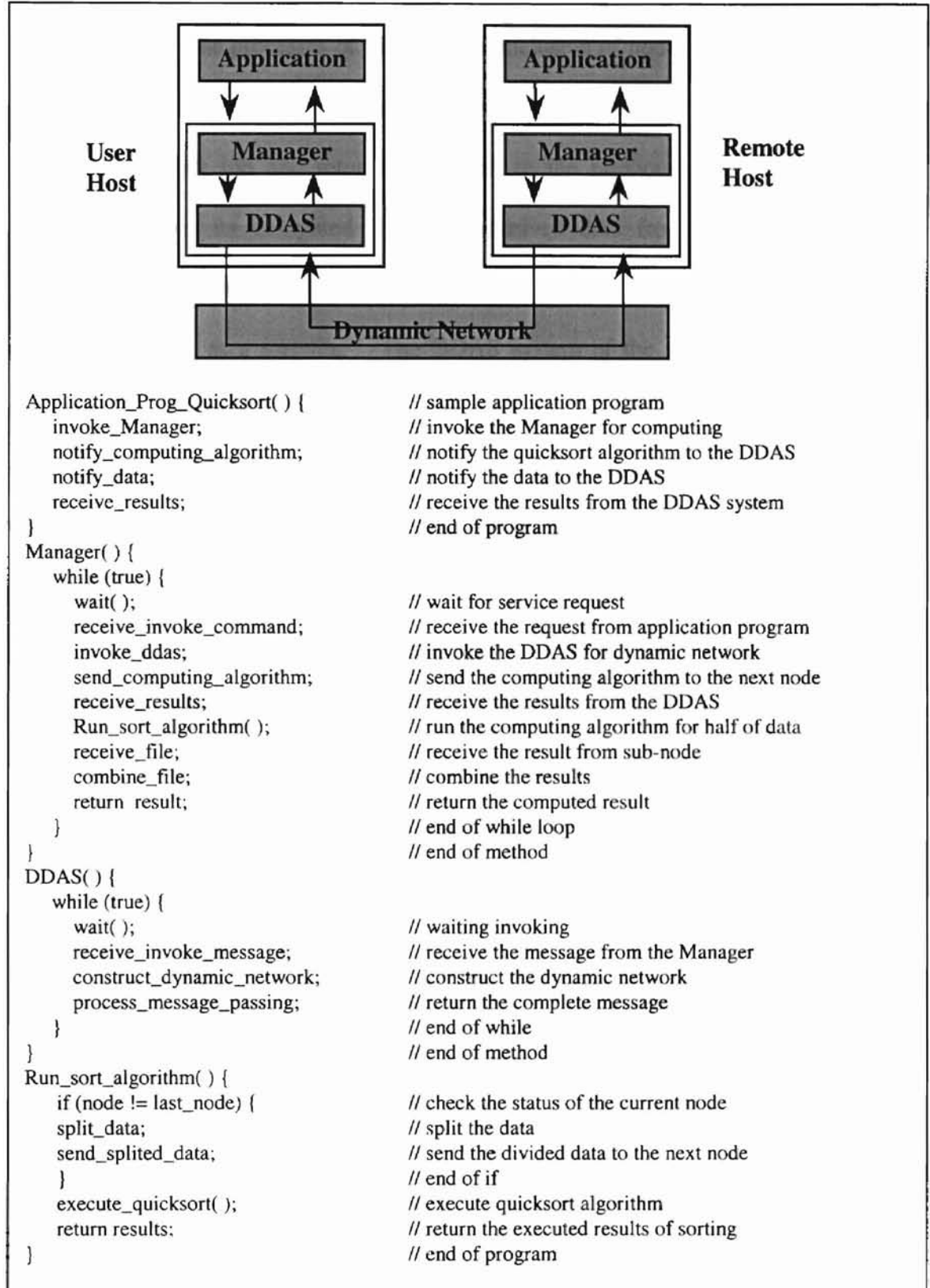


Figure 16. A sample diagram and pseudo-codes for a distributed job.

This process is repeated by Host\_ N<sub>0</sub> and Host\_N<sub>1</sub>. Now two more nodes are included in the computation. The processes of division and inclusion of nodes continue until either no more nodes are required or no node is available. The last node, say Host\_N<sub>n</sub>, computes the sorting job and returns the result to the previous one, say Host\_N<sub>n-1</sub>. Host\_N<sub>n-1</sub> combines its computed data and received data from Host\_N<sub>n</sub>. These combined data are already sorted and send to the previous node and so on. Eventually all data reach Host\_ N<sub>0</sub> as a sort file. The DDAS system of the Host\_N<sub>0</sub> combines the returned data and terminates the computing processes. If there are no computation jobs in the Host\_N<sub>0</sub>, the DDAS system send the “delete” command to the remote hosts. When the “delete” command is received, the remote hosts, from Host\_N<sub>1</sub> to Host\_N<sub>m</sub>, delete all of the computation results. After that, the DDAS system sends “kill” command to the remote hosts. With this command, the remote hosts terminate their processes. After processing those commands, the DDAS system of the Host\_N<sub>0</sub> invokes the telnet function and deletes all of the DDAS components in the remote machines. As a result of executing these processes, the application program of the Host\_N<sub>0</sub> gets its final results and there are no DDAS-related jobs in the remote hosts. It provides better management of the network in the DDAS system.

## VI. TEST AND DISCUSSION

### 6.1. Test Environment

We validated the DDAS system using four machines, namely “csa” (a.cs.okstate.edu), “csz” (z.cs.okstate.edu), “chester” (chester.cs.okstate.edu) and “eslabsvr” (eslabsvr.wslab.okstate.edu). The “csa”, “csz” and “chester” are the host computer systems of the Computer Science Department and the “eslabsvr” is the host computer system of the Electrical Engineering Department. Table 3 describes the operating systems and JVM versions used for the test. As a main host, “csa” was used. In this system environment, a quick sort algorithm was executed as an application program. The DDAS system resides in an open port in “csa” as a kernel of the application. When an application program invokes the DDAS system, the DDAS system sends its components to the remote hosts that are “csz”, “chester” and “eslabsvr”. After that, it makes the DDAS systems in the remote hosts executable using the implemented telnet function. As a result of these, the DDAS system has constructed its own network. We can check the current status of all remote hosts, send the computation jobs, and receive the computed results.

Host Name	IP Address	OS Name	JVM Version
csa	139.78.113.1	SunOS 5.5.1	1.2
csz	139.78.113.110	Solaris 2.x	1.1.3
chester	139.78.113.102	Solaris 2.x	1.1.3
eslabsvr	139.78.96.1	Solaris 2.x	1.1.1

Table 3. System characteristics.

A system user can set-up and check the system when it is running, since the DDAS provides user control functions such as establishing a network, sending files, executing programs, deleting files, and so on by standard input commands. Table 4 describes the commands and their descriptions. These commands can be entered from the Unix shell prompt.

## 6.2. Example

In the beginning of the execution, a quick sort application and the DDAS system are in “csa” that we call main host. The DDAS system is started in the main host as a kernel application. The application that consists of communication part and algorithm part invokes the manager to facilitate distributed computation. Figure 17 describes the flow chart for the distributed quick sort application using the DDAS system. After invoking the manager, execution of application follows. The manager of the DDAS system reads the system configuration, decides the node hierarchy, and constructs the dynamic network. When constructing the dynamic network, the quick sort algorithm is transferred to all the sub nodes. Table 5 shows the files in each machine after construction of the DDAS system. The main host, “csa”, retains an application program (App.class, Quicksort.class, input.dat) and the DDAS system (DDAS.class, Manager.class, Sender.class, Receiver.class, TelnetIO.class, vns.data). The App.class is the main class of the application program, the Quicksort.class is the computation algorithm of quick sort, and the input.dat is the input data file for sorting. The DDAS.class, Manager.class, Sender.class, Receiver.class, and TelnetIO.class are the core stubs for controlling and communicating in DDAS system, and vns.data is the virtual network setup data file.

Command	Arguments	Description
?		Display the DDAS commands
	command	Display the DDAS command's manual
busy?		Check running processes of all remote machines
	machine	Check running process of the machine
delete		Delete all DDAS files of the remote machines
	machine	Delete DDAS files of the machine
establish		Establish a dynamic network
execute		Execute the program
	program machine	Execute program of the machine
exit		Exit the DDAS program
kill		Kill the processes of all remote machines
	machine	Kill the process of the machine
running?		Check the running process
	machine	Check the running process of the machine
send		Send the DDAS stubs to all remote machines
	program	Send the program to all remote machines
	program machine	Send the program to the machine
set		Set environment
	changeid	Change user I.D.
	changepasswd	Change user password
	addmachine	Add the machine for construction of the network
	deletemachine	Delete the machine from the network
status		Check the status of all remote machine
	machine	Check the status of the machine

Table 4. Input command set of the DDAS system.

MACHINE	FILES
a.cs.okstate.edu	App.class, DDAS.class, Manager.class, Quicksort.class, Sender.class, Receiver.class, TelnetIO.class, vns.data, input.dat
eslabsvr.wslab.okstate.edu	DDAS.class, Manager.class, Quicksort.class,
chester.cs.okstate.edu	Sender.class, Receiver.class,
z.cs.okstate.edu	

Table 5. Files after construction of the dynamic network.

The remote hosts, “wslabsvr”, “chester” and “csz”, restore stubs of the DDAS system (DDAS.class, Manager.class, Sender.class, Receiver.class) and the running algorithm for quick sort (Quicksort.class).

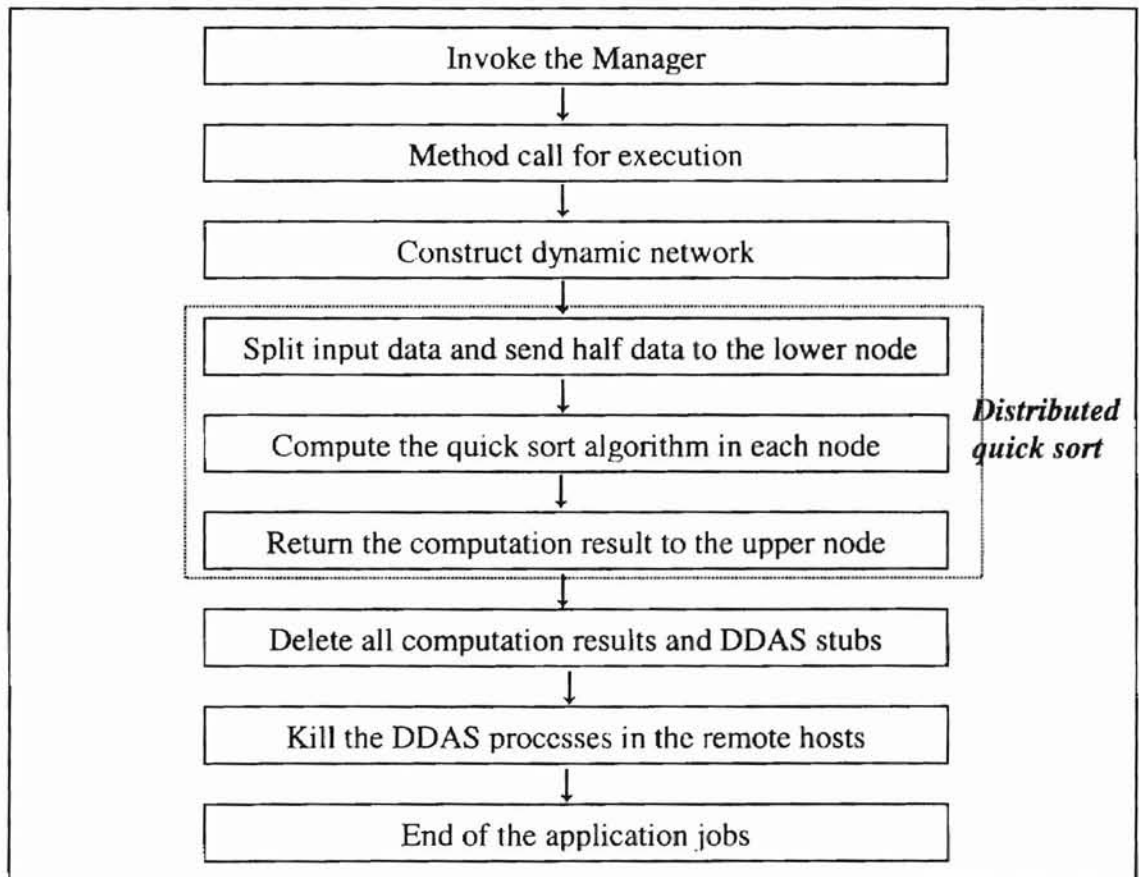


Figure 17. Flow chart for the distributed quick sort application using the DDAS system.

When the construction of dynamic network is completed, the main host splits the data and sends one half to the next node, “eslabsvr”, and performs quick sort algorithm on the other half. The “eslabsvr” receives the data and repeats the process. It sends the data to the next node, “chester”, which also repeats the process. The final node, “csz”, receives the data and runs quick sort algorithm to sort the data received. Table 6 shows the files in each machine when running the application. The input data of the main host is divided into two sets, output1.dat and output2.dat. The output2.dat is transferred to the next node with the name input.dat and output1.dat is processed in the main host. The next node, “eslabsvr”, executes same jobs as the previous host. As a result of these jobs, the files of input.dat, output1.dat and output2.dat are generated in each host except the final host, “csz”. The final host receives output2.dat as an input data, “input.dat”, and sorts it using the sorting algorithm. Figure 18 describes the status of network and distributed computation.

MACHINE	FILES
a.cs.okstate.edu	App.class, DDAS.class, Manager.class, Quicksort.class, Sender.class, Receiver.class, TelnetIO.class, vns.data, input.dat, output1.dat, output2.dat
eslabsvr.wslab.okstate.edu chester.cs.okstate.edu	DDAS.class, Manager.class, Quicksort.class, Sender.class, Receiver.class, input.dat, output1.dat, output2.dat
z.cs.okstate.edu	DDAS.class, Manager.class, Quicksort.class, Sender.class, Receiver.class, input.dat

Table 6. Files after finishing data transfer to the next nodes.

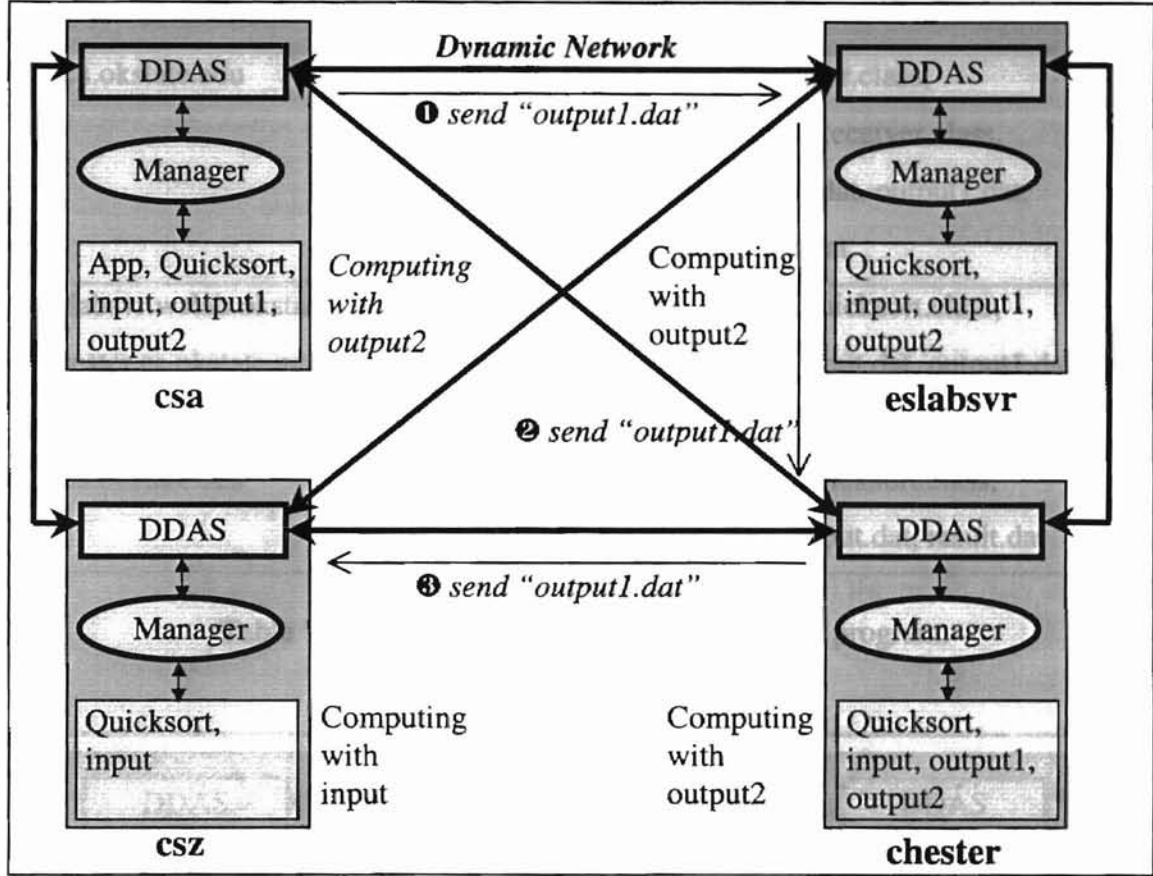


Figure 18. Status of network and distributed computation.

After completing the job assigned to it, each sub-node sends the result to the upper-level node. The last node, "csz", returns its computation result to the upper node, "chester". "chester" receives the data from the lower level node, "csz", merges its computation result and received data, and returns the result to the next upper-node, "eslabsvr". In the same way, "eslabsvr" host sends the computation result to the main host. After receiving the return file from sub-node and combining computation data with return data in the main host, the distributed computations end. Table 7 shows the generated files in each host. The return.dat file is the computation result from the low-level node and the result.dat file is the combination file that adds computation result in



MACHINE	FILES
a.cs.okstate.edu	App.class, DDAS.class, Manager.class, Quicksort.class, Sender.class, Receiver.class, TelnetIO.class, vns.data, input.dat, output1.dat, Output2.dat, return.dat, result.dat
eslabsvr.wslab.okstate.edu chester.cs.okstate.edu	DDAS.class, Manager.class, Quicksort.class, Sender.class, Receiver.class, input.dat, output1.dat, Output2.dat, return.dat, result.dat
z.cs.okstate.edu	DDAS.class, manager.class, Quicksort.class, Sender.class, Receiver.class, input.dat, result.dat

Table 7. Files after execution of an application program.

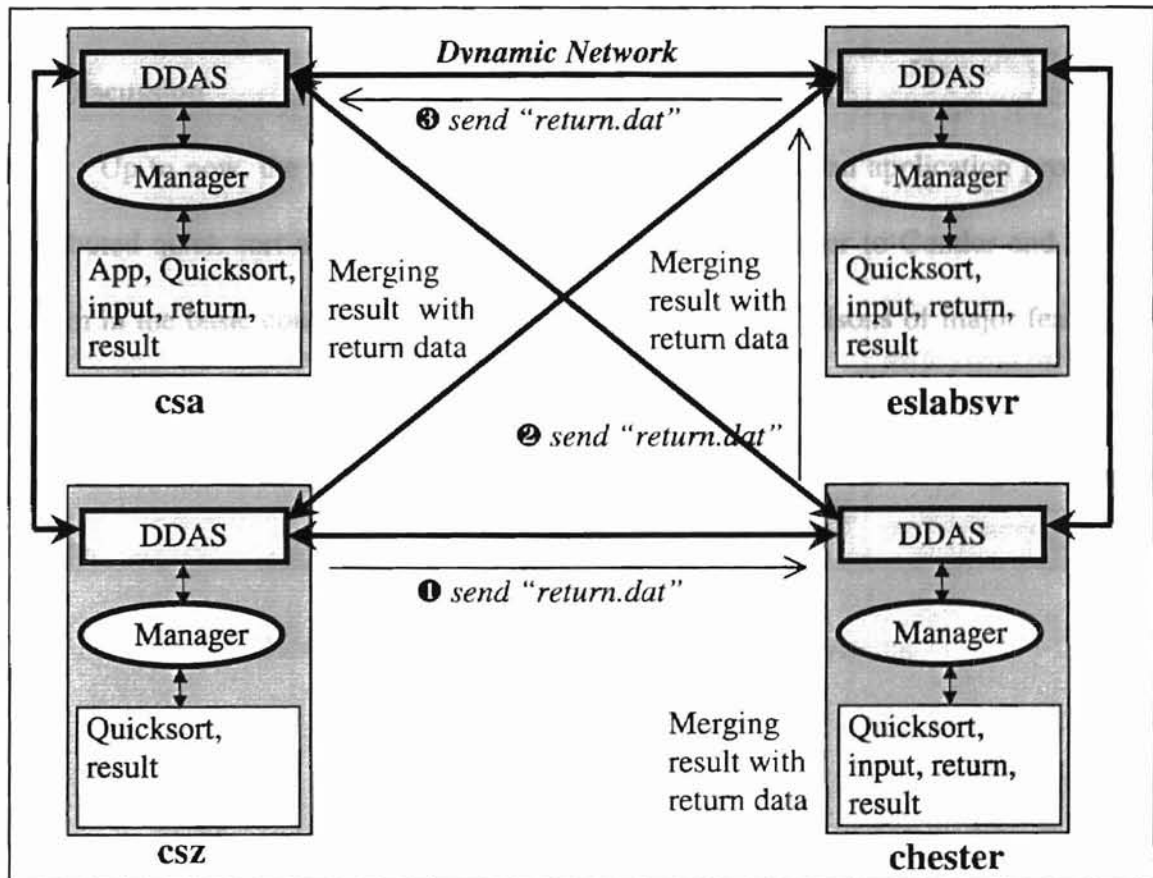


Figure 19. Processes of returning and merging data.

the current node and return.dat from sub-node. Therefore, the result.dat of main host is the final computation result file in computing the quick sort application. Figure 19 describes returning and merging processes.

After the completion of the algorithm, the DDAS system returns the remote hosts to the initial states for efficient management of remote hosts. The DDAS system deletes all the results data from the distributed computation jobs and kills the running DDAS processes in the remote hosts. With these features, the DDAS system controls its own future. Finally, an application program, the DDAS system and the result data are in the main host and there is nothing in remote machines.

### **6.3. Discussion**

Up to now, the DDAS system was used in computing an application program, a distributed quick sort algorithm. The DDAS system is similar to Condor and Piranha system in the basic configuration. Table 8 shows the comparisons of major features of each system. In addition to the comparisons, Table 9 describes the problems of the client-server model and mobile agents and indicates the solutions using the DDAS system.

	<b>Condor</b>	<b>Piranha</b>	<b>DDAS</b>
Network	WAN	LAN	Internet
Remote hosts	Idle machines	Idle machines	Authorized machines
Methodologies of establishing the system	1. Central Manager - main machine 2. Condor Pool - each workstation	Tuplespace - each workstation	DDAS system - main machine
Responsibilities	1. Run two daemons - scheduler & starter 2. Locating the resources 3. Monitoring and informing the user its progress	1. Support uncoupled communication 2. Store task descriptor	1. Construct a network 2. Message-passing for distributed jobs 3. Disconnect the network and delete the nodes
Distributed jobs	Batch system	Task descriptor	DDAS system
Advantage	Does not require special programming	Support heterogeneity	Better security and management
Disadvantage	Jobs are run while there are idle machines in the initial pool	The programmer must supply a handler to deal when a process is forced to vacate a node	Has to gather enough authorized machines for increasing the computation performance

Table 8. Comparisons of DDAS to related systems.

	<b>Client-server model</b>	<b>Mobile agents</b>
<b>Problem</b>	Management problem - Upgrade is required at the same time	Security problem - Difficult to distinguish hostile agents and authorized agents
<b>Cause</b>	Communication fails when server side software or a client side software upgraded; Same communication interfaces are required in both sides.	Mobile processing is generated in the open network system, any mobile program can have access to the hosts.
<b>Solution in the DDAS system</b>	Since remote machine does not have the DDAS system in initial state, the distributed jobs are executed in same DDAS system.	To include a node in the network, login privilege is necessary. Since the distributed jobs have access to only the nodes of own network, security problems are diminished.

Table 9. Problems of client-server and mobile agents and solution in DDAS system.

## **VII. CONCLUSIONS AND FUTURE WORK**

### **7.1. Conclusions**

While client-server model and mobile agents can compute distributed jobs in a connected network only, the DDAS system does not need a connected network. It means there are no occupied DDAS systems in the remote hosts. This feature ensures higher security and provides better system management in distributed computation applications. Throughout running a distributed computing application program, the DDAS system can be one of available distributed applications. Any computation application program can invoke the DDAS system for distributed computing. The set of computers running DDAS forms a fully connected network. Therefore it supports various mapping schemes. In the test case outlined earlier, a linear chain, “csa” → “eslabsvr” → “chester” → “csz”, is used. During the running time, the DDAS system can be created, transferred, executed, killed, and deleted using dynamic network in remote machines. It supports secure distributed system and efficient management system.

### **7.2. Future Work**

Even though the DDAS system provides better security and management of the system, it has a limitation. When a distributed computation process is running and a host of the DDAS system node is too slow or is down, the DDAS system wastes a long time waiting for the results of computation. To deal with this limitation and to pursue an adaptive distributed computation, it needs to implement network monitoring functions which is considered as future work. The monitoring functions can notify the idle machines to the user when the system is invoked. During the computation jobs, the

monitoring functions inform the user of the nodes that completed their work. Implementation of the monitoring functions can increase the performance of DDAS system.

## REFERENCES

- [1] Istabrak Abdul-Fatah and Shikharesh Majumdar, Performance Comparison of Architectures for Client-Server Interactions in CORBA, *Distributed Computing Systems*, May 1998, pp. 2-11, IEEE Computer Society.
- [2] Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974
- [3] Jonathan Aldrich, James Dooley, Scott Mandelsohn and Adam Rifkin, Providing Easier Access to Remote Objects in Client-Server Systems, *System Sciences*, Jan. 1998, Vol. 7, pp. 366-375, IEEE Computer Society.
- [4] Douglas T. Anderson, Pat Dawson, Sandra Honomachl and Michael Tribble, *The Network Technical Guide*, Micro House International Inc., 1997 004.64 N476 11
- [5] Miro Benda, Middleware: any client, any server, *IEEE Internet Computing*, July-Aug. 1997, Vol. 1, Issue 4, pp. 94-96, IEEE Computer Society
- [6] Jurgen Berghoff, Oswald Drobnik, Anselm Lingnau and Christian Monch, Agent-based configuration management of distributed applications, *Configurable Distributed Systems*, May 1996, pp. 52-59, IEEE Computer Society.
- [7] A. Berson, *Client-server Architecture*, 2nd edition, McGraw-Hill, 1996 004.36 B5257 193
- [8] M. Bertocco, F. Ferraris, C. Offelli and M. Parvis, A Client-Server Architecture for Distributed Measurement System, *Instrumentation and Measurement Technology Conference*, May 1998, Vol. 1, pp. 67-72, IEEE 10-11
- [9] Legand L. Burge III, JMAS: A Java-Based Mobile Actor System for Heterogeneous Distributed Parallel Computing, *Thesis*, Dec. 1998, Oklahoma State University 300 B.9541
- [10] L.M. Camarinha-Matos and Walter Vieira, Mobile Agents and Remote Operation, *Intelligent Engineering Systems*, Sept. 1997, pp. 463-468, IEEE
- [11] Nicholas Carriero, Eric Freeman, David Gelerner and David Kaminsky, Adaptive Parallelism and Piranha, *Computer*, Vol. 28, Issue 1, pp. 40-49, IEEE Computer Society.
- [12] Todd Courtois, *Java Networking and Communications*, Prentice-Hall, Inc., 1997 001.501 J6118
- [13] Manfred Dalmeijer, Eric Rietjens, Dieter Hammer, Ad Aerts and Michiel Soede, A Reliable Mobile Agents Architecture, *Object-oriented Real-time Distributed Computing*, April 1998, pp. 64-72, IEEE Computer Society.

- [14] Peter Domel, Mobile Telescript Agents and the Web, *Technologies for the Information Superhighway*, Feb. 1996, pp. 52-57, IEEE Computer Society.
- [15] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers and J. Pruyne, A worldwide flock of Condor: load sharing among workstation clusters, *Report 95-130*, The Faculty of Technical Mathematics and Informatics, Delft, The Netherlands.
- [16] Eric Evans and Daniel Rogers, Using Java Applets and CORBA for Multi-User Distributed Applications, *IEEE Internet Computing*, May-June 1997, Vol. 1, Issue 3, pp. 43-55, IEEE
- [17] X. Evers, J.F.C.M. de Jongh, R. Boontje, D.H.J. Epema and R. van Dantzig, Condor flocking: load sharing between pools of workstations, *Report 93-104*, The Faculty of Technical Mathematics and Informatics, Delft, The Netherlands.
- [18] Jim Farley, *Java Distributed Computing*, O'Reilly & Associates, Inc., 1998 005.133 J41 f
- [19] Sidnie Feit, *TCP/IP: Architecture, Protocols, and Implementations with IPv6 and IP Security*, 2nd edition, McGraw-Hill, 1996
- [20] Stefan Fuenfroeken, Integrating Java-based Mobile Agents into Web Servers under Security Concerns, *System Sciences*, Jan. 1998, Vol. 7, pp. 34-43, IEEE Computer Society.
- [21] David Gelernter and David Kaminsky, Supercomputing out of recycled garbage: preliminary experience with Piranha, *Proceedings of the 1992 International Conference on Supercomputing*, pp. 417-427, ACM, Inc.
- [22] James Gosling, Bill Joy and Guy Steele, *The Java Language Specification*, Addison-Wesley, 1996 005.127 J413 b3
- [23] Robert Gray, David Kotz, Saurab Nog, Daniela Rus and George Cybenko, Mobile Agents: The Next Generation in Distributed Computing, *Parallel Algorithms/Architecture Synthesis*, March 1997, pp. 8-24, IEEE Computer Society.
- [24] David Gunter, Steven Burnett, Gregory L. Field, Lola Gunter, Tom Klejna, Shankar Lakshman, Alexia Prendergast, Mark C. Reynolds and Marcia E. Rolnald, *Client/Server Programming with RPC and DCE*, Que Corporation, 1995
- [25] P.W. Halliden, Security for distributed applications, *Security and Detection*, 1995, pp. 156-160, IEE
- [26] Elliotte Rusty Harold, *Java Network Programming*, O'Reilly & Associates, Inc., 1997
- [27] Stephen J. Hartley, *Concurrent Programming: The Java Programming Language*, Oxford University Press, Inc., 1998 005.2757 H1332 f



- [28] Julius Hrivnac, *DCE Distributed Computing Environment*, <http://hp18.fzu.cz/~hrivnac/DCE/WhatIsIt/index.htm>
- [29] *Java Remote Method Invocation*, <http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html>
- [30] Dag Johansen, Robbert van Renesse and Fred B. Schneider, Operating System Support for Mobile Agents, *Hot Topics in Operating Systems*, May 1995, pp. 42-45, IEEE Computer Society.
- [31] Thomas K. Keyser and Robert P. Davis, Distributed computing approaches toward manufacturing scheduling problems, *IIE Transactions*, Apr. 1998, Vol. 30, Issue 4, pp. 379-390, Institute of Industrial Engineers
- [32] K.M. Khalil, A Dual Network Architecture for High-Performance Distributed Computing Environments, *Local Computer Networks*, Sept. 1993, pp. 354-360, IEEE Computer Society.
- [33] 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, IEEE, <http://www.cs.dartmouth.edu/~agent>
- [34] Hosoon Ku, Luderer, Gottfried W.R. and B. Subbiah, An Intelligent Mobile Agent Framework for Distributed Network Management, *Global Telecommunications Conference*, Nov. 1997, Vol. 1, pp. 160-164, IEEE
- [35] John Lewis and William Loftus, *Java Software Solution: Foundations of Program Design*, Addison-Wesley, 1998 105.133.34126
- [36] Tim Lindholm and Frank Yellin, *The Java Virtual Machine Specification*, 1996, <http://java.sun.com/docs/books/vmspec/html/VMSpecTOC.doc.html>
- [37] Thomas J. Mowbray and Raphael C. Malveau, *CORBA Design Patterns*, John Wiley & Sons, Inc., 1997
- [38] Michael T. Peterson, *DCE: A Guide to Developing Portable Applications*, McGraw-Hill, 1995
- [39] Paul E. Renaud, *Introduction to Client/Server Systems*, John Wiley & Sons, Inc., 1996
- [40] *RFCs by the numbers*, <http://ftp.tu-clausthal.de/pub/TEXT/docs/rfc/rfc>

- [41] Akhil Sahai and Christine Morin, Towards distributed and dynamic network management, *Network Operations and Management Symposium*, Feb. 1998, Vol. 2, pp. 455-464, IEEE
- [42] C. Sashidhar and S.M. Shatz, Design and Implementation Issues for Supporting Callback Procedures in RPC-Based Distributed Software, *Computer Software and Applications Conference*, Aug. 1997, pp. 460-466, IEEE Computer Society.
- [43] Paul L. Schlieve, *Demystifying TCP/IP*, 2nd edition, Wordware Publishing, Inc., 1998
- [44] Kam Hong Shum, Adaptive distributed computing through competition, *Configurable Distributed Systems*, May 1996, pp. 220-227, IEEE Computer Society.
- [45] Joseph Tardo and Luis Valente, Mobile Agent Security and Telescript, *Technologies for the Information Superhighway*, Feb. 1996, pp. 58-63, IEEE Computer Society.
- [46] Hartmut Vogler, Thomas Kunkelmann and Marie-Louise Moschgath, Distributed Transaction Processing as a Reliability Concept for Mobile Agents, *Distributed Computing Systems*, Oct. 1997, pp. 59-64, IEEE Computer Society.
- [47] E.J. Well, LAN/WAN integration: internetworking + application interoperability, *Military Communications Conference*, Oct. 1994, Vol. 1, pp. 220 - 226, IEEE
- [48] Ann Wollrath, Jim Waldo and Roger Riggs, Java-Centric Distributed Computing, *IEEE Micro*, May-June 1997, Vol. 17, Issue 3, pp. 44-53, IEEE

## **APPENDIX A**

### **ABBREVIATIONS AND ACRONYMS**

## ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
ALPANET	Advanced Research Projects Agency NETwork
CORBA	Common Object Request Broker Adapter
DCE	Distributed Computing Environment
DDAS	Dynamic Distributed Agents Server
DOD	Department Of Defense
FTP	File Transfer Protocol
IDL	Interface Description Language
IIOB	Internet Inter-ORB Protocol
IP	Internet Protocol
ISO	International Organization for Standardization
JDK	Java Development Kit
JVM	Java Virtual Machine
LAN	Local Area Network
OMG	Object Management Group
ORB	Object Request Broker
OSF	Open Software Foundation
OSI	Open Systems Interconnection
RMI	Remote Method Calls
RPC	Remote Procedure Call
SMTPS	Simple Mail Transfer Protocol
SQL	Structured Query Language

TCP	Transmission Control protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
WAN	Wide Area Network
WWW	World Wide Web

## **APPENDIX B**

### **PARTIAL DDAS SYSTEM SOURCE CODE**

```

/*****
/*          DDAS Program          */
/*          */
/*          */
/* Function:          */
/* 1. Install Server socket on the open port          */
/* 2. Connect to the Server socket          */
/* 3. Running Receiving thread          */
/* 4. Running Sending thread          */
*****/

import java.net.*;
import java.io.*;
import Synchronization.*;

public class DDAS extends Concurrent {
    protected static String Server = null;
    protected static String msgReceived = null;
    protected static String msgSend = null;
    protected static int nextFlag = 0;
    protected static int countLoop = 1;

    private static final int PORT = 7777;

    public static boolean debug = false;

    // set-up socket as a server
    protected static void openSocket() {
        nextFlag = 0;
        EstablishRendezvous er = null;
        try {
            er = new EstablishRendezvous(PORT); // open server socket
            if(debug)
                System.out.println ("Waiting for connection on port " + PORT);
        } catch (MessagePassingException e) {
            System.err.println(e);
            System.exit(1);
        }

        // install Server socket
        Rendezvous r = null;
        try {
            r = er.serverToClient();
        } catch (MessagePassingException e) {
            System.err.println("Server:" + e);
        }

        // check Client approach
        r.serverGetRequest(); // read the data from Client

        // Running Server
        new Receiver("Calculator", r);
        if(debug) System.out.println("Connected with another
            Host."+"\n");
    }
}

```

```

        // replay to the Client
        r.serverMakeReply("connected"); // send reply to Client
        er.close();
    }

    // connect to the Server socket
    protected static void connectSocket() {
        nextFlag = 1;
        Rendezvous r = null;

        // create a rendezvous to the Server object
        EstablishRendezvous er = null;
        try {
            er = new EstablishRendezvous(Server, PORT);
            r = er.clientToServer();
            if(debug) System.out.println("\nConnected to " + Server
                + " on port " + PORT);
        } catch (MessagePassingException e) {
            System.err.println(e);
            System.exit(1);
        }
        // Send setup signal to Server and wait for reply
        r.clientMakeRequestAwaitReply("setup");

        // runnig Client
        new Sender("ManagerSender", r, er);
    }
}

// Server side thread
class Receiver extends DDAS implements Runnable {
    private Rendezvous r = null;

    // construct
    public Receiver(String name, Rendezvous r) {
        this.r = r;
        new Thread(this).start();
    }

    // running a thread for receiving integer data
    public void run() {
        EstablishRendezvous er = null;

        msgReceived = (String)r.serverGetRequest();
        if(debug) System.out.println(age()+"ms, "+"Received data.");
        if(debug) System.out.println("Received data from the Client: ");
        if(debug) System.out.println(msgReceived);

        r.serverMakeReply("received");
        nextFlag = 1;
        r.close();
    }
}
}

```



```

// Client side thread
class Sender extends DDAS implements Runnable {
    private Rendezvous r = null;
    private EstablishRendezvous er = null;

    // construct
    public Sender(String name, Rendezvous r, EstablishRendezvous er) {
        this.er = er;
        this.r = r;
        new Thread(this).start();
    }

    // running a thread for transferring data
    public void run() {

        pause(breakTime);
        if(debug)
            System.out.println(age()+"ms, sending data to the Client");
        try{
            r.clientMakeRequestAwaitReply (msgSend);
        } catch (MessagePassingException e) {
            System.err.println(e);
            r.close();
            System.exit(1);
        }
        pause(breakTime);
        r.close();
        er.close();
        nextFlag = 0;
    }
}

```

```

/*****
/*                               Manager Program                               */
/*                               */
/*                               */
/* Function:                       */
/* 1. Communicate with application programs                       */
/* 2. Communicate with the DDAS component                       */
/* 3. Provide user interface functions                       */
*****/

```

```

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

```

```

class Manager 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;

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

public static boolean debug = true;

static Quicksort q = new Quicksort();

// static DDAS ddas = new DDAS();
public static void main (String[] strg) throws Exception{
    initialSystem();
    while(true) {
        System.out.print("\nDDAS>> ");
        inputCheck();
    }
}

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

static void inputCheck() throws Exception{
    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)
    throws Exception{
    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:
    //      sendMessage = "delete";
    //      makeSendMessage (sendMessage);

    //messageCommunication();

    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);

//checkReceivedData();
if (msgReceived.equals("return"))
System.out.println("ENDENDEND");
q.concatFile();

break;

// exit service
case 5:
System.out.println("End of program. Bye!\n\n");
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:
    System.out.println("Not serviced command. Type ?");
    break;
}
}

private static int checkReceivedData() {
    StringTokenizer checker = new StringTokenizer(msgReceived);
    int opcodeNumber = 0;

    String[] receivedData = new String[MAX_RECE_DATA];
    int saveBufCnt = checker.countTokens();

    for (int i = 0; i < saveBufCnt; i++)
        receivedData[i] = checker.nextToken();

    opcodeNumber = makeOpcodeToInt(receivedData[0]);
    if(debug) System.out.println("Command No: "+opcodeNumber);

    checkAndMakeData(opcodeNumber, receivedData, saveBufCnt);
    return opcodeNumber;
}

```

```

static void messageCommunication() {
    for (int i = 0; i < machineNumber; i++) {
        //      System.out.println ("Waiting ... ");
        Server = machineNames[i];
        connectSocket();
        while(nextFlag == 1) pause(PAUSE);

        openSocket();
        while(nextFlag == 0) pause(PAUSE);
        System.out.println(machineNames[i] + "\n"+ msgReceived);
        System.out.println();
    }
}

/*
static void messageCommunication() {
    //      for (int i = 0; i < machineNumber; i++) {
    //          System.out.println ("Waiting ... ");
    Server = "a.cs.okstate.edu";
    connectSocket();
    while(nextFlag == 1) pause(PAUSE);

    openSocket();
    while(nextFlag == 0) pause(PAUSE);
    System.out.println(machineNames + "\n"+ msgReceived);
    System.out.println();
    //      }
}
*/

static void messageCommunication(String machine) {
    // System.out.println ("Waiting ... ");
    Server = machine;
    connectSocket();
    while(nextFlag == 1) pause(PAUSE);

    openSocket();
    while(nextFlag == 0) pause(PAUSE);
    System.out.println(machine + "\n"+ msgReceived);
    System.out.println();
}

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

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

```

```

static void setEnvironment(String[] argsOther) {
    if(argsOther[1].equals("changeId")) {
        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);
    }
}

/*
 * Display Environment
 */
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);
}

/*
 * Help service
 * display command + argument & how to use
 */
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 changeId 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(debug) System.out.println (readData);
            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;
}

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

    fileName[0] = "default1.dat";
    fileName[1] = "default2.dat";
    fileName[2] = "default3.dat";

    try {
        for (int i = 0; i < machineNumber; i++) {
            commands = "mftp" + " " + machineNames[i] + " " + fileName[i];
            Process pc =
                Runtime.getRuntime().exec(commands);
            pc.waitFor();
            System.out.println(
                "Sending "+fileName[i]+" & Manager files to
                "+machineNames[i]);
        }
    } 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;
    //String fileName = "output2.dat";

    try {
        //for (int i = 0; i < machineNumber; i++) {
            commands = "ftpdata" + " " + machineNames + " " + fileName
                + " " + desFilename;
            Process pc =
                Runtime.getRuntime().exec(commands);
            pc.waitFor();
            System.out.println(
                "Sending "+fileName+ " & Manager files to "+
                machineNames);
            // }
    } catch (IOException e) {
        System.err.println(e);
    } catch (InterruptedException e) {

```

```

        System.out.println("");
    }
}

private static void executeFiles() {

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

        System.out.println("Running DDAS Program in "+ host + " ...");
        ExecuteFile efl =
            new ExecuteFile(host, msg, command, msgMore, runMsg);
    }
}

private static void deleteAllFiles() {
    for (int i = 0; i < machineNumber; i++) {
        String host = machineNames[i];
        String msg = "/home/kkwan>>";
        String msgMore = "/home/kkwan/java/DDAS>>";
        String command = "rm -r *\r";
        String runMsg = ">>> Running DDAS System";

        System.out.println("Delete All DDAS Files in "+ host + " ...");
        DeleteFile df1 =
            new DeleteFile(host, msg, command, msgMore);
    }
}

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;
    }
}
}
}

```

VITA

**Kwan-Sung Kim**

Candidate for the Degree of  
Master of Science

Thesis: DDAS: DESIGN AND IMPLEMENTATION OF A KERNEL APPLICATION  
FOR ADAPTIVE DISTRIBUTED COMPUTATION

Major Field: Computer Science

Biographical:

Personal Data: Born in Cheju, Korea, February 15, 1965, the son of Kwang-II Kim and Jung-Sim Choi.

Education: Graduated from Ohyun High School, Cheju, Korea in February, 1984; received Bachelor of Engineering in Electronic Engineering from Kyunghee University, Seoul, Korea in February, 1991; completed requirements for the Master of Science in Computer Science at Oklahoma State University in July, 1999.

Professional Experience: Research Assistant, Department of Computer Science, Oklahoma State University, November, 1998 to June, 1999; Assistant Manager, R&D Department, S1 Corporation, July, 1995 to January, 1997; System Developer, R&D Department, S1 Corporation, January, 1991 to June, 1995;