Chapter 2

Distributed System Design Framework

Objective of this chapter:

Chapter two will present a design methodology model of distributed systems to simplify the design and the development of such systems. In addition, we provide an overview of all the design issues and technologies that can be used to build distributed systems.

Key Terms

Network, protocol, interface, Distributed system design, WAN, MAN, LAN, LPN, DAN, Circuit switching, Packet Switching, Message Switching, Server Model, Pool Model, Integrated Model, Hybrid Model.

2.1 Introduction

In Chapter 1, we have reviewed the main characteristics and services provided by distributed systems and their evolution. It is clear from the previous chapter that there are a lot of confusions on what constitute a distributed system, its main characteristics and services, and their designs. In this chapter, we present a framework that can be used to identify the design principles and the technologies to implement the components of any distributed computing system. We refer to this framework as the Distributed System Design Model (DSDM). Generally speaking, the design process of a distributed system involves three main activities: 1) designing the communication network that enables the distributed system computers to exchange information, 2) defining the system structure (architecture) and the services that enable multiple computers to act and behalf as a system rather than a collection of computers, and 3) defining the distributed programming techniques to develop distributed applications. Based on this notion of the design process, the Distributed System Design Model can be described in terms of three layers: (see Figure 2.1): 1) *Network, Protocol, and Interface (NPI) layer*; 2) *System Architecture and Services (SAS) layer*; and 3) *Distributed Computing Paradigms (DCP) layer*. In this chapter, we describe the functionality and the design issues that must be taken into consideration during the design and implementation of each layer. Furthermore, we organize the book chapters into three parts where each part corresponds to one layer in the DSDM.

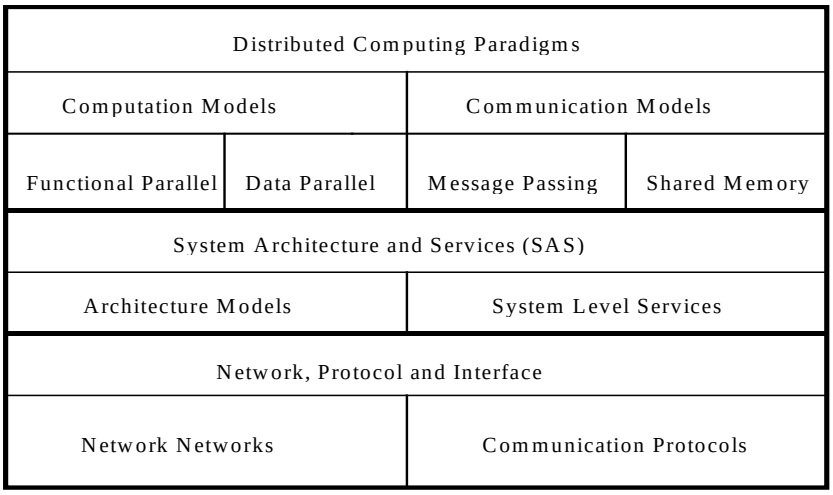


Figure 2.1. Distributed System Design Model.

The communication network, protocol and interface (NPI) layer describes the main components of the communication system that will be used for passing control and information among the distributed system resources. This layer is decomposed into three sub-layers: Network Types, Communication Protocols, and Network Interfaces.

The distributed system architecture and service layer (SAS) defines the architecture and the system services (distributed file system, concurrency control, redundancy management, load sharing and balancing, security service, etc.) that must be supported and supported in order for the distributed system to behave and function as if it were a single image computing system.

The distributed computing paradigms (DCP) layer represents the programmer (user) perception of the distributed system. This layer focuses on the programming paradigms that can be used to develop distributed applications. Distributed computing paradigms can be broadly characterized based on the computation and communication models. Parallel and distributed computations can be described in terms of two paradigms: Functional Parallel and Data Parallel paradigms. In functional parallel paradigm, the computations are assigned to different computers. In data parallel paradigm, all the computers perform the same functions, Same Program Multiple Data Stream (SPMD), but each function operates on different data streams. One can also characterize parallel and distributed computing based on the techniques used for inter-task communications into two main models: Message Passing and Distributed Shared Memory models. In message passing paradigm, tasks communicate with each other by messages, while in distributed shared memory they communicate by reading/writing to a global shared address space.

In the following subsections, we describe the design issues and technologies associated with each layer in the DSDM.

2.2 Network, Protocol and Interface

The first layer (from the bottom-up) in the distributed system design model addresses the issues related to designing the computer network, communications protocols, and host interfaces. The communication system represents the underlying infrastructure used to exchange data and control information among the logical and physical resources of the distributed system. Consequently, the performance and the reliability of distributed system depend heavily on the performance and reliability of the communication system.

Traditionally distributed computing systems have relied entirely on local area networks to implement the communication system. Wide area networks were not considered seriously because of their high-latency and low-bandwidth. However, the current emerging technology has changed that completely. Currently, the WAN operate at Terabit per second transmission rates (Tbps) as shown in Figure 1.2.

A communication system can be viewed as a collection of physical and logical components that jointly perform the communication tasks. The physical components (network devices) transfer data between the host memory and the communication medium. The logical components provide services for message assembly and/or de-assembly, buffering, formatting, routing and error checking. Consequently, the design of a communication system involves defining the resources required to implement the functions associated with each component. The physical components determine the type of computer network to be used (LAN's, MAN's, WAN's), type of network topology (fully connected, bus, tree, ring, mixture, and random), and the type of communication medium (twisted pair, coaxial cables, fiber optics, wireless, and satellite), and how the host accesses the network resources. The logical components determine the type of communication services (packet switching, message switching, circuit switching), type of information (data, voice, facsimile, image and video), management techniques (centralized and/or distributed), and type of communication protocols.

The NPI layer discusses the design issues and network technologies available to implement the communication system components using three sub-layers: 1) Network Type that discusses the design issues related to implement the physical computer network, 2) Communications Protocols that discusses communication protocols designs and their impact on distributed system performance, and 3) Host Network Interface that discusses the design issues and techniques to implement computer network interfaces.

2.2.1 Network Type

A computer network is essentially any system that provides communication between two or more computers. These computers can be in the same room, or can be separated by several thousands of miles. Computer networks that span large geographical distances are fundamentally different from those that span short

distances. To help characterize the differences in capacity and intended use, communications networks are generally classified according to the distance into five categories: 1) Wide Area Network (WAN), 2) Metropolitan Area Network (MAN), 3) Local Area Network (LAN), 4) Local Peripheral Network (LPN), and 5) Desktop Area Network (DAN).

Wide area networks (WANs): WANs are intended for use over large distances that could include several national and international private and/or public data networks. There are two types of WANs: Packet switched and Circuit Switched networks. WANs used to operate at slower speeds (e.g., 1.54 Mbps) than LAN technologies and have high propagation delays. However, the recent advances in fiber optical technology, wavelength division multiplexing technology and the wide deployment of fiber optics to implement the backbone network infrastructure have made their transmission rates higher than the transmission rates of LANs. In fact, it is now approaching Pita transmission rates (Pbps).

Metropolitan area networks (MANs)*:* MANs span intermediate distances and operate at medium-to-high speeds. As the name implies, a MAN can span a large metropolitan area and may or may not use the services of telecommunications carriers. MANs introduce less propagation delay than WANs and their transmission rates range from 56Kbps to 100 Mbps.

Local area networks (LANs)*:* LANs normally used to interconnect computers and different types of data terminal equipment within a single building or a group of buildings or a campus area. LANs provide the highest speed connections (e.g., 100 Mbps, 1 Gbps) between computers because it covers short distances than those covered by WANs and MANs. Most LANs use a broadcast communication medium where each packet is transmitted to all the computers in the network.

Local peripheral networks (LPNs)*: LPNs* can be viewed as special types of LANs [Tolmie and Tanlawy, 1994; Stallings et al, 1994] and it covers an area of a room or a laboratory. LPN is mainly used to connect all the peripheral devices (disk drives, tape drives, etc.) with the computers located in that room or laboratory. Traditionally, input/output devices are confined to one computer system. However, the use of high speed networking standards (e.g., HIPPI and Fiber Channels) to implement LPNs has enabled the remote access to the input/output devices.

Desktop area networks (DANs)*:* DAN is another interesting concept that aims at replacing the proprietary bus within a computer by a standard network to connect all the components (memory, network adapter, camera, video adapter, sound adapter, etc.) using a standard network. The DAN concept is becoming even more important with the latest development in palm computing devices; the palm computers do not need to have huge amount of memory, sound/video capabilities, all these can be taken from the servers that can be connected to the palm devices using a high speed communication link.

Network Topology

The topology of a computer network can be divided into five types: bus, ring, hub, fully connected and random as shown in Figure 2.2.

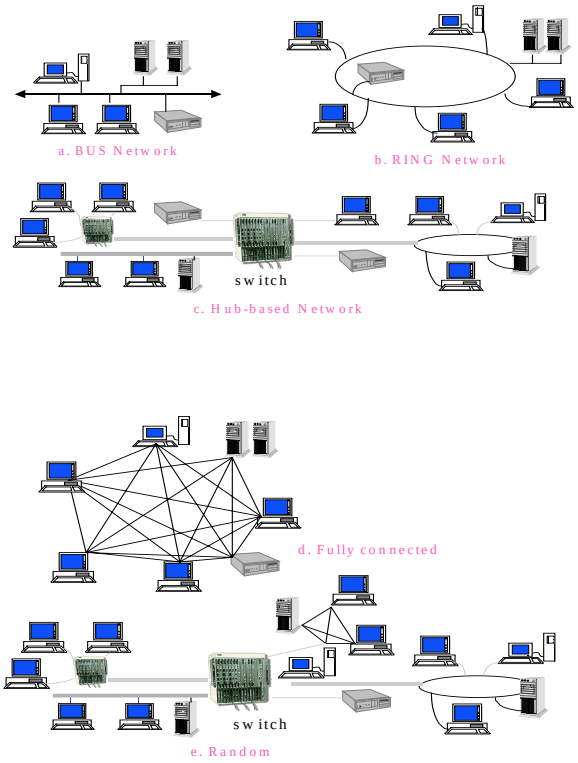


Figure 2.2 Different Types of Network Topologies

In a bus-based network, the bus is time-shared among the computers connected to the bus. The control of the bus is either centralized or distributed. The main limitation of the bus topology is its scalability; when the number of computers sharing the bus becomes large, the contention increases significantly that lead to unacceptable communication delays. In a ring network, the computers are connected using point-to-point communication links that form a closed loop. The main advantages of the ring topology include simplified routing scheme, fast connection setup, a cost proportional to the number of interfaces, and provide high throughput [Weitzman, 1980; Halsall, 1992]. However, the main limitation of the ring topology is its reliability, which can be improved by using double rings. In a hub-based or switched-based network, there is one central routing switch that connects an incoming message on one of its input links to its destination through one of the switch output links. This topology can be made hierarchical where a slave switch can act as a master switch for another cluster and so on. With the rapid deployment of switched-based networks (e.g., Gigabit Ethernet), this topology is expected to play an important role in designing high performance distributed systems. In fully connected network, every computer can reach any other computer in one hob. However, the cost is prohibitively especially when the number of computers to be connected is large. The Random network is a type of network

topology that is a combination of the other types that will lead to an ad-hoc topology.

Network Service

Computer networks can also be classified according to the switching mechanism used to transfer data within the network. These switching mechanisms can be divided into three basic types: Circuit switching, Message switching and Packet switching.

Circuit Switching: Conceptually, circuit switching is similar to the service offered by telephony networks. The communication service is performed in three phases: connection setup, data transmission, and connection release. Circuit-switched connections are reliable and deliver data in the order it was sent. The main advantage of circuit switching is its guaranteed capacity; once the circuit is established, no other network activity is allowed to interfere with the transmission activity and thus can not decrease the capacity of the circuit. However, the disadvantage of Circuit switching is the cost associated with circuit setup and release and the low utilization of network resources.

Message Switching: In a message switching system, the entire message is transmitted along a predetermined path between source and destination computers. The message moves in a store-and-forward manner from one computer to another until it reaches its destination. The message size is not fixed and it could vary from few kilobytes to several megabytes. Consequently, the intermediate communication nodes should have enough storage capacity to store the entire message as being routed to its destination. Message switching could result in long delays when the network traffic is heavy and consist of many long messages. Furthermore, the resource utilization is inefficient and it provides limited flexibility to adjust to fluctuations in network conditions [Weitzman, 1980].

Packet Switching: In this approach, messages are divided into small fixed size pieces, called packets that are multiplexed onto the communications links. A packet, which usually contains only a few hundred bytes of data, is divided into two parts: data and header parts. The header part carries routing and control information that is used to identify the source computer, packet type, and the destination computer; this service is similar to the postal service. Users place mail packages (packets) into the network nodes (mailboxes) that identify the source and the destination of the package. The postal workers then use whatever paths they deem appropriate to deliver the package. The actual path traveled by the package is not guaranteed. Like the postal service, a packet-switched network uses best-effort delivery. Consequently, there is no guarantee that the packet will ever be delivered. Also there are typically several intermediate nodes between the source and the destination that will store and forward the packets. As a result the packets sent from one source may not take the same route to the destination, nor may they be delivered in the same transmission order, and they may be duplicated. The main

advantage of Packet switching is that the communication links are shared by all the network computers and thus improve the utilization of the network resources. The disadvantage is that as network activity increases, each machine sharing the connection receives less of the total connection capacity, which results in a slower communication rate. Furthermore, there is no guarantee that the packets will be received in the same order of their transmission or without any error or duplication.

The main difference between circuit switching and packet switching is that in circuit switching there is no need for intermediate network buffer. Circuit switching provides a fast technique to transmit large data, while packet switching is useful to transmit small data blocks between a random number of geographically dispersed users. Another variation of these services is the virtual circuit switching which combines both packet and circuit switching in its service. The communication is done by first establishing the connection, transferring data, and finally disconnecting the connection. However, during the transmission phase, the data is transferred as small packets with headers that define only the virtual circuit these packets are using. In this service, we have the advantages of both circuit switching and packet switching. In fact, the virtual circuit switching is the service adopted in ATM networks where packets are referred to as cells as will be discussed later in Chapter 3.

2.2.2 Communication Protocols

A protocol is a set of precisely defined rules and conventions for communication between two parties. A communication protocol defines the rules and conventions that will be used by two or more computers on the network to exchange information. In order to manage the complexity of the communication software, a hierarchy of software layers is commonly used for its implementation. Each layer of the hierarchy is responsible for a well-defined set of functions that can be implemented by a specific set of protocols. The Open Systems Interconnection (OSI) reference model, which is proposed by the International Standards Organization (ISO), has seven layers as shown in Figure 2.3. In what follows, we briefly describe the functions of each layer of the OSI reference model from the bottom-up [Jain, 1993].

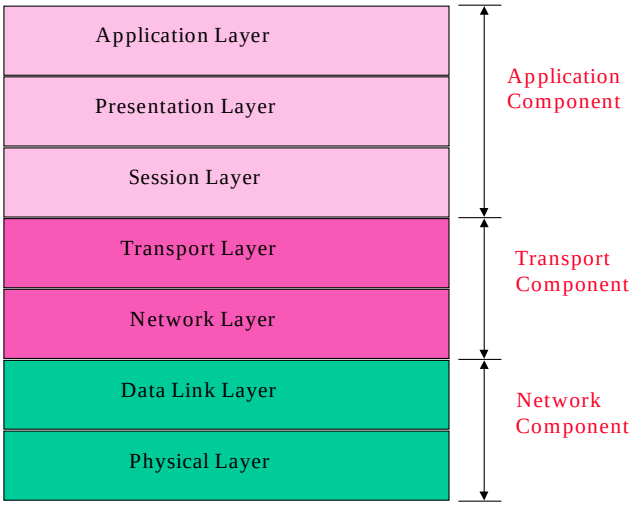


Figure 2.3 The OSI reference model

Physical Layer: It is concerned with transmitting raw bits over a communication channel. Physical layer recognizes only individual bits and cannot recognize characters or multi-character frames. The design issues here largely deal with mechanical, electrical, procedural interfaces and physical transmission medium. The physical layer consists of the hardware that transmits sequences of binary data by analog or digital signaling and using either electric signals, light signals or electro-magnetic signals.

Data Link Layer: It defines the functional and procedural methods to transfer data between two neighboring communication nodes. This layer includes mechanisms to deliver data reliably between two adjacent nodes, group bits into frames and to synchronize the data transfer in order to limit the flow of bits from the physical layer. In local area networks, the data link layer is divided into two sublayers: the medium access control (MAC) sub-layer, which defines how to share the single physical transmission medium among multiple computers and the logical link control (LLC) sub-layer that defines the protocol to be used to achieve error control and flow control. LLC protocols can be either bit or character based protocols. However, most of the networks use bit-oriented protocols [Tanenbaum, 1988].

Network Layer: The network layer addresses the routing scheme to deliver packets from the source to the destination. This routing scheme can be either static (the routing path is determined a priori) or dynamic (the routing path is determined based on network conditions). Furthermore, this layer provides techniques to prevent and remove congestion once it occurs in the network; congestion occurs when some nodes receive more packets than they can process and route. In wide area networks, where the source and destination computers could be interconnected by different types of networks, the network layer is responsible for internetworking; that is converting the packets from one network format to another. In a single local area network with broadcast medium, the network layer is redundant and can be

eliminated since packets can be transmitted from any computer to any other computer by just one hop [Coulouris and Dollimore, 1988]. In general, the network layer provides two types of services to the transport layer: connection-oriented and connectionless services. The connection oriented service uses circuit switching while the connectionless service uses the packet switching technique.

Transport Layer: It is an end-to-end layer that allows two processes running on two remote computers to exchange information. The transport layer provides to the higher-level processes efficient, reliable and cost-effective communication services. These services allow the higher level layers to be developed independent of the underlying network-technology layers. The transport layer has several critical functions related to achieving reliable data delivery to the higher layer such as detecting and correcting erroneous packets, delivering packets in order, and providing a flow control mechanism. Depending on the type of computer network being used, achieving these functions may or may not be trivial. For instance, operating over a packet-switching network with widely varying inter-packet delays presents a challenging task for efficiently delivering ordered data packets to the user; in this network, packets will experience excessive delays that makes decision on the cause of the delay a very difficult task. The delay could be caused by a network failure or by the network being congested. The transport protocol's task is to resolve this issue that could be by itself a time consuming task.

The session, presentation and application layers form the upper three layers in the OSI reference model. In contrast to the lower four layers, which are concerned with providing reliable end-to-end communication, the upper layers are concerned with providing user-oriented services. They take error-free channel provided by the transport layer, and add additional features that are useful to a wide variety of user applications.

Session Layer: It provides mechanisms for organizing and structuring dialogues between application layer processes. For example, the user can select the type of synchronization and control needed for a session such as alternate two-way or simultaneous operations, establishment of major and/or minor synchronization points and techniques for starting data exchange.

Presentation Layer: The main task of this layer focuses on the syntax to be used for representing data; it is not concerned with the semantics of the data. For example, if the two communicating computers use different data representation schemes, this layer task is then to transform data from the formats used in the source computer into a standard data format before transmission. At the destination computer, the received data is transformed from the standard format to the format used by the destination computer. Data compression and encryption for network security are issues of this layer as well [Tanenbaum, 1988; Coulouris and Dollimore, 1988].

Application Layer: This layer supports end-user application processes. This layer contains service elements (protocols) to support application processes such as job

management function, file transfer protocol, mail service, programming language support, virtual terminal, virtual file system, just to name a few.

2.2.3 Network Interfaces

The main function of host-network interface is to transmit data from the host to the network and deliver the data received from the network to the host. Consequently, The host-network interface interacts with upper layer software to perform functions related to message assembly and de-assembly, formatting, routing and error checking. With the advances in processing and memory technology, these communication functions can now be implemented in the hardware of the network interface. A tradeoff is usually made regarding how these functions are going to be distributed between the host and the network interface. The more functions allocated to the network interface, the fewer loads imposed on the host to perform the communication functions; however, the cost of the network interface will increase. The network interface can be a passive device used for temporary storing the received data. In this case, the network interface is under the control of the processor that performs all the necessary functions to transfer the received message to the destination remote process. A more sophisticated network interface can execute most of the communication functions such as assembling complete message packets, passing these packets to the proper buffer, performing flow control, managing the transmit and receive of message packets, interrupting the host when the entire message has been received. In the coming Chapters, we will discuss in more detail the design issues in host network interfaces.

2.3 Distributed System Architectures and Services

The main issues addressed in this layer are related to the system architecture and the functions to be offered by the distributed system. The architecture of a distributed system identifies the main hardware and software components of the system and how they interact with each other to deliver the services provided by the system. In addition to defining the system architecture and how its components interact, this layer defines also the system services and functions that are required to run distributed applications.

The architecture of a distributed system can be described in terms of several architectural models that define the system structure and how the components collaborate and interact with each other. The components of a distributed system must be independent and be able to provide a significant service or function to the system users and applications. In what follows, we describe the architectural models and the system services and functions that should be supported by distributed systems.

2.3.1 Architectural Models

The distributed system architectural models can be broadly grouped into four models: Server Model, Pool Model, Integrated Model, and Hybrid Model [colorois, ohio-os].

Server Model

The majority of distributed systems that have been built so far are based on the *server model* (which is also referred to as the workstation or the client/server model). In this model each user is provided with a workstation to run the application tasks. The need for workstations is primarily driven by the user requirements of a high-quality graphical interface and guaranteed application response time. Furthermore, the server model supports sharing the data between users and applications (e.g., shared file servers and directory servers). The server model consists of workstations distributed across a building or a campus and connected by a local area network (see Figure 2.4). Some of the workstations could be located in offices, and thus be tied to a single user, whereas others may be in public areas where are used by different users. In both cases, at any instant of time, a workstation is either setting idle or has a user logged into it.

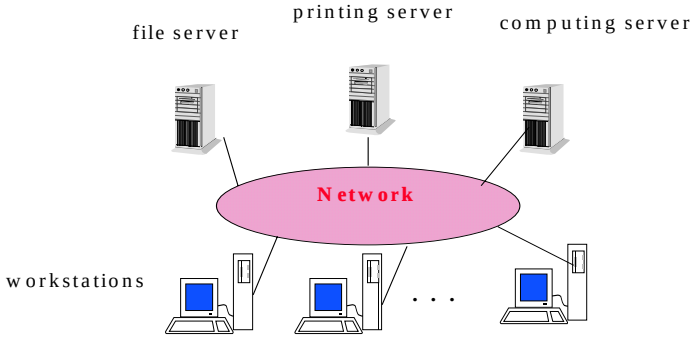


Figure 2.4 Server Model

In this architecture, we do need communication software to enable the applications running on the workstations to access the system servers. The term server refers to application software that is typically running on a fast computer that offers a set of services. Examples of such servers include compute engines, database servers, authentication/authorization servers, gateway servers, or printers. For example, the service offered by an authentication/authorization server is to validate user identities and authorize access to system resources.

In this model, a client sends one or more requests to the server and then waits for a response. Consequently, distributed applications are written as a combination of clients and servers. The programming in this model is called *synchronous programming*. The server can be implemented in two ways: single or concurrent

server. If the server is implemented as a single thread of control, it can support only one request at a time; that is a client request that finds its server busy must wait for all the earlier requests to complete before its request can be processed. To avoid this problem, important servers are typically implemented as concurrent servers; the services are developed using multiple lightweight processes (that we refer to interchangeably as threads) in order to process several requests concurrently. In concurrent server, after the request is received a new thread (child thread) is created to perform the requested service whereas the parent thread keeps listening at the same port for the next service requests. It is important to note that the client machine participates significantly in the computations performed in this model; that is not all the computations are done at the server and the workstations are acting as input/output devices.

Pool Model

An alternative approach to organize distributed system resources is to construct a processor pool. The processor pool can be a rack full of CPUs or a set of computers that are located in a centralized location. The pool resources are dynamically allocated to user processes on demand. In the server model, the processing powers of idle workstations cannot be exploited or used in a straightforward manner. However, in the processor pool model, a user process is allocated CPUs or computing resources as much as needed and when that process is finished, all its computing resources are returned to the pool so other processes can use them. There is no concept of ownership here; all the processors belong equally to every process in the system. Consequently, the processor pool model does not need any additional software to achieve load balancing as it is required in the server model to improve the system utilization and performance, especially when the number of computing resources is large; when the number is large, the probability of finding computers idle or lightly loaded is typically high.

In the pool model, programs are executed on a set of computers managed as a processor service. Users are provided with terminals or low-end workstations that are connected to the processor pool via a computer network as shown in Figure 2.5.

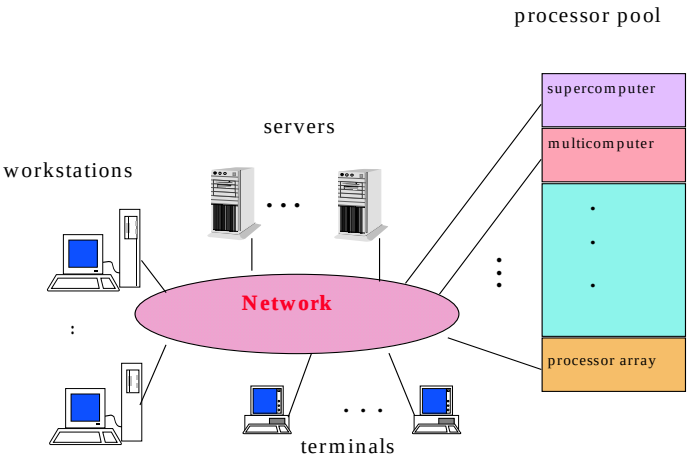


Figure 2.5 Processor Pool Model

The processor pool model provides a better utilization of resources and increased flexibility, when compared to the server model. In addition, programs developed for centralized systems are compatible with this model and can be easily adapted. Finally, processor heterogeneity can be easily incorporated into the processor pool. The main disadvantages of this model are the increased communication between the application program and the terminal, and the limited capabilities provided by the terminals. However, the wide deployment of high speed networks (e.g., Gigabit Ethernet) will make the remote access to the processor pool resources (e.g., supercomputers, high speed specialized servers) is attractive and cost-effective. Furthermore, the introduction of hand-held computers (palm, cellular, etc.) will make this model even more important; we can view the hand-held computers as terminals and most of the computations and the services (e.g., Application Service Provides) are provided by the pool resources.

Integrated Model

The integrated model brings many of the advantages of using networked resources and centralized computing systems to distributed systems by allowing users to access different system resources in a manner similar to that used in a centralized, single-image, multi-user computing system. In this model each computer is provided with appropriate software so it can perform both the server and the client roles. The system software located in each computer is similar to the operating system of a centralized multi-user computing system, with the addition of networking software.

In the integrated model, the set of computing resources forming the distributed system are managed by a single distributed operating system that makes them appear to the user as a single computer system, as shown in Figure 2.6. The individual computers in this model have a high degree of autonomy and run a

complete set of standard software. A global naming scheme that is supported across the distributed system allows individual computers to share data and files without regard to their location. The computing and storage resources required to run user applications or processes are determined at runtime by the distributed operating system such that the system load is balanced and certain system performance requirement is achieved. However, the main limitation of this approach is the requirement that the user processes across the whole system must interact using only one uniform software system (that is the distributed operating system). As a result, this approach requires that the distributed operating system be ported to every type of computer available in this system. Further, existing applications must be modified to support and interoperate with the services offered by the distributed operating system. This requirement limits the scalability of this approach to develop distributed systems with large number of heterogeneous logical and physical resources.

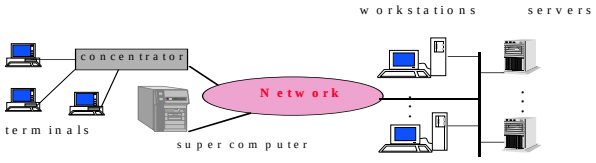


Figure2.6 Integrated Model

Hybrid Model

This model can be viewed as a collection of two or more of the architectural models discussed above. For example, the server and pool models can be used to organize the access and the use of the distributed system resources. The Amoeba system is an example of such a system. In this model, users run interactive applications on their workstations to improve user response time while other applications run on several processors taken from the processor pool. By combining these two models, the hybrid model has several advantages: providing the computing resources needed for a given application, parallel processing of user tasks on pool's processors and the ability to access the system resources from either a terminal or a workstation.

2.3.2 System Level Services

The design of a distributed computing environment can follow two approaches: top-down or bottom up. The first approach is desirable when the functions and the services of a distributed system are well defined. It is typically used when designing special-purposed distributed applications. The second approach is desirable when the system is built using existing computing resources running traditional operating systems. The structure of the existing operating systems (e.g., Unix) is usually designed to support a centralized time-sharing environment and does not support the distributed computing environment. An operating system is the software that provides the functions that allow resources to be shared between tasks, and provides a level of abstraction above the computer hardware that facilitates the

use of the system by user and applications programs. However, the required system-level services are greater in functionality than might normally exist in an operating system. Therefore, a new set of system-wide services must be added on top of the individual operating systems in order to run efficiently distributed applications. Examples of such services include distributed file system, load balancing and scheduling, concurrency control, redundancy management, security service, just to name a few. The distributed file system allows the distributed system users to transparently access and manipulate files regardless of their locations. The load scheduling and balancing involves distributing the loads across the overall system resources such that the overall load of the system is well balanced. The concurrency control allows concurrent access to the distributed system resources as if they were accessed sequentially (serializable access). Redundancy management addresses the consistency and integrity issues of the system resources when some system files or resources are redundantly distributed across the system to improve performance and system availability. The security service involves securing and protecting the distributed system services and operations by providing the proper authentication, authorization, and integrity schemes.
In Part II chapters, we will discuss in detail the design and implementation issues of these services.

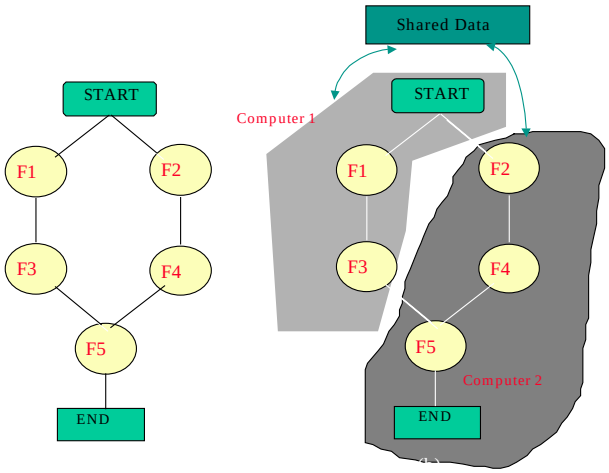
2.4 Distributed Computing Paradigms

In the first layer of the distributed system design model, we address the issues related to designing the communication system, while in the second layer, we address the system architecture and the system services to be supported by a distributed system. In the third layer, we address the programming paradigms and communication models needed to develop parallel and distributed applications. The distributed computing paradigms can be classified according to two models: Computation and Communication models. The computation model defines the programming model to develop parallel and distributed applications while the communication model defines the techniques used by processes or applications to exchange control and data information. The computation model describes the techniques available to the users to decompose and run concurrently the tasks of a given distributed application. In broad terms, there are two computing models: *Data Parallel, and Functional Parallel*. The communication model can be broadly grouped into two types: *Message Passing,* and *Shared Memory*. The underlying communication system can support either one or both of these paradigms. However, supporting one communication paradigm is sufficient to support the other type; a message passing can be implemented using shared memory and vice versa. The type of computing and communication paradigms used determine the type of distributed algorithms that can be used to run efficiently a given distributed application; what is good for a message passing model might not be necessarily good when it is implemented using shared memory model.

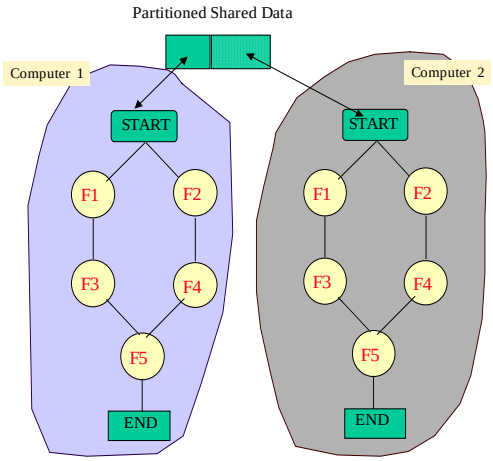
2.4.1 Computation Models

Functional Parallel Model

In this model, the computers involved in a distributed application execute different threads of control or tasks, and interact with each other to exchange information and synchronize the concurrent execution of their tasks. Different terms have been used in the literature to describe this type of parallelism such as control parallelism, and asynchronous parallelism. Figure 2.7(a) shows the task graph of a distributed application with five different functions (F1-F5). If this application is programmed based on the functional parallel model and run on two computers, one can allocate functions F1 and F3 to computer 1 and functions F2, F4 and F5 to computer 2 (see Figure 2.7(b)). In this example, the two computers must synchronize their executions such that computer 2 can execute function F5 only after functions F2 and F4 have been completed and computer 2 has received the partial results from computer 1. In other words, the parallel execution of these functions must be serializable; that is the parallel execution of the distributed application produces identical results to the sequential execution of this application [Casavant, et al, 1996; Quinn, 1994].



(a) (b)



(c)

Figure2.7 (a) A block of a Task with five functions, (b) Functional Parallel Model, and (c) SPMD Data Parallel Model.

Another variation to the functional parallel model is the host-node programming model. In this model, the user writes two programs: the host and node programs. The host program controls and manages the concurrent execution of the application tasks by downloading the node program to each computer as well as the required data. In addition, the host program receives the results from the node program. The node program contains most of the compute-intensive tasks of the application. The number of computers that will run the node program is typically determined at runtime.

In general, parallel and distributed applications developed based on the functional parallel model might lead to race conditions and produce imbalance conditions; this occurs because the task completion depends on many variables such as the task size, type of computer used, memory size available, current load on the communication system, etc. Furthermore, the amount of parallelism that can be supported by this paradigm is limited by the number of functions associated with the application. The performance of a distributed application can be improved by decomposing the application functions into smaller functions; that depends on the type of application and the available computing and communication resources.

Data Parallel Model

In the data parallel model, which is also referred to as synchronous model, the entire data set is partitioned among the computers involved in the execution of a distributed application such that each computer is assigned a subset of the whole data sets [Hillis and Steele, 1986]. In this model, each computer runs the same program but each operates on a different data set, referred to as Single Program Multiple Data (SPMD). Figure 2.7(c) shows how the distributed application shown

in Figure 2.7 (a) can be implemented using data parallel model. In this case, every computer executes the five functions associated with this application, but each computer operates on different data sets.

Data parallel model has been argued favorably by some researchers because it can be used to solve large number of important problems. It has been shown that the majority of real applications can be solved using data parallel model [Fox, Williams and Messina, 1994]. Furthermore, it is easier to develop applications based on data parallel paradigm than those written based on the functional parallel paradigm. In addition, the amount of parallelism that can be exploited in functional parallel model is fixed and is independent of the size of the data sets, whereas in the data parallel model, the data parallelism increases with the size of the data [Hatcher and Quinn, 1991]. Other researchers favored the functional parallel model since all large scale applications can be mapped naturally into functional paradigm.
In summary, we do need to efficiently exploit both the functional and data parallelism in a given large distributed application in order to achieve a high performance distributed computing environment.

2.4.2 Distributed Communications Models

Message Passing Model
The Message Passing model uses a micro-kernel (or a communication library) to pass messages between local and remote processes as well as between processes and the operating system. In this model, messages become the main technique for all interactions between a process and its environment, including other processes. In this model, application developers need to be explicitly involved in writing the communication and synchronization routines required for two remote processes or tasks to interact and collaborate on solving one application. Depending on the relationship between the communicating processes, one can identify two types of message passing paradigms: peer-to-peer message passing, and master-slave message passing. In the peer-to-peer Message Passing, any process can communicate with any process in the system. This type is usually referred to by message passing. In the master-slave type, the communications are only between the master and the slave processes as in the remote procedure call paradigm. In what follows, we briefly describe these two types of message passing.

In the peer-to-peer message passing model, there are two basic communications primitives: SEND and RECEIVE that are available to the users. However, there are many different forms to implement the SEND and RECEIVE primitives. This depends on the required type of communication between the source and destination processes: blocking or non-blocking, synchronous or asynchronous. The main limitations of this model is that the programmers must consider many issues while writing a distributed program such as synchronizing request and response messages, handling data representations especially when heterogeneous computers are involved in the transfer, managing machine addresses, and handling system failures that could be related to communications network or computer failures [Singhal and

Mukesh, 1994]. In addition to all of these, debugging and testing Message Passing programs are difficult because their executions are time-dependent and the asynchronous nature of the system.

The remote procedure calls mechanism has been used to alleviate some of the difficulties encountered in programming parallel and distributed applications. The procedure call mechanism within a program is a well-understood technique to transfer control and data between the calling and called programs. The RPC is an extension of this concept to allow a calling program on one computer to transfer control and data to the called program on another computer. The RPC system hides all the details related to transferring control and data between processes and give them the illusion of calling a local procedure within a program. The remote procedure call model provides a methodology for communication between the client and server parts of a distributed application. In this model, the client requests a service by making what appears to be a procedure call. If the relevant server is remote, the call is translated into a message using the underlying RPC mechanism and then sent over the communication network. The appropriate server receives the request, executes the procedure and returns the result to the client.

Shared Memory Model

In message passing model, the communication between processes is controlled by a protocol and involves explicit cooperation between processes. In Shared memory model, communication is not explicitly controlled and it requires the use of a global shared memory. The two forms of communication models can be compared using the following analogies: message communication resembles the operation of a postal service in sending and receiving mail. A simpler form of message communication can be achieved using a shared mailbox scheme. On the other hand, the shared memory scheme can be compared to a bulletin board, sometimes found in a grocery store or in a supermarket where users post information such as ads for merchandise or help wanted notices. The shared memory acts as a central repository for existing information that can be read or updated by anyone involved.

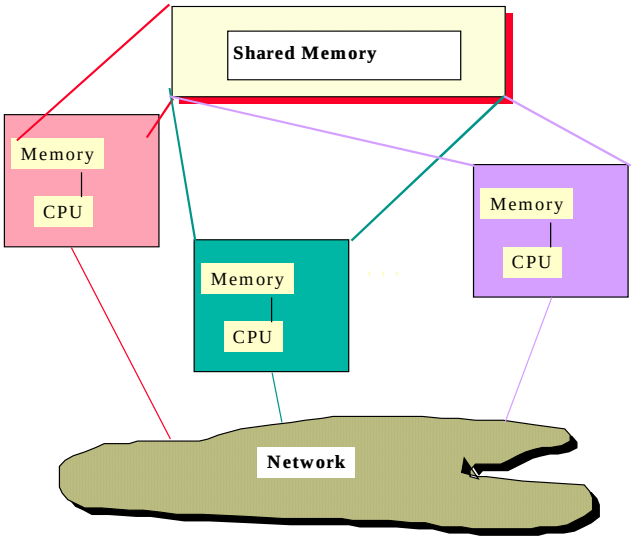


Figure2.8 Distributed Shared Memory Model

Most of distributed applications have been developed based on the message passing model. However, the current advances in networking and software tools have made it possible to implement distributed applications based on shared memory model. In this approach, a global virtual address space is provided such that processes or tasks can use this address space to point to the location where shared data can be stored or retrieved. In this model, application tasks or processes can access shared data by just providing a pointer or an address regardless of the location of where the data is stored. Figure 2.8 shows how a Distributed Shared Memory (DSM) system can be built using the physical memory systems available in each computer.

The advantages of the DSM model include easy to program, easy to transfer complex data structures, no data encapsulation is required as is the case in message passing model, and portability (program written for multiprocessor systems can be ported easily to this environment) [Stumm and Zhou, 1990]. The main differences between message passing and shared memory models can be highlighted as follows: 1) The communication between processes using shared memory model is simpler because the communicated data can be accessed by performing reading operations as if they were local. In the message passing system, a message must be passed from one process to another. Many other issues must be considered in order to transfer efficiently the inter-process messages such as buffer management and allocation, routing scheme, flow control, and error control; and 2) Message passing system is scalable and can support a large number of heterogeneous computers interconnected by a variety of processor interconnect schemes. However, in shared memory model, this approach is not as scalable as the Message Passing model because the complexity of the system increases significantly when the number of computers involved in the distributed shared memory becomes large.

2.5 Summary

Distributed computing systems field is relatively new and as a result there is no general consensus on what constitute a distributed system and how to characterize and design such type of computing systems. In this chapter, we have presented the design issues of distributed systems in a three layer design model: 1) *Network, Protocol, and Interface (NPI)* layer, 2) *System Architecture and Services (SAS)* layer, and 3) *Distributed Computing Paradigms (DCP)* layer. Each layer defines the design issues and technologies that can be used to implement the distributed system components of that layer.

The NPI layer addresses the main issues encountered during the design of the communication system. This layer is decomposed into three sub-layers: Networks, Communication Protocols and Network Interfaces. Each sub-layer denotes one important communication component (subsystem) required to implement the distributed system communication system. The SAS layer represents the designers, developers, and system managers' view of the system. It defines the main components of the system, system structure or architecture, and the system level services required to develop distributed computing applications. Consequently, this layer is decomposed into two sub-layers: architectural models and system level services. The architectural models describe the structure that interconnects the main components of the system and how they perform their functions. These models can be broadly classified into four categories: server model, pool model, integrated model, and hybrid model. The majority of distributed systems that are currently in use or under development are based on the server model (which is also referred to as workstation or client/server model). The distributed system level services could be provided by augmenting the basic functions of an existing operating system. These services should support global system state or knowledge, inter-process communication, distributed file service, concurrency control, redundancy management, load balancing and scheduling, fault tolerance and security.

The Distributed Computing Paradigm (DCP) layer represents the programmer (user) perception of the distributed system. It focuses on the programming models that can be used to develop distributed applications. The design issues of this can be classified into two models: Computation and Communication Models. The computation model describes the mechanisms used to implement the computational tasks associated with a given application. These mechanisms can broadly be described by two models: Functional Parallel, and Data Parallel. The communication models describe how the computational tasks exchange information during the application execution. The communication models can be grouped into two types: Message Passing (MP) and Shared Memory (SM).

2.6 Problems

1. Explain about Distributed System Reference Model.

2. What are the main issues involved in designing a high performance distributed computing system?

3. With the rapid deployment of ATM-based networks, hub-based networks are expected to be widely used. Explain why these networks are attractive.

4. Compare the functions of data link layer with those offered by the transport layers in the ISO OSI reference model.

5. Suppose you wanted to perform the task of finding all the primes in a list of numbers, using a distributed system,

- Develop three distributed algorithms, based on each of the following programming models to sort the prime numbers in the list: (i) funcational, (ii) data, and (iii) remote procedure call.
- Choose one of the algorithms you described in part-1 and show how this algorithm can be implemented using each of the following two communication models: (i) message passing and (ii) shared memory.

6. Compare the distributed system architectural models by showing their advantages and disadvantages. For each architectural model, define the set of applications that are most suitable for that model.

7. You are asked to design a distributed system lab that supports the computing projects and assignments of computer engineering students. Show how the Distributed System Reference Model can be used to design such a system.

References

1. Liebowitz B.H., and Carson, J.H., ``Multiple Processor Systems for Real-Time Applications", Prentice-Hall, 1985.

2. Weitzman, Cay. ``Distributed micro/minicomputer systems: structure, implementation, and application". Englewood Cliffs, N.J.: Prentice-Hall, c1980.

3. Halsall, Fred. ``Data communications, computer networks and open systems". 3rd ed. Addison-wesley 1992.

4. LaPorta, T.F., and Schwartz, M., ``Architectures, Features, and Implementations of High-Speed Transport Protocols", IEEE Network Magazine", May 1991.

5. Mullender, S., Distributed Systems, Second Edition, Addison-Wesley, 1993.

6. Coulouris, G.F., Dollimore, J., Distributed Systems: Concepts and Design, Addison-Wesley, 1988.

7. Hillis, W. D. and Steele, G. Data parallel algorithms, Comm. ACM, 29:1170, 1986.

8. Hatcher, P. J., and Quinn, M. J., Data-Parallel Programming on MIMD Computers. MIT Press, Cambridge, Massachusetts, 1991.

9. Singhal, Mukesh. ``Advanced concepts in operating systems : distributed, database, and multiprocessor operating systems". McGraw-Hill, c1994.

10. IBM, ``Distributed Computing Environment Understanding the Concepts". IBM Corp. 1993.

11. M. Stumm and S. Zhou, "Algorithms Implememting Distributed Shared Memory", Computer; Vol.23, No. 5, May 1990, pp. 54-64.

12. B. Nitzberg and V. Lo, "Distributed Shared Memory: A Survey of Issues and Algorithms", Computer, Aug. 1991, pp. 52-60.

13. K. Li and P. Hudak, "Memory Coherence in Shared Virtual Memory Systems", ACM Trans. Computer Systems, Vol.7, No. 4, Nov. 1989, pp. 321-359.

14. K. Li and R. Schaefer, "A Hypercube Shared Virtual Memory System", 1989 Inter. Conf. on Parallel Processing, pp. 125-132.

15. B. Fleisch and G. Popek, "Mirage : A Coherent Distributed Shared Memory Design", Proc. 14th ACM Symp. Operating System Principles, ACM ,NY 1989, pp. 211-223.

16. J. Bennet, J. Carter, and W. Zwaenepoel, "Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence", Porc. 1990 Conf. Principles and Practice of Parallel Programming, ACM Press, New York, NY 1990, pp. 168-176.

17. U. Ramachandran and M. Y. A. Khalidi, "An Implementation of Distributed Shared Memory", First Workshop Experiences with building Distributed and Multiprocessor Systems, Usenix Assoc., Berkeley, Calif., 1989, pp. 21-38.

18. M. Dubois, C. Scheurich, and F. A. Briggs, "Synchronization, Coherence, and Event Ordering in Multiprocessors", Computer, Vol. 21, No. 2, Feb. 1998, pp. 9-21.

19. J. K. Bennet, "The Design and Implementation of Distributed Smalltalk", Proc. of the Second ACM conf. on Object-Oriented Programming Systems, Languages and Applications, Oct. 1987, pp. 318-330.

20. R. Katz, 5. Eggers, D. Wood, C. L. Perkins, and R. Sheldon, "Implementing a Cache Consistency Protocol", Proc. of the 12th Annu. Inter. Symp. on Computer Architecture, June 1985, pp. 276-283.

21. P. Dasgupta, R. J. LeBlane, M. Ahamad, and U Ramachandran, "The Clouds Distributed Operating System," IEEE Computer, 1991, pp.34-44

22. B. Fleich and G. Popek, "Mirage: A Coherence Distributed Shared Memory Design," Proc. 14th ACM Symp. Operating System Principles, ACM, New York, 1989,pp.21 1-223.

23. D. Lenoskietal, "The Directoiy-Based Cache Coherence Pro to col for the Dash Multiprocessor, "Proc. 17th Int'l Symp. Computer Architecture, IEEE CS Press, Los Alamitos, Calif., Order No. 2047, 1990, pp. 148-159.

24. R. Bisiani and M. Ravishankar, "Plus: A Distributed Shared-Memoiy System," Proc. 17th Int'l Symp. Computer Architecture, WEE CS Press, Los Alamitos,Calif., Order No. 2047,1990, pp.115-124.

25. J. Bennett, J. Carter, And W. Zwaenepoel. "Munin: Distributed Shared Memory based on Type-Specific Memoiy Coherence, "Proc. 1990 Conf Principles and Practice of Parallel Programming, ACM Press, New York, N.Y., 1990, pp.168-176.

26. D. R. Cheriton, "Problem-oriented shared memoiy : a decentralized aproach to distributed systems design ",Proceedings of the 6th Internation Conference on Distributed Computing Systems. May 1986, pp. 190-197.

27. Jose M. Bernabeu Auban , Phillip W. Hutto, M. Yousef A. Khalidi, Mustaque Ahamad, Willian F. Appelbe, Partha Dasgupta, Richard J. Leblanc and Umarkishore Ramachandran, "Clouds--a distributed, object-based operating system: architecture and kernel implication ", European UNIX Systems User Group Autumn Conference, EUUG, October 1988, pp.25-38.

28. Francois Armand, Frederic Herrmann, Michel Gien and Marc Rozier, "Chorus, a new technology for building unix systems", European UNIX systems User Group Autumn Conference, EUUG, October 1988, ppi-18.

29. G. Delp. ``The Architecture and Implementation ofMemnet: A High-speed Shared Memoy Computer Communication Network, doctoral disseration", University of Delaware, Nework, Del., 1988.

30. Zhou et al., "A Heterogeneous Distributed Shared Memory," to be published in IEEE Trans. Parallel and Distributed Systems.

31. Geoffrey C. Fox, Roy D. Williams, and Paul C. Messina. ``Parallel Computing Works!". Morgan Kaufmann, 1994.

32. D.E.Tolmie, A.Tanlawy(ed), “High Performance Networks – Technology and Protocols”, Norwell, Massachusetts, Kluwer Academic Publishers, 1994.

33. W. Stallings, “Advances in Local and Metropolitan Area Networks”, Los Alamitos, California, IEEE Computer Society Press, 1994.

34. B.N.Jain, “Open Systems Interconnection: Its Architecture and Protocols”, New York, McGraw-Hill, 1993.

35. T.L.Casavant, et al (ed), “Parallel Computers: Theory and Practice”, IEEE Computer Society Press, 1996

36. M.J.Quinn, “Parallel Computing: Theory and Practice”, New York, McGraw-Hill, 1994

37. A. S. Tanenbaum, Computer Networks, 2nd Edition, Prentice-Hall, 1988.