MODULE-1: Introduction to Distributed System

VIT | Vidyalankar Institute of Technology
Accredited A+ by NAAC

Prepared by Prof. Amit K. Nerurkar

Certificate

This is to certify that the e-book titled "INTRODUCTION TO DISTRIBUTED SYSTEM" comprises all elementary learning tools for a better understating of the relevant concepts. This e-book is comprehensively compiled as per the predefined eight parameters and guidelines.

Signature Date: 20-03-2020

Prof. Amit K. Nerurkar

Assistant Professor

Department of Computer Engineering

⚠ *DISCLAIMER: The information contained in this e-book is compiled and distributed for educational purposes only. This e-book has been designed to help learners understand relevant concepts with a more dynamic interface. The compiler of this e-book and Vidyalankar Institute of Technology give full and due credit to the authors of the contents, developers and all websites from wherever information has been sourced. We acknowledge our gratitude towards the websites YouTube, Wikipedia, and Google search engine. No commercial benefits are being drawn from this project.*

Module 1 Introduction of Distributed Systems

1.1 DEFINITION, ISSUES, GOALS, TYPES OF DISTRIBUTED SYSTEMS, DISTRIBUTED SYSTEM MODELS, HARDWARE CONCEPTS, SOFTWARE CONCEPT, MODELS OF MIDDLEWARE, SERVICES OFFERED BY MIDDLEWARE, CLIENT SERVER MODEL

DEFINITION OF A DISTRIBUTED SYSTEM

Q.1 Give definition of a Distributed System.

(A) *A distributed system is a collection of independent computers that appears to its users as a single coherent system.*

This definition has two aspects. The first one deals with hardware: the machines are autonomous. The second one deals with software: the users think they are dealing with a single system.

One important characteristic of distributed systems is that differences between the various computers and the ways in which they communicate are hidden from users. The same holds for the internal organization of the distributed system. Another important characteristic is that users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.

Distributed systems should also be relatively easy to expand or scale. This characteristic is a direct consequence of having independent computers, but at the same time, hiding how these computers actually take part in the system as a whole. A distributed system will normally be continuously available, although perhaps certain parts may be temporarily out of order. Users and applications should not notice that parts are being replaced or fixed, or that new parts are added to serve more users or applications.

To support heterogeneous computers and networks while offering a single-system view, distributed systems are often organized by means of a layer of software that is logically placed between a higher-level layer consisting of users and applications, and a layer underneath consisting of operating systems, as shown in Fig. below. Accordingly, such a distributed system is sometimes called **middleware.**

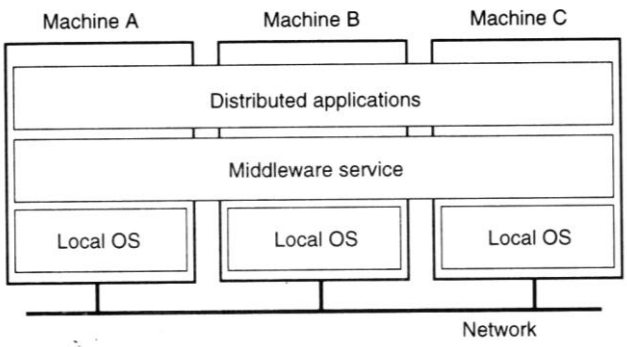


Fig.: A distributed system organized as middleware. Note that the middleware layer extends over multiple machines.

Let us now take a look at several examples of distributed systems. As a first example, consider a network of workstations in a university or company department. In addition to each user's personal workstation, there might be a pool of processors in the machine room that are not assigned to specific users but are allocated dynamically as needed. Such a system might have a single file system, with all files accessible from all machines in the same way and using the same path name. Furthermore, when a user types a command, the system could look for the best place to execute that command, possibly on the user's own workstation, possibly on an idle workstation belonging to someone else, and possibly on one of the unassigned processors in the machine room. If the system as a whole looks and acts like a classical single-processor timesharing system (i.e., multi-user), it qualifies as a distributed system.

GOALS

Q.2 Discuss Goals of Distributed Systems.

(A) Distributed computing systems come into existence in some very natural ways. For example, several applications are inherently distributed in nature and require a distributed computing system for their realization.

Some examples of inherently distributed applications are a computerized worldwide airline reservation system, a computerized banking system in which a customer can deposit / withdraw money from his or her account from any branch of the bank, and a factory automation system controlling robots and machines all along an assembly line.

Information Sharing among Distributed Users

Another reason for the emergence of distributed computing systems was a desire for efficient person-to-person communication facility by sharing information over great distances. In a distributed computing system, information generated by one of the users can be easily and efficiently shared by the users working at other nodes of the system. This facility may be useful in many ways.

Figure: Information Sharing

Resource Sharing

Information is not the only thing that can be shared in a distributed computing system. Sharing of software resources such as software libraries and database as well as hardware resources such as printers, hard disks, and plotters can also be done in a very effective way among all the computers and the users of a single distributed computing system.

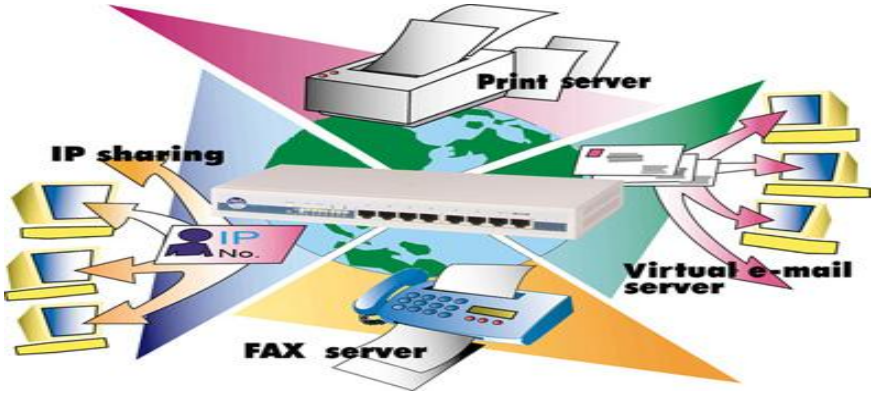


Figure: Resource Sharing

Better Price-Performance Ratio

This is one of the most important reasons for the growing popularity of distributed computing systems. With the rapidly increasing power and reduction in the price of microprocessors, combined with the increasing speed of communication networks, distributed computing systems potentially have a much better price-performance ratio than a single large centralized system.

Another reason for distributed computing systems to be more cost-effective than centralized systems is that they facilitate resource sharing among multiple computers.

Shorter Response Times and Higher Throughput

Due to multiplicity of processors, distributed computing systems are expected to have better performance than single-processor centralized systems. The two most commonly used performance metrics are response time and throughput of user processes. That is, the multiple processors of a distributed computing system can be utilized properly for providing shorter response times and higher throughput than a single-processor centralized system.

Higher Reliability

Reliability refers to the degree of tolerance against errors and component failures in a system.

A reliable system prevents loss of information even in the event of component failures.

An important aspect of reliability is *availability*, which refers to the fraction of time for which a system is available for use. In comparison to a centralized system, a distributed computing system also enjoys the advantage of increased availability. For example, if the processor of a centralized system fails (assuming that it is a single-processor centralized system), the entire system breaks down and no useful work can be performed. However, in the case of a distributed computing system, a few parts of the system can be down without interrupting the jobs of the users who are using the other parts of the system. For example, if a workstation of a distributed computing system that is based on the workstation-server model fails, only the user of that workstation is affected. Others users of the system are not as affected by this failure. Similarly, in a distributed computing system based on the processor-pool model, if some of the processors in the pool are down at any moment, the system can continue to function normally, simply with some loss in performance that is proportional to the number of processors that are down. In this case, none of the users is affected and the users cannot even known that some of the processors are down.

The advantage of higher reliability is an important reason for the use of distributed computing systems for critical applications whose failure may be disastrous. However, often reliability comes at the cost of performance. Therefore, it is necessary to maintain a balance between the two.

Extensibility and Incremental Growth

Another major advantage of distributed computing systems is that they are capable of incremental growth. That is, it is possible to gradually extend the power and functionality of a distributed computing system by simply adding additional resources (both hardware and software) to the system as and when the need arises.

Extensibility is also easier in a distributed computing system because addition of new resources to an existing system can be performed without significant disruption of the normal functioning of the system.

Better Flexibility in Meeting Users' Needs

Different types of computers are usually more suitable for performing different types of computations. For example, computers with ordinary power are suitable for ordinary data processing jobs, whereas high-performance computers are more suitable for complex data.

HARDWARE CONCEPTS

Q.3 Explain Hardware Concepts.

(A) Various classification schemes for multiple CPU computer systems have been proposed over the years, but none of them have really caught on and been widely adopted. For our purposes, we consider only systems built from a collection of independent computers. In Fig. below, we divide all computers into two groups: those that have shared memory, usually called **multiprocessors,** and those that do not, sometimes called **multicomputers.** The essential difference is this: in a multiprocessor, there is a single physical address space that is shared by all CPUs. If any CPU writes, for example, the

value 44 to address 1000, any other CPU subsequently reading from *its* address 1000 will get the value 44. All the machines share the same memory.

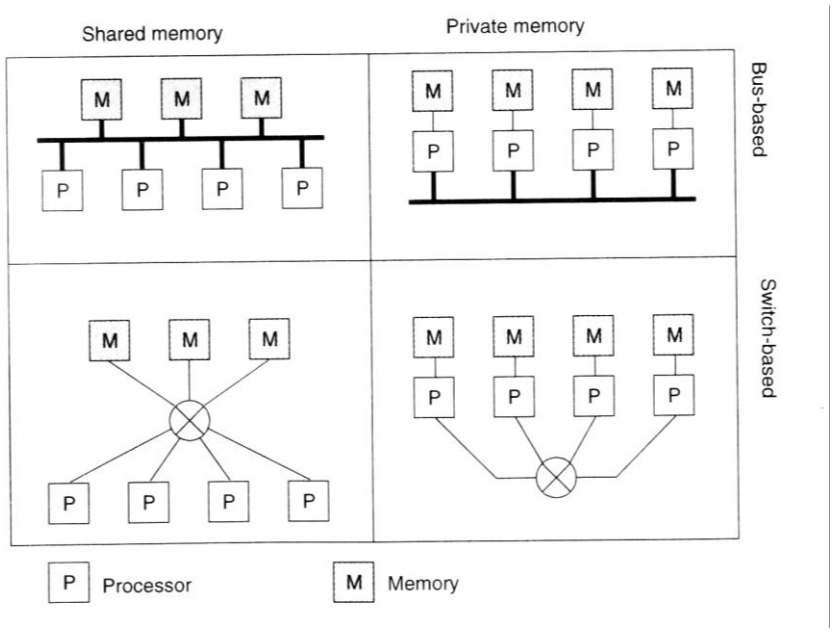


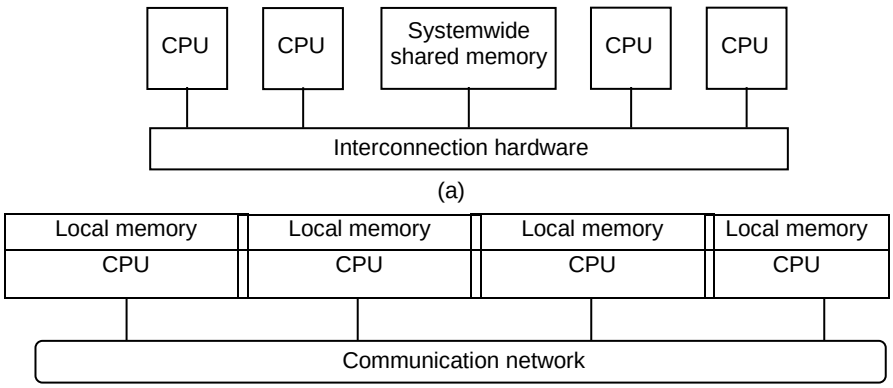
Fig. : Different basic organizations of processors and memories in distributed systems

In contrast, in a multicomputer, every machine has its own private memory. After one CPU writes the value 44 to address 1000. if another CPU reads address 1000 it will get whatever value was there before. The write of 44 does not affect *its* memory at all. A common example of a multicomputer is a collection of personal computers connected by a network.

Computer architectures consisting of interconnected multiple processors are basically of two types:

1. Tightly coupled systems. In these systems, there is a single systemwide primary memory (address space) that is shared by all the processors (a).If any processor writes, for example, the value 100 to the memory location x, any other processor subsequently reading from location x will get the value 100. Therefore, in these systems, any communication between the processors usually takes place through the shared memory.
2. Loosely coupled systems. In these systems, the processors do not share memory and each processor has its own local memory (b). If a processor writes the value 100

to the memory location x, this write operation will only change the contents of its local memory and will not affect the contents of the memory of any other processor. Hence, if another processor reads the memory location x, it will get whatever value was there before in that location of its own local memory. In these systems, all physical communication between the processors is done by passing messages across the network that interconnects the processors.



Difference between tightly coupled and loosely coupled multiprocessor systems: (a) a tightly coupled multiprocessor system, (b) loosely coupled multiprocessor system.

Usually, tightly coupled systems are referred to as *parallel processing systems*, and loosely coupled systems are referred to as *distributed computing systems*, or simply distributed systems.

In contrast to the tightly coupled systems, the processors of distributed computing systems can be located far from each other to cover a wider geographical area. Furthermore, in tightly coupled systems, the number of processors that can be usefully deployed is usually small and limited by the bandwidth of the shared memory. This is not the case with distributed computing systems that are more freely expandable and can have an almost unlimited number of processors.

Definition

In short, a distributed computing system is basically a collection of processors interconnected by a communication network in which each processor has its own local memory and other peripherals, and the communication between any two processors of the system takes place by message passing over the communication network. For a

particular processor, its own resources are local, whereas the other processors and their resources are *remote*. Together, a processor and its resources are usually referred to as a *node* or site or *machine* of the distributed computing system.

VIDEO

SOFTWARE CONCEPTS

Q.4 Explain Software Concepts.

(A) To understand the nature of distributed systems, we will therefore first take a look at operating systems in relation to distributed computers. Operating systems for distributed computers can be roughly divided into two categories: tightly-coupled systems and loosely-coupled systems. In tightly-coupled systems, the operating system essentially tries to maintain a single, global view of the resources it manages. Loosely-coupled systems can be thought of as a collection of computers each running their own operating system. However, these operating systems work together to make their own services and resources available to the others.

This distinction between tightly-coupled and loosely-coupled systems is related to the hardware classification. A tightly-coupled operating system is generally referred to as a **distributed operating** system **(DOS),** and is used for managing multiprocessors and homogeneous multi-computers. Like traditional uniprocessor operating systems, the main goal of a distributed operating system is to hide the intricacies of managing the underlying hardware such that it can be shared by multiple processes.

The loosely-coupled **network operating system** (NOS) is used for heterogeneous multicomputer systems. Although managing the underlying hardware is an important issue for a NOS, the distinction from traditional operating systems comes from the fact local services are made available to remote clients. In the following sections we will first take a look at tightly-coupled and loosely-coupled operating systems.

To actually come to a distributed system, enhancements to the services of network operating systems are needed such that a better support for distribution transparency is

provided. These enhancements lead to what is known as **middleware,** and lie at the heart of modern distributed systems. Middleware is also discussed in this section below summarizes the main issues with respect to DOS, NOS, and middleware.

DOS (Distributed Operating System)

Q.5 What is DOS & NOS?

(A)

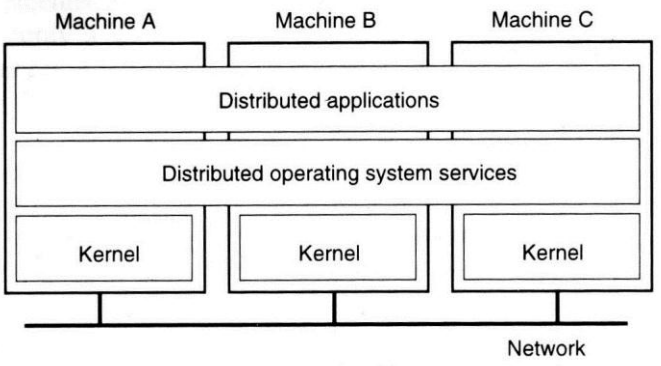


Figure . General structure of a multicomputer operating system.

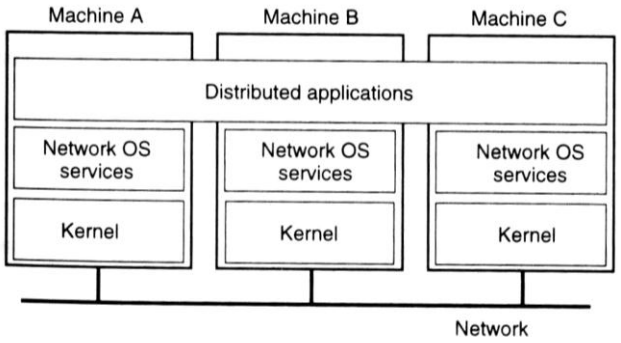


Fig. : General structure of a network operating system

An operating system as a program that controls the resources of a computer system and provides its users with an interface or virtual machine that is more convenient to use than the bare machine. According to this definition, the two primary tasks of an operating systems are as follows:

1. To present users with a virtual machine that is easier to program than the underlying hardware.
2. To manage the various resources of the system. This involves performing such tasks as keeping track of who is using which resource, granting resource requests, accounting for resource usage, and mediating conflicting requests from different programs and users.

The operating systems commonly used for distributed computing systems can be broadly classified into two types – *network operating systems and distributed operating systems*. The three most important features commonly used to differentiate between these two types of operating systems are system image, autonomy, and fault tolerance capability.

1. **System image.** The most important feature used to differentiate between the two types of operating systems is the image of the distributed computing system from the point of view of its users. In case of a network operating system, the users view the distributed computing system as a collection of distinct machine connected by a communication subsystem. That is, the users are aware of the fact that multiple computers are being used. On the other hand, a distributed operating system hides the existence of multiple computers and provides a single-system image to its users. That is, it makes a collection of networked machines act as a *virtual uniprocessor*. The difference between the two types of operating systems based on this feature can be best illustrated with the help of examples. Two such examples are presented below.

   In the case of a network operating system, although a user can run a job on any machine of the distributed computing system, he or she is fully aware of the machine on which his or her job is executed. This is because, by default, a user's job is executed on the machine on which the user is currently logged. If the user wants to execute a job on a different machine, he or she should either log on to that machine by using some kind of "remote login" command or use a special command for remote execution to specify the machine on which the job is to be executed. In either case, the user knows the machine on which the job is executed. On the other hand, a distributed operating system dynamically and automatically allocates jobs to the various machines of the system for processing. Therefore, a user of a distributed operating system generally has no knowledge of the machine on which a job is executed. That is, the selection of a machine for executing a job is entirely manual in the case of network operating systems but is automatic in the case of distributed operating systems.

   With a network operating system, a user is generally required to know the location of a resource to access it, and different sets of system calls have to be used for accessing local and remote resources. On the other hand, users of a distributed operating system need not keep track of the locations of various resources for accessing them, and the same set of system calls is used for accessing both local and remote resources. For instance, users of a network operating system are usually aware of where each of their files is stored and must use explicit *file transfer* commands for moving a file from one machine to another, but the users of a distributed operating system have no knowledge of the location of their files within the system and use the same command to access a file irrespective off whether it is on the local machine or on a remote machine. That is, control over file placement is done manually by the users in a network operating system but automatically by the system in a distributed operating system.

Notice that the key concept behind this feature is “transparency”. A distributed operating system has to support several forms of transparency to achieve the goal of providing a single-system image to its users. Moreover, it is important to note here that with the current state of the art in distributed operating systems, this goal is not fully achievable. Researchers are still working hard to achieve this goal.

2. **Autonomy.** A network operating system is built on a set of existing centralized operating systems and handles the interfacing and coordination of remote operations and communications between these operating systems. That is, in the case of a network operating system, each computer of the distributed computing system has its own local operating system (the operating systems of different computers may be the same or different), and there is essentially no coordination at all among the computers except for the rule that when two processes of different computers communicate with each other, they must use a mutually agreed on communication protocol. Each computer functions independently of other computers in the sense that each one makes independent decisions about the creation and termination of their own processes and management of local resources. Notice that due to the possibility of difference in local operating systems, the system calls for different computers of the same distributed computing system may be different in this case.

   On the other hand, with a distributed operating system, there is a single systemwide operating system and each computer of the distributed computing system runs a part of this global operating system. The distributed operating system tightly interweaves all the computers of the distributed computing system in the sense that they work in close cooperation with each for the efficient and effective utilization of the various resources of the system. That is, processes and several resources are managed globally (some resources are managed locally). Moreover, there is a single set of globally valid system calls available on all computers of the distributed computing system.

   The set of system calls that an operating system supports are implemented by a set of programs called the *kernel* of the operating system. The kernel manages and controls the hardware of the computer system to provide the facilities and resources that are accessed by other programs through system calls. To make the same set of system calls globally valid, with a distributed operating system identical kernels are run on all the computers of a distributed computing system. The kernels of different computers often cooperate with each other in making global decisions, such as finding the most suitable machine for executing a newly created process in the system.

   In short, it can be said that the degree of autonomy of each machine of a distributed computing system that uses a network operating system is considerably high as compared to that of machines of a distributed computing system that uses a distributed operating system.

3. **Fault tolerance capability.** A network operating system provides little or no fault tolerance capability in the sense that if 10% of the machines of the entire distributed computing system are down at any moment, at least 10% of the users are unable to continue with their work. On the other hand, with a distributed operating system, most of the users are normally unaffected by the failed machines and can continue to perform their work normally, with only a 10% loss in performance of the entire distributed computing system. Therefore, the fault tolerance capability of a distributed operating system is usually very high as compared to that of a network operating system.

Q.6 Explain Design issues in distributed systems.

(A) Design issues in distributed systems are:

1. Transparency
2. Reliability
3. Flexibility
4. Performance
5. Scalability
6. Heterogeneity
7. Security
8. Emulation of existing operating systems

1. **Transparency:** A truly transparent distributed O.S permits easy and convenient access to all resources from any site in the system. Thus, the location of a resource or service in the system need not be known to the users.

   The concept of transparency can be applied to several aspects of a distributed system:

   a) **Location transparency:** The users cannot tell where resources are located.
   b) **Migration transparency:** Resources can more at will without changing their names.
   c) **Replication transparency:** The users cannot tell how many copies exists.
   d) **Cancurrency transparency:** Multiple users can share resources automatically.
   e) **Parallelism transparency:** Activities can happen in parallel without users knowing.
   f) **Access transparency:** Hide differences in data representation and how a resource is accessed.
   g) **Relocation transparency:** Hide that a resource may be moved to another location while in use.
   h) **Failure Transparency:** Hide the failures and recovers of resource.
   i) **Persistence transparency:** Hide whether a resource is in memory or on disk.

2. **Heterogeneity:** Most distributed systems as they are used today are built on top of a heterogeneous multicomputer.
   This means that the computers that form part of the system may vary widely with respect to, For example, processor type, memory sizes and I/O bandwidth. In fact, some of the computers may actually be high performance parallel systems, such as multiprocessors or homogeneous multiprocessors.
   Also, the interconnection network may be highly heterogeneous as well.

Another example of heterogeneity is the construction of large-scale multicomputers using existing networks and backbones.

Example – A campus-wide distributed system that is running on top of local area networks from different depts, connected through a high-speed backbone.

3. **Security:** In order that the users can trust the system and rely on it, the various resources of a computer system must be protected against destruction and unauthorized access.

   Therefore, as compared to a centralized system, enforcement of security in a distributed system has the following additional requirements:

   i) It should be possible for the sender of a message to know that the message was received by the intended receiver.
   ii) It should be possible for the receiver of a message to know that the message was sent by the genuine sender.
   iii) It should be possible for both the sender and receiver of a message to be guaranteed that the contents of the msg. were not changed while it was in transfer.

   Cryptography is the only known practical method for dealing with these security aspects of a distributed system.

4. **Reliability:** In general, distributed systems are expected to be more reliable than centralised systems due to the existence of multiple instances of resources.

   A fault is a mechanical or algorithm defect that may generate an error. A fault in a system causes system failure.

   For higher reliability, the fault handling mechanisms of a distributed operating. System must be designed properly to avoid faults, to tolerate faults, and to detect and recover from faults.

   Commonly used methods are:

   i) Fault avoidance
   ii) Fault Tolerance
   iii) Fault detection and Recovery

   i) Fault avoidance: Fault avoidance deals with designing the components of the system in such a way that the occurance of faults is minimized.

   ii) Fault Tolerance: Fault tolerance is the ability of a system to continue functioning in the event of partial system failure.

   To improve the fault tolerance ability of a distributed system. Following are some important concepts:

   a) Redundancy technique b) Distributed control

   a) Redundancy technique: The basic idea behind redundancy techniques is to avoid single points of failure by replicating critical hardware and software components, so that if one of them fails, the others can be used to continue.

b) **Distributed Control:** For better reliability, many of the particular algorithms or protocols used in a distributed operating system must employ a distributed control mechanism to avoid single points of failure.

Example – A highly available distributed file system should have multiple and independent file servers.

Controlling multiple and independent storage devices.

iii) **Fault Detection and Recovery:** The fault detection and recovery method of improving reliability deals with the use of hardware and software mechanisms to determine the occurance of a failure and then to correct the system to a state acceptable for continued operation.

Commonly used techniques for this are:

a) Atomic transactions
b) Stateless servers
c) Acknowledgements and time-based retransmissions of messages.

a) Atomic transactions: An atomic transaction is a computation consisting of a collection of operations that take place indivisibly in the presence of failures and concurrent computations.

b) Stateless servers: In a distributed system, a server may be implemented by using as stateless server or stateful server. These are based on whether or not the history of the serviced requests between a client and a server affects the execution of the next service request.

The stateful approach does depend on the history of the serviced requests, but the stateless approach does not depend on it.

Stateless servers have a distinct advantage over stateful servers in the event of failure i.e. the stateless service paradigm makes crash recovery very easy because no client. State information is maintained by the server.
On the other hand the stateful service paradigm requires complex crash recovery procedures.

c) Acknowledgements and timeout-based retransmissions of messages: Handling of lost messages usually involves return of acknowledgment message and retransmissions on the basis of timeouts i.e. the receiver must return an acknowledgment message for every message received, and if the sender does not receive any acknowledgment for a message within a fixed timeout period, it assumes that the message was lost and retransmits the message.

5. **Flexibility:** Another important issue in the design of distributed operating systems is flexibility. The design of a distributed operating system should be flexible due to the following reasons:
   i) Ease of modification ii) Ease of enhancement

i) **Ease of modification:** Some parts of the design often need to be replaced / modified either because some bug is detected in the design or because the design is no longer suitable for the changed system environment or new-user requirements. Therefore, it should be easy to incorporate changes in the system in a user transparent manner or with minimum interruption caused to the user.

ii) **Ease of Enhancement:** In every system, new functionalities have to be added from time to time to make it more powerful and easy to use. Therefore, it should be easy to add new services to the system.

The most important design factor that influences the flexibility of a distributed OS is the model used for designing its Kernel.

The two commonly used models for kernel design in distributed operating systems are the

- monolithic kernel
- and the microkernel

6. **Performance:** If a distributed system is to be used, its performance must be at least as good as a centralized system.

Some design principle considered useful. For better performance are as follows:

a) Batch if possible.
b) Cache whenever possible.
c) Minimize copying of data
d) Minimize network traffic
e) Take advantage of fine-grain parallelism for multiprocessing.

- Batching often helps in improving performance greately.
- Caching of data at clients, site frequently improves overall system performance.
- Data copying overhead (e.g. moving data in and out of buffers) involves a substantial CPU cost of many operations.
- System performance may also be improved by reducing intermode communication costs.
- Performance can also be improved by taking advantage of fine-grain parallelism for multiprocessing.

7. **Scalability:** Scalability refers to the capability of a system to adopt to increased service load.

Some guideline principles for designing scalable distributed systems are as follows:

i) **Avoid centralized entities:** In a design of a dos, use of centralized entities such as a single central file server or a single database for the entire system makes the distributed system nonscalable. The use of Centralized entities should be avoided in the design.

ii) **Avoid centralized algorithms:** A centralized algorithm is one that operates by collecting information from all nodes, processing this information on a single node and then distributing the results to other nodes. The use of such algorithms in the design of a distributed operating system is also not acceptable from a scalability point of view.

iii) **Perform most operations on client workstations:** If possible, an operation should be performed on the client's own workstation rather than on a server machine. This is because a server is a common resource for several clients and hence server cycles are more precious than the cycles of client workstations. This principle enhances the scalability of the system.

8. **Emulation of Existing Operating system:** For commercial success, it is important that a newly designed distributed operating system be able to emulate existing popular operating systems such as UNIX.

MODELS OF MIDDLEWARE

Q.7 Explain Middleware

(A) Neither a distributed operating system or a network operating system really qualifies as a distributed system. A distributed operating system is not intended to handle a collection of *independent* computers, while a network operating system does not provide a view of a *single coherent system.* The question comes to mind whether it is possible to develop a distributed system that has the best of both worlds: the scalability and openness of network operating systems and the transparency and related ease of use of distributed operating systems. The solution is to be found in an additional layer of software that is used in network operating systems to more or less hide the heterogeneity of the collection of underlying platforms but also to improve distribution transparency. Many modern distributed systems are constructed by means of such an additional layer of what is called **middleware.**

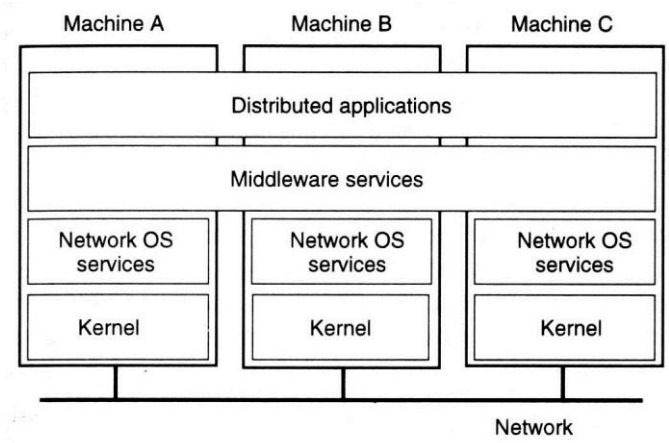


Fig. : General structure of a distributed system as middleware.

Video

Q.8 Compare DOS, NOS & Middleware.

(A) A brief comparison between distributed operating systems, network operating systems, and (middleware-based) distributed systems is given below.

Item	Distributed OS		Network OS	Middleware-based DS
	Multiproc.	Multicomp.		
Degree of transparency	Very high	High	Low	High
Same OS on all nodes?	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

Q.9 Explain Components or Services.

(A)

- Threads package. It provides a simple programming model for building concurrent applications. It includes operations to create and control multiple threads of execution in a single process and to synchronize access to global data within an application.
- Remote Procedure Call (RPC) facility. It provides programmers with a number of powerful tools necessary to build client-server applications. In fact, the DCE RPC facility is the basic for all communication in DCE because the programming model underlying all of DCE is the client-server model. It is easy to use, is network-and protocol-independent, provides secure communication between a client and a server,

and hides differences in data requirements by automatically converting data to the appropriate forms needed by clients and servers.

- Distributed Time Service (DTS). It closely synchronizes the checks of all the computers in the system. It also permits the use of time values from external time sources, such as those of the U.S. National Institute for Standards and Technology (NIST), to synchronize the clocks of the computers in the system with external time. This facility can also be used to synchronize the clocks of the computers of one distributed environment with the clocks of the computers of another distributed environment.
- Name services. The name services of DCE include the Cell Directory Service (CDS), the Global Directory Service (GDS), and the Global Directory Agent (GDA). These services allow resources such as servers, files, devices, and so on, to be uniquely named and accessed in a location-transparent manner.
- Security Service. It provides the tools needed for authentication and authorization to protect system resources against illegitimate access.
- Distributed File Service (DFS). It provides a systemwide file system that has such characteristics as location transparency, high performance, and high availability. A unique feature of DCE DFS is that it can also provide file services to clients of other file systems.

CLIENT SERVER MODELS

Q.10 What is Clients and Servers.

(A) In the basic client-server model, processes in a distributed system are divided into two (possibly overlapping) groups. A server is a process implementing a specific service, for example, a file system service or a database service. A client is a process that requests a service from a server by sending it a request and subsequently waiting for the server's reply. This client-server interaction, also known as request-reply behavior is shown in Fig. below.

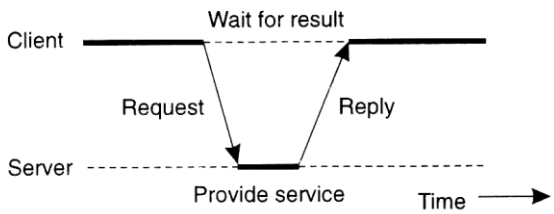


Fig. : General interaction between a client and a server

Communication between a client and a server can be implemented by means of a simple connectionless protocol when the underlying network is fairly reliable as in many local-area networks. In these cases, when a client requests a service, it simply packages a message for the server, identifying the service it wants, along with the necessary input data. The message is then sent to the server. The latter, in turn, will always wait for an

incoming request, subsequently process it, and package the results in a reply message that is then sent to the client.

Many client-server systems use a reliable connection oriented protocol. Although this solution is not entirely appropriate in a local area network due to relatively low performance, it works perfectly fine in wide-area systems in which communication is inherently unreliable. For example, virtually all Internet application protocols are based on reliable TCP/IP connections. In this case, whenever a client requests a service, it first sets up a connection to the server before sending the request. The server generally uses that same connection to send the reply message, after which the connection is torn down. The trouble is that setting up and tearing down a connection is relatively costly, especially when the request and reply messages are small.

When distinguishing only clients and servers, we miss the point that a server may sometimes need to act as a client, as shown in Fig. below, leading to a **(physically) three-tiered architecture.**

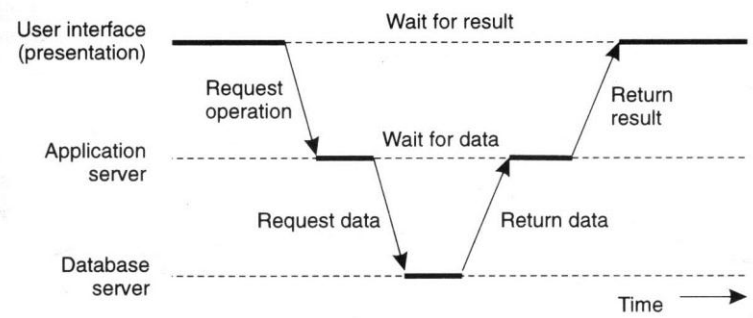


Fig. : An example of a server acting as client

In this architecture, programs that form part of the processing level reside on a separate server, but may additionally be partly distributed across the client and server machines. A typical example of where a three-tiered architecture is used is in transaction processing. In this case, a separate process, called the transaction monitor, coordinates all transactions across possibly different data servers.

Video

DISCUSS THE SIMILARITIES BETWEEN PARALLEL AND DISTRIBUTED SYSTEMS.

Parallel and Distributed Systems:

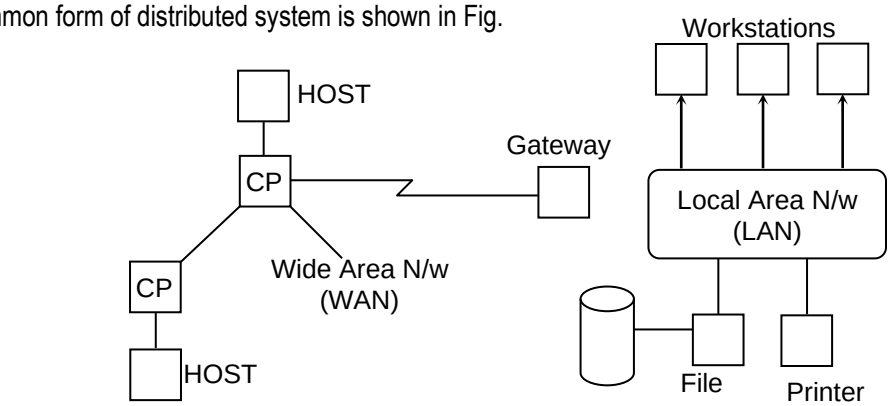
"A distributed system is a collection of independent computers that appears to its users as a single coherent system."

This definition has two aspects:

i) The first one deals with hardware: The machines are autonomous.

ii) The second one deals with software: The users think they are dealing with a single system.

A common form of distributed system is shown in Fig.



CP: Communication processor

Distributed systems are designed to allow many users to work together while the only goal of parallel systems is to achieve maximum speedup on a single problem.

Example– Param Padma

- the fastest Indian supercomputer, operates at a speed of 10,000 G flops and is be used for applications like weather forecasting.

- This distinction is difficult to maintain because the design spectrum is really a continuum.
- Sometimes, the term 'distributed system' is used in the broadest sense to denote any system in which multiple interconnected CPUs work together.
- Tightly-coupled Systems tend to be used more as parallel systems (working on a single problem)
- and loosely-coupled ones tend to be used as distributed systems (working on many unrelated problems)
- although this is not always true.
- The definition of a distributed system excludes multiprocessor systems and vector or array processors since each node is expected to passes some local memory.
  Such systems may be a part of a distributed system, but do not constitute distributed systems by themselves.
- However multiprocessor systems and vector or array processors are included in parallel systems as multiple processors are running in parallel.

- Similarities:

	Parallel Systems		Distributed Systems
i)	Multiple processors run in parallel.	i)	Multiple processors run in parallel.
ii)	Algorithm for mutual exclusion, deadlock handling etc. need to be implemented.	ii)	Algorithm for mutual exclusion, deadlock handling etc. need to be implemented.

- Differences:

	Parallel Systems		Distributed System
i)	The goal of parallel systems is to achieve maximum speedup on a single problem.	i)	The goal of distributed systems is to allow many users to work together.
ii)	These are generally tightly-coupled systems, working on a single problem.	ii)	These are generally loosely coupled systems, working on many unrelated problems.
iii)	Each node need not possess local memory.	iii)	Each node must possess local memory.
iv)	They include multiprocessors and vector or array processors.	iv)	They exclude multiprocessors and vector or array processors.

- Compare and Contrast Centralized and Decentralized Systems

1)	The reason number one for the trend towards distributed system is that, these systems potentially have a much better price / performance ratio than a single large centralized system.
2)	A distributed system may have more total computing power than a mainframe.
3)	By distributing the workload over many machines, if a single m/c crashes the system as a whole still service. Therefore, distributed systems are more reliable than centralized ones.
4)	Another potential advantage of distributed systems over a centralized one is that some applications are inherently distributed.
5)	Computing power can be added in small increments i.e. more processors can be added to the system as per the requirement.

6)	Cost of distributed system is higher than centralized system.
7)	Distributed system uses intelligent terminals whereas centralized system used dumb terminals.

Disadvantages of Distributed System

1. **Software:** Little software exists at present for distributed systems. Currently, there are not much experienced people for designing and implementing distributed software. But as more research is done, this problem will diminish.
2. **Networking:** The second potential problem is due to communication network. It can lose message which requires special software to handle and it can become overloaded when the network saturates, it must either be replaced or a second one must be added. Once the system comes to depend on the network, its loss or saturation can negate most of the advantages the distributed system was built to achieve.
3. **Security:** The sharing of data over distributed system may also give easy access to secret data.

   Despite those potential problems, advantages outweight the disadvantages.
4. **Communication:** Communication which is essential in a distributed system is quite slow which affects the performance of the system.

References

1. http://micronica.com.au/catalog/sharer/
2. https://www.youtube.com/watch?v=mm3r8EG4wLQ&index=6&list=PLmPpJG5-RKf0RUQ7VYjHn6pnoWT2__4CM
3. https://www.youtube.com/watch?v=YY0SvestyaQ
4. https://www.youtube.com/watch?v=rYK-kTBUrK4
5. Distributed Systems by P K Sinha
6. Distributed Operating Systems by Tanenbaum

Quiz

A system in which the components of an information system are distributed to multiple locations in a computer network is known as a:

1. database system
2. networked system
3. distributed system
4. communication system
5. none of the above

Which of the following define the application architecture for an information system?

1. the implementation technology for all software to be developed in-house
2. the technology to be used to implement the user interface
3. the distribution of stored data across a network
4. the degree to which the information system will be centralized or distributed
5. all of the above

Which of the following is NOT an information system application layer?

1. data layer
2. database layer
3. presentation layer
4. application logic layer
5. data manipulation layer

An application system can be mapped into how many different layers?

1. one
2. two
3. three
4. four
5. none of the above

A server that hosts services for e-mail, calendaring, and other work group functionality is known as a(n):

1. groupware server
2. communication server
3. exchange server
4. application server
5. transaction server

GQ

1. What is distributed computing?
2. Explain distributed computing system?
3. Explain tightly coupled system. How are they different from distributed computing system?
4. Define Distributed computing system. What are remote source.
5. Explain Distributed computing systems models with diagram.
6. Explain briefly DC models.
7. List different DC models with examples.
8. Write short note :
   (i) Minicomputer model (ii) Workstation model
   (iii) Workstation-server model (iv) Hybrid model
9. Compare and contrast workstation and workstation-server model.
10. Explain goals of Distributed computing.
11. Why Distributed computing is gaining popularity?
12. Explain :
    (i) Resource sharing (ii) Reliability
    (iii) Flexibility (iv) Autonomy
13. What is OS? Explain DOS.
14. What is NOS and DOS?
15. Compare DOS and NOS as platforms for Distributed Systems.
16. Explain :
    (i) System Image (ii) Autonomy (iii) Fault tolerance capability
17. Explain primary use of OS. How can DCS classified.
18. Explain in brief fault tolerance capability.
19. Explain design issues in D.S.
20. Write short notes on :
    (i) Transparency (ii) Security (iii) Heterogeneity (iv) Performance
21. Explain Distributed Computing Environment (DCE).
22. How was DCE created? What are its components?
23. Explain DCE cells.
24. Compare and contrast between parallel and distributed systems.
25. Explain parallel systems.
26. List and elaborate advantages and disadvantages of Distributed system.
27. Compare and Contrast centralized and decentralized systems.
28. What are the desirable features of an Open Distributed Systems?
29. What are the major issues in designing a distributed operating system?
30. Write short note on DCOM.
31. Explain the reasons why the use and popularity of distributed systems are rapidly increasing, despite the increased complexity and difficulty of building it.