(MR22-1CS0159) DISTRIBUTED OPERATING SYSTEMS

[III Year B Tech CBCS POOL-2 3/0/0/3]

UNIT–I:

Introduction to Distributed Systems, What is a Distributed System? , Hardware concepts, Software concepts, Design issues.

UNIT–II:

Communication in Distributed Systems, Lay red Protocols, ATM networks, The Client – server model, Remote Procedure call, Group communication

UNIT–III:

Synchronization in Distributed System, Clock Synchronization, Mutual Exclusion, Election algorithms, Atomic transactions, Deadlocks in Distributed Systems.

UNIT–IV:

Process and processors in Distributed System threads, System Models, Processors allocation, Scheduling in Distributed System, Fault tolerance, Real-time Distributed System.

Distributed File Systems, Distributed File System Design, Distributed File System implementation, Trends in Distributed File System.

UNIT–V:

Distributed Shared Memory, Introduction, What is Shared memory? Consistency models, Page based Distributed Shared memory, Shared – variable Distributed Shared memory, Object based Distributed Shared Memory

UNIT–I:

Introduction to Distributed Systems, What is a Distributed System? , Hardware concepts, Software concepts, Design issues.

Introduction to the course:

Operating Systems History

Uniprocessor Operating Systems Generally speaking, the roles of operating systems are

(1) resource management (CPU, memory, I/O devices) and

(2) to provide a virtual interface that is easier to use than hardware to end users and other applications. For example, when saving a file we do not need to know what block on the hard drive we want to save the file.

The operating system will take care of where to store it. In other words, we do not need to know the low-level complexity.

Uniprocessor operating systems are operating systems that manage computers with only one processor/core. The structure of uniprocessor operating systems include

1. Monolithic model. In this model one large kernel is used to handle everything. The examples of this model include MS-DOS and early UNIX.

2. Layered design. In this model the functionality is decomposed into N layers. Each layer can only interact with with layer that is directly above/below it.

3. Microkernel architecture. In this model the kernel is very small and only provides very basic services: inter-process communication and security. All other additional functionalities are implemented as standard processes in user-space

Distributed Systems History

Minicomputer model: In this model, each user has its local machine. The machines are interconnected, but the connection may be transient (e.g., dialing over a telephone network). All the processing is done locally but you can fetch remote data like files or databases.

Workstation model: In this model, you have local area networks (LANs) that provide a connection nearly all of the time. An example of this model is the Sprite operating system. You can submit a job to your local workstation. If your workstation is busy, Sprite will automatically

transmit the job to another idle workstation to execute the job and return the results. This is an early example of resource sharing where processing power on idle machines are shared.

Client-server model: This model evolved from the workstation model. In this model there are powerful workstations who serve as dedicated servers while the clients are less powerful and rely on the servers to do their jobs.

Processor pool model: In this model the clients become even less powerful (thin clients). The server is a pool of interconnected processors. The thin clients basically rely on the server by sending almost all their tasks to the server.

Cluster computing systems / Data centers: In this model the server is a cluster of servers connected over high-speed LAN.

Grid computing systems: This model is similar to cluster computing systems except for that the server are now distributed in location and are connected over wide area network (WAN) instead of LAN.

WAN-based clusters / distributed data centers: Similar to grid computing systems but now it is clusters/data centers rather than individual servers that are interconnected over WAN. Virtualization

Cloud computing: Infrastructures are managed by cloud providers. Users only lease resources on demand and are billed on a pay-as-you-go model.

Distributed Operating System

Distributed operating systems are operating systems that manage resources in a distributed system. However, from a user perspective a distributed OS will look no different from a centralized OS because all of the details about distribution are automatically handled by the OS and are transparent to the user.

The definition of distributed systems deals with two aspects that:

¬ Deals with hardware: The machines linked in a distributed system are autonomous.

¬ Deals with software: A distributed system gives an impression to the users that they are dealing with a single system.

Features of Distributed Systems:

- Communication is hidden from users
- Applications interact in uniform and consistent way

Introduction of Distributed System

- High degree of scalability
- A distributed system is functionally equivalent to the systems of which it is composed.
- Resource sharing is possible in distributed systems.
- Distributed systems act as fault tolerant systems
- Enhanced performance Issues in distributed systems
- Concurrency
- Distributed system function in a heterogeneous environment. So adaptability is a major issue.
- Latency
- Memory considerations: The distributed systems work on both local and shared memory.
- Synchronization issues
- Applications must need to adapt gracefully without affecting other parts of the systems in case of failures.
- Since they are widespread, security is a major issue.
- Limits imposed on scalability
- They are less transparent.
- Knowledge about the dynamic network topology is a must.

QOS parameters The distributed systems must offer the following QOS:

- Performance
- Reliability
- Availability
- Security

Differences between centralized and distributed systems

Centralized Systems	Distributed Systems
In Centralized Systems, several jobs are done on a particular central processing unit(CPU)	In Distributed Systems, jobs are distributed among several processors. The Processor are interconnected by a computer network
They have shared memory and shared variables.	They have no global state (i.e.) no shared memory and no shared variables.
Clocking is present.	No global clock.

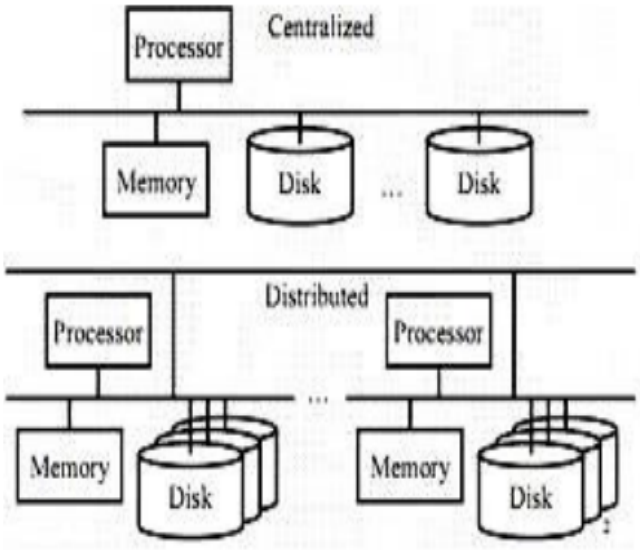


Fig 1.1: Distributed and Centralized systems

There are essentially three flavors of distributed OS's:

Distributed operating system (DOS),

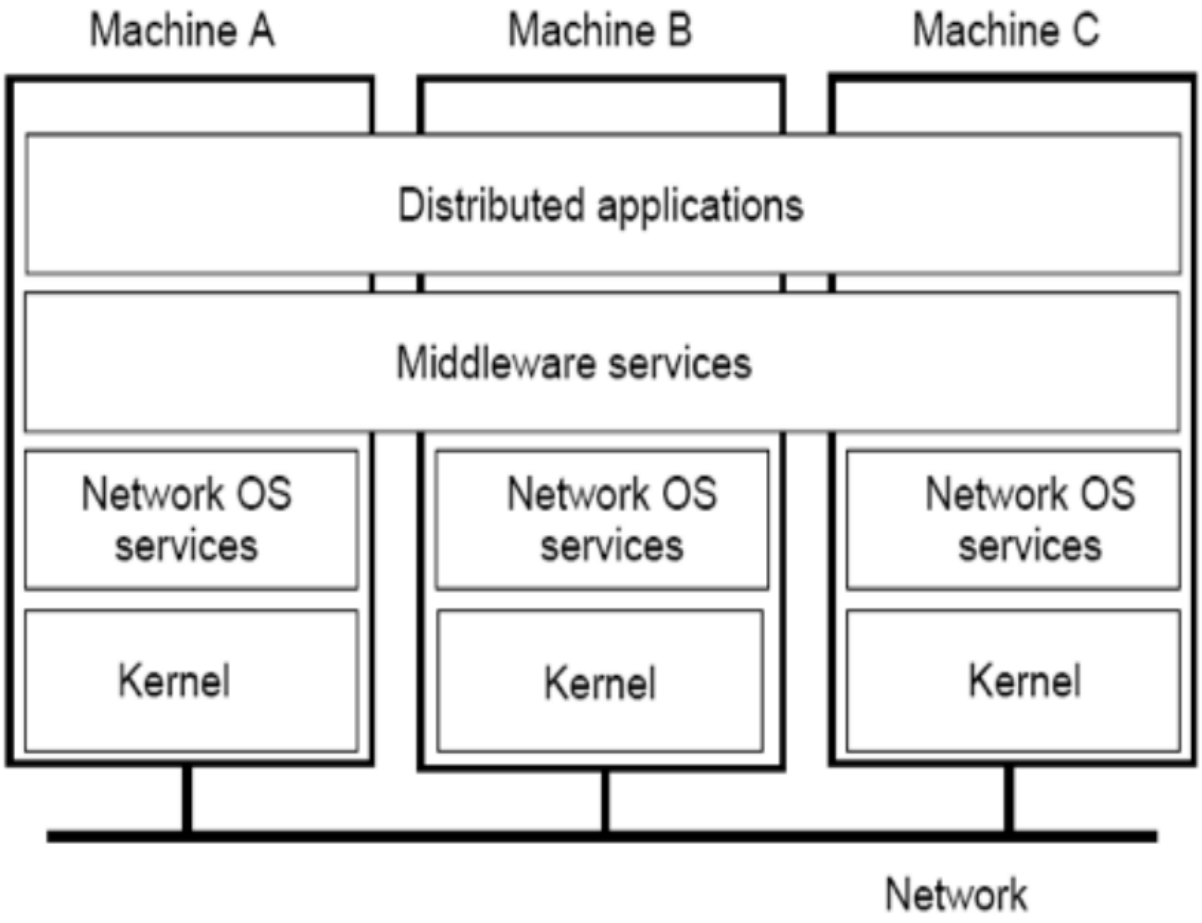
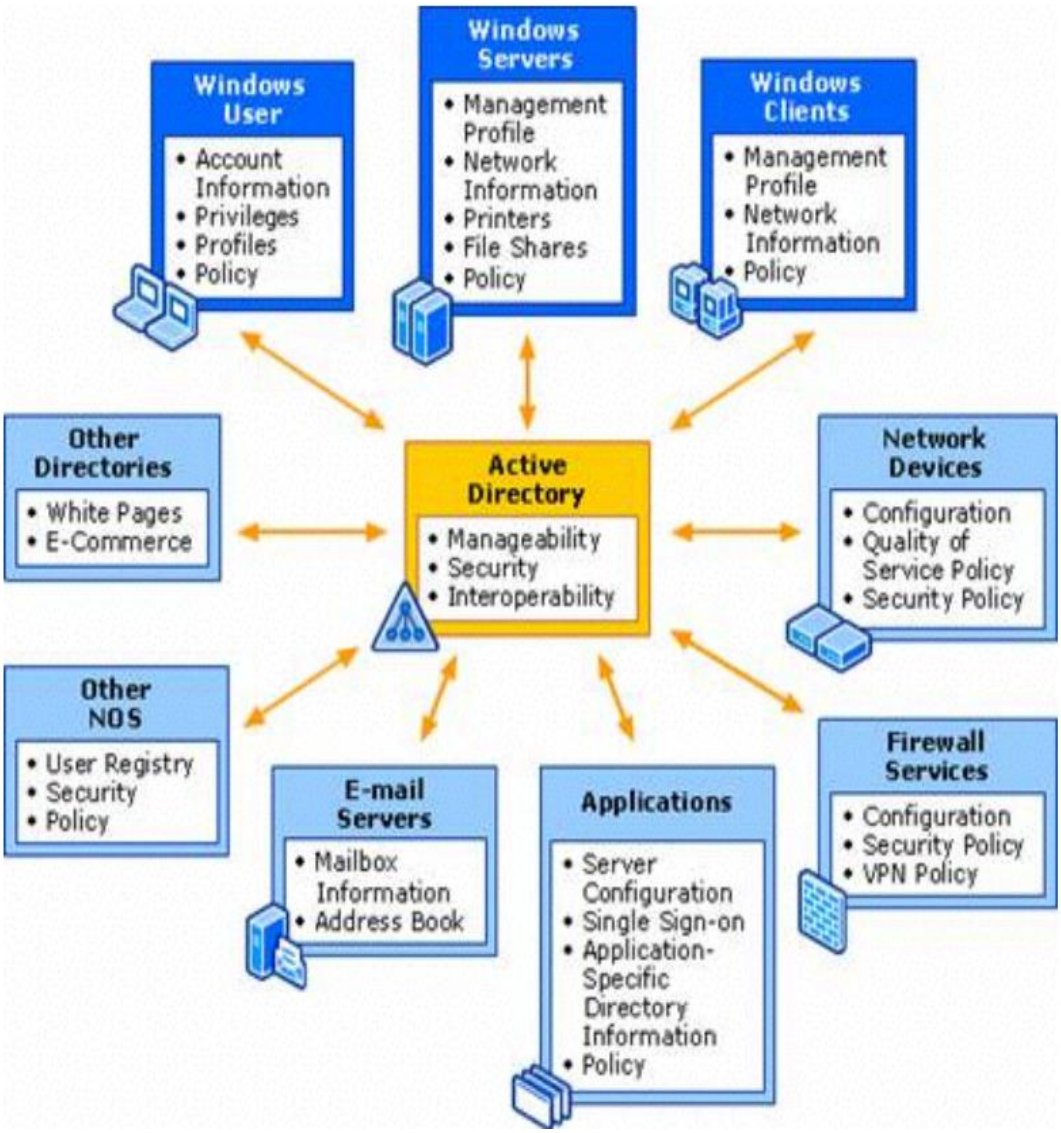


Fig. 1-22. General structure of a distributed system as middleware.

Networked operating system (NOS),



And

Middleware.

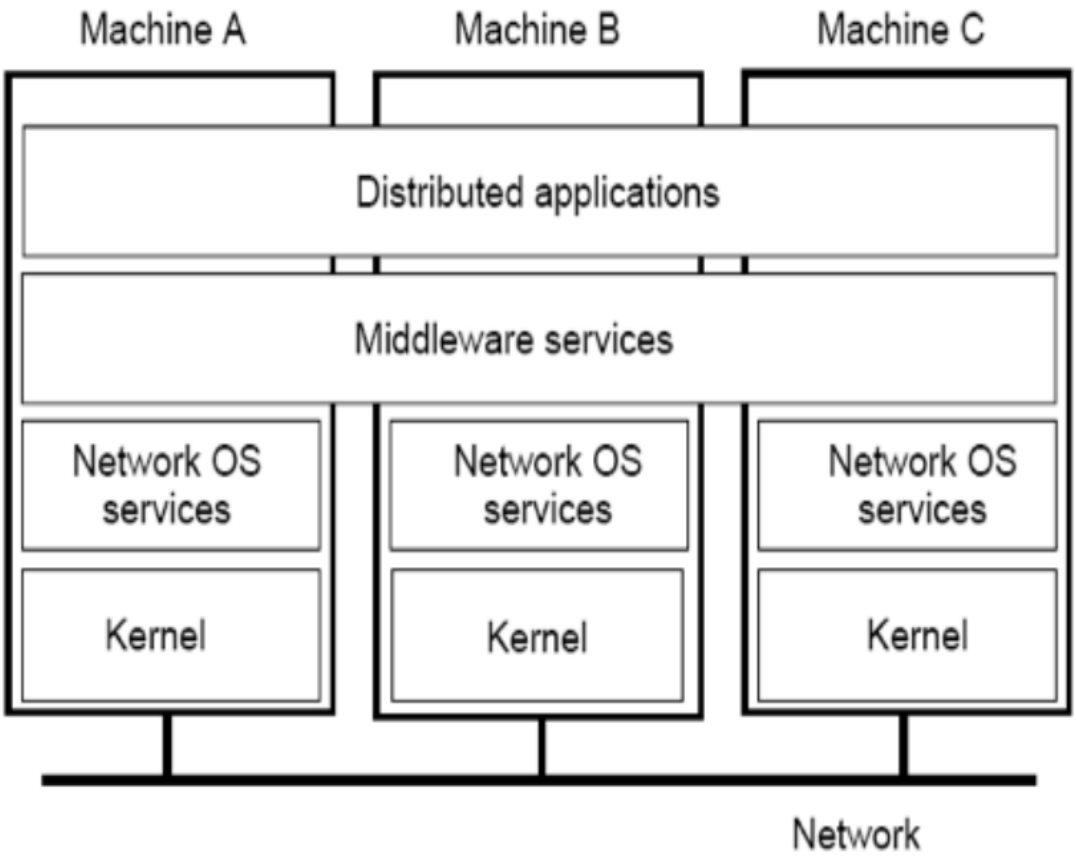


Fig. 1-22. General structure of a distributed system as middleware.

DOS provides the highest level of transparency and the tightest form of integration. In a distributed system managed by DOS, everything operates above the DOS kernel will see the system as a single logical machine.

In NOS you are still allowed to manage loosely-coupled multiple machines but it does not necessarily hide anything from user. Middleware takes a NOS and adds a software layer on top to make it behave or look like a DOS.

Architectural Styles

Most distributed systems can be described by one of the architectures discussed in this lecture. It is important to understand the differences between them so that we can decide on the architecture before implementing a new system.

1.Layered Architectures

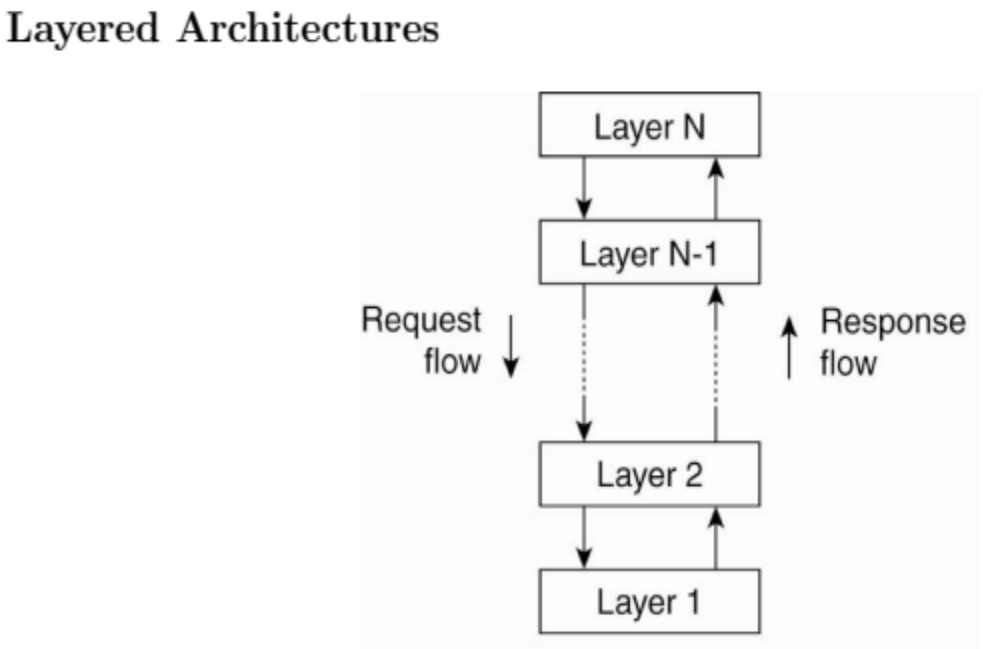


Figure 2.2: Layered Design

Layered Design A layered architecture looks like a stack, as seen in the figure above. The system is partitioned into a sequence of layers and each layer can communicate to the layer above or below.

For example, layer i can communicate with layer i+1 and layer i−1 but not the others (e.g. layer i+2). This is the main restriction of a layered design.

The layered architecture is especially common in web applications where this architecture is divided across the client and the server. Common instances of these systems are multitiered architectures and network stack

2.Object-Based Style :

Object-Based Style

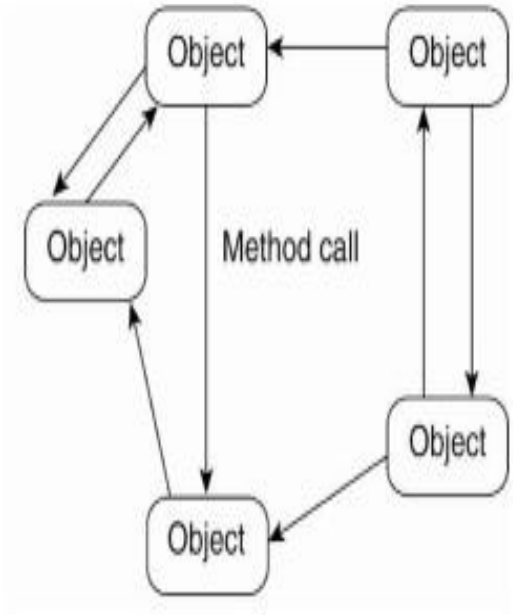


Figure 2.3: Object-based Style

Object-based Style In this architecture, each component corresponds to an object. Unlike in standard OOP programming, objects can be distributed across multiple machines.

As shown in the figure above, the system can have many objects. Each object have has its own states and exposes its own interface which other objects can use.

All objects can communicate with any other object without restriction, making this a “generalized” version of the layered design. Components interact with each other via remote procedure calls.

3.Event-Based Architecture :

Event-Based Architecture

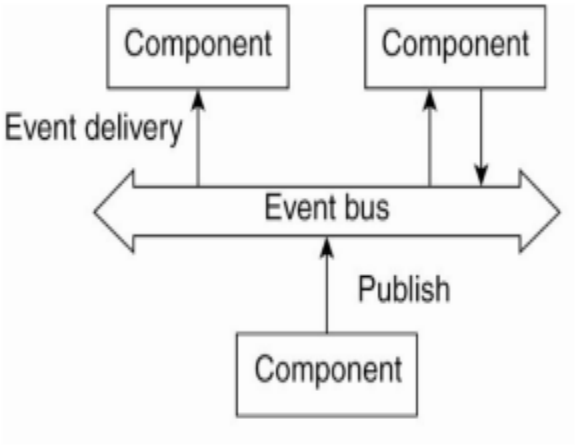


Figure 2.4: Event-based Architecture

Event-based Architecture An event-based architecture has many components that communicate using a publisher-subscriber (pubsub) model via an event bus instead of through direct communication. In this architecture, a component that sends an event to the event bus is a publisher, and a component

asynchronously. After a component sends information by publishing an event, the event bus then checks for subscriptions matching the recipient information enclosed in the newly published event.

If one or more matching subscriptions is found, the event bus will deliver the data to the appropriate component(s).

A component posts information and some component may come along later and retrieve the information. Unlike in the event-based architecture, data posted in the shared data space have no specific information about the recipient.

Therefore, posted data can be in the shared data space for a while until some component actively retrieve this data. From this sense, the components in the data-space architecture are loosely coupled in space and time.

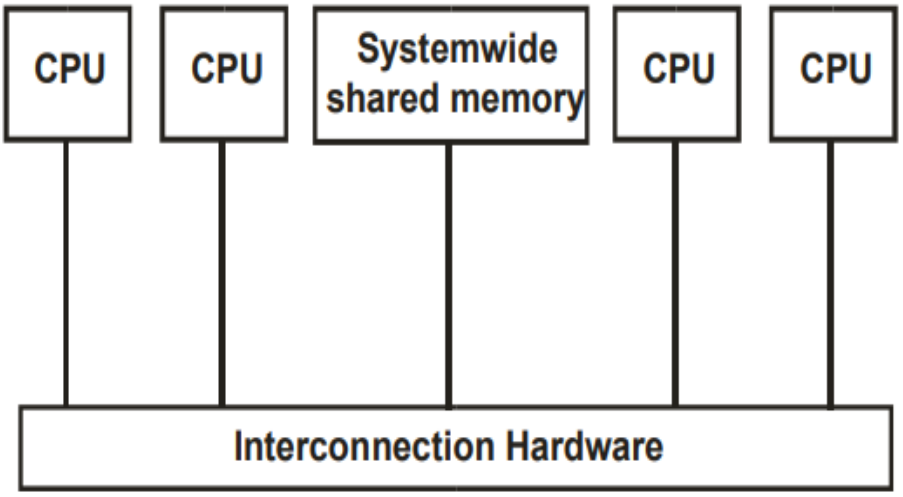
Computer architectures consisting of interconnected, multiple processors are basically of two types:

1. **Tightly coupled systems:**

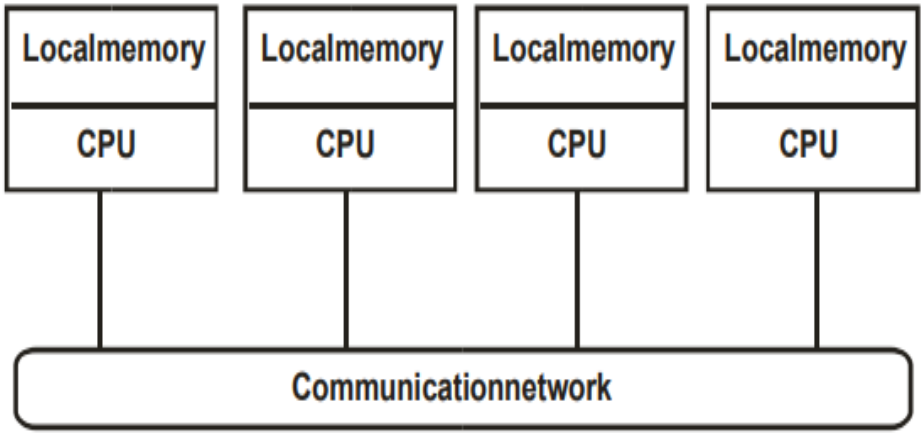
   In these systems, there is a single system wide primary memory (address space) that is shared by all the processors [Fig. 1.1(a)]. If any processor writes, for example, the value 100 to the memory location x, any other processor subsequently reading from location x will get the value 100. Therefore, 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 [Fig. 1.1(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. In these systems, all physical communication between the processors is done by passing messages across the network that interconnects the processors.



(a)



(b)

Difference between tightly coupled and loosely coupled multiprocessor systems

(a) tightly coupled multiprocessor system;
(b) a loosely coupled multiprocessor system

• 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 processor 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.

• 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.

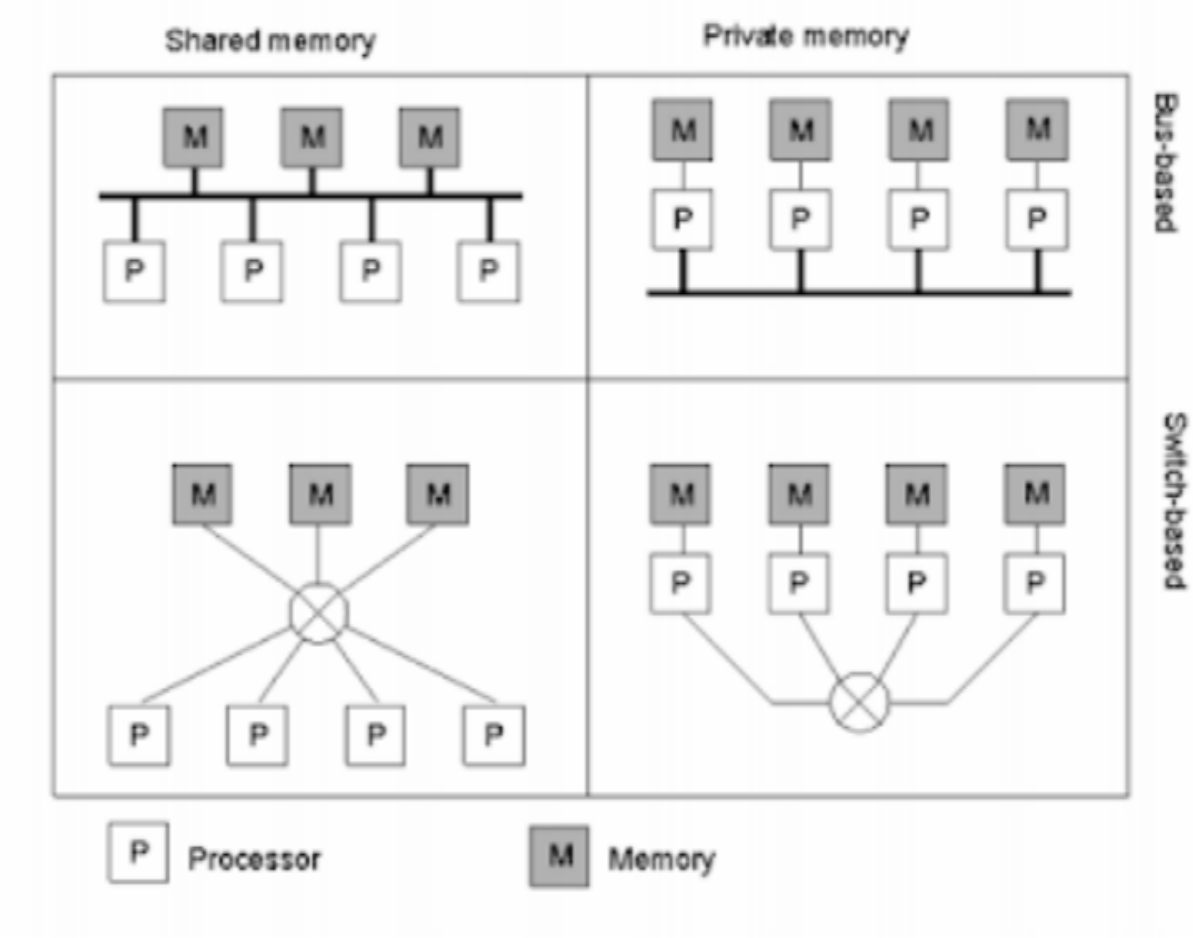
– Hardware and Software Concepts

Hardware concepts :-

Hardware in Distributed Systems can be organized in several different ways:

♣ Shared Memory (Multiprocessors , which have a single address space).

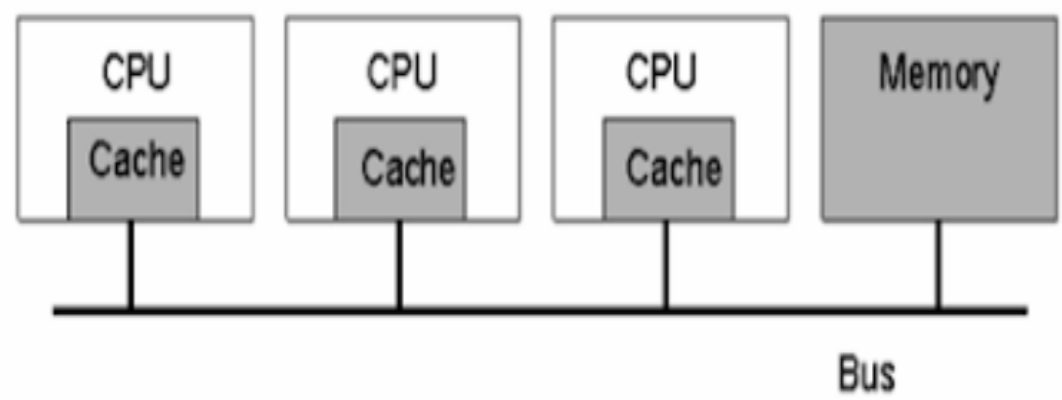
♣ Private Memory (Multicomputers, each CPU has a direct connection to its local memory)



Multiprocessors – Bus Based

♣ Have limited scalability

♣ Cache Memory help avoid bus overloading.



Multiprocessors – Switch Based

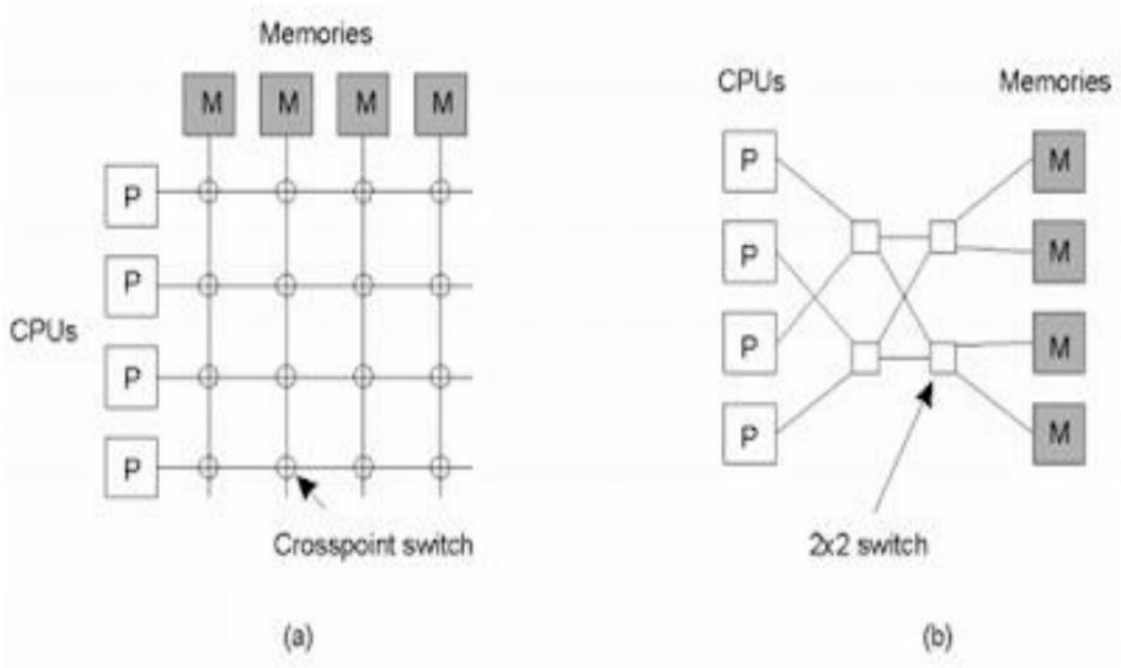
• Different CPUs can access different memories simultaneously

• The number of switches limits the number of CPUs that can access memory simultaneously

a) A crossbar switch

b) An omega switching network

Multiprocessors – Switch Based



Multicomputers :-

Homogeneous:

• All CPUs and memory are identical;
• Connected through a broadcast shared multi access network (like Ethernet) in bus based systems;
• Messages routed through an interconnectionn network in switch-based multicomputers (e.g., grids, hipercubes…). Heterogeneous:
• The most usual topology;
• Computers may vary widely with respect to processor type, memory size, I/O bandwidth;
• Connections are also diverse (a single multicomputer can simultaneously use LANs, Wide Area ATM, and frame relay networks);
• Sophisticated software is needed to build applications due to the inherent heterogeneity; Examples: SETI@home, WWW…

Software Concepts :-

- There are three largely used operating system types:
- a) Distributed operating system
- b) Network operating system
- c) Middleware operating system
- Distributed operating system:

In distributed OS, a common set of services is shared among multiple processors in such a way that they are meant to execute a distributed application effectively and also provide services to separate independent computers connected in a network as shown in fig below

- It communicates with all the computer using message passing interface(MPI).
- It follows the tightly coupled architecture pattern.
- It uses Data structure like queue to manages the messages and avoid message loss between sender and receiver computer.
- Eg Automated banking system, railway reservation system etc.

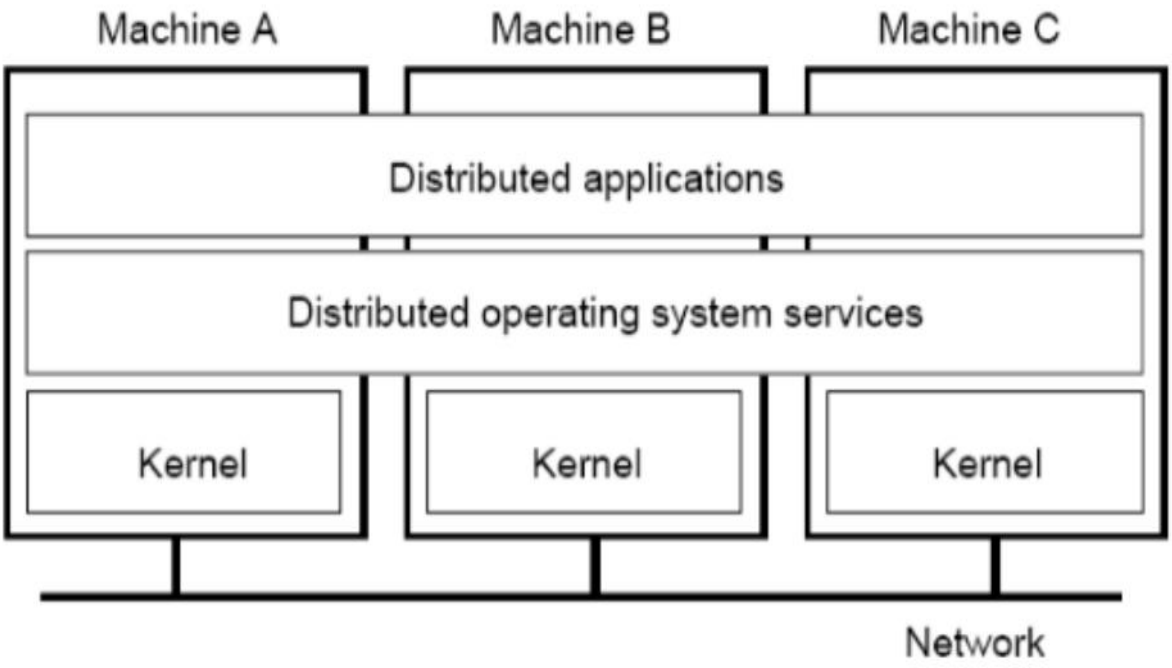


Fig. General structure of a multicomputer operating system

Network operating system:

⎫ It is specifically designed for hetrogeneous multicomputer system, where multiple hardware and network platforms are supported.

⎫ It has multiple operating system running on different hardware platforms connected in network.
⎫ It provides to each computer connected in network.

⎫ It follows the loosely coupled architecture pattern which allow user to use services provided by the local machine itself as shown in fig below.

⎫ Eg Remote login where user workstation is used to log in to the remoter server and execute remote commands over the network.

⎫ Eg Centralized file storage system

Design Issues of Distributed System

A distributed System is a collection of autonomous computer systems that are physically separated but are connected by a centralized computer network that is equipped with distributed system software. These are used in numerous applications, such as online gaming, web applications, and cloud computing. However, creating a distributed system is not simple, and there are a number of design considerations to take into account. The following are some of the major design issues of distributed systems.

Table of Content

Design Issues of Distributed System

Certainly! Here is a detailed breakdown of the key design issues of distributed systems:

1. Scalability

- **Challenges**
  - **Handling Increased Load:** As the number of users or requests increases, the system must scale accordingly without performance degradation.
  - **Geographic Distribution:** Ensuring performance across geographically dispersed locations.
- **Strategies to Achieve Scalability**
  - **Horizontal Scaling:** Adding more nodes to the system.
  - **Vertical Scaling:** Enhancing the capacity of existing nodes.
  - **Sharding:** Dividing the database into smaller, more manageable pieces.

2. Reliability

- **Fault Tolerance**
  - **Redundancy:** Using duplicate components to take over in case of failure.

    - **Failover Mechanisms:** Automatically switching to a standby system when the primary system fails.
- **Redundancy and Replication**
    - **Data Replication:** Storing copies of data on multiple nodes to ensure availability and reliability.
    - **Consensus Algorithms:** Ensuring consistency among replicated data (e.g., Paxos, Raft).

3. Availability

- **Uptime and Downtime Considerations**
    - **High Availability Architectures:** Designing systems to minimize downtime.
    - **Monitoring and Alerting:** Using tools to detect and respond to issues promptly.
- **Techniques to Improve Availability**
    - **Load Balancers:** Distributing incoming requests across multiple servers.
    - **Geographic Redundancy:** Deploying servers in different geographic locations to avoid single points of failure.

4. Consistency

- **Data Consistency Models**
    - **Strong Consistency:** Ensuring that all nodes see the same data at the same time.
    - **Eventual Consistency:** Allowing for temporary discrepancies between nodes, with eventual convergence.
- **Trade-offs between Consistency and Availability (CAP Theorem)**
    - **CAP Theorem:** Understanding the trade-off between Consistency, Availability, and Partition Tolerance.

5. Latency

- **Sources of Latency**
    - **Network Delays:** Time taken for data to travel across the network.
    - **Processing Delays:** Time taken for nodes to process requests.
- **Minimization Techniques**
    - **Caching:** Storing frequently accessed data closer to the user.
    - **Data Compression:** Reducing the amount of data that needs to be transferred.

6. Load Balancing

- **Load Distribution Methods**
    - **Round Robin:** Distributing requests evenly across servers.
    - **Least Connections:** Directing traffic to the server with the fewest active connections.
- **Dynamic vs. Static Load Balancing**
    - **Dynamic:** Adapting to changing loads in real-time.
    - **Static:** Using predetermined load distribution strategies.

7. Security

- **Authentication and Authorization**
    - **Identity Verification:** Ensuring that users are who they claim to be.
    - **Access Control:** Restricting access to resources based on user roles.
- **Data Encryption and Secure Communication**
    - **Encryption:** Protecting data in transit and at rest.

    - **Secure Protocols:** Using HTTPS, SSL/TLS for secure communications.

8. Architectural Design Patterns

- **Client-Server Model**
    - **Centralized Servers:** Handling requests from multiple clients.
- **Peer-to-Peer Model**
    - **Decentralized Network:** Nodes act as both clients and servers.
- **Microservices Architecture**
    - **Service Decomposition:** Breaking down applications into smaller, independent services.
- **Service-Oriented Architecture (SOA)**
    - **Service Reusability:** Designing services to be reused across different applications.

9. Communication Issues

- **Network Protocols**
    - **TCP/IP:** Ensuring reliable, ordered, and error-checked delivery of data.
    - **UDP:** Providing faster, connectionless communication.
- **Message Passing vs. Shared Memory**
    - **Message Passing:** Communicating by sending messages between nodes.
    - **Shared Memory:** Direct access to a common memory space.
- **Synchronous vs. Asynchronous Communication**
    - Synchronous: Blocking operations until a response is received.
    - Asynchronous: Allowing operations to proceed without waiting for a response.

10. Data Management

- **Data Distribution and Partitioning**
    - **Horizontal Partitioning:** Distributing rows of a database across different nodes.
    - **Vertical Partitioning:** Distributing columns of a database across different nodes.
- **Database Replication**
    - **Master-Slave Replication:** One master node with multiple read-only slave nodes.
    - **Multi-Master Replication:** Multiple nodes capable of both read and write operations.
- **Handling Distributed Transactions**
    - **Two-Phase Commit:** Ensuring all nodes agree on a transaction's outcome.
    - **Distributed Ledger Technologies:** Using blockchain for immutable and verifiable transactions.