Introduction to Distributed System

Content

- Characterization of Distributed Systems
- Issues, Goals, and Types of distributed systems
- Distributed System Models, Hardware concepts, Software Concept.
- Middleware: Models of Middleware
- Services offered by middleware
- Client Server model.

Introduction

Definition:

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

Important aspects of this definition:

- Distributed system consist of components
- Users think they are dealing with a single system.

Characteristics of Distributed System

- Differences between the various computers and the ways in which they communicate are mostly hidden from users
- Users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.
- Distributed system should also be relatively easy to expand or scale.

Issues of Distributed System

- **Heterogeneity**

  The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Internet consists of many different sorts of network their differences are masked by the fact that all of the computers attached to them use the Internet protocols to communicate with one another.For eg., a computer attached to an Ethernet has an implementation of the Internet protocols over the Ethernet, whereas a computer on a different sort of network will need an implementation of the Internet protocols for that network.

- **Openness**

  The openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways.The openness of distributed systems is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

Issues of Distributed System

- **Security**

  Many of the information resources that are made available and maintained in distributed systems have a high intrinsic value to their users.Their security is therefore of considerable important. Security for information resources has three components: confidentiality, integrity, and availability.

- **Scalability**

  Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet. A system is described as scalable ***if it will remain effective when there is a significant increase in the number of resources and the number of users***.

Issues of Distributed System

- **Transparency**

  Transparency can be achieved at different levels. Easiest to do is to hide the distribution from 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 move at will without changing their names

  c) Replication transparency: The users cannot tell how many copies exist.

  d) Concurrency transparency: Multiple users can share resources automatically.

  e) Parallelism transparency: Activities can happen in parallel without users knowing.

Issues of Distributed System

- **Quality of service**

  Once users are provided with the functionality that they require of a service, such as the file service in a distributed system, we can go on to ask about the quality of the service provided.

  The main nonfunctional properties of systems that affect the quality of the service experienced by clients and users are reliability, security and performance.

  Adaptability to meet changing system configurations and resource availability has been recognized as a further important aspect of service quality.

Issues of Distributed System

- **Performance**

  Always the hidden data in the background is the issue of performance.

  Building a transparent, flexible, reliable distributed system, more important lies in its performance.

  In particular, when running a particular application on a distributed system, it should not be appreciably worse than running the same application on a single processor.

Issues of Distributed System

- **Failure handling**

  Computer systems sometimes fail.

  When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation.

  Failures in a distributed system are partial – that is, some components fail while others continue to function. Therefore the handling of failures is particularly difficult.

Issues of Distributed System

- **Concurrency**

  Both services and applications provide resources that can be shared by clients in a distributed system.

  There is therefore a possibility that several clients will attempt to access a shared resource at the same time.

  Object that represents a shared resource in a distributed system must be responsible for ensuring that it operates correctly in a concurrent environment.

  This applies not only to servers but also to objects in applications.

Issues of Distributed System

- **Reliability**

  One of the original goals of building distributed systems was to make them more reliable than single-processor systems.

  The idea is that if a machine goes down, some other machine takes over the job. ***A highly reliable system must be highly available***, but that is not enough.

  Data entrusted to the system must not be lost or garbled in any way, and if files are stored redundantly on multiple servers, all the copies must be kept consistent.

  In general, the more copies that are kept, the better the availability, but the greater the chance that they will be inconsistent, especially if updates are frequent.

Goals of Distributed System

A distributed system should make resources easily accessible; it should be reasonably hide the fact that resources are distributed across a network; it should be open; and it should be scalable.

1. Making resources accessible
2. Distribution Transparency
3. Openness
4. Scalability
5. Reliability
6. Performance

Goals of Distributed System

Making resources accessible

- The main goal of DS is to make it easy for users to access remote resources and to share them in a controlled and efficient way.
- Resources can be anything: printers, computers, storage facilities, data, files, web pages, networks etc.
- There are many reasons for wanting to share resources.
- One obvious reason is that of economics.
- E.g. it is cheaper to let a printer be shared by several users in a small office than having to buy and maintain a separate printer for each user.

Goals of Distributed System

Distribution Transparency

- An important goal of a DS is to hide the fact that its processes and resources are physically distributed across multiple computers.
- DS that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.

Goals of Distributed System

Distribution Transparency : Types of Transparency

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Figure 1-2. Different forms of transparency in a distributed system (ISO, 1995).

Goals of Distributed System

Openness

- An open DS is a system that offers services according to standard rules that describe the syntax and semantics of those services.
- E.g. in CN, standard rule govern the format, contents, and meaning of messages sent and received. Such rules are formalized in protocols.

Goals of Distributed System

Open Distributed System

Be able to interact with services from other open systems, irrespective of the underlying environment:

- Systems should conform to well-defined interface
- System should support portability of applications
- System should easily interoperate

Achieving openness

At least make the distributed system independent from heterogeneity of the underlying environment:

- Hardware
- Platforms
- Languages

Goals of Distributed System

Scalability

- Important goal
- Scalability can be measured along three dimensions:
  - Size scalability: number of users and/or processes
  - Geographical scalability: maximum distance between nodes
  - Administrative scalability: number of administrative domain

Goals of Distributed System

Scalability Problems:

- Having multiple copies leads to inconsistencies
- Modifying one copy makes that copy different from the rest
- Need global synchronization on each modification

Goals of Distributed System

Reliability

- The main goal was to increase the dependability of distributed systems relative to single processor systems.
- According to the concept, when one machine malfunctions, another one adjust to it.

Goals of Distributed System

Performance

- If a distributed system cannot achieve the desired performance, it is pointless to develop it as transparent, flexible and reliable.
- A distributed system ought to offer the highest level of performance.

Types of Distributed System

- Distributed Computing Systems
  - Cluster Computing Systems
  - Grid Computing Systems
- Distributed Information Systems
  - Transaction Processing Systems
  - Enterprise Application Integration
- Distributed Pervasive Systems
  - Home Systems
  - Electronic Health Care System
  - Sensor Networks

Distributed Computing System

- An important class of distributed systems is the one used for high-performance computing tasks.
- Roughly speaking, one can make a distinction between two subgroups.
- In cluster computing the underlying hardware consists of a collection of similar workstations or PCs, closely connected by means of a high- speed local-area network. In addition, each node runs the same operating system.
- The situation becomes quite different in the case of grid computing.
- This subgroup consists of distributed systems that are often constructed as a federation of computer systems, where each system may fall under a different administrative domain, and may be very different when it comes to hardware, software, and deployed network technology.

Cluster Computing System

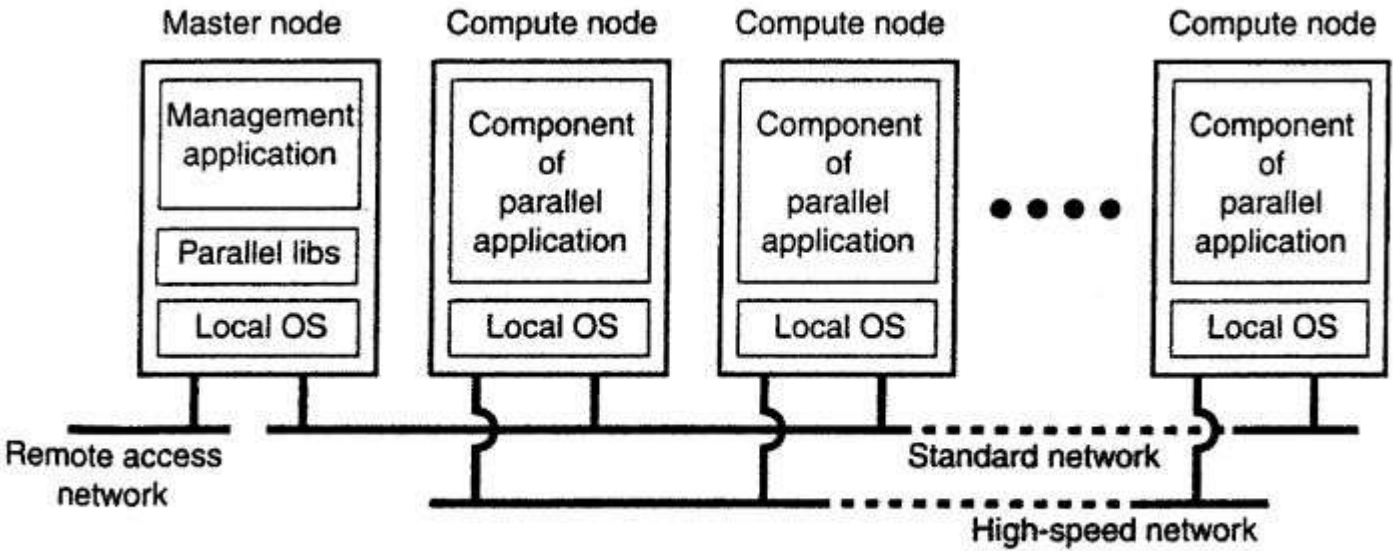


Figure 1-6. An example of a cluster computing system.

Cluster Computing System

- cluster computing is used for parallel programming in which a single (compute intensive) program is run in parallel on multiple machines.
- One well-known example of a cluster computer is formed by Linux-based Beowulf clusters, of which the general configuration is shown in Fig. 1-6.
- Each cluster consists of a collection of compute nodes that are controlled and accessed by means of a single master node.
- The master typically handles the allocation of nodes to a particular parallel program, maintains a batch queue of submitted jobs, and provides an interface for the users of the system.
- As such, the master actually runs the middleware needed for the execution of programs and management of the cluster, while the compute nodes often need nothing else but a standard operating system.

Grid Computing System

- Grid computing systems have a high degree of heterogeneity: no assumptions are made concerning hardware, operating systems, networks, administrative domains, security policies, etc.
- A key issue in a grid computing system is that resources from different organizations are brought together to allow the collaboration of a group of people or institutions.
- Such a collaboration is realized in the form of a virtual organization.
- The people belonging to the same virtual organization have access rights to the resources that are provided to that organization.
- Typically, resources consist of compute servers (including supercomputers, possibly implemented as cluster computers), storage facilities, and databases. In addition, special networked devices such as telescopes, sensors, etc., can be provided as well.

Grid Computing System

- Given its nature, much of the software for realizing grid computing evolves around providing access to resources from different administrative domains, and to only those users and applications that belong to a specific virtual organization.
- For this reason, focus is often on architectural issues.
- An architecture proposed by Foster et al. (2001). is shown in Fig. 1-7

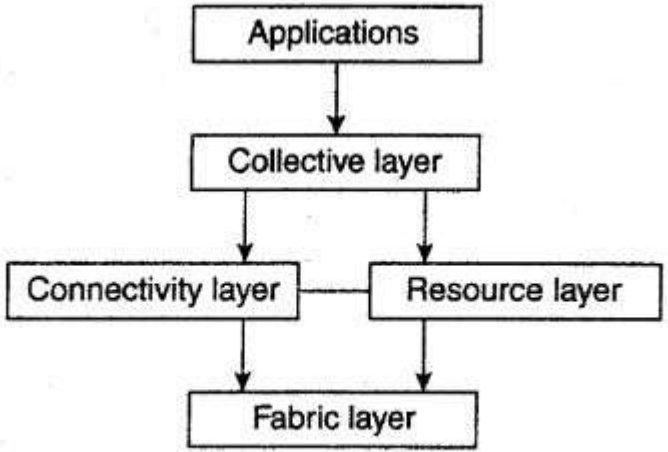


Figure 1-7. A layered architecture for grid computing systems.

Grid Computing System

- The lowest fabric layer provides interfaces to local resources at a specific site. Note that these interfaces are tailored to allow sharing of resources within a virtual organization.
- Typically, they will provide functions for querying the state and capabilities of a resource

Grid Computing System

- The connectivity layer consists of communication protocols for supporting grid transactions that span the usage of multiple resources.
- For example, protocols are needed to transfer data between resources, or to simply access a resource from a remote location.
- In addition, the connectivity layer will contain security protocols to authenticate users and resources.
- Note that in many cases human users are not authenticated; instead, programs acting on behalf of the users are authenticated.
- In this sense, delegating rights from a user to programs is an important function that needs to be supported in the connectivity layer.

Grid Computing System

- The resource layer is responsible for managing a single resource.
- It uses the functions provided by the connectivity layer and calls directly the interfaces made available by the fabric layer.
- For example, this layer will offer functions for obtaining configuration information on a specific resource, or, in general, to perform specific operations such as creating a process or reading data.
- The resource layer is thus seen to be responsible for access control, and hence will rely on the authentication performed as part of the connectivity layer.

Grid Computing System

- The next layer in the hierarchy is the collective layer.
- It deals with handling access to multiple resources and typically consists of services for resource discovery, allocation and scheduling of tasks onto multiple resources, data replication, and so on.
- Unlike the connectivity and resource layer, which consist of a relatively small, standard collection of protocols,
- the collective layer may consist of many different protocols for many different purposes, reflecting the broad spectrum of services it may offer to a virtual organization.

Grid Computing System

- Finally, the application layer consists of the applications that operate within a virtual organization and which make use of the grid computing environment.
- the collective, connectivity, and resource layer form the heart of what could be called a grid middleware layer.
- These layers jointly provide access to and management of resources that are potentially dispersed across multiple sites.

Distributed Information System

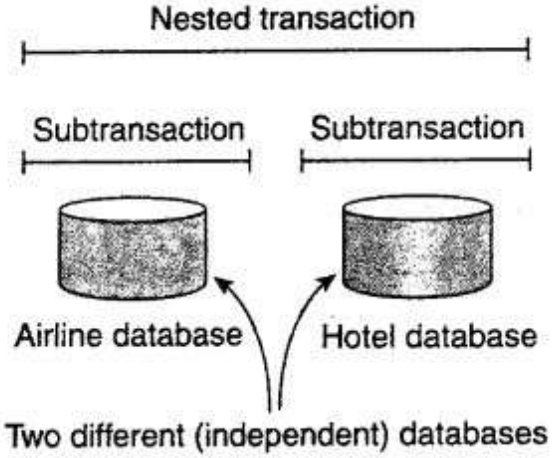
Distributed Transaction Processing:

- ➔ It works across different servers using multiple communication models.
- ➔ The four characteristics that transactions have:
  - ◆ Atomic: the transaction taking place must be indivisible to the others.
  - ◆ Consistent: The transaction should be consistent after the transaction has been done.
  - ◆ Isolated: Concurrent transactions do not interfere with each other.
  - ◆ Durable: Once a transaction commits, the changes are permanent.

Distributed Information System

Distributed Transaction Processing:

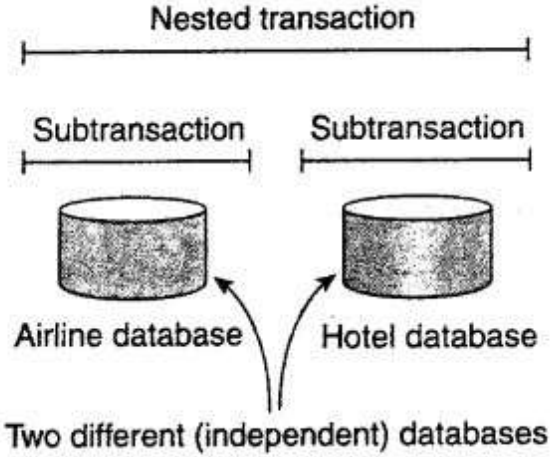
- A nested transaction is constructed from a number of subtransactions
- The top-level transaction may fork off children that run in parallel with one another, on different machines, to gain performance or simplify programming.
- Each of these children may also execute one or more subtransactions, or fork off its own children.



Distributed Information System

Distributed Transaction Processing:

- Nested transactions are important in distributed systems, for they provide a natural way of distributing a transaction across multiple machines.
- They follow a logical division of the work of the original transaction.
- For example, a transaction for planning a trip by which three different flights need to be reserved can be logically split up into three subtransactions. Each of these subtransactions can be managed separately and independent of the other two.



Distributed Information System

Distributed Transaction Processing:

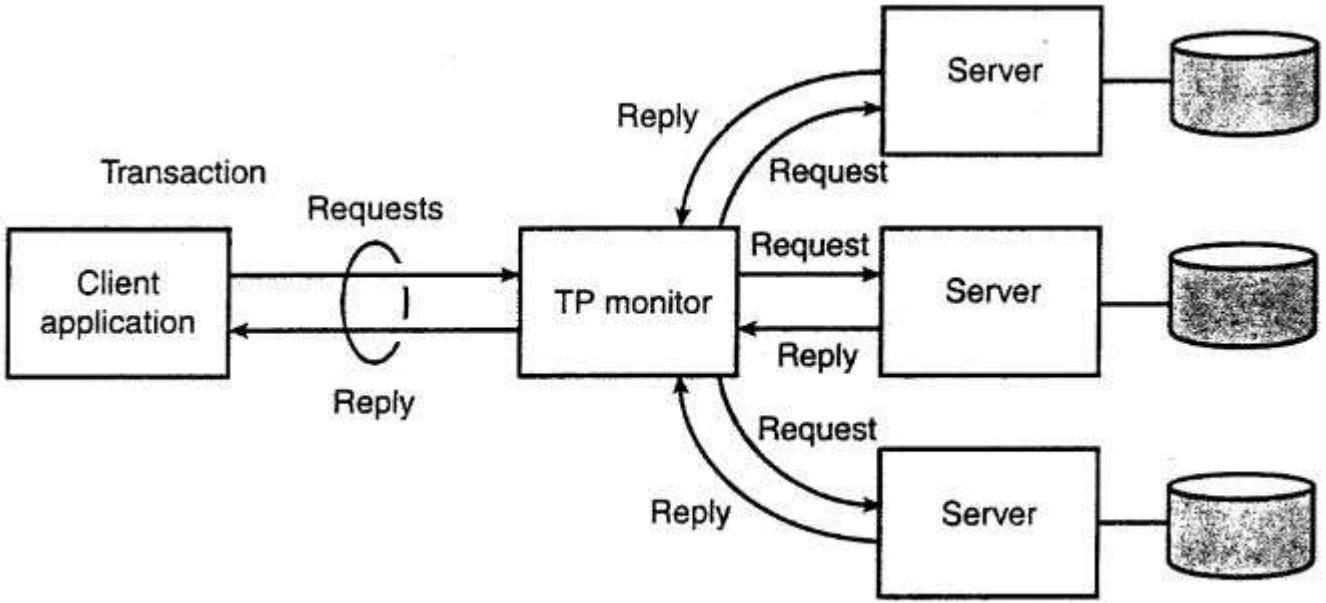


Figure 1-10. The role of a TP monitor in distributed systems.

Distributed Information System

Distributed Transaction Processing:

- In the company's middleware systems, the component that manages distributed (or nested) transactions has formed the application integration core at the server or database.
- This was referred to as the Transaction Processing Monitor(TP Monitor).
- Its main task was to allow an application to access multiple servers/databases by providing a transactional programming model.
- Many requests are sent to the database to get the result, to ensure each request gets successfully executed and deliver result to each request, this work is handled by the TP Monitor.

Distributed Information System

Enterprise Application Integration:

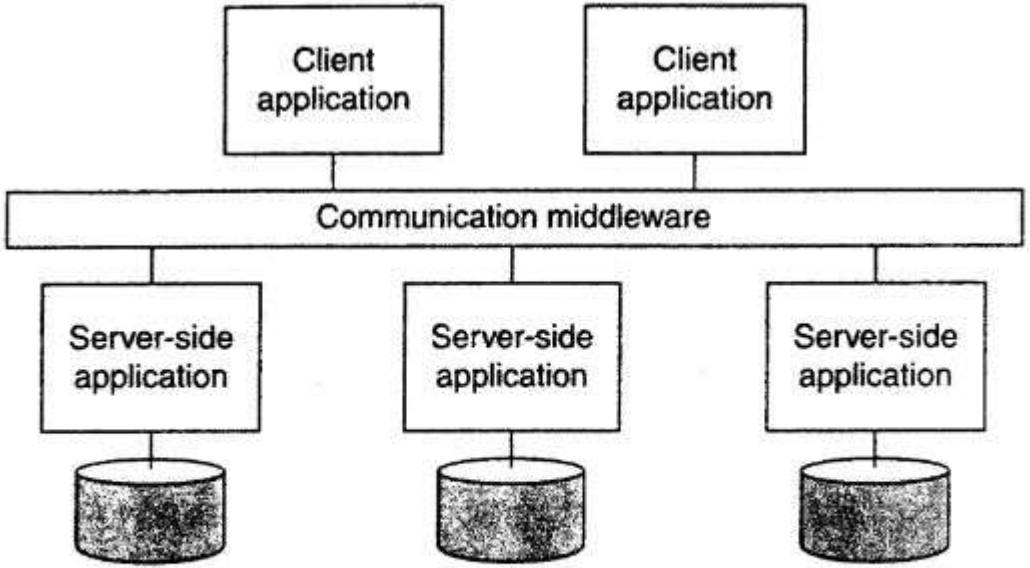


Figure 1-11. MiddJeware as a communication facilitator in enterprise application integration.

Distributed Information System

Enterprise Application Integration:

- Enterprise Application Integration (EAI) is the process of bringing different businesses together.
- The databases and workflows associated with business applications ensure that the business uses information consistently and that changes in data done by one business application are reflected correctly in another’s.
- Many organizations collect different data from different plate forms in the internal systems and then they use those data are used in the Trading system /physical medium.

Distributed Information System

Enterprise Application Integration:

- Remote Procedure Calls (RPC), a software element that sends a request to every other application component.
- An app can have a different database for managing different data and then they can communicate with each other on different platforms.
- Suppose, if you login into your android device and watch your video on YouTube then you go to your laptop and open YouTube you can see the same video is in your watch list.

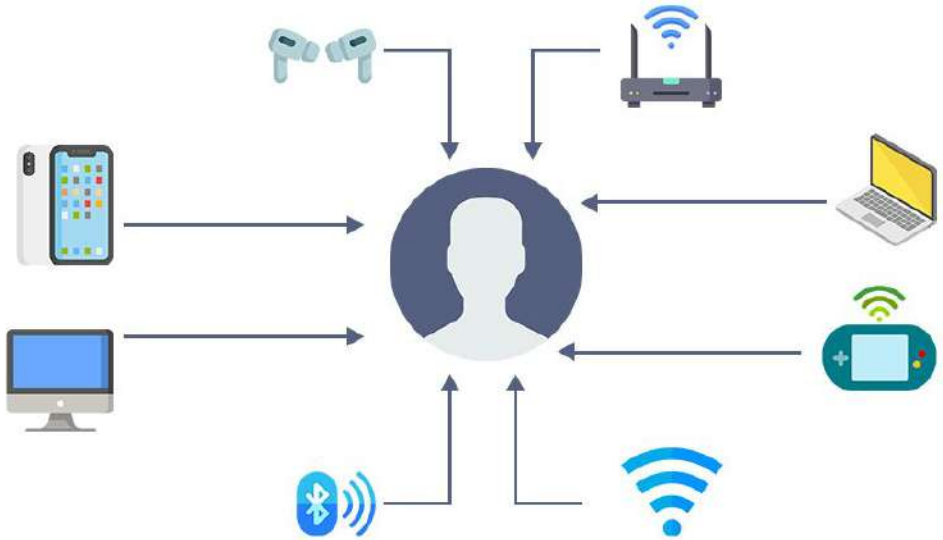
Distributed Pervasive System

- Pervasive Computing is also abbreviated as ubiquitous (Changed and removed) computing
- it is the new step towards integrating everyday objects with microprocessors so that this information can communicate.
- a computer system available anywhere in the company or as a generally available consumer system that looks like that same everywhere with the same functionality but that operates from computing power, storage, and locations across the globe.

Distributed Pervasive System

Home System

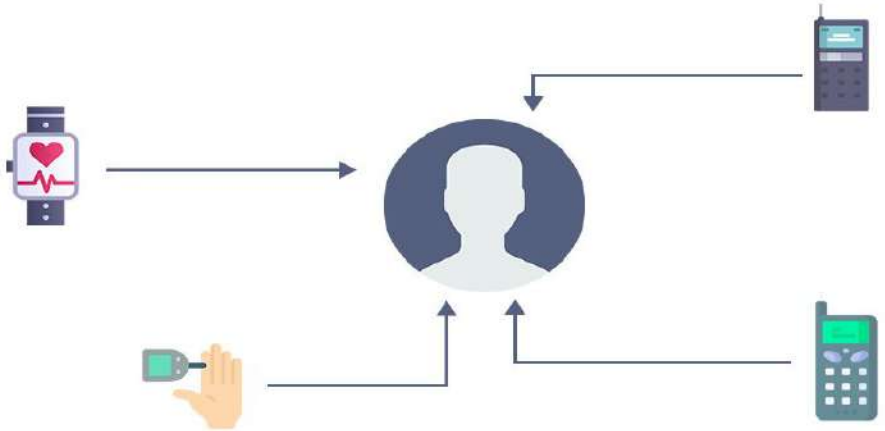
Nowadays many devices used in the home are digital so we can control them from anywhere and effectively.



Distributed Pervasive System

Electronic Health System

Nowadays smart medical wearable devices are also present through which we can monitor our health regularly.

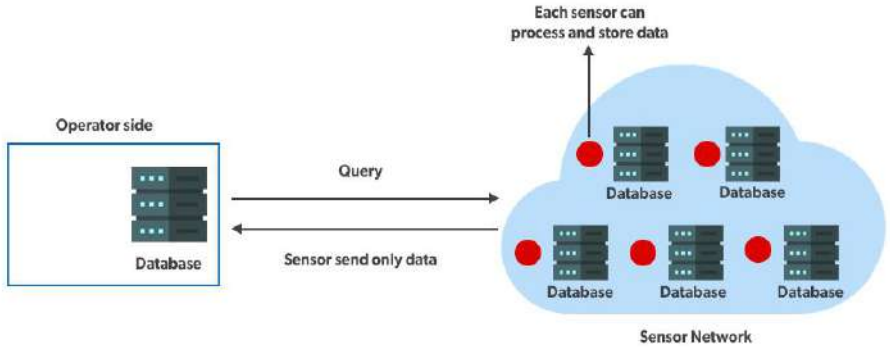


Distributed Pervasive System

Sensor Network (IoT devices)

Internet devices only send data to the client to act according to the data send to the device.

They can store and process the data to manage it efficiently.



Distributed System Models: Physical Model

- A physical model is basically a representation of the underlying hardware elements of a distributed system.
- It encompasses the hardware composition of a distributed system in terms of computers and other devices and their interconnections.
- It is primarily used to design, manage, implement and determine the performance of a distributed system.
- A physical model majorly consists of the following components:
  - **Nodes**
  - **Links**
  - **Middleware**
  - **Network topology**
  - **Communication Protocols**

Distributed System Models: Physical Model

Nodes

- Nodes are the end devices that have the ability of processing data, executing tasks and communicating with the other nodes.
- These end devices are generally the computers at the user end or can be servers, workstations etc.
- Nodes provision the distributed system with an interface in the presentation layer that enables the user to interact with other back-end devices, or nodes, that can be used for storage and database services, or processing, web browsing etc.
- Each node has an Operating System, execution environment and different middleware requirements that facilitate communication and other vital tasks.

Distributed System Models: Physical Model

Links

- Links are the communication channels between different nodes and intermediate devices.
- These may be wired or wireless.
- Wired links or physical media are implemented using copper wires, fibre optic cables etc.
- The choice of the medium depends on the environmental conditions and the requirements.
- Generally, physical links are required for high performance and real-time computing.
- Different connection types that can be implemented are as follows:
  - **Point-to-point links** – It establishes a connection and allows data transfer between only two nodes.
  - **Broadcast links** – It enables a single node to transmit data to multiple nodes simultaneously.
  - **Multi-Access links** – Multiple nodes share the same communication channel to transfer data. Requires protocols to avoid interference while transmission.

Distributed System Models: Physical Model

Middleware

- These are the softwares installed and executed on the nodes.
- By running middleware on each node, the distributed computing system achieves a decentralised control and decision-making.
- It handles various tasks like communication with other nodes, resource management, fault tolerance, synchronisation of different nodes and security to prevent malicious and unauthorised access.

Distributed System Models: Physical Model

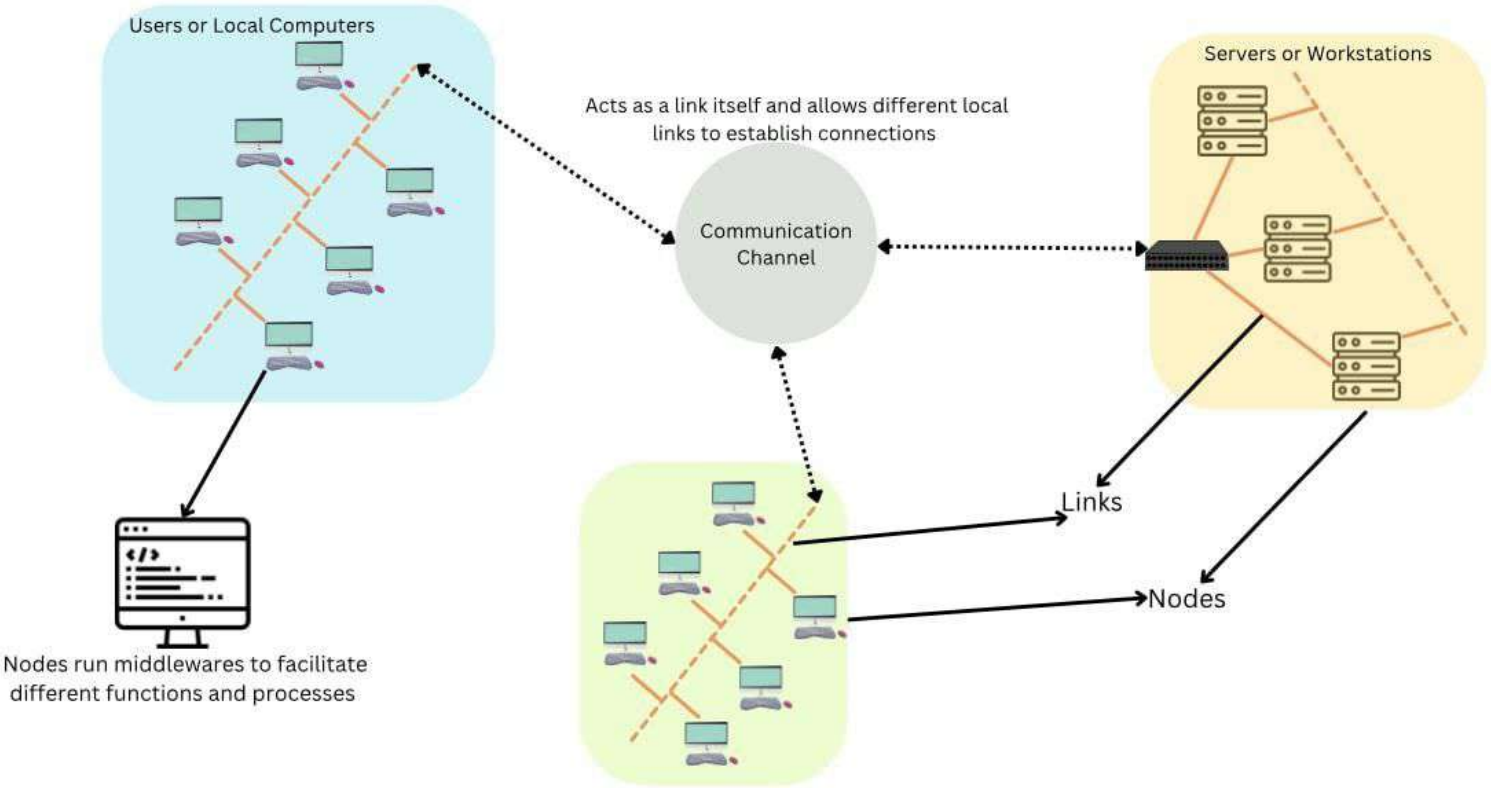
Network Topology

This defines the arrangement of nodes and links in the distributed computing system. The most common network topologies that are implemented are bus, star, mesh, ring or hybrid. Choice of topology is done by determining the exact use cases and the requirements.

Communication Protocols

Communication protocols are the set rules and procedures for transmitting data from in the links. Examples of these protocols include TCP, UDP, HTTPS, MQTT etc. These allow the nodes to communicate and interpret the data.

Distributed System Models: Physical Model

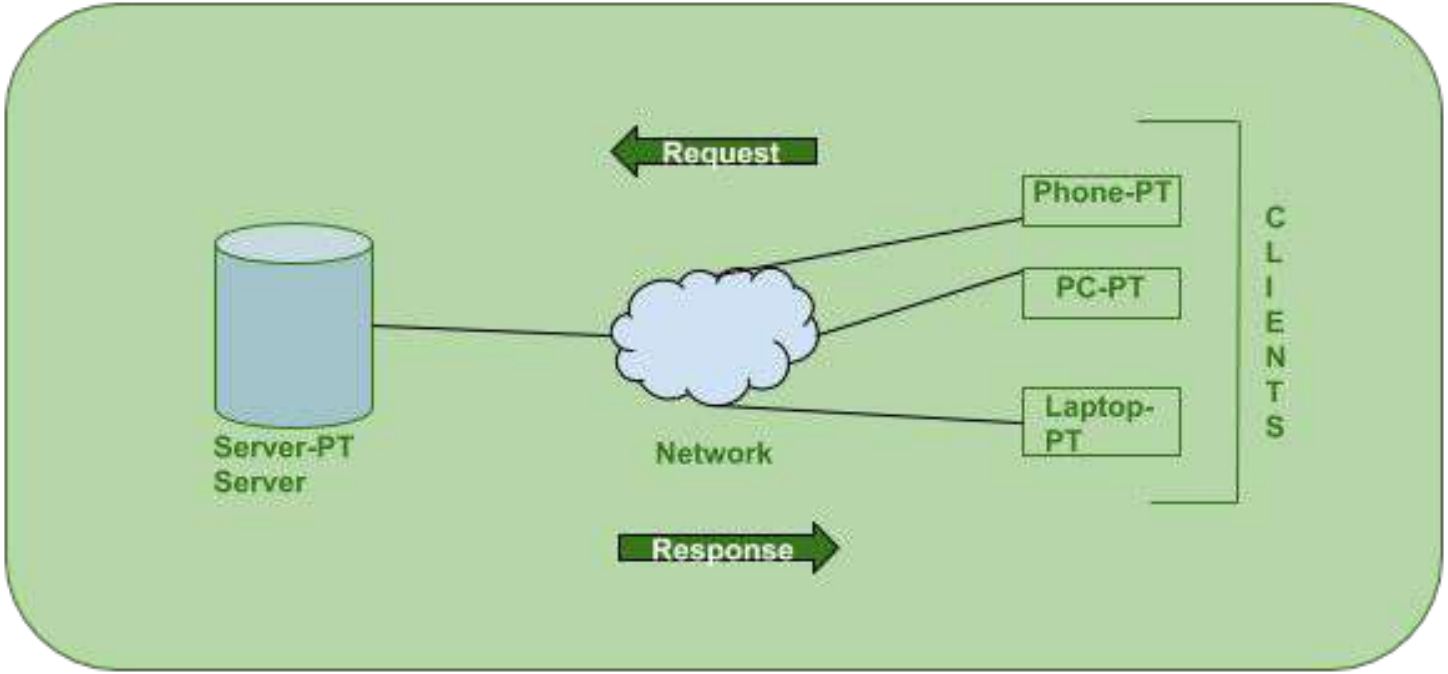


Distributed System Models: Architectural Model

- Architectural model in distributed computing system is the overall design and structure of the system, and how its different components are organised to interact with each other and provide the desired functionalities.
- It is an overview of the system, on how will the development, deployment and operations take place.
- Construction of a good architectural model is required for efficient cost usage, and highly improved scalability of the applications.
- The key aspects of architectural model are
    - Client server model
    - Peer to peer model
    - Layered model
    - Micro services model

Distributed System Models: Architectural Model

Client Server Model



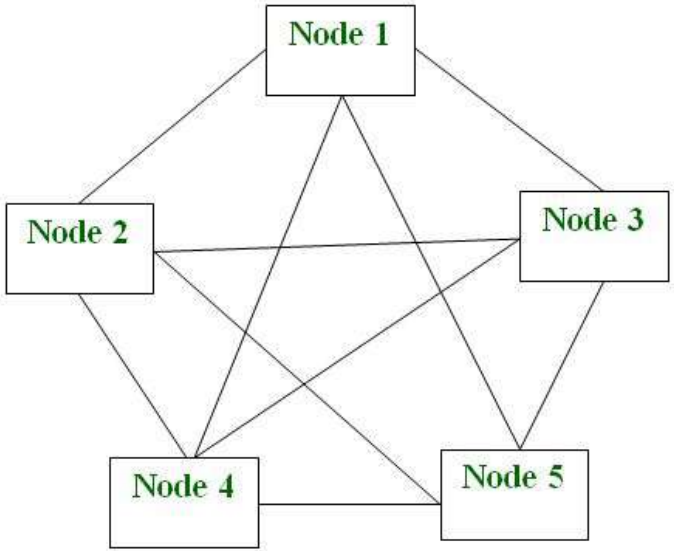
Distributed System Models: Architectural Model

Client Server Model

- It is a centralised approach in which the clients initiate requests for services and servers respond by providing those services.
- It mainly works on the request-response model where the client sends a request to the server and the server processes it, and responds to the client accordingly.
- It can be achieved by using TCP/IP, HTTP protocols on the transport layer.
- This is mainly used in web services, cloud computing, database management systems etc.

Distributed System Models: Architectural Model

Peer to Peer Model



P2P Architecture

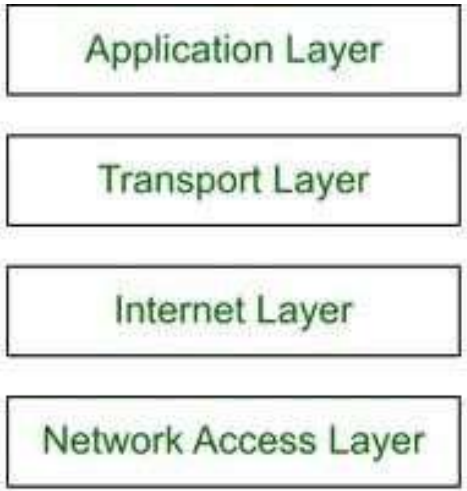
Distributed System Models: Architectural Model

Peer to Peer Model

- It is a decentralised approach in which all the distributed computing nodes, known as peers, are all the same in terms of computing capabilities and can both request as well as provide services to other peers.
- It is a highly scalable model because the peers can join and leave the system dynamically, which makes it an ad-hoc form of network.
- The resources are distributed and the peers need to look out for the required resources as and when required.
- The communication is directly done amongst the peers without any intermediaries according to some set rules and procedures defined in the P2P networks.

Distributed System Models: Architectural Model

Layered Model



Various Layers of the TCP/ IP Model

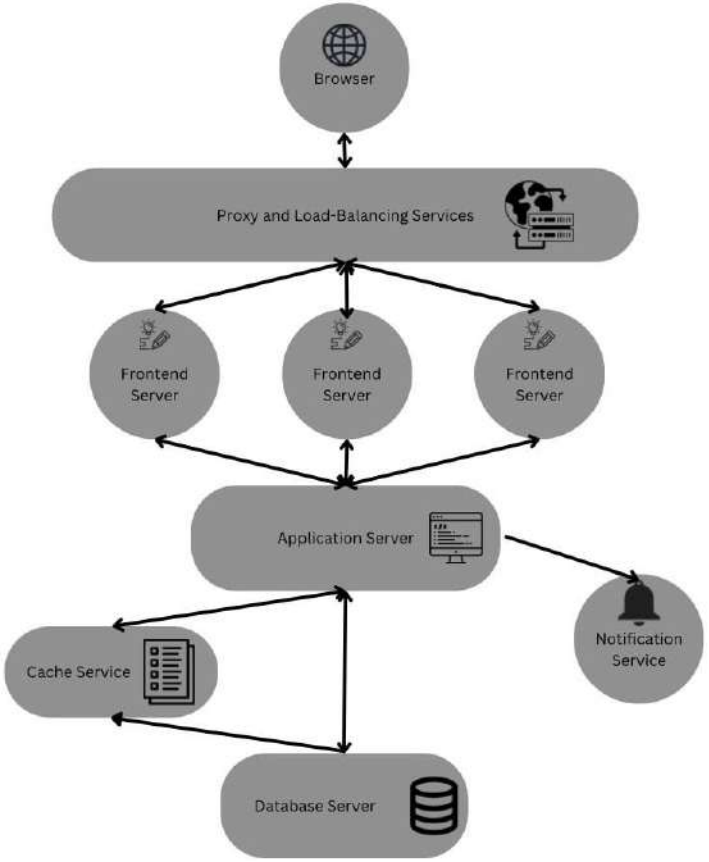
Distributed System Models: Architectural Model

Layered Model

- It involves organising the system into multiple layers, where each layer will provision a specific service.
- Each layer communicated with the adjacent layers using certain well-defined protocols without affecting the integrity of the system.
- A hierarchical structure is obtained where each layer abstracts the underlying complexity of lower layers.

Distributed System Models: Architectural Model

Micro Services Model



Distributed System Models: Architectural Model

Micro Services Model

- In this system, a complex application or task, is decomposed into multiple independent tasks and these services running on different servers.
- Each service performs only a single function and is focussed on a specific business-capability.
- This makes the overall system more maintainable, scalable and easier to understand.
- Services can be independently developed, deployed and scaled without affecting the ongoing services.

Distributed System Models: Fundamental Model

- The fundamental model in a distributed computing system is a broad conceptual framework that helps in understanding the key aspects of the distributed systems.
- These are concerned with more formal description of properties that are generally common in all architectural models.
- It represents the essential components that are required to understand a distributed system's behaviour. Three fundamental models are as follows:
    - Interaction Model
    - Remote Procedure Call
    - Failure Model
    - Security Model

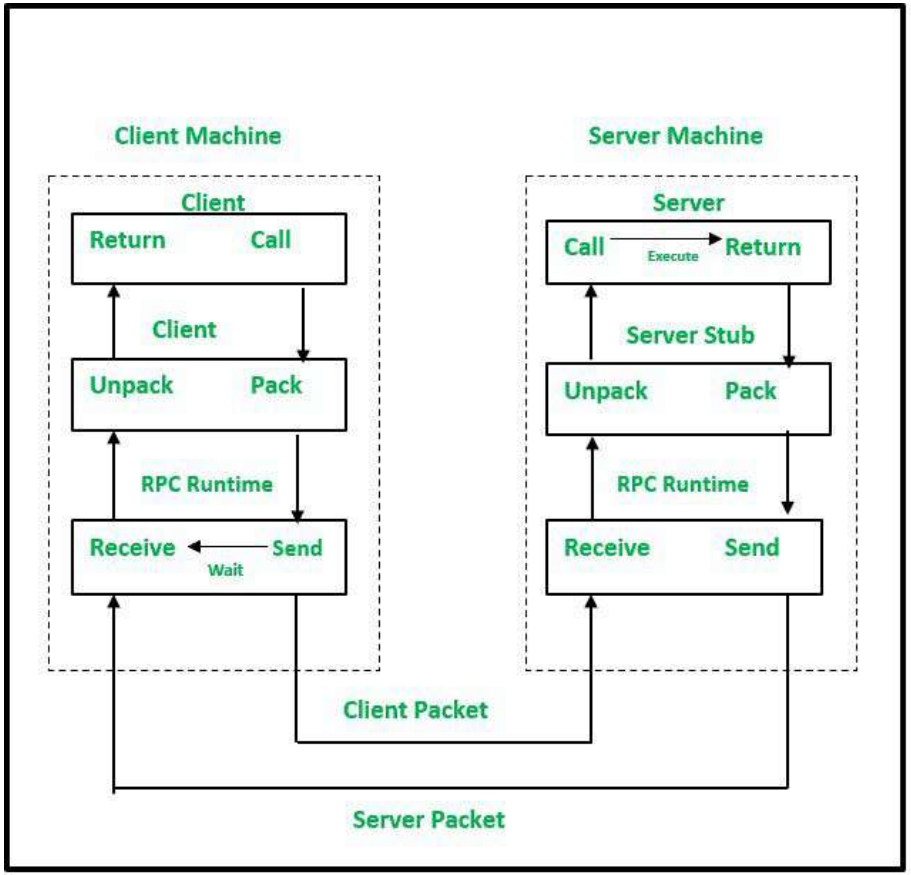
Distributed System Models: Fundamental Model

Interaction Model

- Distributed computing systems are full of many processes interacting with each other in highly complex ways.
- Interaction model provides a framework to understand the mechanisms and patterns that are used for communication and coordination among various processes.
- Different components that are important in this model are –
    - Message Passing – It deals with passing messages that may contain, data, instructions, a service request, or process synchronisation between different computing nodes. It may be synchronous or asynchronous depending on the types of tasks and processes.
    - Publish/Subscribe Systems – In this the publishing process can publish a message over a topic and the processes that are subscribed to that topic can take it up and execute the process for themselves. It is more important in an event-driven architecture.

Distributed System Models: Fundamental Model

RPC



Distributed System Models: Fundamental Model

RPC

- It is a communication paradigm that has an ability to invoke a new process or a method on a remote process as if it were a local procedure call.
- The client process makes a procedure call using RPC and then the message is passed to the required server process using communication protocols.
- These message passing protocols are abstracted and the result once obtained from the server process, is sent back to the client process to continue execution.

Distributed System Models: Fundamental Model

Failure Model

- This model addresses the faults and failures that occur in the distributed computing system.
- It provides a framework to identify and rectify the faults that occur or may occur in the system.
- Fault tolerance mechanisms are implemented so as to handle failures by replication and error detection and recovery methods.
- Different failures that may occur are:
  - Crash failures – A process or node unexpectedly stops functioning.
  - Omission failures – It involves a loss of message, resulting in absence of required communication.
  - Timing failures – The process deviates from its expected time quantum and may lead to delays or unsynchronised response times.
  - Byzantine failures – The process may send malicious or unexpected messages that conflict with the set protocols.

Distributed System Models: Fundamental Model

Security Model

- Distributed computing systems may suffer malicious attacks, unauthorised access and data breaches.
- Security model provides a framework for understanding the security requirements, threats, vulnerabilities, and mechanisms to safeguard the system and its resources.
- Various aspects that are vital in the security model are –
    - Authentication – It verifies the identity of the users accessing the system. It ensures that only the authorised and trusted entities get access. It involves –
    - Password-based authentication – Users provide a unique password to prove their identity.
    - Public-key cryptography – Entities possess a private key and a corresponding public key, allowing verification of their authenticity.
    - Multi-factor authentication – Multiple factors, such as passwords, biometrics, or security tokens, are used to validate identity.

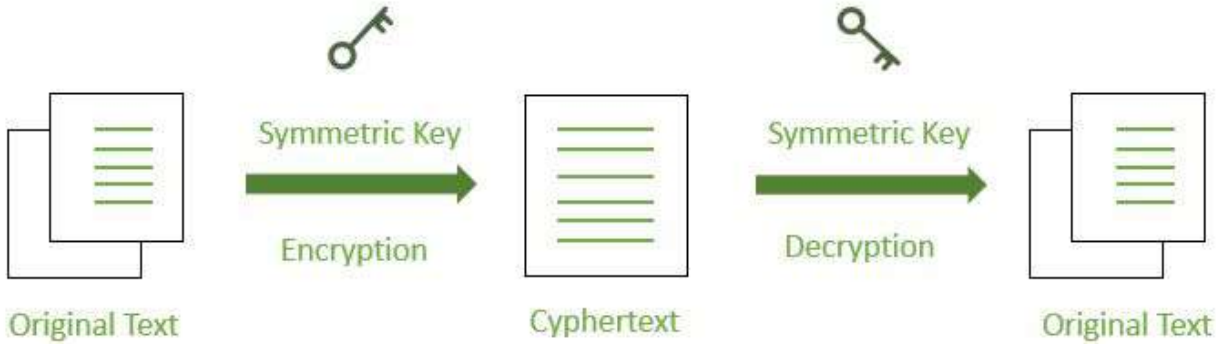
Distributed System Models: Fundamental Model

Security Model

- Encryption – It is the process of transforming data into a format that is unreadable without a decryption key. It protects sensitive information from unauthorized access or disclosure.
- Data Integrity – Data integrity mechanisms protect against unauthorised modifications or tampering of data. They ensure that data remains unchanged during storage, transmission, or processing. Data integrity mechanisms include:
    - Hash functions – Generating a hash value or checksum from data to verify its integrity.
    - Digital signatures – Using cryptographic techniques to sign data and verify its authenticity and integrity.

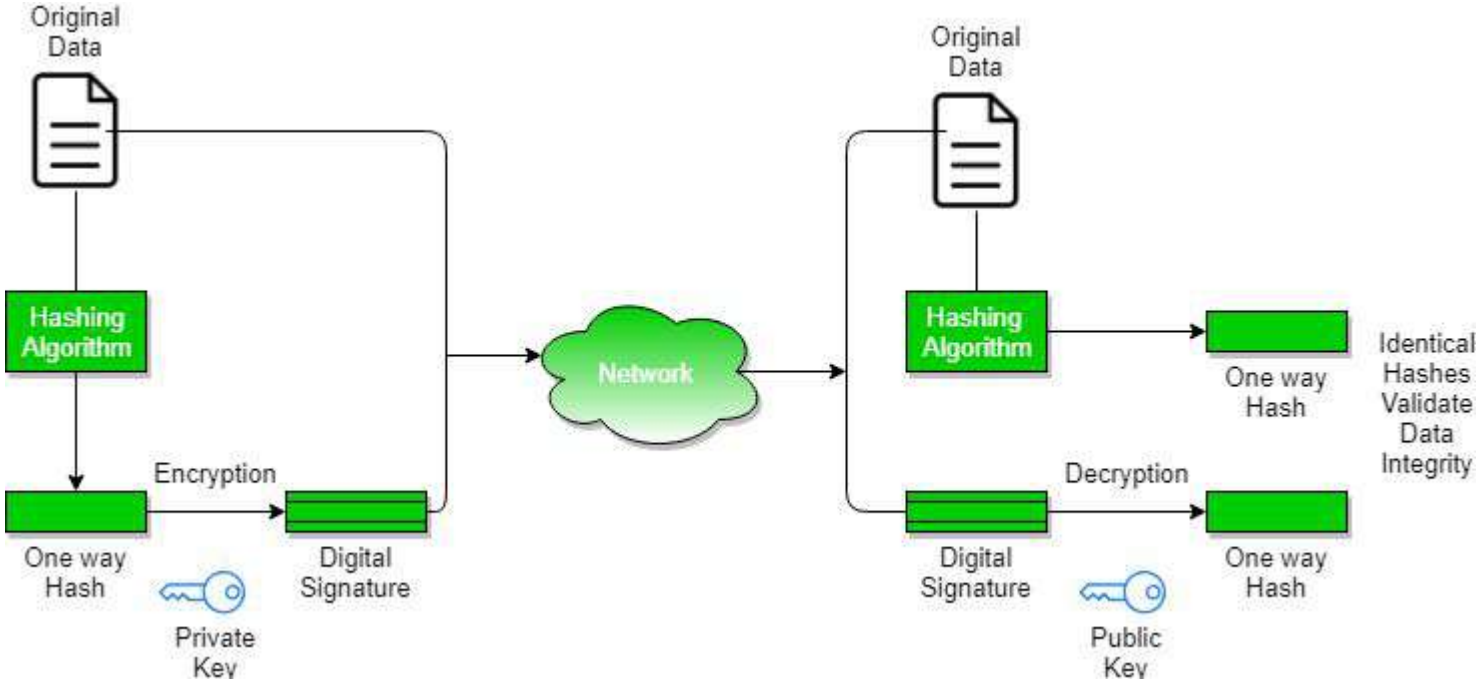
Distributed System Models: Fundamental Model

Security Model



Distributed System Models: Fundamental Model

Security Model



Hardware Concepts

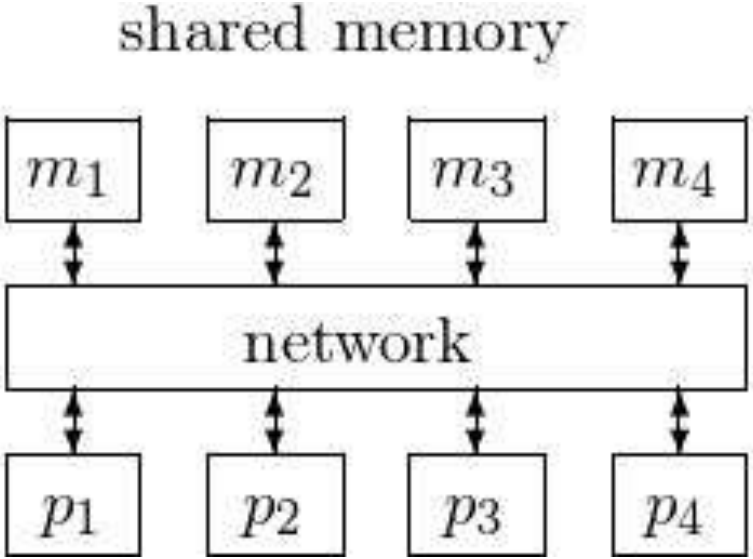
Types:

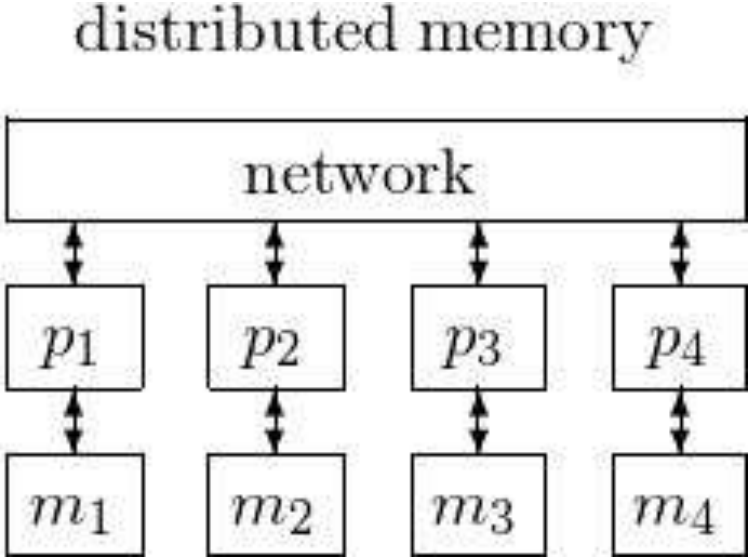
1. Multiprocessor
2. Multicomputer

Hardware Concepts : Multiprocessor

- A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM.
- The main objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.
- There are two types of multiprocessors: Shared memory multiprocessor and distributed memory multiprocessor.
- In shared memory multiprocessors, all the CPUs shares the common memory
- In a distributed memory multiprocessor, every CPU has its own private memory.

Hardware Concepts : Multiprocessor





Hardware Concepts : Multiprocessor

Benefits of using a Multiprocessor

- Enhanced performance.
- Multiple applications.
- Multi-tasking inside an application.
- High throughput and responsiveness.
- Hardware sharing among CPUs.

Hardware Concepts : Multiprocessor

Advantages:

- Improved performance:Multiprocessor systems can execute tasks faster than single-processor systems, as the workload can be distributed across multiple processors.

- Better scalability:Multiprocessor systems can be scaled more easily than single-processor systems, as additional processors can be added to the system to handle increased workloads.

Hardware Concepts : Multiprocessor

Advantages:

- Increased reliability: Multiprocessor systems can continue to operate even if one processor fails, as the remaining processors can continue to execute tasks.
- Reduced cost: Multiprocessor systems can be more cost-effective than building multiple single-processor systems to handle the same workload.
- Enhanced parallelism: Multiprocessor systems allow for greater parallelism, as different processors can execute different tasks simultaneously.

Hardware Concepts : Multiprocessor

Disadvantages:

- Increased complexity: Multiprocessor systems are more complex than single-processor systems, and they require additional hardware, software, and management resources.
- Higher power consumption: Multiprocessor systems require more power to operate than single-processor systems, which can increase the cost of operating and maintaining the system.

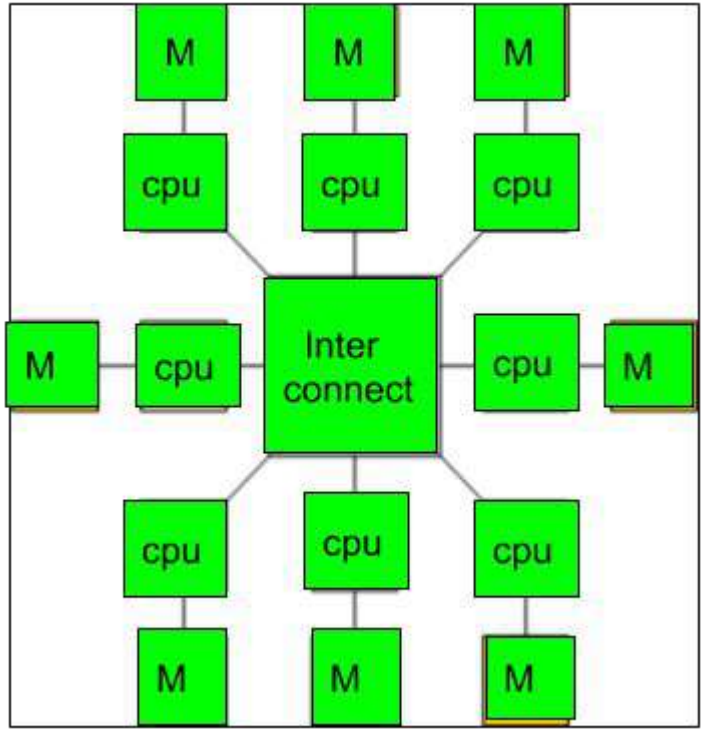
Hardware Concepts : Multiprocessor

Disadvantages:

- Difficult programming: Developing software that can effectively utilize multiple processors can be challenging, and it requires specialized programming skills.
- Synchronization issues: Multiprocessor systems require synchronization between processors to ensure that tasks are executed correctly and efficiently, which can add complexity and overhead to the system.
- Limited performance gains: Not all applications can benefit from multiprocessor systems, and some applications may only see limited performance gains when running on a multiprocessor system.

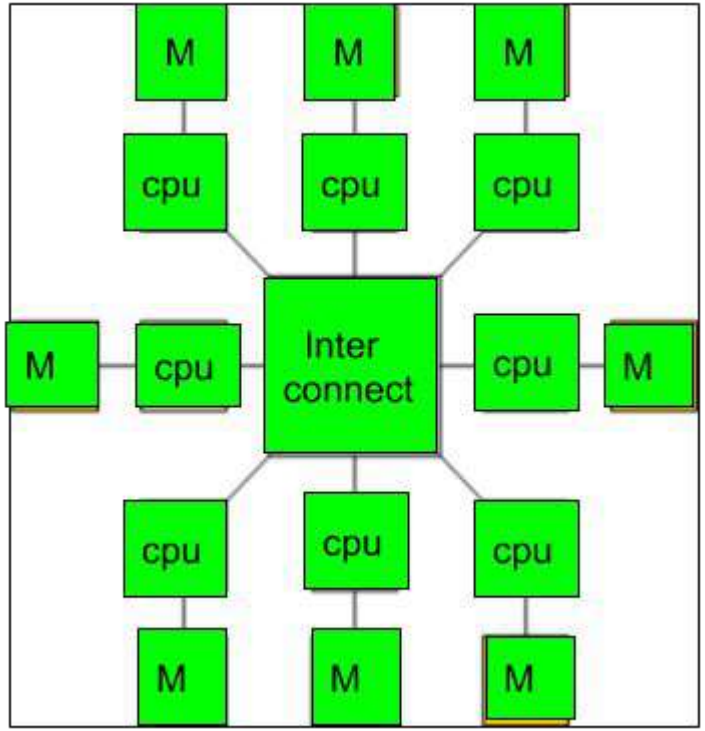
Hardware Concepts : Multicomputer

- A multicomputer system is a computer system with multiple processors that are connected together to solve a problem. Each processor has its own memory and it is accessible by that particular processor and those processors can communicate with each other via an interconnection network.



Hardware Concepts : Multicomputer

- As the multicomputer is capable of messages passing between the processors, it is possible to divide the task between the processors to complete the task. Hence, a multicomputer can be used for distributed computing. It is cost effective and easier to build a multicomputer than a multiprocessor.



Hardware Concepts : Multicomputer

Advantages:

- Improved performance: Multicomputer systems can execute tasks faster than single-computer systems, as the workload can be distributed across multiple computers.

- Better scalability: Multicomputer systems can be scaled more easily than single-computer systems, as additional computers can be added to the system to handle increased workloads.

Hardware Concepts : Multicomputer

Advantages:

- Increased reliability: Multicomputer systems can continue to operate even if one computer fails, as the remaining computers can continue to execute tasks.
- Reduced cost: Multicomputer systems can be more cost-effective than building a single large computer system to handle the same workload.
- Enhanced parallelism: Multicomputer systems allow for greater parallelism, as different computers can execute different tasks simultaneously.

Hardware Concepts : Multicomputer

Disadvantages:

- Increased complexity: Multicomputer systems are more complex than single-computer systems, and they require additional hardware, software, and management resources.

- Higher power consumption: Multicomputer systems require more power to operate than single-computer systems, which can increase the cost of operating and maintaining the system.

Hardware Concepts : Multicomputer

Disadvantages:

- Difficult programming: Developing software that can effectively utilize multiple computers can be challenging, and it requires specialized programming skills.
- Synchronization issues: Multicomputer systems require synchronization between computers to ensure that tasks are executed correctly and efficiently, which can add complexity and overhead to the system.
- Network latency: Multicomputer systems rely on a network to communicate between computers, and network latency can impact system performance.

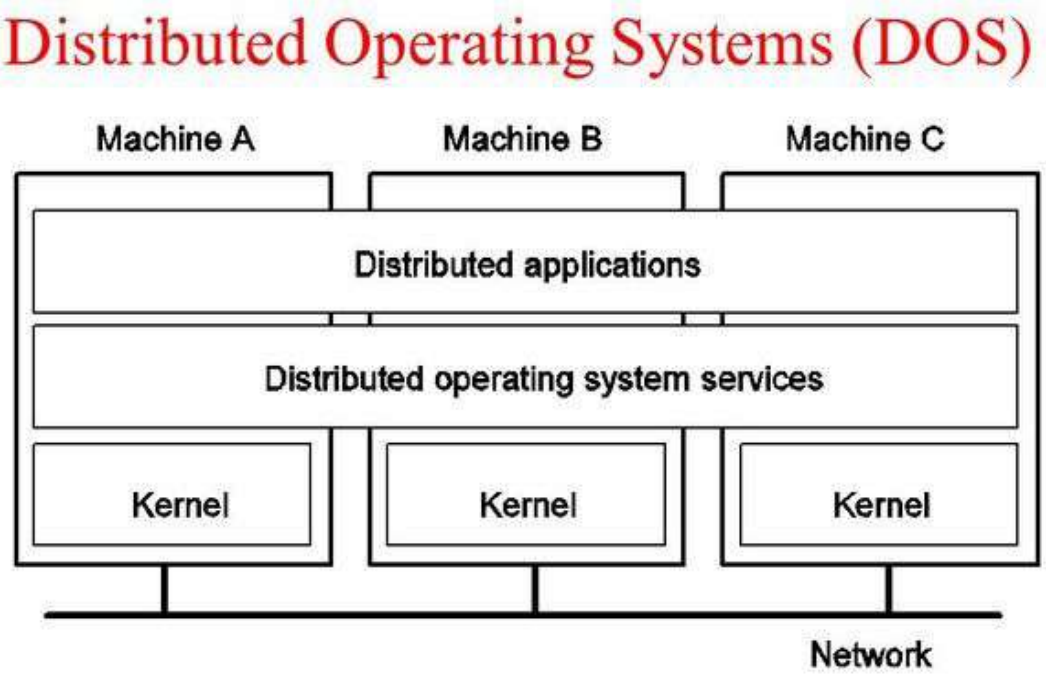
Software Concept

- Used to provide interaction between a user and the hardware
- Types:
  - Distributed operating system
  - Network operating system
  - middleware

Software Concept: Distributed Operating System

- Basic type of OS
- It is DS that abstracts resources, such as memory or CPUs and exposes common services and primitives that in turn are used by applications.
- Through single communication channel it links several computers
- Each of the system has its own processor and memory
- CPUs may communicate through high speed buses
- It is also called as tightly coupled system

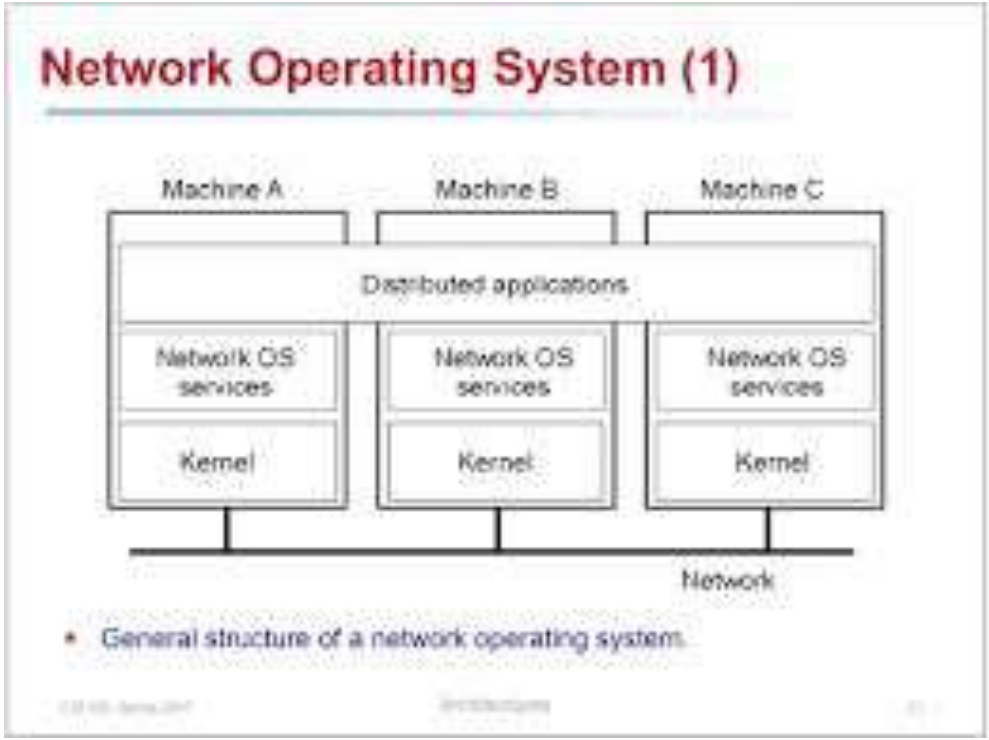
Software Concept: Distributed Operating System



Software Concept: Network Operating System

- It is created specifically to handle workstations, database sharing, application sharing, file and printer access sharing and other network wide computer functions.
- Designed for heterogeneous computer system
- It has multiple OS that are running on different hardware platform.
- Also called as loosely coupled system

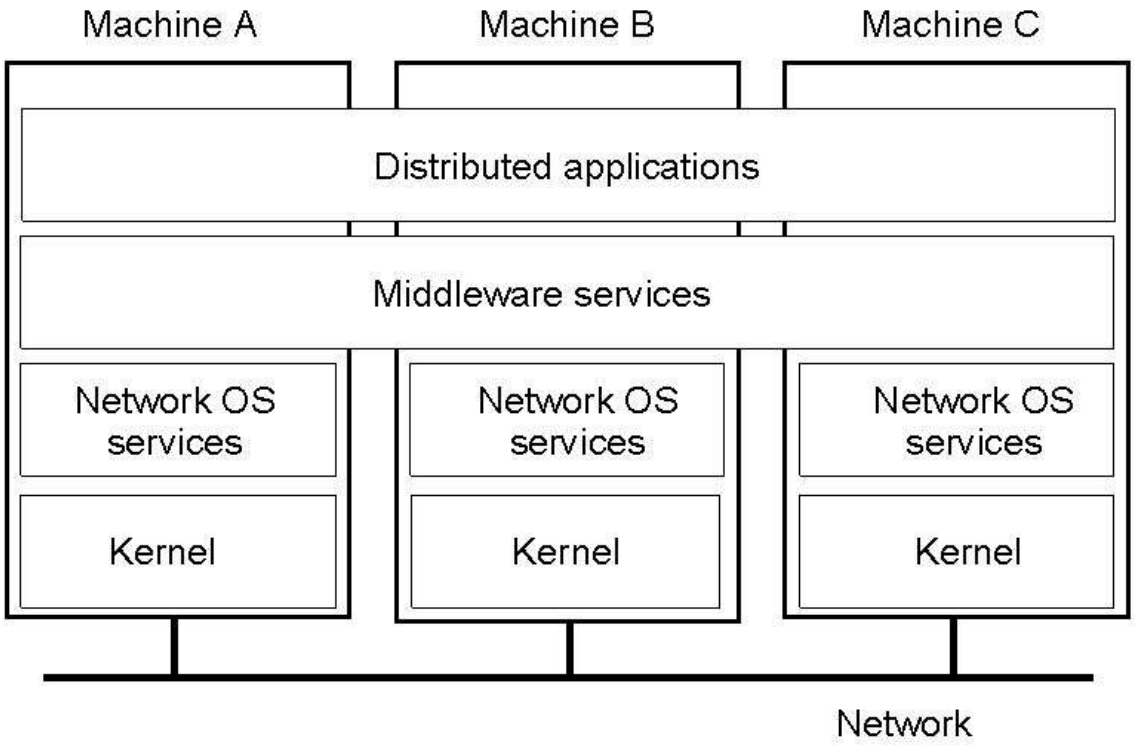
Software Concept: Network Operating System



Software Concept: Middleware

- It refers to software that offers additional services above and beyond those offered by the OS to allow data management and communication across the various distributed system components.
- Complex distributed applications are supported and made easier by middleware
- It enables interoperability between applications that run on different OS, by supplying services so that the application can exchange data in standard format.
- It lies in between the application software that may be working on different OS
- It comes in many forms, including database middleware, transactional middleware, intelligent middleware, message oriented middleware.
- Provides variety of services such as including control services, communication services and security services.

Software Concept: Middleware



Software Concept: Middleware

- It refers to software that offers additional services above and beyond those offered by the OS to allow data management and communication across the various distributed system components.
- Complex distributed applications are supported and made easier by middleware
- It enables interoperability between applications that run on different OS, by supplying services so that the application can exchange data in standard format.
- It lies in between the application software that may be working on different OS
- It comes in many forms, including database middleware, transactional middleware, intelligent middleware, message oriented middleware.
- Provides variety of services such as including control services, communication services and security services.

Models of Middleware

- Remote Procedure call
- Message oriented Middleware(MOM)
- Distributed Object Technology

Models of Middleware: Remote Procedure call

- It is one of the successful middleware models used in modern distributed applications for communication.
- It uses local call to call a procedure resides on the remote machine to get the result.
- Hidden communication is done between client and server.
- For eg If a user wants to get the sum of two numbers stored on remote server by using local method call, the user calls method with parameters, in turn server receives a RPC call from the client and returns the appropriate result to the client.
- Therefore, though the method was executed remotely it appears like a local to the called machine.
- This is a synchronous technology where both the client and server should be present during the communication.

Models of Middleware: Message oriented Middleware(MOM)

- It is another model used to send and receive the communication messages between clients and servers.
- It uses data structures like queue to store and retrieve messages.
- When the client is sending the messages faster than the receiver receiving it or the client is sending the message when the receiver is not available. So it uses queuing mechanism between the client and server to avoid the message been misplaced.
- It is asynchronous mechanism where messages can be sent even though the receiver is not available
- For eg Email system

Models of Middleware: Distributed Object Technology

- The distributed object technology has changed the scope of middleware technologies to one step up where objects are distributed to the remote server to facilitate the client.
- Eg RMI and CORBA
- The distributed object mechanism hides the communication interfaces and their details to provide access to the remote object efficiently.

Services offered by middleware

- Messaging middleware facilitates communications between distributed applications and services.
- Object or ORB middleware enables software components or objects to communicate and interact with a program -- such as containers -- across distributed systems.
- Remote Procedure Call (RPC) middleware provides a protocol that enables a program to request a service from another program located on a separate computer or network.
- Data or database middleware enables direct access to, and interaction with, databases; it typically includes SQL database software.

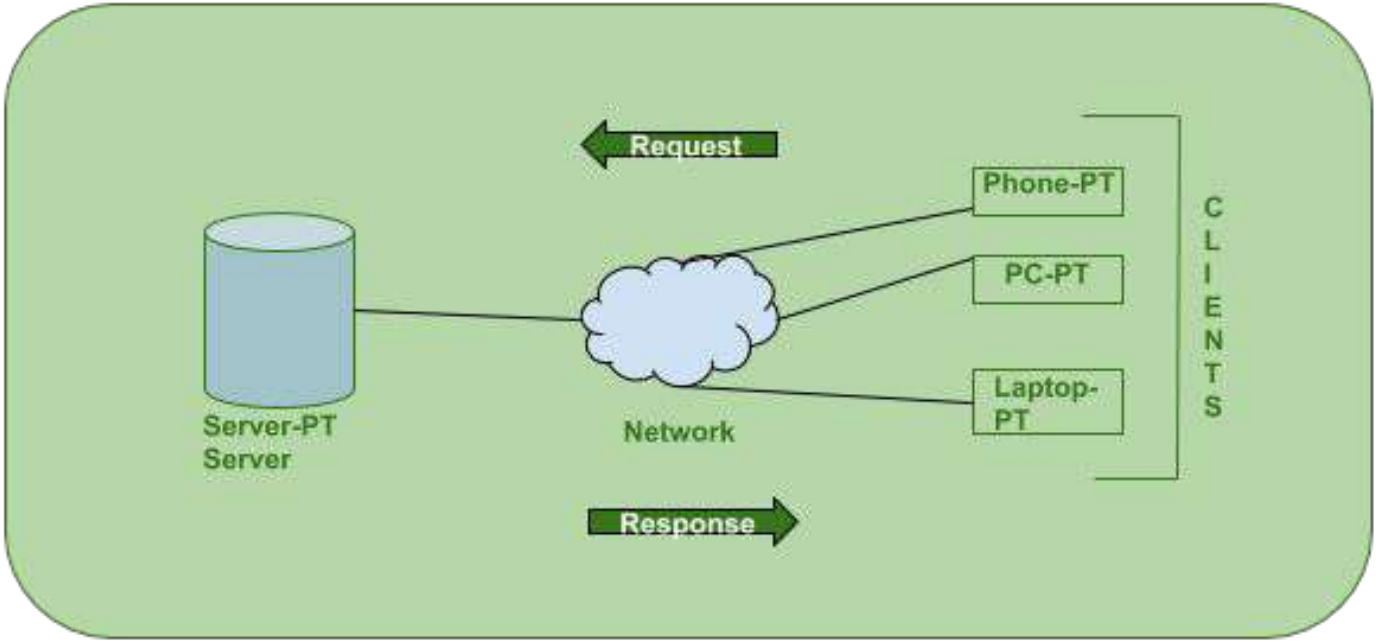
Services offered by middleware

- Transaction or transactional middleware ensures transactions move from one phase to the next via transaction process monitoring.
- Content-centric middleware allows client-side requests for specific content and abstracts and delivers it
- Embedded middleware facilitates communication and integration between embedded apps and real-time operating systems.
- API middleware enables developers to create and manage their application's APIs.
- Asynchronous data streaming middleware enables data sharing between applications by replicating data streams in an intermediate store.

Client Server Model

- The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients.
- In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client.
- Clients do not share any of their resources.
- Examples of Client-Server Model are Email, World Wide Web, etc.

Client Server Model

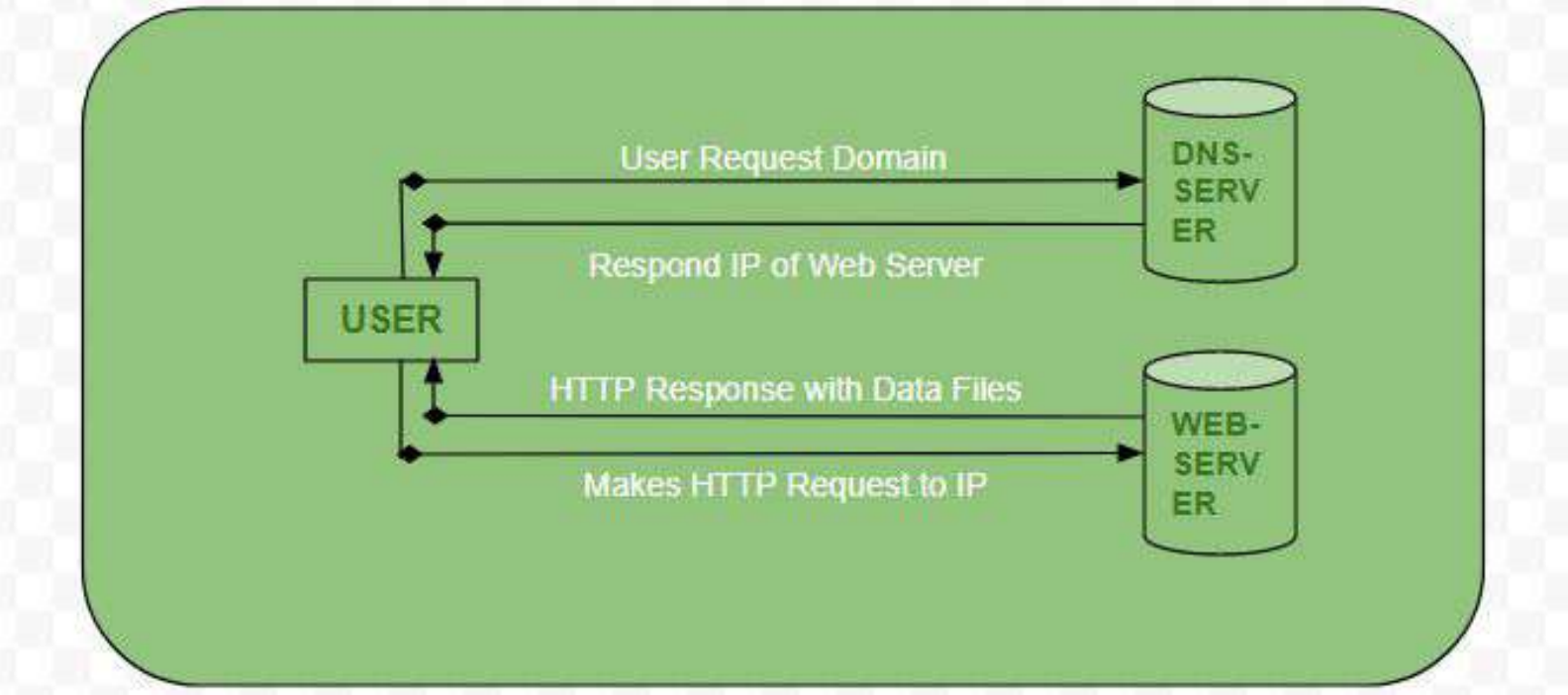


Client Server Model

How the browser interacts with the servers ?

- There are few steps to follow to interacts with the servers a client.
- User enters the URL(Uniform Resource Locator) of the website or file. The Browser then requests the DNS(DOMAIN NAME SYSTEM) Server.
- DNS Server lookup for the address of the WEB Server.
- DNS Server responds with the IP address of the WEB Server.
- Browser sends over an HTTP/HTTPS request to WEB Server’s IP (provided by DNS server).
- Server sends over the necessary files of the website.
- Browser then renders the files and the website is displayed.

Client Server Model



Client Server Model

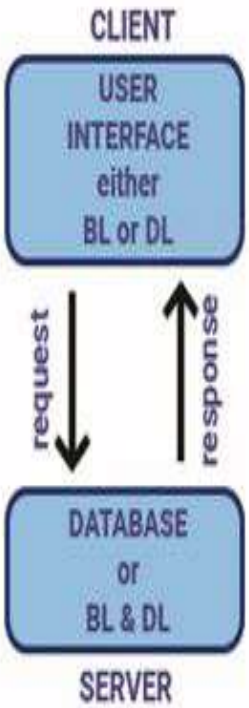
Types

- 1 tier architecture
- 2-tier architecture
- 3-tier architecture
- N-tire architecture

1-tier architecture

Used to describe a specific type of software architecture in which all the necessary components for the functioning of an application are accessible under the same package

2-tier architecture



2-tier architecture

The best environment is possessed by this architecture, where the client’s side stores the user interface and the server houses the database, while either the client’s side or the server’s side manages the database logic and business logic.

3-tier architecture

- Unlike 2-tier architecture that has no intermediary, in 3-tier client-server architecture, middleware lies between the client and the server.
- If the client places a request to fetch specific information from the server, the request will first be received by the middleware.
- It will then be dispatched to the server for further action. The same pattern will be followed when the server sends a response to the client.
- The framework of 3-tier architecture is categorized into three main layers, presentation layer, application layer, and database tier.
- All three layers are controlled at different ends.
- While the presentation layer is controlled at the client's device,
- the middleware and the server handle the application layer and the database tier respectively.
- Due to the presence of a third layer that provides data control, 3-tier architecture is more secure, has invisible database structure, and provides data integrity.

3-tier architecture

n-tier architecture

- N-tier architecture is also called multi-tier architecture.
- It is the scaled form of the other three types of architecture.
- This architecture has a provision for locating each function as an isolated layer that includes presentation, application processing, and management of data functionalities.