UNIT-I

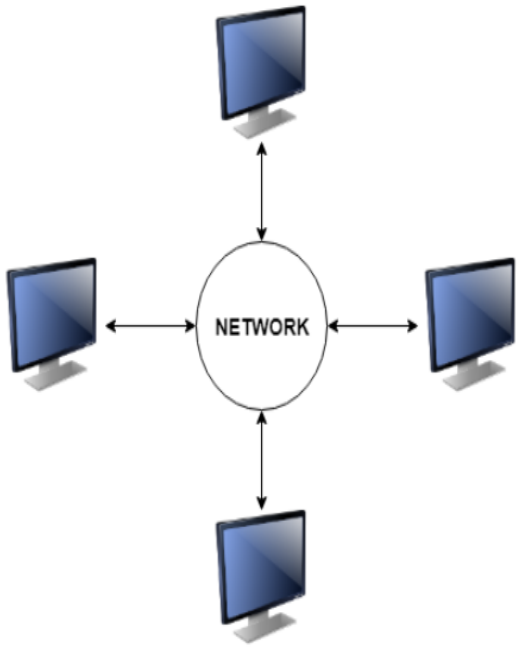
UNIT-I:

Characterization of Distributed Systems: Introduction, Examples of Distributed Systems, Resource Sharing and the Web, Challenges.

System Models: Introduction, Architectural Models- Software Layers, System Architecture, Variations, Interface and Objects, Design Requirements for Distributed Architectures, Fundamental Models- Interaction Model, Failure Model, Security Model.

A distributed system contains multiple nodes that are physically separate but linked together using the network. All the nodes in this system communicate with each other and handle processes in tandem. Each of these nodes contains a small part of the distributed operating system software.

A diagram to better explain the distributed system is:



DISTRIBUTED OPERATING SYSTEM

Types of Distributed Systems

The nodes in the distributed systems can be arranged in the form of client/server systems or peer to peer systems. Details about these are as follows:

Client/Server Systems

In client server systems, the client requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server. Both the client and server usually communicate via a computer network and so they are a part of distributed systems.

Peer to Peer Systems

The peer to peer systems contains nodes that are equal participants in data sharing. All the tasks are equally divided between all the nodes. The nodes interact with each other as required as share resources. This is done with the help of a network.

Advantages of Distributed Systems

Some advantages of Distributed Systems are as follows:

- All the nodes in the distributed system are connected to each other. So nodes can easily share data with other nodes.
- More nodes can easily be added to the distributed system i.e. it can be scaled as required.
- Failure of one node does not lead to the failure of the entire distributed system. Other nodes can still communicate with each other.
- Resources like printers can be shared with multiple nodes rather than being restricted to just one.

Disadvantages of Distributed Systems

Some disadvantages of Distributed Systems are as follows:

- It is difficult to provide adequate security in distributed systems because the nodes as well as the connections need to be secured.
- Some messages and data can be lost in the network while moving from one node to another.
- The database connected to the distributed systems is quite complicated and difficult to handle as compared to a single user system.
- Overloading may occur in the network if all the nodes of the distributed system try to send data at once.

A distributed system is the collection of autonomous computers that are connected using a communication network and they communicate with each other by passing messages to one another.

The different processors have their own local memory.

They use a distribution middleware.

They help in sharing different resources and capabilities to provide users with a single and integrated coherent network.

Distributed computing is a field of computer science that studies distributed systems and the computer program that runs in a distributed system is called a distributed program.

A distributed system requires concurrent Components, communication network and a synchronization mechanism. A distributed system allows resource sharing, including

software by systems connected to the network.

Examples of distributed systems / applications of distributed computing:

Intranets, Internet, WWW, email.

Telecommunication networks: Telephone networks and Cellular networks.

Network of branch office computers -Information system to handle automatic processing of orders,

Real-time process control: Aircraft control systems,

Electronic banking,

Airline reservation systems,

Sensor networks,

Mobile and Pervasive Computing systems.

Characteristics of Distributed Systems:-

Key characteristics of distributed systems are

- **Resource Sharing**

Resource sharing means that the existing *resources* in a distributed system can be accessed or remotely accessed across multiple computers in the system. Computers in distributed systems shares resources like *hardware* (disks and printers), *software* (files, windows and data objects) and *data*. Hardware resources are shared for reductions in cost and convenience. Data is shared for consistency and exchange of information.

Resources are managed by a software module known as a resource manager. Every resource has its own management policies and methods.

- **Heterogeneity**

In distributed systems components can have variety and differences in Networks, Computer hardware, Operating systems, Programming languages and implementations by different developers.

- **Openness**

Openness is concerned with extensions and improvements of distributed systems. The distributed system must be open in terms of *Hardware* and *Softwares*. In order to make a distributed system open,

1. A detailed and well-defined interface of components must be published.

2. Should standardize the interfaces of components

3. The new component must be easily integrated with existing components

- **Concurrency**

Concurrency is a property of a system representing the fact that multiple activities are executed at the same time. The concurrent execution of activities takes place in different components running on multiple machines as part of a distributed system. In addition, these activities may perform some kind of interactions among them. Concurrency *reduces the latency* and *increases the throughput* of the distributed system.

- **Scalability**

Scalability is mainly concerned about how the distributed system handles the *growth* as the number of users for the system increases. Mostly we scale the distributed system by adding more computers in the network. Components should not need to be changed when we scale the system. Components should be designed in such a way that it is scalable.

- **Fault Tolerance**

In a distributed system hardware, software, network anything can fail. The system must be designed in such a way that it is available all the time even after something has failed.

- **Transparency**

Distributed systems should be perceived by users and application programmers as a whole rather than as a collection of cooperating components. Transparency can be of various types like access, location, concurrency, replication, etc.

Challenges for a Distributed System

Designing a distributed system does not come as easy and straight forward. A number of challenges need to be overcome in order to get the ideal system. The major challenges in distributed systems are listed below:

1. Heterogeneity:

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity (that is, variety and difference) applies to all of the following:

- Hardware devices: computers, tablets, mobile phones, embedded devices, etc.
- Operating System: Ms Windows, Linux, Mac, Unix, etc.
- Network: Local network, the Internet, wireless network, satellite links, etc.
- Programming languages: Java, C/C++, Python, PHP, etc.
- Different roles of software developers, designers, system managers

Different programming languages use different representations for characters and data structures such as arrays and records. These differences must be addressed if programs written in different languages are to be able to communicate with one another. Programs written by different developers cannot communicate with one another unless they use common standards, for example, for network communication and the representation of primitive data items and data structures in messages. For this to happen, standards need to be agreed and adopted – as have the Internet protocols.

Middleware: The term middleware applies to a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages. Most middleware is implemented over the Internet protocols, which themselves mask the differences of the underlying networks, but all middleware deals with the differences in operating systems

and hardware

Heterogeneity and mobile code : The term mobile code is used to refer to program code that can be transferred from one computer to another and run at the destination – Java applets are an example. Code suitable for running on one computer is not necessarily suitable for running on another because executable programs are normally specific both to the instruction set and to the host operating system.

2. Transparency:

Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components. In other words, distributed systems designers must hide the complexity of the systems as much as they can. Some terms of transparency in distributed systems are:

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 may be copied in several places
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or a disk

3. Openness

The openness of a computer system is the characteristic that determines whether the system can be extended and reimplemented 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. If the well-defined interfaces for a system are published, it is easier for developers to add new features or replace sub-systems in the future. Example: Twitter and Facebook have API that allows

developers to develop theirs own software interactively.

4. 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. For example, a data structure that records bids for an auction may be accessed very frequently when it gets close to the deadline time. For an object to be safe in a concurrent environment, its operations must be synchronized in such a way that its data remains consistent. This can be achieved by standard techniques such as semaphores, which are used in most operating systems.

5. 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 importance. Security for information resources has three components:
confidentiality (protection against disclosure to unauthorized individuals)
integrity (protection against alteration or corruption),
availability for the authorized (protection against interference with the means to access the resources).

6. Scalability

Distributed systems must be scalable as the number of user increases. The scalability is defined as

A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity

Scalability has 3 dimensions:

- Size
  - Number of users and resources to be processed. Problem associated is

overloading

- Geography
  - Distance between users and resources. Problem associated is communication reliability
- Administration
  - As the size of distributed systems increases, many of the system needs to be controlled. Problem associated is administrative mess

7. 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. The handling of failures is particularly difficult.

UNIT-I

System Models: Introduction, Architectural Models- Software Layers, System Architecture, Variations, Interface and Objects, Design Requirements for Distributed Architectures, Fundamental Models- Interaction Model, Failure Model, Security Model.

Introduction:-

An architectural **model** of a **distributed system** defines the way in which the components of the **system** interact with each other and the way in which they are mapped onto an underlying network of computers. E.gs. include the client-server **model** and the peer process **model**.

Distributed System Models is as follows:

1. Architectural Models
2. Interaction Models
3. Fault Models

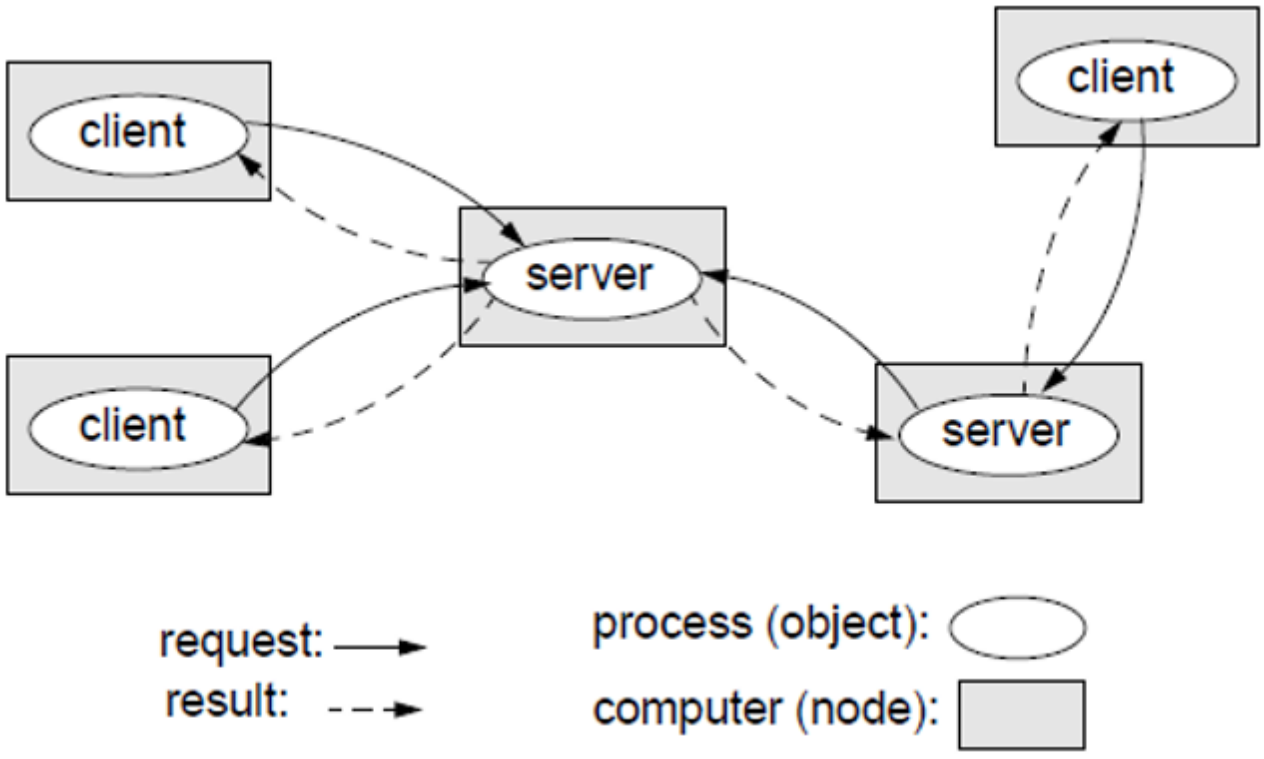
1. Architectural Models

Architectural model describes responsibilities distributed between system components and how are these components placed.

a)Client-server model

☞ The system is structured as a set of processes, called servers, that offer services to the users, called clients.

- The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC) or remote method invocation (RMI):
- The client sends a request (invocation) message to the server asking for some service;
- The server does the work and returns a result (e.g. the data requested) or an error code if the work could not be performed.



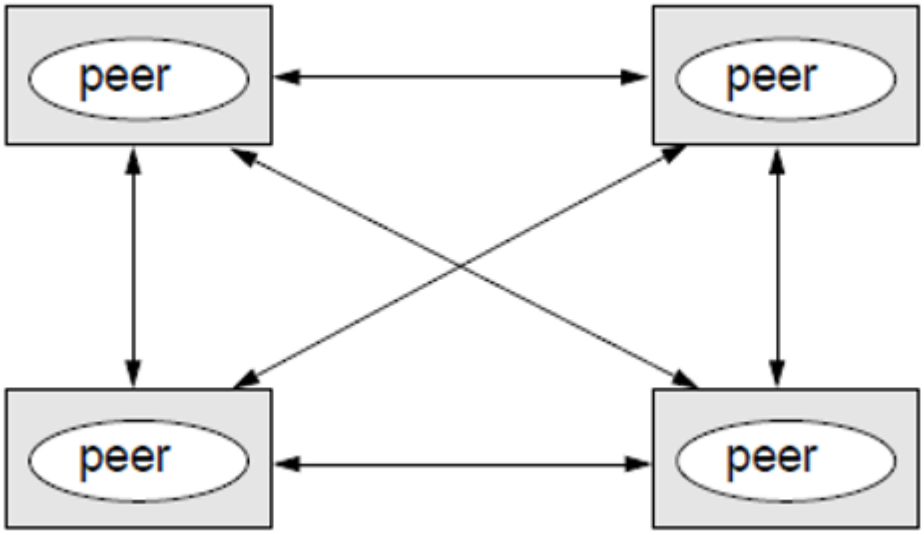
A server can itself request services from other servers; thus, in this new relation, the server itself acts like a client.

b)Peer-to-peer

☞ All processes (objects) play similar role.

- Processes (objects) interact without particular distinction between clients and servers.
- The pattern of communication depends on the particular application.
- A large number of data objects are shared; any individual computer holds only a small part of the application database.
- Processing and communication loads for access to objects are distributed across many computers and access links.

- This is the most general and flexible model.



- Peer-to-Peer tries to solve some of the above
- It distributes shared resources widely -> share computing and communication loads.

☞ Problems with peer-to-peer:

- High complexity due to
  - Cleverly place individual objects
  - retrieve the objects
  - maintain potentially large number of replicas.

2.Interaction Model

Interaction model are for handling time i. e. for process execution, message delivery, clock drifts etc.

- Synchronous distributed systems

Main features:

- Lower and upper bounds on execution time of processes can be set.
- Transmitted messages are received within a known bounded time.
- Drift rates between local clocks have a known bound.

Important consequences:

1. In a synchronous distributed system there is a notion of global physical time (with a known relative precision depending on the drift rate).
2. Only synchronous distributed systems have a predictable behavior in terms of timing.

Only such systems can be used for hard real-time applications.

3. In a synchronous distributed system it is possible and safe to use timeouts in order to detect failures of a process or communication link.

☞ It is difficult and costly to implement synchronous distributed systems.

- Asynchronous distributed systems

☞ Many distributed systems (including those on the Internet) are asynchronous. - No bound on process execution time (nothing can be assumed about speed, load, and reliability of computers). - No bound on message transmission delays (nothing can be assumed about speed, load, and reliability of interconnections) - No bounds on drift rates between local clocks.

Important consequences:

1. In an asynchronous distributed system there is no global physical time. Reasoning can be only in terms of logical time (see lecture on time and state).
2. Asynchronous distributed systems are unpredictable in terms of timing.
3. No timeouts can be used.

☞ Asynchronous systems are widely and successfully used in practice.

In practice timeouts are used with asynchronous systems for failure detection.

However, additional measures have to be applied in order to avoid duplicated messages, duplicated execution of operations, etc.

3. Fault Models

☞ Failures can occur both in processes and communication channels. The reason can be both software and hardware faults.

☞ Fault models are needed in order to build systems with predictable behavior in case of faults (systems which are fault tolerant).

☞ such a system will function according to the predictions, only as long as the real faults behave as defined by the “fault model”.