A quality product by
Brainheaters™ LLC

Brainheaters Notes

DC

COMPS Semester-8

'C' SCHEME - 2022-2023

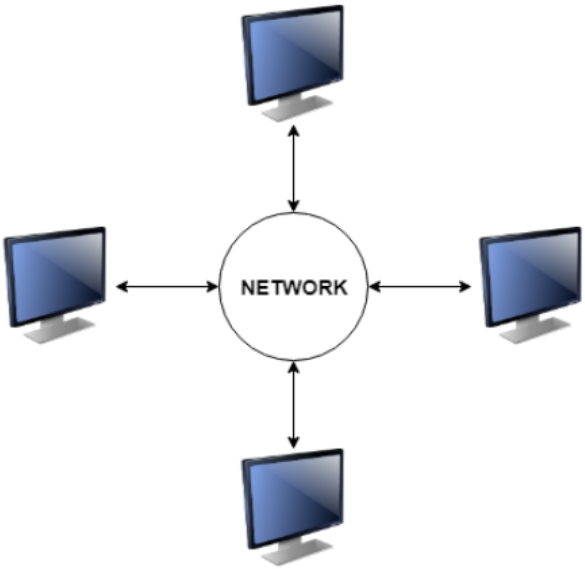
MODULE-1

Q1. Distributed System? Explain its Characteristics and name its types.

(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Distributed computing and distributed databases, a distributed system is a collection of independent components located on different machines that share messages with each other in order to achieve common goals.

- As such, the distributed system will appear as if it is one interface or computer to the end-user.
- The hope is that together, the system can maximize resources and information while preventing failures, as if one system fails, it won't affect the availability of the service.



DISTRIBUTED OPERATING SYSTEM

- The machines that are a part of a distributed system may be computers, physical servers, virtual machines, containers, or any other node that can connect to the network, have local memory, and communicate by passing messages.

There are mainly two types, they are as follows:

1. Client/Server Systems
   - In this system, the client requests the server for a resource. On the other hand, the server provides this resource to the client. One client contacts only a single server at a time. Whereas a single server can deal with multiple clients simultaneously. The clients and servers connect through a computer network in the system.
2. Peer to Peer Systems
   - In this system, the nodes play an important role. All the work equally divides among the nodes. Furthermore, these nodes can share data or resources as per the requirement. Again, they require a network to connect.

Features/Characteristics of Distributed System

The features are as follows:

1. Resource Sharing
   - The main important feature of this system is that it allows users to share resources.
   - Moreover, they can share resources in a secure and controlled manner. Resources can be of any type.
   - For example, some common resources which are shared can be printers, files, data, storage, web pages, etc.
2. Openness
   - This means that the services which the system provides are openly displayed through interfaces.

- Moreover, these interfaces provide only the syntax of the services.
- For example, the type of functions, their return types, parameters, etc. These interfaces use Interface Definition Languages (IDL).

3. Concurrency
    - It means that several tasks take place at different nodes of the system simultaneously.
    - Moreover, these tasks can also interact with each other. It results in increasing the efficiency of the system.
4. Scalability
    - It refers to the fact that the efficiency of the system should not change when more nodes are added to the system.
    - Moreover, the performance for the system with 100 nodes should be equal to the system with 1000 nodes.
5. Fault Tolerance
    - It means that the user can still work with the system in the case, hardware, or software fails.
6. Transparency
    - It is the most important feature of the system. The main goal of a distributed OS is to hide the fact that the resources are being shared.
    - Furthermore, transparency means that the user should not know that the resources he is using are shared.
    - Moreover, for the user, the system should be a separate individual unit.

Q2. What is a Distributed System? Explain any issues in such systems.

(P4 - Appeared 1 Time) (5-10 Marks)

Ans: A distributed operating system is an important type of operating system. An operating system is basically a program that acts as an interface between the system hardware and the user.

- Moreover, it handles all the interactions between the software and the hardware.
- A distributed operating system is one in which several computer systems are connected through a single communication channel.
- Moreover, these systems have their individual processors and memory.
- Furthermore, these processors communicate through high-speed buses or telephone lines.
- These individual systems that connect through a single channel are considered as a single unit.
- We can also call them loosely coupled systems. The individual components or systems of the network are nodes.

Issues in designing distributed systems:

1. Heterogeneity
    - The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks.
    - The Internet consists of many different sorts of networks; 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.
2. 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.

3. 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, integrity, and availability.
4. 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.
5. 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.
6. 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. Therefore any programmer who takes an implementation of an object that was not intended for use in a distributed system must do whatever is necessary to make it safe in a concurrent environment.

7. Transparency
    - Transparency can be achieved at two 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.
        - Location transparency: The users cannot tell where resources are located
        - Migration transparency: Resources can move at will without changing their names
        - Replication transparency: The users cannot tell how many copies exist.
        - Concurrency transparency: Multiple users can share resources automatically.
        - Parallelism transparency: Activities can happen in parallel without users knowing.
8. 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.

9. 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.
10. 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. Unfortunately, achieving this is easier said than done.

Q3. Explain Goals/Features of Distributed Systems. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: The four important goals that should be met for an efficient distributed system are as follows:

1. Connecting Users and Resources:
    - The main goal of a distributed system is to make it easy for users to access remote resources and to share them with others in a controlled way.
    - It is cheaper to let a printer be shared by several users than buying and maintaining printers for each user.
    - Collaborating and exchanging information can be made easier by connecting users and resources.
2. Transparency:
    - It is important for a distributed system to hide the location of its process and resource. A distributed system that can portray itself as a single system is said to be transparent.
    - The various transparencies need to be considered are access, location, migration, relocation, replication, concurrency, failure and persistence.
    - Aiming for distributed transparency should be considered along with performance issues.
3. Openness:
    - Openness is an important goal of a distributed system in which it offers services according to standard rules that describe the syntax and semantics of those services.
    - Open distributed system must be flexible making it easy to configure and add new components without affecting existing components. An open distributed system must also be extensible.
4. Scalable:
    - Scalability is one of the most important goals which are measured along three different dimensions.
    - First, a system can be scalable with respect to its size which can add more users and resources to a system.
    - Second, users and resources can be geographically apart.

- Third, it is possible to manage even if many administrative organizations are spanned.

Features/Characteristics of Distributed System

The features are as follows:

1. Resource Sharing
    - The main important feature of this system is that it allows users to share resources.
    - Moreover, they can share resources in a secure and controlled manner.
    - Resources can be of any type. For example, some common resources which are shared can be printers, files, data, storage, web pages, etc.
2. Openness
    - This means that the services which the system provides are openly displayed through interfaces.
    - Moreover, these interfaces provide only the syntax of the services.
    - For example, the type of functions, their return types, parameters, etc. These interfaces use Interface Definition Languages (IDL).
3. Concurrency
    - It means that several tasks take place at different nodes of the system simultaneously.
    - Moreover, these tasks can also interact with each other. It results in increasing the efficiency of the system.
4. Scalability
    - It refers to the fact that the efficiency of the system should not change when more nodes are added to the system.
    - Moreover, the performance for the system with 100 nodes should be equal to the system with 1000 nodes.

5. Fault Tolerance

- It means that the user can still work with the system in the case, hardware, or software fails.

6. Transparency

- It is the most important feature of the system. The main goal of a distributed OS is to hide the fact that the resources are being shared.
- Furthermore, transparency means that the user should not know that the resources he is using are shared.
- Moreover, for the user, the system should be a separate individual unit.

Q4. Write a Short note on Distributed Systems and explain its limitations. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: Distributed System is a collection of self-governing computer systems efficient of transmission and cooperation among each other by the means of interconnections between their hardware and software.

- It is a collection of loosely coupled processors that appears to its users as a single systematic system.
- Distributed systems have various limitations such as in distributed systems there is not any presence of a global state.
- This differentiates distributed system computing from databases in which a steady global state is maintained.
- Distributed system limitations have the impact on both design and implementation of distributed systems.

There are mainly two limitations of the distributed system which are as following:

1. Absence of a Global Clock:
    - In a distributed system there are a lot of systems and each system has its own clock.

- Each clock on each system is running at a different rate or granularity leading to them being asynchronous.
- In starting the clocks are regulated to keep them consistent, but only after one local clock cycle they are out of the synchronization and no clock has the exact time.
- Time is known for a certain precision because it is used for the following in distributed system:
    - Temporal ordering of events
    - Collecting up-to-date information on the state of the integrated system
    - Scheduling of processes
- There are restrictions on the precision of time by which processes in a distributed system can synchronize their clocks due to asynchronous message passing.
- Every clock in a distributed system is synchronized with a more reliable clock, but due to transmission and execution time lapses the clocks become different.
- Absence of global clock make more difficult the algorithm for designing and debugging of distributed system.

2. Absence of Shared Memory:
    - Distributed systems do not have any physically shared memory, all computers in the distributed system have their own specific physical memory.
    - As computers in the distributed system do not share the common memory, it is impossible for any one system to know the global state of the full distributed system.
    - Processes in the distributed system obtains a coherent view of the system but in actuality that view is a partial view of the system.

- As in distributed systems there is an absence of a global state, it is challenging to recognize any global property of the system.
- The global state in a distributed system is divided by many computers into smaller entities.

Q5. Difference between Grid and Cluster Computing Models. (P4 - Appeared 1 Time) (5-10 Marks)

Ans:

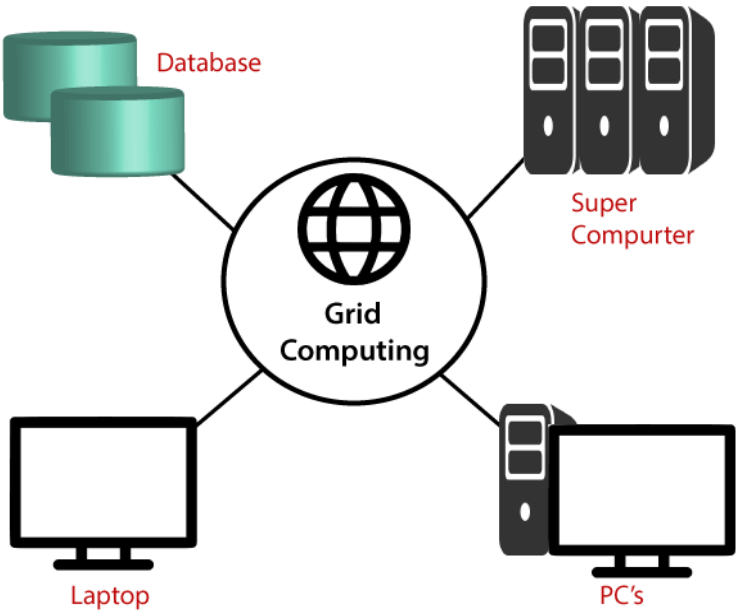
Cluster Computing	Grid Computing
Nodes must be homogeneous i.e. they should have the same type of hardware and operating system.	Nodes may have different Operating systems and hardwares. Machines can be homogeneous or heterogeneous.
Computers in a cluster are dedicated to the same work and perform no other task.	Computers in a grid contribute their unused processing resources to the grid computing network.
Computers are located close to each other.	Computers may be located at a huge distance from one another.
Computers are connected by a high speed local area network bus.	Computers are connected using a low speed bus or the internet.

Computers are connected in a centralized network topology.	Computers are connected in a distributed or decentralized network topology.
Scheduling is controlled by a central server.	It may have servers, but mostly each node behaves independently.
Whole system has a centralized resource manager.	Every node manages its resources independently.
Whole system functions as a single system.	Every node is autonomous, and anyone can opt out anytime.
Cluster computing is used in areas such as WebLogic Application Servers, Databases, etc.	Grid computing is used in areas such as predictive modeling, Automation, simulations, etc.
It has Centralized Resource management.	It has Distributed Resource Management.

Q6. Write a Short note on the Grid Computing model. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: Grid Computing can be defined as a network of computers working together to perform a task that would rather be difficult for a single machine.

- All machines on that network work under the same protocol to act as a virtual supercomputer.
- The task that they work on may include analyzing huge datasets or simulating situations that require high computing power.
- Computers on the network contribute resources like processing power and storage capacity to the network.
- Grid Computing is a subset of distributed computing, where a virtual supercomputer comprises machines on a network connected by some bus, mostly Ethernet or sometimes the Internet.
- It can also be seen as a form of Parallel Computing where instead of many CPU cores on a single machine, it contains multiple cores spread across various locations.
- The concept of grid computing isn't new, but it is not yet perfected as there are no standard rules and protocols established and accepted by people.



Working:

A Grid computing network mainly consists of these three types of machines

1. Control Node: A computer, usually a server or a group of servers which administers the whole network and keeps the account of the resources in the network pool.
2. Provider: The computer contributes its resources to the network resource pool.
3. User: The computer that uses the resources on the network.

Advantages of Grid Computing:

1. It is not centralized, as there are no servers required, except the control node which is just used for controlling and not for processing.
2. Multiple heterogeneous machines i.e. machines with different Operating Systems can use a single grid computing network.
3. Tasks can be performed parallelly across various physical locations and the users don't have to pay for them (with money).

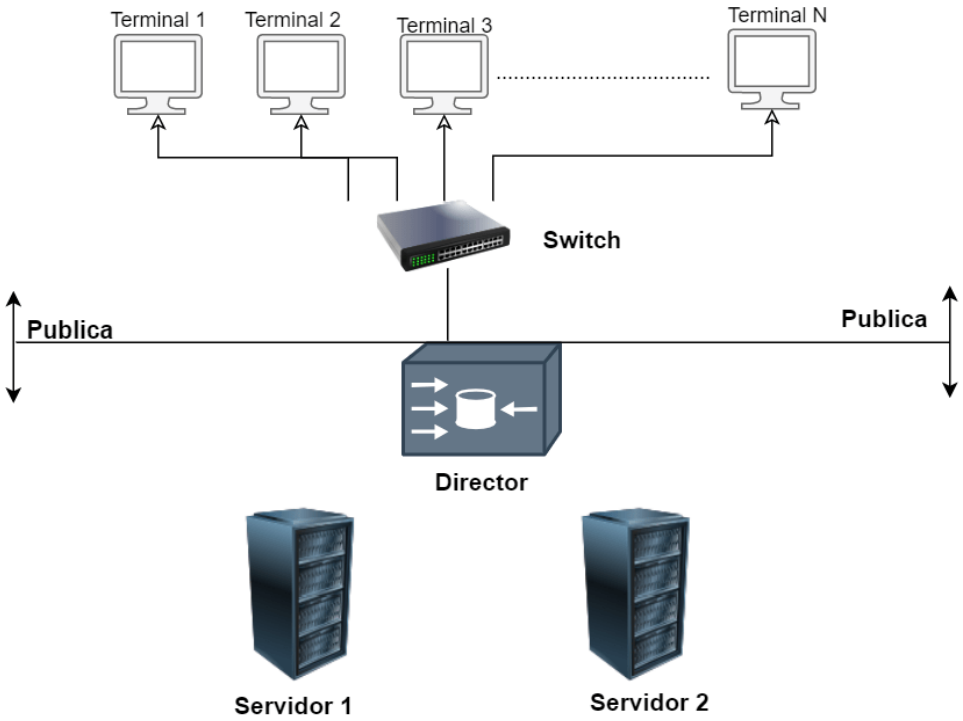
Disadvantages of Grid Computing :

1. The software of the grid is still in the involution stage.
2. A super fast interconnect between computer resources is the need of hour.
3. Licensing across many servers may make it prohibitive for some applications.
4. Many groups are reluctant to share resources .

Q7. Write a Short note on Cluster Computing models. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: Cluster computing defines several computers linked on a network and implemented like an individual entity. Each computer that is linked to the network is known as a node.

- Cluster computing provides solutions to solve difficult problems by providing faster computational speed, and enhanced data integrity.
- The connected computers implement operations all together thus generating the impression like a single system (virtual device). This procedure is defined as the transparency of the system.



Advantages of Cluster Computing

The advantages of cluster computing are as follows -

- Cost-Effectiveness - Cluster computing is considered to be much more cost effective. These computing systems provide boosted implementation concerning the mainframe computer devices.

- Processing Speed - The processing speed of cluster computing is validated with that of the mainframe systems and other supercomputers demonstrated around the globe.
- Increased Resource Availability - Availability plays an important role in cluster computing systems. Failure of some connected active nodes can be simply transformed onto different active nodes on the server, providing high availability.
- Improved Flexibility - In cluster computing, better description can be updated and improved by inserting unique nodes .

Types of Cluster Computing

The types of cluster computing are as follows -

1. High Availability (HA) and Failover Clusters
    - These cluster models generate the availability of services and resources in an uninterrupted technique using the system's implicit redundancy.
    - The basic term of Cluster is that if a node declines, then applications and services can be made available to different nodes.
    - These methods of clusters deliver as the element for critical missions, mails, documents, and application servers.
2. Load Balancing Clusters
    - This cluster allocates all the incoming traffic/requests for resources from nodes that run the equal programs and machines.
    - In this cluster model, some nodes are answerable for tracking orders, and if a node declines, therefore the requests are distributed amongst all the nodes available.
    - Such a solution is generally used on web server farms.

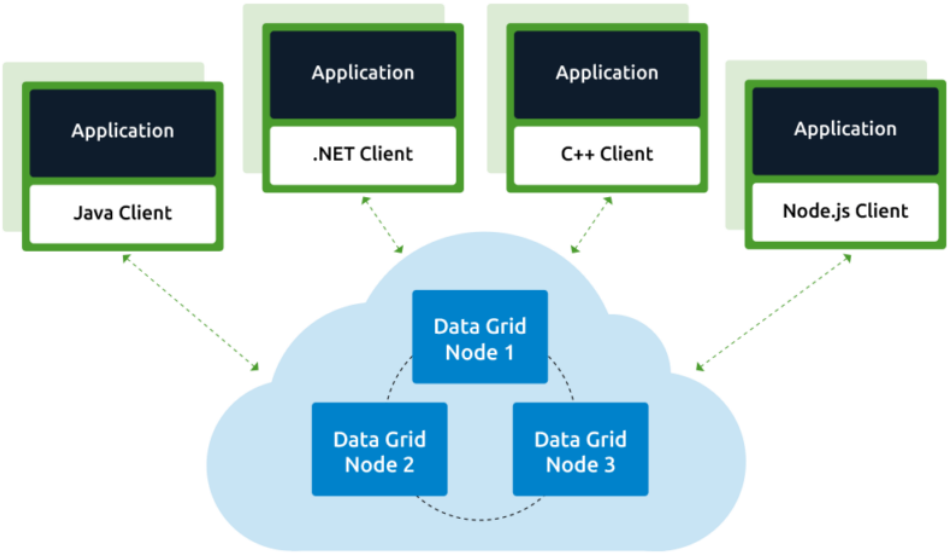
3. HA & Load Balancing Clusters
    - This cluster model associates both cluster features, resulting in boost availability and scalability of services and resources.
    - This kind of cluster is generally used for email, web, news, and FTP servers.
4. Distributed & Parallel Processing Clusters
    - This cluster model boosts availability and implementation for applications that have huge computational tasks.
    - A large computational task has been divided into smaller tasks and distributed across the stations.
    - Such clusters are generally used for numerical computing or financial analysis that need high processing power.

Q8. Explain working of Grid Computing model in Distributed system with its advantages and disadvantages.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Grid computing is the practice of leveraging multiple computers, often geographically distributed but connected by networks, to work together to accomplish joint tasks.

- It is typically run on a "data grid," a set of computers that directly interact with each other to coordinate jobs.
- Grid computing works by running specialized software on every computer that participates in the data grid.
- The software acts as the manager of the entire system and coordinates various tasks across the grid. Specifically, the software assigns subtasks to each computer so they can work simultaneously on their respective subtasks.
- After the completion of subtasks, the outputs are gathered and aggregated to complete a larger-scale task.

- The software lets each computer communicate over the network with the other computers so they can share information on what portion of the subtasks each computer is running, and how to consolidate and deliver outputs.



- With grid computing, specialized software runs on every computer that participates in the data grid.
- This controller software acts as the manager of the entire system and coordinates various tasks across the grid.

Advantages Of Grid Computing :

- They are not that expensive.
- They are quite efficient and reliable machines and can solve complex problems in limited time.
- They are scalable.
- Grid Computing follows distributed computing architecture.
- Grid Computing is application oriented.
- They work in a decentralized management system.

- They can use existing hardware.
- Can easily associate with other organizations.
- Tasks and instructions can be performed in parallel

Disadvantages Of Grid Computing:

- They are not interactive for job submissions.
- Grid system is not fully evolved.
- Learning curve to get started
- Difficult in sharing resources across different admins.
- Grid environment can work with smaller servers.
- Some applications may not work with full potential.

Q9. Explain Distributed System Models.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Distributed System Models is as follows:

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

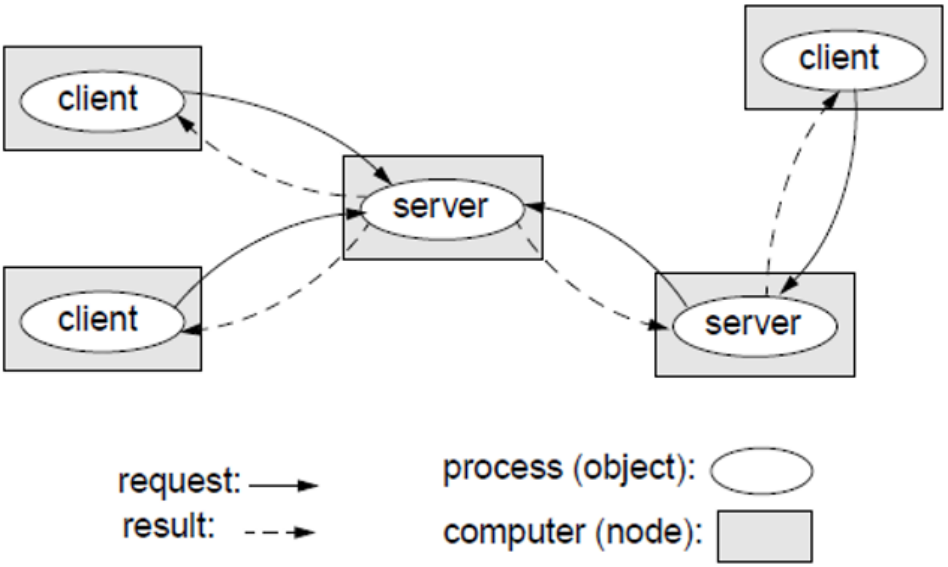
1. Architectural Models

- Architectural model describes responsibilities distributed between system components and how these components are 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):

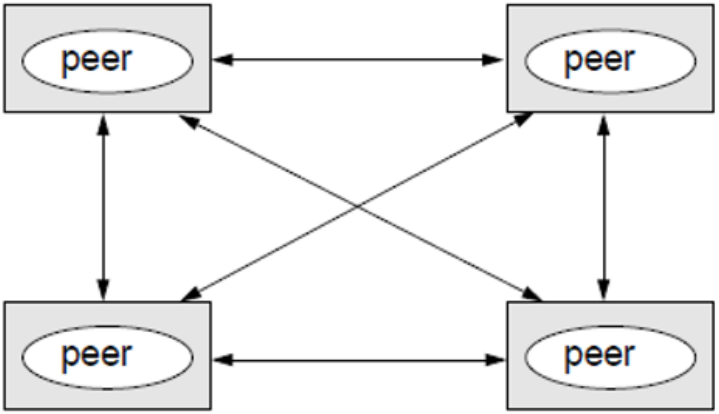
- The client sends a request (invocation) message to the server asking for some service;
- The server does the work and returns a result (e.g. the data requested) or an error code if the work could not be performed.



- 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 a 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 a potentially large number of replicas.

2 .Interaction Model

- Interaction models 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 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.

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 duplicate messages, duplicated execution of operations, etc.

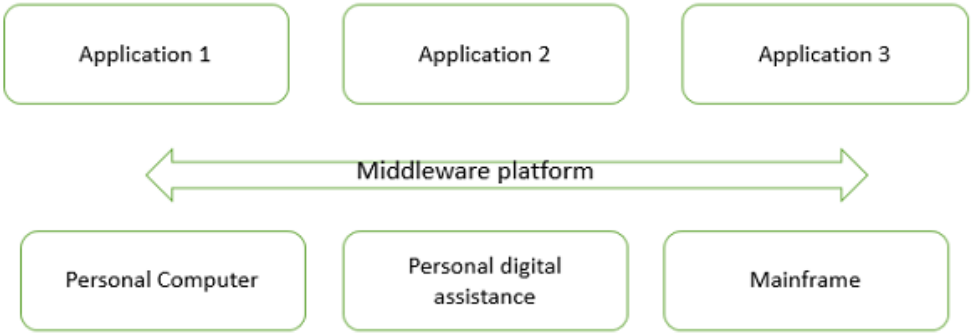
3. Fault Models

- 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”.

Q10. What is the role of Middleware in a Distributed System and mention the services provided by it.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: In a distributed system, middleware is a software component that serves between two or more applications.

Middleware usually resides between the operating system and the end user or end-user application. It provides essential features that the operating system doesn't offer. The term usually refers to large software products, such as database managers, transaction monitors, and web servers.



Middleware as an infrastructure for distributed system

Middleware can perform numerous functions such as:

- It manages connections to various backend resources. Middleware components can create connection pools to provide fast and efficient access to popular backend databases.
- Second, middleware software can implement logic based on client requirements. For example, the middleware component can detect that the language header of the client browser making a particular request is set to English. As a result, queries sent to the backend will adjust to returning only English-based results.
- Third, middleware is essential in concurrency, load balancing, and transaction management. It's scaled up and down to distribute incoming client requests across multiple servers, virtual machines, or cloud availability zones.
- Finally, middleware plays an essential role in protecting access to backend resources. It requires a secure connection using technology such as SSL and a username/password combination or digital certificate authentication.

Q11. Specify and Explain the models of Middleware.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: The first middle ware model has started with the distributed file system, where the files were stored and distributed over the network. Models of middle ware are as follows:

- Remote Procedure call:
    - It is one of the successful middleware models used in modern distributed applications for communication.
    - It uses a local call to call a procedure residing 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 a 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.
- 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 queues 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 a queuing mechanism between the client and server to avoid the message being misplaced.
    - It is asynchronous mechanism where messages can be sent even though the receiver is not available For eg Email system
- 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.

- Remote Method Invocation
    - These objects are distributed and located by using the RMI registry.
    - The client can access remote objects by using the interfaces.

Disadvantage

- It didn't support the concept of heterogeneity and is compatible with java platform only. Common object request broker architecture(CORBA)-
- It is one of the most popular distributed object technologies where objects can be accessible from remote locations through ORB.
- Server and client communicate with each other through object request broker bus.
- To map the semantics of objects and fetch the appropriate object an interface definition language is used.
- It is evolved with service based middleware where service models are used.
- In the service model, the services are published by the service providers and consumed by the service consumer.

Q12. Difference Between NOS and DOS.(P4 - Appeared 1 Time) (5-10 Marks)

Ans:

Features	Network operating system	Distributed operating system
Scalability	Higher Scalable Less Scalable	

Definition	Network operating systems are server-based operating systems that provide networking-related functionality.	The distributed operating system manages a set of independent, networked, communicating computers and makes them look like an ordinary centralized operating system.
Objective	Its primary objective is to give local services to remote users.	Its main objective is to manage hardware resources.
Communication	Its communication is file-based or shared folder-based.	Its communication is mostly message-based or shared memory-based.
Resource Management	Resources are managed at every node.	In the distributed operating system, global central or distributed management is used to manage resources.
Coupled System	The loosely coupled system is used in heterogeneous computers.	The tightly coupled system is used in homogeneous computers.
Fault Tolerance	Its fault tolerance is less.	Its fault tolerance is high.
Nodes	All nodes can have a different operating system.	All nodes have the same operating system.

Ease of Implementation	High	Less
Rate of autonomy	The rate of autonomy is high.	The rate of autonomy is less.

Q13. Write a short note on NOS. Types of NOS and its features(P4 - Appeared 1 Time) (5-10 Marks)

Ans: A network operating system(NOS) is software that connects multiple devices and computers on the network and allows them to share resources on the network. Let's see what are the functions of the network operating system.

Functions of the NOS :

Following are the main functions of NOS :

- Creating and managing user accounts on the network.
- Controlling access to resources on the network.
- Provide communication services between the devices on the network.
- Monitor and troubleshoot the network.
- Configuring and Managing the resources on the network.

Types of Network operating systems : There are mainly two types of networks, one is peer to peer and another is client/server. Now let's see each type one by one.

- Peer to Peer –
  Peer-to-peer network operating systems allow sharing resources and files with small-sized networks and having fewer resources. In general, peer-to-peer network operating systems are used on LAN.

- Client/server –
  Client-server network operating systems provide users access to resources through the central server. This NOS is too expensive to implement and maintain. This operating system is good for the big networks which provide many services.

Features of network operating systems :

Let's see what are the functions of the network operating system.

- Printers and application sharing on the network.
- File systems and database sharing.
- Provide good security by using functionality like user authentication and access control.
- Create backups of data.
- Inter-networking.

Q14. What is DOS? Give its types and advantages.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Dos is one of the first operating systems designed and developed by software giant Microsoft.

- The amazing features and characteristics of the dos operating system made them very popular among users, and soon it ruled the IT industry. Dos entirely changed the market as it was user-friendly to use and handle.
- DOS is a single-user and single-tasking operating system that is operated with the help of hard disk drives. It is a text-based, or character user interface operating system that does not support graphics and animations.
- Do use and utilize the command line for executing commands and instructions given to the computer system to receive desired output. Dos acts as an interface between the user and the computer system.

- Dos is a light weighted operating system that also requires less memory for processing and execution; therefore, the time required for the booting process is considerably lesser than other windows operating systems.
- Dos manages disk and disk files. It also acts as a bridge between computer hardware and software applications.

Types of MS-DOS Commands:

There are mainly two types of MS-DOS commands:

1. Internal Commands: Internal commands are those commands that are loaded automatically in the memory when DOS is loaded into memory during the booting process. These commands are easier to learn and use. They require no external files for their storage as in the case of external commands. These are for performing a basic operation on files and in directories. They do not need any external file support. These commands are used for common jobs such as copying and erasing files.
2. External Commands: These external commands are for performing advanced tasks and they do not need some external file support as they are not stored in COMMAND.com. The external commands are used less frequently and are stored in some external files which are stored in some secondary storage devices. Whenever an external command is to be executed then the external file in which that particular command is stored is transferred from the secondary storage disk to the main memory(RAM).

Advantages

The advantages of MS-DOS Operating System are as follows -

- MS-DOS is a lightweight system and it allows direct access to all hardware with the help of the command line.

- This operating system is very lightweight.
- It also does not support multitasking therefore there is less overhead and less latency.
- MS_DOS boots the system faster than any other operating system.

Disadvantages

The disadvantages of MS-DOS Operating System are as follows -

- It has a command line user interface therefore it is very less user friendly.
- Very few applications are supported in DOS.
- It is deprecated by Microsoft that's why no more updates will be available to MS DOS.
- It does not support multitasking. So, only one application can be run on this OS at a time.

MODULE-2

Q1. What is IPC? Give its Functions and Characteristics.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Interprocess Communication is a process of exchanging the data between two or more independent processes in a distributed environment is called Interprocess communication.

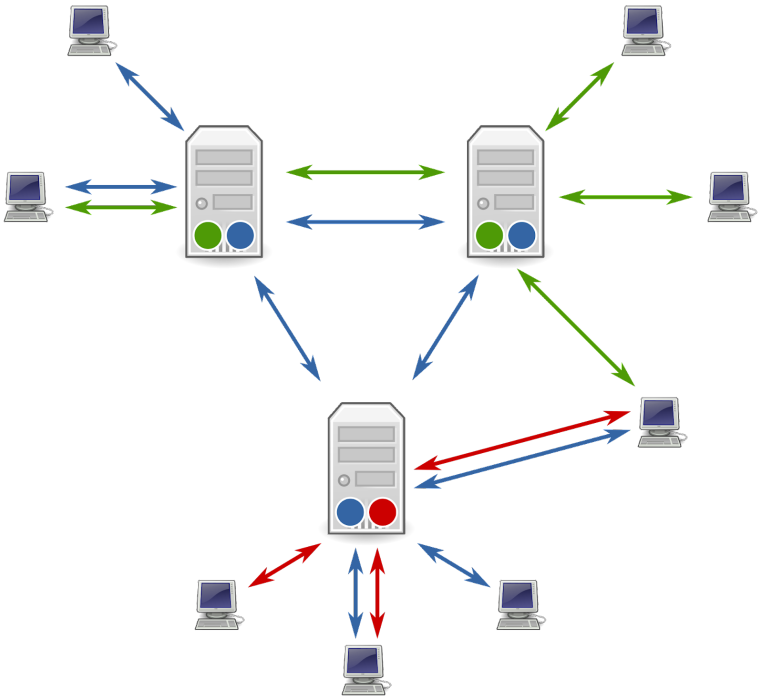
- Interprocess communication on the internet provides both Datagram and stream communication.

Inter process communication (IPC) is used for exchanging data between multiple threads in one or more processes or programs.

- The Processes may be running on single or multiple computers connected by a network. The full form of IPC is Inter-process communication.
- It is a set of programming interfaces which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system. This allows a specific program to handle many user requests at the same time.
- Since every single user request may result in multiple processes running in the operating system, the process may require communicating with each other. Each IPC protocol approach has its own advantages and limitations, so it is not unusual for a single program to use all of the IPC methods.

Examples Of Interprocess Communication:

1. N number of applications can communicate with the X server through network protocols.
2. Servers like Apache spawn child processes to handle requests.
3. Pipes are a form of IPC: grep foo file | sort



It has two functions:

1. Synchronization:
   Exchange of data is done synchronously which means it has a single clock pulse.
2. Message Passing:
   When processes wish to exchange information. Message passing takes several forms such as: pipes, FIFO, Shared Memory, and Message Queues.

Characteristics Of Inter-process Communication:

There are mainly five characteristics of inter-process communication in a distributed environment/system.

1. Synchronous System Calls:
   In the synchronous system calls both sender and receiver use blocking system calls to transmit the data which means the

sender will wait until the acknowledgment is received from the receiver and receiver waits until the message arrives.

2. Asynchronous System Calls:
   In the asynchronous system calls, both sender and receiver use non-blocking system calls to transmit the data which means the sender doesn't wait for the receiver to acknowledge.
3. Message Destination:
   A local port is a message destination within a computer, specified as an integer. Sport has exactly one receiver but many senders. Processes may use multiple ports from which to receive messages. Any process that knows the number of a port can send the message to it.
4. Reliability:
   It is defined as validity and integrity.
5. Integrity:
   Messages must arrive without corruption and duplication to the destination.
6. Validity:
   Point to point message services are defined as reliable, If the messages are guaranteed to be delivered without being lost is called validity.
7. Ordering:
   It is the process of delivering messages to the receiver in a particular order. Some applications require messages to be delivered in the sender order i.e the order in which they were transmitted by the sender.

Q2. Explain IPC in Distributed Systems.(P4 - Appeared 1 Time) (5-10 Marks)

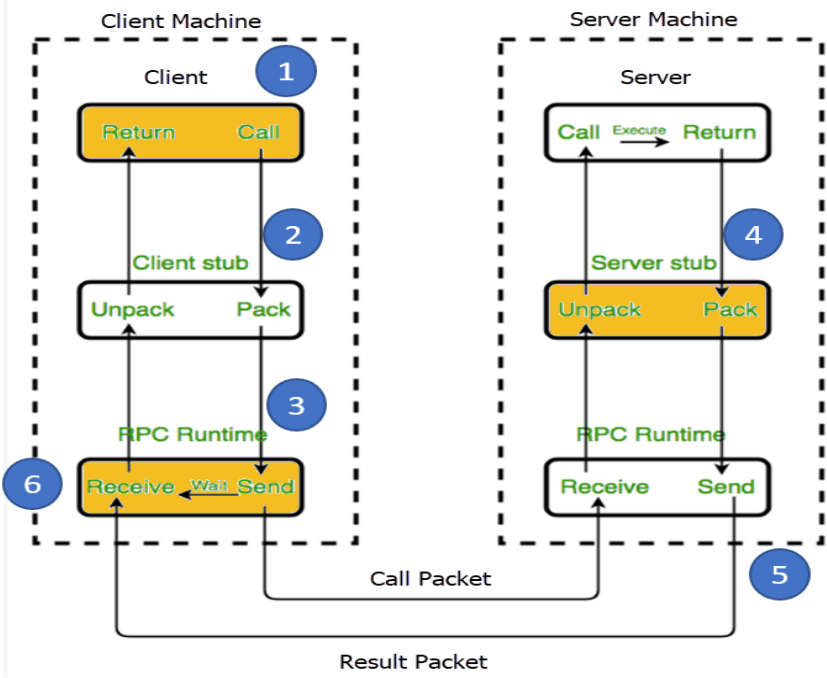
Ans: Inter-process communication (IPC) is the activity of sharing data across multiple and commonly specialized processes using communication protocols.

- Typically, applications using IPC are categorized as clients and servers, where the client requests data and the server responds to client requests.
- Many applications are both clients and servers, as commonly seen in distributed computing.
- Methods for achieving IPC are divided into categories which vary based on software requirements, such as performance and modularity requirements, and system circumstances, such as network bandwidth and latency.
  There are several reasons for implementing inter-process communication systems:
    - Sharing information; for example, web servers use IPC to share web documents and media with users through a web browser.
    - Distributing labor across systems; for example, Wikipedia uses multiple servers that communicate with one another using IPC to process user requests.
    - Privilege separation; for example, HMI software systems are separated into layers based on privileges to minimize the risk of attacks. These layers communicate with one another using encrypted IPC.

Q3. Explain RPC model and its types.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Remote Procedure Call (RPC) is an interprocess communication technique.

- The Full form of RPC is Remote Procedure Call. It is used for client-server applications. RPC mechanisms are used when a computer program causes a procedure or subroutine to execute in a different address space, which is coded as a normal procedure call without the programmer specifically coding the details for the remote interaction.
- This procedure call also manages low-level transport protocols, such as User Datagram Protocol, Transmission Control Protocol/Internet Protocol etc. It is used for carrying the message data between programs.



RPC Architecture

RPC architecture has mainly five components of the program:

1. Client
2. Client Stub
3. RPC Runtime
4. Server Stub
5. Server

Following steps take place during the RPC process:

1. Step 1) The client, the client stub, and one instance of RPC runtime execute on the client machine.
2. Step 2) A client starts a client stub process by passing parameters in the usual way. The client stub stores within the client's own address space. It also asks the local RPC Runtime to send back to the server stub.
3. Step 3) In this stage, RPC is accessed by the user by making regular Local Procedural Cal. RPC Runtime manages the transmission of messages between the network across client and server. It also performs the job of retransmission, acknowledgment, routing, and encryption.
4. Step 4) After completing the server procedure, it returns to the server stub, which packs (marshalls) the return values into a message. The server stub then sends a message back to the transport layer.
5. Step 5) In this step, the transport layer sends back the result message to the client transport layer, which returns a message to the client stub.
6. Step 6) In this stage, the client stub marshalls (unpack) the return parameters, in the resulting packet, and the execution process returns to the caller.

Types of RPC:

Callback RPC: In a Callback RPC, a P2P (Peer-to-Peer)paradigm opts between participating processes. In this way, a process provides both client and server functions which are quite helpful. Callback RPC's features include:

- The problems encountered with interactive applications that are handled remotely
- It provides a server for clients to use.
- Due to the callback mechanism, the client process is delayed.
- Deadlocks need to be managed in callbacks.
- It promotes a Peer-to-Peer (P2P) paradigm among the processes involved.

RPC for Broadcast: A client's request that is broadcast all through the network and handled by all servers that possess the method for handling that request is known as a broadcast RPC. Broadcast RPC's features include:

- You have an option of selecting whether or not the client's request message ought to be broadcast.
- It also gives you the option of declaring broadcast ports.
- It helps in diminishing physical network load.

Batch-mode RPC: Batch-mode RPC enables the client to line and separate RPC inquiries in a transmission buffer before sending them to the server in a single batch over the network. Batch-mode RPC s features include:

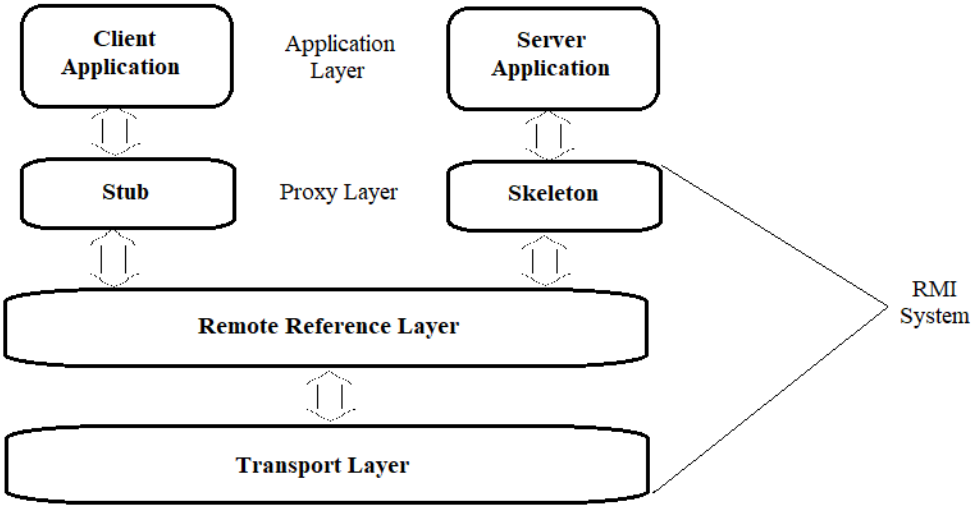
- It diminishes the overhead of requesting the server by sending them all at once using the network.
- It is used for applications that require low call rates.
- It necessitates the use of a reliable transmission protocol.

Q4. Explain working of RMI.(P4 - Appeared 1 Time) (5-10 Marks)

Ans:Java Remote Method Invocation (RMI)

- The RMI stands for Remote Method Invocation is an API mechanism.
- This mechanism allows an object residing in one system (JVM) to access an object running on another JVM.
- This mechanism generally creates distributed applications in java.
- The RMI provides remote communication between the java applications using two objects called stub and skeleton.

The Architecture of Remote Method Invocation (RMI)



- A remote object is an object whose method can be invoked from another JVM.
- Java RMI application contains two types of programs such as server program and client program.
- In the server-side program, a remote object is created, and a reference of that remote object is made available for the client-side using a registry.

- The client-side program requests the remote objects on the server and tries to invoke its methods.

Components of RMI Architecture

Stub

- The stub is an object, acts as a gateway for the client-side.
- All the outgoing requests are routed through it.
- It resides at the client-side and represents the remote object.
- When the caller invokes a method on the stub object, it does the following tasks:
- It initiates a connection with remote Virtual Machine (JVM),
- It writes and transmits (marshalls) the parameters to the remote Virtual Machine (JVM),
- It waits for the result
- It reads (us marshals) the return value or exception, and
- It finally returns the value to the caller.

Skeleton

- The skeleton is an object, acts as a gateway for the server-side object.
- All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:
- It reads the parameter for the remote method
- It invokes the method on the actual remote object, and
- It writes and transmits (marshalls) the result to the caller.

Transport Layer

- This layer connects the client and the server.
- It manages the existing connection and also sets up new connections.
- It is responsible for the transportation of data from one machine to another.
- The default connection is set up only in the TCP/IP protocol.

RRL(Remote Reference Layer)

- It is the layer that manages the references made by the client to the remote object.
- This layer gets a stream-oriented connection from the transport layer.
- It is responsible for dealing with the semantics of remote invocations.
- It is also responsible for handling duplicated objects and for performing implementation-specific tasks with remote objects.

Working of RMI

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called invoke() of the object remoteRef. It passes the request to the RRL on the server-side.
- The RRL on the server-side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

Packages used in RMI

- java.rmi - Provides the RMI package.
- java.rmi.activation - Provides support for RMI Object Activation.
- java.rmi.dgc - Provides classes and interface for RMI distributed garbage-collection (DGC).
- java.rmi.registry - Provides a class and two interfaces for the RMI registry.
- java.rmi.server - Provides classes and interfaces for supporting the server side of RMI.
- javax.activity - Contains Activity service-related exceptions thrown by the ORB machinery during unmarshalling.

- javax.management.remote.rmi - The RMI connector is a connector for the JMX Remote API that uses RMI to transmit client requests to a remote MBean server.
- javax.rmi - Contains user APIs for RMI-IIOP.
- javax.rmi.CORBA - Contains portability APIs for RMI-IIOP.
- javax.transaction - Contains three exceptions thrown by the ORB machinery during unmarshalling.
- org.omg.stub.java.rmi - Contains RMI-IIOP Stubs for the Remote types that occur in java.rmi package.

Q5. Difference between Message and Stream-Oriented Communication.(P4 - Appeared 1 Time) (5-10 Marks)

Ans:

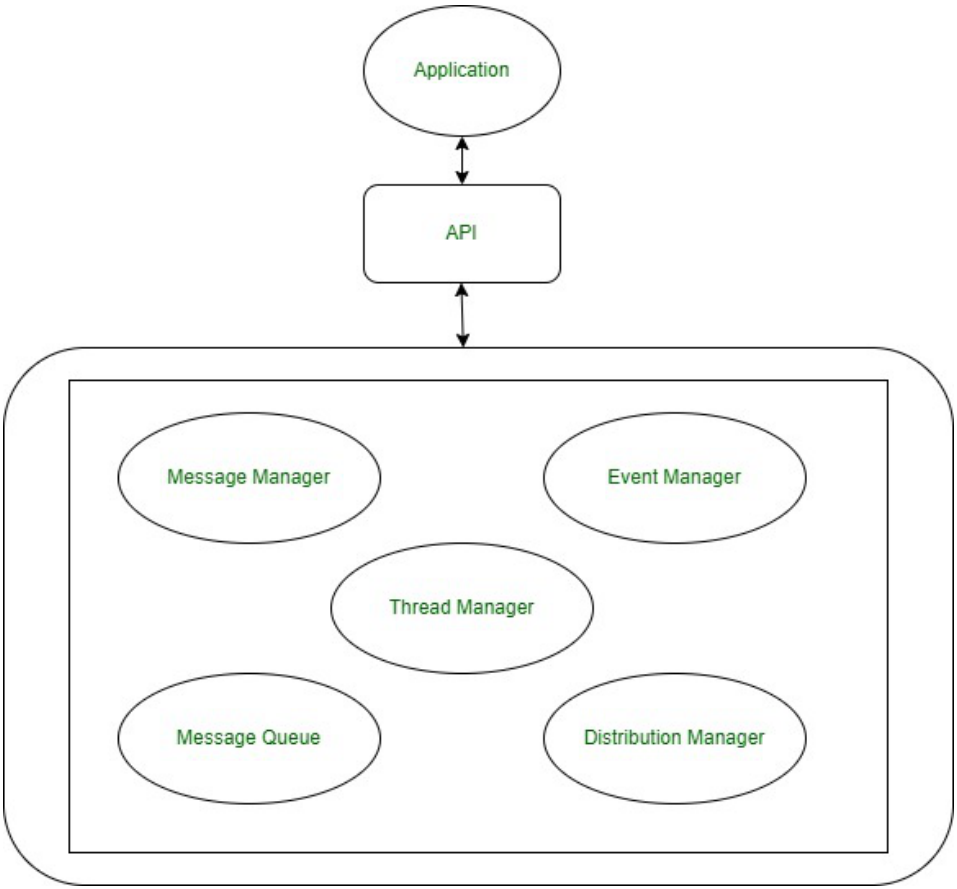
Message oriented communication	Stream oriented communication
UDP (user datagram protocol) uses manage oriented communication	TCP (transmission contact protocol) uses stream oriented communication
Data is sent by application in discrete packages called messages.	Data is sent with no particular structure.
Communication is connection less, data is sent without any setup.	Communication is oriented, connection established before comm.
It is unreliable best effort delivery without acknowledgement.	It is reliable, data acknowledged.

Re transmission is not performed.	Lost data is reframed automatically.
Low overhead.	High overhead.
No flow control.	Flow control using sent protocol like sliding
Transmission speed is very high as compared to stream-oriented.	Transmission speed is lower as compared to message oriented.
Suitable for applications like audio, video where speed is critical than loss of messages.	Suitable for applications like e-mail systems where data must be persistent through delivery late.

Q6. Explain Message-Oriented Communication.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Message Oriented Communication in Distributed Systems is one of the most popular communications paradigms that allows us to achieve high performance, scalability, and fault tolerance simultaneously.

- Message-oriented middleware (MOM) is software or hardware infrastructure supporting sending and receiving messages between distributed systems.
- MOM allows application modules to be distributed over heterogeneous platforms and reduces the complexity of developing applications that span multiple operating systems and network protocols.



- Message Passing refers to the communication between processes, whereas message queues refer to a linked list of messages stored within the kernel.
- Messages are put into the queue by a message producer and retrieved from it by a consumer.
- A message queue is software-engineered for sending messages between processes, applications, and servers. It enables an asynchronous communication methodology.
- A message queue receives and sends messages between loosely coupled microservices, applications, sockets, and other endpoints.

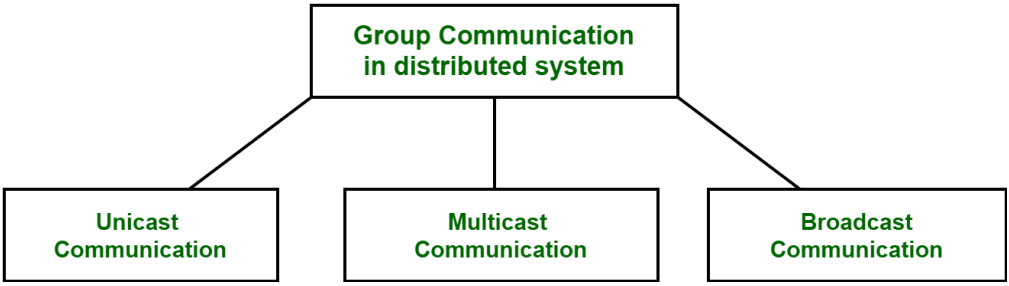
- The benefit of using a message queue is that the sender application does not need to wait for the receiver application to process the data. It can continue its work immediately, while the receiver pulls in the data later on.

Features and Capabilities

1. Unified messaging
2. Provisioning and monitoring
3. Dynamic scaling
4. Management and control tools
5. Dynamic scaling
6. Flexible service quality
7. Secure communication
8. Integration with other tool

Q7. What is group communication? Explain its types.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Communication between two processes in a distributed system is required to exchange various data, such as code or a file, between the processes.

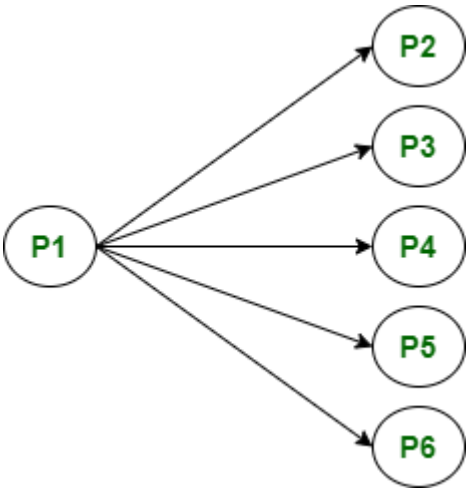


- When one source process tries to communicate with multiple processes at once, it is called Group Communication. A group is a collection of interconnected processes with abstraction.

- This abstraction is to hide the message passing so that the communication looks like a normal procedure call. Group communication also helps the processes from different hosts to work together and perform operations in a synchronized manner, therefore increasing the overall performance of the system.

Types of Group Communication in a Distributed System:

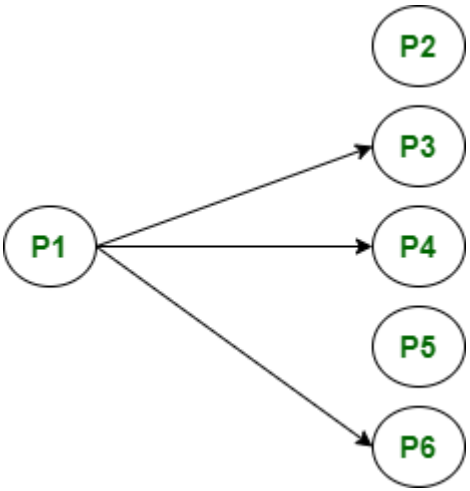
- Broadcast Communication : When the host process tries to communicate with every process in a distributed system at same time. Broadcast communication comes in handy when a common stream of information is to be delivered to each and every process in the most efficient manner possible. Since it does not require any processing whatsoever, communication is very fast in comparison to other modes of communication. However, it does not support a large number of processes and cannot treat a specific process individually.



A broadcast Communication: P1 process communicating with every *process in the system*

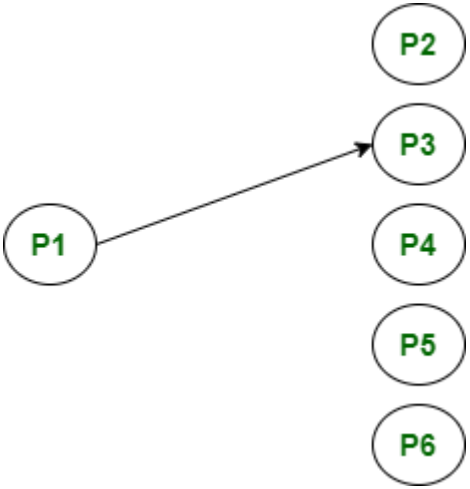
- Multicast Communication : When the host process tries to communicate with a designated group of processes in a

distributed system at the same time. This technique is mainly used to find a way to address the problem of a high workload on the host system and redundant information from processes in the system. Multitasking can significantly decrease time taken for message handling.



A multicast Communication: P1 process communicating with only a group *of the process in the system*

- Unicast Communication : When the host process tries to communicate with a single process in a distributed system at the same time. Although, the same information may be passed to multiple processes. This works best for two processes communicating as only it has to treat a specific process only. However, it leads to overheads as it has to find the exact process and then exchange information/data.



A unicast Communication: P1 process communicating with only P3 process

MODULE-3

Q1. Why is clock synchronization important in distributed systems?(P4 - Appeared 1 Time) (5-10 Marks)

Ans: The various clocks in the system even if set to a common time value at an instant, drift apart due to unavoidable reasons. Each node in distributed systems can share their resources with other nodes.

- So, there is a need for proper allocation of resources to preserve the state of resources and help coordinate between the several processes. To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.
- Hence some kind of continuous mechanism for synchronization is needed so that they can coordinate and work together to achieve the objectives of the distributed system. Two types of synchronization are possible- external synchronization and internal synchronization.
- To maintain the data i.e. to send, receive and manage the data between the systems at the same time in a synchronized manner you need a clock that has to be synchronized. This process to synchronize data is known as Clock Synchronization.
- As the distributed system has its own clocks. The time among the clocks may also vary. So, it is possible to synchronize all the clocks in a distributed environment.

Types of Clock Synchronization

- Physical clock synchronization
- Logical clock synchronization
- Mutual exclusion synchronization

Q2. How clock synchronization is implemented.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Clock synchronization is a method of synchronizing clock values of any two nodes in a distributed system with the use of external reference clock or internal clock value of the node. During the synchronization, many factors affect a network.

- Synchronization in distributed systems is achieved via clocks. The physical clocks are used to adjust the time of nodes. Each node in the system can share its local time with other nodes in the system. The time is set based on UTC (Universal Time Coordination).
- Clock synchronization is a method of synchronizing clock values of any two nodes in a distributed system with the use of external reference clock or internal clock value of the node. During the synchronization, many factors affect a network.
- To maintain the data i.e. to send, receive and manage the data between the systems at the same time in a synchronized manner you need a clock that has to be synchronized. This process to synchronize data is known as Clock Synchronization.
- As the distributed system has its own clocks. The time among the clocks may also vary. So, it is possible to synchronize all the clocks in a distributed environment.

Q3. What are types of clock synchronization?(P4 - Appeared 1 Time) (5-10 Marks)

Ans: As the distributed system has its own clocks. The time among the clocks may also vary. So, it is possible to synchronize all the clocks in a distributed environment.

Types of Clock Synchronization

- Physical clock synchronization

- Logical clock synchronization
- Mutual exclusion synchronization

(1) Physical Clock: - Time isn't a big issue in traditional centralized systems, where one or more CPUs share a common bus. The entire system shares the same understanding of time, right or wrong, it is consistent. - In distributed systems, this is not the case. Every system, though, has its own timer that keeps the clock running. These clocks are based on the oscillation of a piezoelectric crystal or a similar integrated circuit. They are not flawless, but they are relatively precise, reliable, and accurate. This implies that the clocks will differ from the correct time. Every timer is different in terms of characteristics — characteristics that might change with time, temperature. Thus, each system's time will drift away from the true time at a different rate — and perhaps in a different direction (slow or fast).It is feasible to coordinate physical clocks across several systems, but it will never be accurate. The drifting away from the real-time from each clock is something that happens in a distributed system.

(2) Logical Clock: - Logical clocks mean creating a protocol on all computers in a distributed system so that the computers can keep a uniform ordering of happenings inside some virtual time range. - In a distributed system, a logical clock is a technique for recording temporal and causative links. Because distributed systems may lack a physically synchronized global clock, a logical clock provides for the global ordering of occurrences from various processes in certain systems.

(3) Mutual Exclusion:- . Mutual exclusion ensures that concurrent access of processes to a shared resource or data is serialized, that is, executed in a mutually exclusive manner. Mutual exclusion in a distributed system states that only one process is allowed to execute the critical section (CS) at any

given time. In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion. Message passing is the sole means for implementing distributed mutual exclusion. The decision as to which process is allowed access to the CS next is arrived at by message passing, in which each process learns about the state of all other processes in some consistent way.

Q4. What is the Election algorithm and its types?(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected by another processor.

- Election algorithm basically determines where a new copy of the coordinator should be restarted. Election algorithm assumes that every active process in the system has a unique priority number.
- The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has the highest priority number.
- Then this number is sent to every active process in the distributed system. We have two election algorithms for two different configurations of a distributed system.

.Bully Algorithm

- This algorithm was proposed by Garcia-Molina.
- When the process notices that the coordinator is no longer responding to requests, it initiates an election. A process, P, holds an election as follows:
    1. P sends an ELECTION message to all processes with higher numbers.
    2. If no one responds, P wins the election and becomes the coordinator.

3. If one of the higher-ups answers, it takes over. P's job is done.
    - A process can get an ELECTION message at any time from one of its lower numbered colleagues.
    - When such a message arrives, the receiver sends an OK message back to the sender to indicate that he is alive and will take over. The receiver then holds an election, unless it is already holding one.
    - All processes give up except one that is the new coordinator. It announces its victory by sending all processes a message telling them that starting immediately it is the new coordinator.
    - If a process that was previously down comes back up, it holds an election. If it happens to the highest numbered process currently running, it will win the election and take over the coordinator's job. Thus the biggest guy in town always wins, hence the name "bully algorithm".

Ring Algorithm

- This algorithm uses a ring for its election but does not use any token. In this algorithm it is assumed that the processes are physically or logically ordered so each processor knows its successor.
- When any process notices that a coordinator is not functioning, it builds an ELECTION message containing its own process number and sends the message to its successor. If the successor is down the sender skips over the successor and goes to the next member along the ring until a process is located.
- At each step the sender adds its own process number to the list in the message making itself a candidate to elected as coordinator

- The message gets back to the process that started it and recognizes this event as the message consists of its own process number.
- At that point the message type is changed to COORDINATOR and circulated once again to inform everyone who the coordinator is and who are the new members. The coordinator is selected with the process having the highest number.
- When this message is circulated once it is removed and normal work is preceded.

Q6. Compare Physical and Logical clocks.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Physical Clocks:

- The physical clocks are needed to adjust the time of nodes. All the nodes in the system can share their local time with all the other nodes in the system.
- Basically, in physical synchronization, physical clocks are used to time stamp an event on that computer. Physical clocks keep the time of the day. It will be consistent across systems.

Logical Clocks

- If two systems do not interact with each other then there is no need for synchronization. So, what usually matters is that processes agree on the order in which events occur rather than the time at which they occurred.
- Logical clocks do not need exact time. So, absolute time is not a constraint in logical clocks. Logical clocks just bothers about the message to be delivered and not about the timings of the events that occurred.
- Using Interrupts, computers generally update a software clock. More the interrupts, the higher the overhead. Accuracy will be more.

- The most common logical clock synchronization algorithm for distributed systems is Lamport's algorithm. It is used in the situation where ordering is important, not the time.

Q7. What is mutual exclusion in a distributed system and explain its requirements.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Mutual exclusion ensures that concurrent access of processes to a shared resource or data is serialized, that is, executed in a mutually exclusive manner.

- Mutual exclusion in a distributed system states that only one process is allowed to execute the critical section (CS) at any given time. In a distributed system, shared variables (semaphores) or a local kernel cannot be used to implement mutual exclusion. Message passing is the sole means for implementing distributed mutual exclusion. The decision as to which process is allowed access to the CS next is arrived at by message passing, in which each process learns about the state of all other processes in some consistent way.
- The decision as to which process is allowed access to the CS next is arrived at by message passing, in which each process learns about the state of all other processes in some consistent way. The design of distributed mutual exclusion algorithms is complex because these algorithms have to deal with unpredictable message delays and incomplete knowledge of the system state.
- There are three basic approaches for implementing distributed mutual exclusion:
    1. Token-based approach.
    2. Non-token-based approach.
    3. Quorum-based approach.

- In the token-based approach, a unique token (also known as the PRIVILEGE message) is shared among the sites. A site is allowed to enter its CS if it possesses the token and it continues to hold the token until the execution of the CS is over. Mutual exclusion is ensured because the token is unique.
- In Distributed systems, we neither have shared memory nor a common physical clock and therefore we can not solve mutual exclusion problems using shared variables. To eliminate the mutual exclusion problem in distributed systems, an approach based on message passing is used.
- A site in a distributed system does not have complete information of the state of the system due to lack of shared memory and a common physical clock.

Requirements of Mutual exclusion Algorithm:

- **No Deadlock:**
  Two or more sites should not endlessly wait for any message that will never arrive.
- **No Starvation:**
  Every site who wants to execute a critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical section
- **Fairness:**
  Each site should get a fair chance to execute a critical section. Any request to execute a critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system.
- **Fault Tolerance:**
  In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Q8. Difference between token and non-token based Algorithms.(P4 - Appeared 1 Time) (5-10 Marks)

Ans:

Token Based Algorithm	Non-Token Based Algorithm
Uses token to enter into Critical section	No token is required to enter into critical section
Uses sequence number to execute the critical section	Uses timestamps to enter the critical section
Every request has a sequence number to enter the critical section	Every request has a timestamp to enter the critical section
Higher sequence number will be having the lowest priority	Higher the timestamp, lower the priority
Example: Suzuki-kasami's broadcast algorithm	Example: Ricart-agrawal's algorithm

Q9. Explain Rica-Agarwala's non-token based algorithm.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Ricart–Agrawala algorithm is an algorithm for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala. This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm. Like Lamport's Algorithm, it also follows a permission based approach to ensure mutual exclusion.

In this algorithm:

- Two types of messages (REQUEST and REPLY) are used and communication channels are assumed to follow FIFO order.

- A site sends a REQUEST message to all other sites to get their permission to enter a critical section.
- A site sends a REPLY message to another site to give its permission to enter the critical section.
- A timestamp is given to each critical section request using Lamport's logical clock.
- Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section requests is always in the order of their timestamp.

Algorithm:

- To enter Critical section:
    - When a site S_i wants to enter the critical section, it sends a timestamped REQUEST message to all other sites.
    - When a site S_j receives a REQUEST message from site S_i, It sends a REPLY message to site S_i if and only if
        - Site S_j is neither requesting nor currently executing the critical section.
        - In case Site S_j is requesting, the timestamp of Site S_i's request is smaller than its own request.

Otherwise the request is deferred by site S_j.

- To execute the critical section:
    - Site S_i enters the critical section if it has received the REPLY message from all other sites.
- To release the critical section:
    - Upon exiting the site S_i sends a REPLY message to all the deferred requests.

Message Complexity:

Ricart–Agrawala algorithm requires invocation of $2(N - 1)$ messages per critical section execution. These $2(N - 1)$ messages involves

- (N – 1) request messages
- (N – 1) reply messages

Drawbacks of Ricart–Agrawala algorithm:

- Unreliable approach: failure of any one of the nodes in the system can halt the progress of the system. In this situation, the process will starve forever.
  The problem of failure of nodes can be solved by detecting failure after some timeout.

Performance:

- Synchronization delay is equal to maximum message transmission time
- It requires 2(N – 1) messages per Critical section execution

Q10. What does a non token-based algorithm mean and give its types and explain one of them in brief.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: A site communicates with other sites in order to determine which sites should execute the critical section next. This requires the exchange of two or more successive rounds of messages among sites.

- This approach uses timestamps instead of sequence numbers to order requests for the critical section. Whenever a site makes a request for a critical section, it gets a timestamp.
- Timestamp is also used to resolve any conflict between critical section requests. All algorithms which follow a non-token based approach maintain a logical clock. Logical clocks get updated according to Lamport's scheme

Example: Lamport's algorithm, Ricart–Agrawala algorithm

Types of Non-Token based algorithm

1. Lamport's Algorithm
2. Ricart-Agrawala Algorithm
3. Maekawa's Algorithm

- Lamport's Distributed Mutual Exclusion Algorithm is a permission based algorithm proposed by Lamport as an illustration of his synchronization scheme for distributed systems.
- In permission based timestamp is used to order critical section requests and to resolve any conflict between requests.
- In Lamport's Algorithm, critical section requests are executed in the increasing order of timestamps i.e a request with smaller timestamp will be given permission to execute critical section first than a request with larger timestamp.

In this algorithm:

- Three types of messages (REQUEST, REPLY and RELEASE) are used and communication channels are assumed to follow FIFO order.A site sends a REQUEST message to all other sites to get their permission to enter a critical section.
- A site sends a REPLY message to a requesting site to give its permission to enter the critical section.
- A site sends a RELEASE message to all other sites upon exiting the critical section.
- Every site, Si, keeps a queue to store critical section requests ordered by their timestamps. request_queuei denotes the queue of site Si
- A timestamp is given to each critical section request using Lamport's logical clock.
- Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section requests is always in the order of their timestamp.

Algorithm:

To enter Critical section:

When a site Si wants to enter the critical section, it sends a request message Request(tsi, i) to all other sites and places the request on request_queuei. Here, Tsi denotes the timestamp of Site Si
When a site Sj receives the request message REQUEST(tsi, i) from site Si, it returns a timestamped REPLY message to site Si and places the request of site Si on request_queuej To execute the critical section:
A site Si can enter the critical section if it has received the message with timestamp larger than (tsi, i) from all other sites and its own request is at the top of request_queuei
To release the critical section:
When a site Si exits the critical section, it removes its own request from the top of its request queue and sends a timestamped RELEASE message to all other sites
When a site Sj receives the timestamped RELEASE message from site Si, it removes the request of Si from its request queue

Ricart–Agrawala algorithm is an algorithm for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala. This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm. Like Lamport's Algorithm, it also follows a permission based approach to ensure mutual exclusion.
In this algorithm:

- Two types of messages (REQUEST and REPLY) are used and communication channels are assumed to follow FIFO order.
- A site sends a REQUEST message to all other sites to get their permission to enter a critical section.
- A site sends a REPLY message to another site to give its permission to enter the critical section.
- A timestamp is given to each critical section request using Lamport's logical clock.

- Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section requests is always in the order of their timestamp.

Algorithm:

- To enter Critical section:
  - When a site S_i wants to enter the critical section, it send a timestamped REQUEST message to all other sites.
  - When a site S_j receives a REQUEST message from site S_i, It sends a REPLY message to site S_i if and only if
    - Site S_j is neither requesting nor currently executing the critical section.
    - In case Site S_j is requesting, the timestamp of Site S_i's request is smaller than its own request.

Maekawa's Algorithm is a quorum based approach to ensure mutual exclusion in distributed systems. As we know, In permission based algorithms like Lamport's Algorithm, Ricart-Agrawala Algorithm etc. a site request permission from every other site but in quorum based approach, A site does not request permission from every other site but from a subset of sites which is called quorum.

In this algorithm:
Three types of messages (REQUEST, REPLY and RELEASE) are used.

- A site sends a REQUEST message to all other sites in its request set or quorum to get their permission to enter the critical section.
- A site sends a REPLY message to a requesting site to give its permission to enter the critical section.
- A site sends a RELEASE message to all other sites in its request set or quorum upon exiting the critical section.
- The construction of request set or Quorum:

A request set or Quorum in Maekawa's algorithm must satisfy the following properties:

$\forall i \forall j : i \neq j, 1 \leq i, j \leq N :: Ri \cap Rj \neq \emptyset$

i.e there is at least one common site between the request sets of any two sites.

$\forall i : 1 \leq i \leq N :: Si \in Ri$

$\forall i : 1 \leq i \leq N :: |Ri| = K$

Any site Si is contained in exactly K sets.

$N = K(K - 1) + 1$ and $|Ri| = \sqrt{N}$

Algorithm:

To enter Critical section:

When a site Si wants to enter the critical section, it sends a request message REQUEST(i) to all other sites in the request set Ri.

When a site Sj receives the request message REQUEST(i) from site Si, it returns a REPLY message to site Si if it has not sent a REPLY message to the site from the time it received the last RELEASE message. Otherwise, it queues up the request.

To execute the critical section:

A site Si can enter the critical section if it has received the REPLY message from all the site in request set Ri

To release the critical section:

When a site Si exits the critical section, it sends RELEASE(i) message to all other sites in request set Ri

When a site Sj receives the RELEASE(i) message from site Si, it send REPLY message to the next site waiting in the queue and deletes that entry from the queue

In case the queue is empty, site Sj updates its status to show that it has not sent any REPLY message since the receipt of the last RELEASE message.

Q11. Write a short note on- Suzuki-Kasami's Broadcast Algorithm.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: If a site without a token needs to enter a CS, broadcast a REQUEST for a token message to all other sites.

nToken: (a) Queue of request sites (b) Array LN[1..N], the sequence number of the most recent execution by a site j.

Token holder sends token to requestor, if it is not inside CS. Otherwise, send after exiting CS.

Token holders can make multiple CS accesses.

Design issues:

a).Distinguishing outdated REQUEST messages from current REQUEST messages

b) Determining which site has an outstanding request for the CS.

Algorithm

Requesting the critical section

If the requesting site 5i does not have the token, then it increments its sequence number, RNi[i], and sends a

REQUEST(i, sn) message to all other sites. (sn is the updated value of RNdi].)

When a site 5 j receives this message, it sets RNj[i] to max(RNj[i], sn). If 5 j has the idle token, then it sends the token to 5 i if RNj [i]=LN[i]+1.Executing the critical section.

Site 5i executes the CS when it has received the token. Releasing the critical section. Having finished the execution of the CS, site 5 i takes the following actions:

It sets the LN[i] element of the token array equal to RNdi].

For every site 5 j whose 10 is not in the token queue, it appends its 10 to the token queue if RNi [j]=LN[j]+1.

If token queue is nonempty after the above update, then it deletes the top site

Example

Step 1: S1 has a token, S3 is in queue.

Site	Seq.,Vector RN	Token Vect. LN	Token Queue
S1	10,15,9	10,15,8	3
S2	10,16,9		
S3	10,15,9		

Step 2: S3 gets token, S2 in queue

Site	Seq.,Vector RN	Token Vect. LN	Token Queue
S1	10,16,9		
S2	10,16,9		
S3	10,16,9	10,15,9	2

Step 3: S2 gets token, queue empty

Site	Seq.,Vector RN	Token Vect. LN	Token Queue
S1	10,16,9		
S2	10,16,9	10,16,9	<empty>
S3	10,16,9		

Q12. Explain Centralized approach in distributed systems.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Centralized systems are systems that use client/server architecture where one or more client nodes are directly connected to a central server. This is the most commonly used type of system in many organizations where a client sends a request to a company server and receives the response.

Characteristics of Centralized System –

Presence of a global clock: As the entire system consists of a central node(a server/ a master) and many client nodes(a computer/ a slave), all client nodes sync up with the global clock(the clock of the central node).

One single central unit: One single central unit which serves/coordinates all the other nodes in the system.

Dependent failure of components: Central node failure causes the entire system to fail. This makes sense because when the server is down, no other entity is there to send/receive responses/requests.

Architecture of Centralized System –

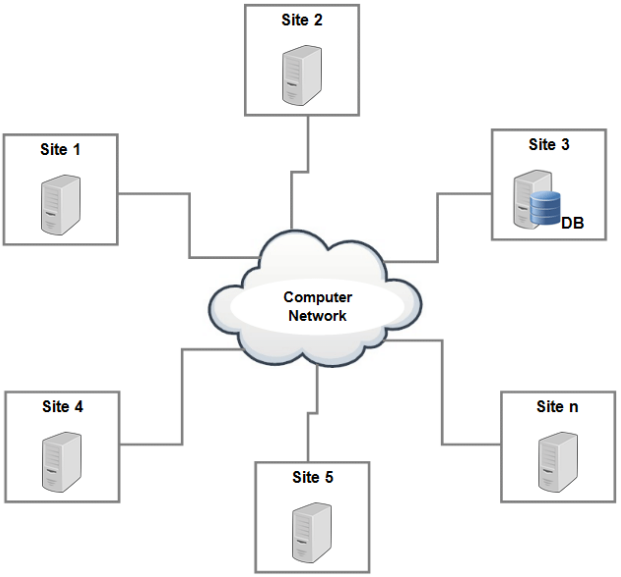
Client-Server architecture. The central node that serves the other nodes in the system is the server node and all the other nodes are the client nodes.

It consists following types of architecture:

- Client-server
- Application Layering

Client Server

Processes in a distributed system are split into two (potentially overlapping) groups in the fundamental client-server architecture. A server is a program that provides a particular service, such as a database service or a file system service. A client is a process that sends a request to a server and then waits for the server to respond before requesting a service from it.



Application Layering

However, many individuals have urged a distinction between the three levels below, effectively adhering to the layered architectural approach we previously described, given that many client-server applications are intended to provide user access to databases:

- The user-interface level
- The processing level
- The data level

Everything required to connect with the user, such as display management directly, is contained at the user-interface level. Applications are often found at the processor level. The actual data that is used to make decisions is managed at the data level.

Limitations of Centralized System –

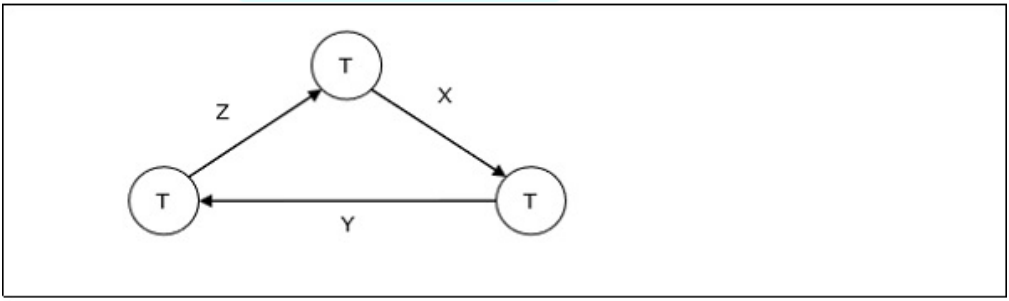
- Can't scale up vertically after a certain limit – After a limit, even if you increase the hardware and software capabilities of the server node, the performance will not increase appreciably leading to a

cost/benefit ratio < 1.

- Bottlenecks can appear when the traffic spikes – as the server can only have a finite number of open ports to which can listen to connections from client nodes.

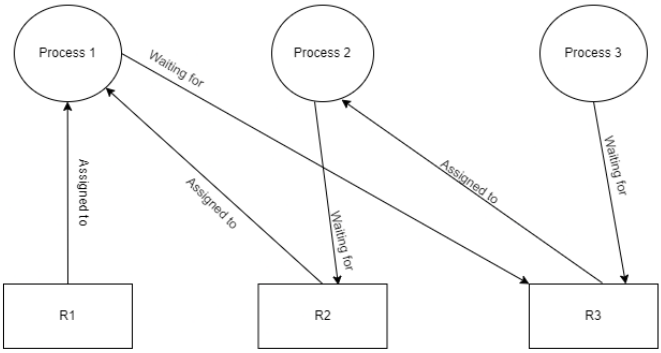
Q13. What is a deadlock and give its types. Also explain deadlock prevention and detection.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Deadlock is a state of a database system having two or more transactions, when each transaction is waiting for a data item that is being locked by some other transaction. A deadlock can be indicated by a cycle in the wait-for-graph. This is a directed graph in which the vertices denote transactions and the edges denote waits for data items.
For example, in the following wait-for-graph, transaction T1 is waiting for data item X which is locked by T3. T3 is waiting for Y which is locked by T2 and T2 is waiting for Z which is locked by T1. Hence, a waiting cycle is formed, and none of the transactions can proceed.

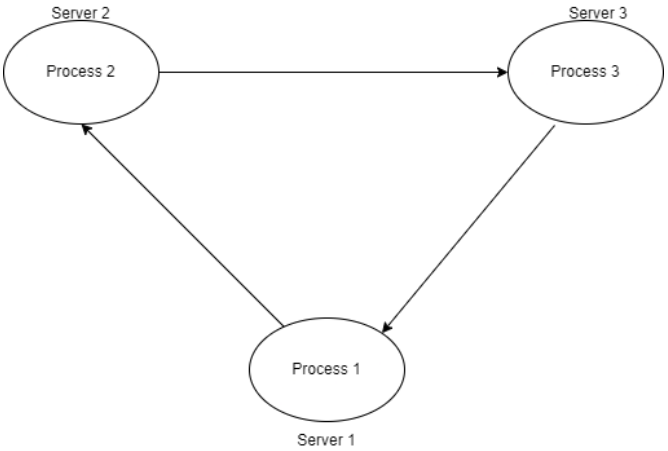


Types of Distributed Deadlock:
There are two types of Deadlocks in Distributed System:
Resource Deadlock: A resource deadlock occurs when two or more processes wait permanently for resources held by each other.

- A process that requires certain resources for its execution, and cannot proceed until it has acquired all those resources.
- It will only proceed to its execution when it has acquired all required resources.
- It can also be represented using AND condition as the process will execute only if it has all the required resources.
- Example: Process 1 has R1, R2, and requests resources R3. It will not execute if any one of them is missing. It will proceed only when it acquires all requested resources i.e. R1, R2, and R3.



Communication Deadlock: On the other hand, a communication deadlock occurs among a set of processes when they are blocked waiting for messages from other processes in the set in order to start execution but there are no messages in transit between them. When there are no messages in transit between any pair of processes in the set, none of the processes will ever receive a message. This implies that all processes in the set are deadlocked.



Distributed Deadlock Prevention

Just like in centralized deadlock prevention, in distributed deadlock prevention approach, a transaction should acquire all the locks before starting to execute. This prevents deadlocks.

The site where the transaction enters is designated as the controlling site. The controlling site sends messages to the sites where the data items are located to lock the items. Then it waits for confirmation. When all the sites have confirmed that they have locked the data items, transaction starts. If any site or communication link fails, the transaction has to wait until they have been repaired.

Though the implementation is simple, this approach has some drawbacks -

- Pre-acquisition of locks requires a long time for communication delays. This increases the time required for transactions.
- In case of site or link failure, a transaction has to wait for a long time so that the sites recover. Meanwhile, in the running sites, the items are locked. This may prevent other transactions from executing.

- If the controlling site fails, it cannot communicate with the other sites. These sites continue to keep the locked data items in their locked state, thus resulting in blocking.

Distributed Deadlock Detection

Just like a centralized deadlock detection approach, deadlocks are allowed to occur and are removed if detected. The system does not perform any checks when a transaction places a lock request. For implementation, global wait-for-graphs are created. Existence of a cycle in the global wait-for-graph indicates deadlocks. However, it is difficult to spot deadlocks since transactions wait for resources across the network. Alternatively, deadlock detection algorithms can use timers. Each transaction is associated with a timer which is set to a time period in which a transaction is expected to finish. If a transaction does not finish within this time period, the timer goes off, indicating a possible deadlock.

Another tool used for deadlock handling is a deadlock detector. In a centralized system, there is one deadlock detector. In a distributed system, there can be more than one deadlock detectors. A deadlock detector can find deadlocks for the sites under its control. There are three alternatives for deadlock detection in a distributed system, namely.

- Centralized Deadlock Detector - One site is designated as the central deadlock detector.
- Hierarchical Deadlock Detector - A number of deadlock detectors are arranged in hierarchy.
- Distributed Deadlock Detector - All the sites participate in detecting deadlocks and removing them.

MODULE-4

Q1. Explain scheduling in a distributed system and give its features.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Scheduling is essentially a decision-making process that enables resource sharing among a number of activities by determining their execution order on the set of available resources. The emergence of distributed systems brought new challenges on scheduling in computer systems, including clusters, grids, and more recently clouds.
The techniques that are used for scheduling the processes in distributed systems are as follows:

- **Task Assignment Approach:** In the Task Assignment Approach, the user-submitted process is composed of multiple related tasks which are scheduled to appropriate nodes in a system to improve the performance of a system as a whole.
- **Load Balancing Approach:** In the Load Balancing Approach, as the name implies, the workload is balanced among the nodes of the system.
- **Load Sharing Approach:** In the Load Sharing Approach, it is assured that no node would be idle while processes are waiting for their processing. t

The features of a good scheduling algorithm in a distributed system.

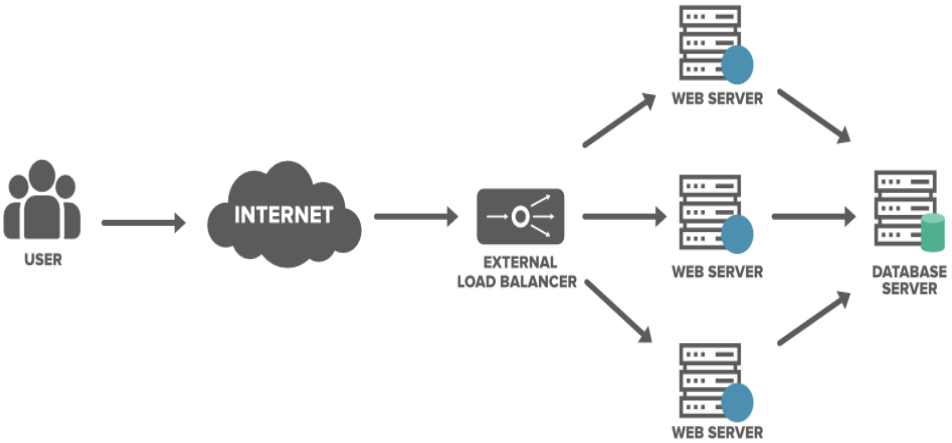
- **Fault Tolerance:** A good global scheduling algorithm should not be stopped when system nodes are crashed or temporarily crashed. Algorithm configuration should also be even if the nodes are separated by multiple nodes.

- **Scalability:** A good global scheduling algorithm should be used for marketing, which means that the algorithm should work well even as the number of nodes increases. The scheduling algorithm will query the workload of all categories in the corrupted system and select a node with the least configuration load. For distributed applications, the configuration algorithm will balance the load between nodes but this is not the case that nodes that are more efficient as they are assigned will have a better response time and less loaded nodes may have a negative response time. so the load should be shared instead of balancing i.e. resources in the nodes should be shared among the nodes until the tasks being performed are not affected.
- **No a priori knowledge about the processes:** Scheduling algorithms work based on the characteristics and resource requirement of the processes; this information should be provided by the user. This will obviously put extra overhead on users, so a good global scheduling algorithm should require less prior knowledge.

Q2. Explain load balancing in distributed systems and its types.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: A load balancer is a device that acts as a reverse proxy and distributes network or application traffic across a number of servers. Load adjusting is the approach to conveying load units (i.e., occupations/assignments) across the organization which is associated with the distributed system.

Load adjusting should be possible by the load balancer. The load balancer is a framework that can deal with the load and is utilized to disperse the assignments to the servers. The load balancers allocate the primary undertaking to the main server and the second assignment to the second server.



Types of load balancing algorithms

Dynamic load balancing algorithms

- Least connection: Checks which servers have the fewest connections open at the time and sends traffic to those servers. This assumes all connections require roughly equal processing power.
- Weighted least connection: Gives administrators the ability to assign different weights to each server, assuming that some servers can handle more connections than others.
- Weighted response time: Averages the response time of each server, and combines that with the number of connections each server has open to determine where to send traffic. By sending traffic to the servers with the quickest response time, the algorithm ensures faster service for users.
- Resource-based: Distributes load based on what resources each server has available at the time. Specialized software (called an "agent") running on each server measures that server's available CPU and memory, and the load balancer queries the agent before distributing traffic to that server.

Static load balancing algorithms

- Round robin: Round robin load balancing distributes traffic to a list of servers in rotation using the Domain Name System (DNS). An authoritative nameserver will have a list of different A records for a domain and provide a different one in response to each DNS query.
- Weighted round robin: Allows an administrator to assign different weights to each server. Servers deemed able to handle more traffic will receive slightly more. Weighting can be configured within DNS records.
- IP hash: Combines incoming traffic's source and destination IP addresses and uses a mathematical function to convert it into a hash. Based on the hash, the connection is assigned to a specific server

Q3. Explain working of Task Assignment approach, its goals and the need for it with an example.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: A Distributed System is a Network of Machines that can exchange information with each other through Message-passing.

- It can be very useful as it helps in resource sharing. In this article, we will see the concept of the Task Assignment Approach in Distributed systems.

Working of Task Assignment Approach:

In the working of the Task Assignment Approach, the following are the assumptions:

- The division of an individual process into tasks.
- Each task's computing requirements and the performance in terms of the speed of each processor are known.
- The cost incurred in the processing of each task performed on every node of the system is known.

- The IPC (Inter-Process Communication) cost is known for every pair of tasks performed between nodes.
- Other limitations are also familiar, such as job resource requirements and available resources at each node, task priority connections, and so on.

Goals of Task Assignment Algorithms:

- Reducing Inter-Process Communication (IPC) Cost
- Quick Turnaround Time or Response Time for the whole process
- A high degree of Parallelism
- Utilization of System Resources in an effective manner

Need for Task Assignment in a Distributed System:

- The need for task management in distributed systems was raised for achieving the set performance goals. For that optimal assignments should be carried out concerning cost and time functions such as task assignment to minimize the total execution and communication costs, completion task time, total cost of 3 (execution, communication, and interference), total execution and communication costs with the limit imposed on the number of tasks assigned to each processor, and a weighted product of cost functions of total execution and communication costs and completion task time.
- All these factors are countable in task allocation and turn, resulting in the best outcome of the system.

Q4. What are the issues due to which load sharing algorithms are used?Explain different policies in it.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: In a distributed system,proper utilization of resources is not required to balance load on all nodes but it is necessary and sufficient to prevent the nodes from being idle.This is known as load sharing.

- Load-sharing is much simpler than load-balancing since it only attempts to ensure that no node is idle when a heavy node exists.
- Load sharing algorithms require that proper decisions be made regarding load estimation policy,process transfer policy,state information exchange policy,location policy,priority assignment policy,and migration limiting policy.
- Other policies for load sharing are as follows:

1.Load estimation policies for Load-sharing algorithms

- Load-sharing algorithms attempt to avoid nodes from being idle but it is sufficient to know whether a node is busy or idle.Thus these algorithms normally employ the simplest load estimation policy of counting the total number of processes on a node.
- In modern systems counting the total number of processes on a node is not suitable.Therefore measuring CPU utilization should be used to estimate the load of a node in these systems.

2.Process transfer policies for Load-sharing algorithms

- Load sharing algorithms normally use all-or-nothing strategy.This strategy uses the threshold value of all the nodes fixed at one.A Node becomes a receiver node when it has no process,& becomes a sender node when it has more than one process.
- To avoid processing power on nodes having zero process load-sharing algorithms,a threshold value of two is used instead of one.
- When CPU utilization is used as the load estimation policy,the double-threshold policy should be used as the process transfer policy.

3.Location policies for Load-sharing algorithms

Location policy decides whether the sender node or the receiver node of the process takes the initiative to search for suitable node in the system.The location policy can be the following:

Sender-initiated location policy

- In this policy heavily loaded nodes search for lightly loaded nodes.
- When the node becomes overloaded,it either broadcasts or randomly probes the other nodes one by one to find a node that is able to receive remote processes.
- A node is viable to receive a process only if the transfer of the process to the receiver's node will not increase the receiver node's load above its threshold value.
- When broadcasting,a suitable node is known to be present as soon as reply arrives at the sender node.

Receiver-initiated location policy

- In this policy lightly loaded nodes search for heavily loaded nodes
- When the node becomes under-loaded,it either broadcasts or randomly probes the other nodes one by one to indicate its willingness to receive remote processes.
- A node is viable to receive a process only if the transfer of the process to the receivers will not increase the receiver node’s load above its threshold value.
- When broadcasting,a suitable node is known to be present as soon as reply arrives at the receiving node.

4.State information exchange policies for Load-sharing algorithms:

In load-sharing algorithms it is not necessary for the nodes to periodically exchange state information,but needs to know the state of other nodes when it is either under loaded or overloaded.A node shares state

information with other nodes only when its state changes.Commonly used policies are:

i.Broadcast when state changes

- In sender-initiated/receiver-initiated location policy a node broadcasts State Information Request when it becomes overloaded/under-loaded.
- It is called broadcast-when-idle policy when a receiver-initiated policy is used with a fixed threshold value of one.

ii.Poll when state changes

- In large networks polling mechanism is used
- Polling mechanism randomly asks different nodes for state information until an appropriate one or probe limit is reached.
- It is called poll-when-idle policy when a receiver-initiated policy is used with a fixed threshold value of one.

5.Priority assignment policy for Load-balancing algorithms

In a distributed operating system that supports process migration,a priority assignment rule for scheduling the local & remote process of a node should be planned.

One of the following priority assignment rules can be used:

i.Selfish priority assignment rule

- In this the local processes are given higher priority than remote processes.
- This rule has the worst response time performance & best response time performance is achieved for local processes.

ii.Altruistic priority assignment rule

- Remote processes are given higher priority than local processes in this rule.
- It has the best response time performance.

iii.Intermediate priority assignment rule

- Priority in this rule is decided depending on the number of local & remote processes on a node.
- When the number of local processes is greater or equal to the number of remote processes than local processes are given higher priority or else remote processes are given higher priority than local processes.
- Performance of this rule is between the two above policies.Overall response time performance is close to altruistic policy.

6.Migration limiting policy for Load-balancing algorithms:

- This policy determines the total number of times a process can migrate.

One of the following two policies may be used

a.Uncontrolled:A remote process arriving at a node is treated just as a process originating at a node due to which a process can migrate any number of times.

b.Controlled:This avoids the instability of the uncontrolled policy & uses a migration count parameter to fix a limit on the number of times a process can migrate.

- A long process may be allowed to migrate more times compared to a short process.

Q5. Compare Load Sharing and balancing.(P4 - Appeared 1 Time) (5-10 Marks)

Ans:

Load balancing	Load sharing
It evenly distributes network traffic or load over many channels. It	It directs a portion of the network's traffic or load to one connection

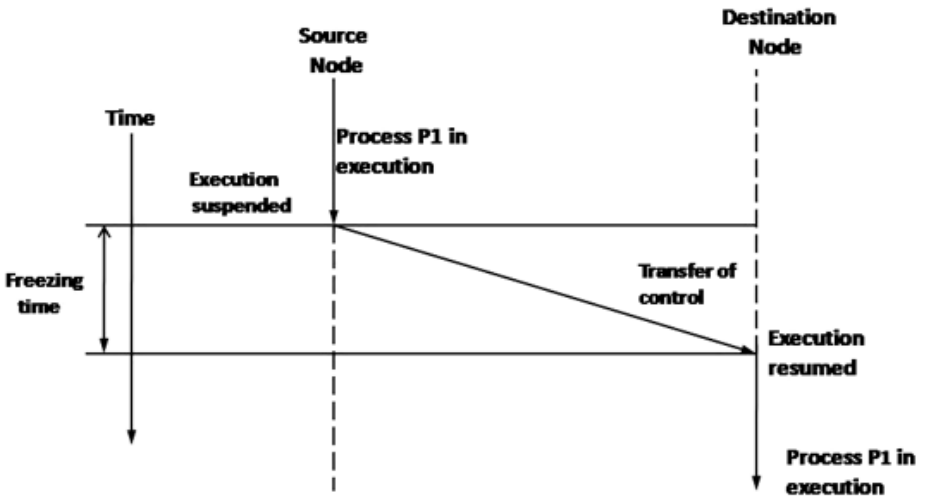
may use both static and dynamic load balancing.	while the balance is routed through other channels.
It isn't easy to have accurate load balancing.	Load sharing is relatively easier than load balancing.
No instance in load balancing is sharing the load.	All instances in it are sharing the load.
Load balancing employs the establishment of ratios, least connections, fastest, round robin, and experimental techniques.	Load Spreading is based on sharing network traffic or load among connections depending on destination IP or MAC address choices.

Q6. What are the steps involved in Process Migration?(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Process migration is the relocation of a process from its source node to another destination node. The way a process migrates from one node to another is shown in the figure below.

A process can either be migrated before it starts execution on its source node which is called a non-preemptive process or during the course of its execution that is known as preemptive process migration.

Preemptive process migration is more costly compared to non-preemptive because the process environment must accompany the process to its new node.



Steps involved in process migration:

i. Process is selected for migration.

ii. Select the destination node for the process to be migrated.

iii. Transfer of selected process to the destination node.

- Migration policy is responsible for the first two steps while the third step is handled by the migration mechanism.
- Migration of a process is complex that involves handling various activities to meet the requirements for a good process migration.
- The sub activities involved are:

i. Freezing the process on its source node and restarting it on destination node

ii. Transferring the processes address space from restarting from its source to destination node.

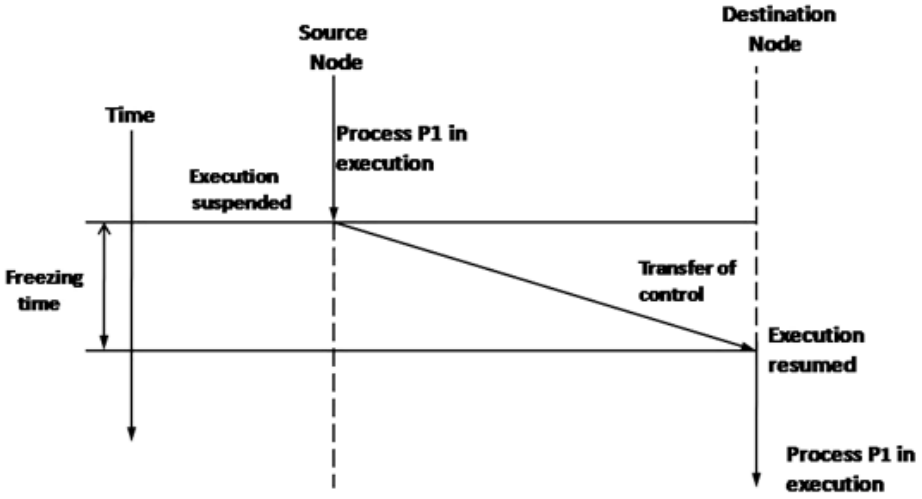
iv. Handling communication between processes that have been placed at different nodes.

- A preemptive process migration facility allows the transfer of an executing process from one node to another. On the other hand, in a system supporting only non-preemptive migration facilities, a process can only be transferred prior to beginning its execution.

- Preemptive process migration is costlier than non-preemptive process migration since the process state, which must accompany the process to its new node, becomes much more complex after execution begins.

Q7. What is Process Migration? Explain the methods used for this.(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Process migration is the relocation of a process from its source node to another destination node. The way a process migrates from one node to another is shown in the figure below.



A process can either be migrated before it starts execution on its source node which is called a non-preemptive process or during the course of its execution that is known as preemptive process migration.

Methods of Process Migration

The methods of Process Migration are:

1. Homogeneous Process Migration: Homogeneous process migration implies relocating a process in a homogeneous environment where all

systems have a similar operating system as well as architecture. There are two unique strategies for performing process migration. These are i) User-level process migration ii) Kernel level process migration.

- User-level process migration: In this procedure, process migration is managed without converting the operating system kernel. User-level migration executions are more simple to create and handle but have usually two issues: i) Kernel state is not accessible by them. ii) They should cross the kernel limit utilizing kernel demands which are slow and expensive.
- Kernel level process migration: In this procedure, process migration is finished by adjusting the operating system kernel. Accordingly, process migration will become more simple and more proficient. This facility permits the migration process to be done faster and relocate more types of processes.

Q8. What is Process Management?(P4 - Appeared 1 Time) (5-10 Marks)

Ans: Process management is the application of knowledge, skills, tools, techniques and systems to define, visualize, measure, control, report and improve processes with the goal to meet customer requirements profitably.

- It can be differentiated from program management in that program management is concerned with managing a group of interdependent projects.
- From another viewpoint, process management includes program management. In project management, process management is the use of a repeatable process to improve the outcome of the project .
- In a Distributed Operating system, the main goal of process management is to make the best possible use of the processing resources of the entire system by sharing them among all the processes.

- Three important concepts are used in distributed operating systems to achieve this goal:
    1. Processor Allocation: It deals with the process of deciding which process should be assigned to which processor.
    2. Process Migration: It deals with the movement of a process from its current location to the processor to which it has been assigned.
    3. Threads: They deal with fine-grained parallelism for better utilization of the processing capability of the system.

Q9. What is Code Migration? (P4 - Appeared 1 Time) (5-10 Marks)

Ans: Code migration is the movement of programming code from one system to another. There are three distinct levels of code migration with increasing complexity, cost and risk. Simple migration involves the movement from language to a newer version. A second, more complicated level of migration involves moving to a different programming language. Migrating to an entirely new platform or operating system is the most complex type of migration.

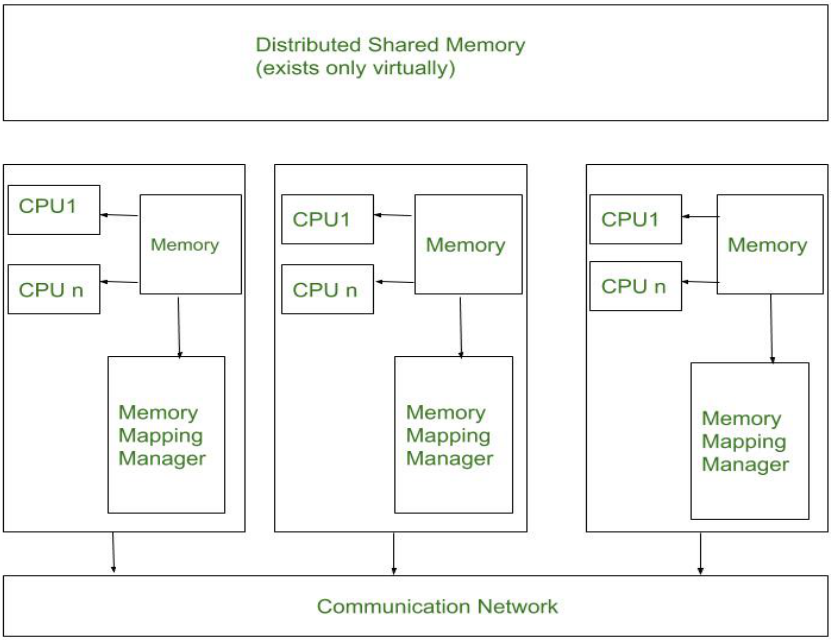
- The first type of code migration is a simple movement from one version of a language to a newer, but syntactically different version. This is the easiest of migration routes as the basic structure and much of the programming constructs usually do not change.
- In many cases, the old code would actually work, but new and improved routines or modularization can be improved by retooling the code to fit the nature of the new language. Therefore migrating the code would lead to more efficiency in execution.
- The second level of code migration would be migrating to a completely different programming language.

- This could be caused by porting to a new software system or implementing a different relational database management system (RDMS).
- This type of migration often requires that programmers learn an entirely new language, or new programmers be brought in to assist with the migration. In this case, the entire program must be rewritten from the ground up. Even though most of the constructs are likely to exist in both languages, the precise syntax is usually completely different.
- The most complex example of code migration is migrating to an entirely new platform and/or operating system (OS). This not only changes the programming language, but also the machine code behind the language.
- While most modern programming languages shield the programmer from this low level code, knowledge of the OS and how it operates is essential to producing code that is efficient and executes as expected.

MODULE-5

Q1. Explain Distributed Shared Memory Architecture and what are its Design issues. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: A distributed shared memory (DSM) system is a collection of many nodes/computers which are connected through some network and all have their local memories. The DSM system manages the memory across all the nodes. All the nodes/computers transparently interconnect and process. The DSM also makes sure that all the nodes are accessing the virtual memory independently without any interference from other nodes. The DSM does not have any physical memory, instead, a virtual space address is shared among all the nodes, and the transfer of data occurs among them through the main memory.



Architecture of Distributed Shared Memory (DSM) :

Every node consists of 1 or additional CPU's and a memory unit. High-speed communication network is employed for connecting the nodes. A straightforward message passing system permits processes on completely different nodes to exchange one another.

Memory mapping manager unit :

Memory mapping manager routine in every node maps the native memory onto the shared computer storage. For mapping operation, the shared memory house is divided into blocks.

Information caching may be a documented answer to deal with operation latency. DMA uses information caching to scale back network latency. the most memory of the individual nodes is employed to cache items of the shared memory house.

Memory mapping manager of every node reads its native memory as an enormous cache of the shared memory house for its associated processors. The bass unit of caching may be a memory block. Systems that support DSM, information moves between secondary memory and main memory also as between main reminiscences of various nodes.

Communication Network Unit :

Once a method accesses information within the shared address, the house mapping manager maps the shared memory address to the physical memory. The mapped layer of code is enforced either within the operating kernel or as a runtime routine.

Physical memory on every node holds pages of shared virtual–address houses. Native pages are a unit gift in some node's memory. Remote pages in some other node's memory.

Issues to Design and Implementation of DSM:

- Granularity
- Structure of shared memory space
- Memory coherence and access synchronization

- Data location and access
- Replacement strategy
- Thrashing
- Heterogeneity

1. Granularity: Granularity refers to the block size of a DSM system. Granularity refers to the unit of sharing and the unit of data moving across the network when a network block shortcoming then we can utilize the estimation of the block size as words/phrases. The block size might be different for the various networks
2. Structure of shared memory space: Structure refers to the design of the shared data in the memory. The structure of the shared memory space of a DSM system is regularly dependent on the sort of applications that the DSM system is intended to support.
3. Memory coherence and access synchronization: In the DSM system the shared data things ought to be accessible by different nodes simultaneously in the network. The fundamental issue in this system is data irregularity. The data irregularity might be raised by synchronous access. To solve this problem in the DSM system we need to utilize some synchronization primitives, semaphores, event count, and so on.
4. Data location and access: To share the data in the DSM system it ought to be possible to locate and retrieve the data as accessed by clients or processors. Therefore the DSM system must implement some form of data block finding system to serve network data to meet the requirement of the memory coherence semantics being utilized.
5. Replacement strategy: In the local memory of the node is full, a cache miss at the node implies not just a get of the gotten to information block from a remote node but also a replacement. A

data block of the local memory should be replaced by the new data block. Accordingly, a position substitution methodology is additionally vital in the design of a DSM system.

6. Thrashing: In a DSM system data blocks move between nodes on demand. In this way on the off chance that 2 nodes compete for write access to the single data item. The data relating to the data block might be moved back and forth at such a high rate that no genuine work can get gone. The DSM system should utilize an approach to keep away from a situation generally known as thrashing.
7. Heterogeneity: The DSM system worked for homogeneous systems and need not address the heterogeneity issue. In any case, assuming the underlined system environment is heterogeneous, the DSM system should be designed to deal with heterogeneity, so it works appropriately with machines having different architectures.

Q2. Define Replication. Give its types. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: In a distributed system data is stored over different computers in a network. Therefore, we need to make sure that data is readily available for the users. Availability of the data is an important factor often accomplished by data replication. Replication is the practice of keeping several copies of data in different places.

The first and foremost thing is that it makes our system more stable because of node replication. It is good to have replicas of a node in a network due to following reasons:

- If a node stops working, the distributed network will still work fine due to its replicas which will be there. Thus it increases the fault tolerance of the system.

- It also helps in load sharing where loads on a server are shared among different replicas.
- It enhances the availability of the data. If the replicas are created and data is stored near to the consumers, it would be easier and faster to fetch data.

Types of Replication

- Active Replication
- Passive Replication

Active Replication:

- The request of the client goes to all the replicas.
- It is to be made sure that every replica receives the client request in the same order else the system will get inconsistent.
- There is no need for coordination because each copy processes the same request in the same sequence.
- All replicas respond to the client's request.

Advantages:

- It is really simple. The codes in active replication are the same throughout.
- It is transparent.
- Even if a node fails, it will be easily handled by replicas of that node.

Disadvantages:

- It increases resource consumption. The greater the number of replicas, the greater the memory needed.
- It increases the time complexity. If some change is done on one replica it should also be done in all others.

Passive Replication:

- The client request goes to the primary replica, also called the main replica.
- There are more replicas that act as backup for the primary replica.

- Primary replica informs all other backup replicas about any modification done.
- The response is returned to the client by a primary replica.
- Periodically the primary replica sends some signal to backup replicas to let them know that it is working perfectly fine.
- In case of failure of a primary replica, a backup replica becomes the primary replica.

Advantages:

- The resource consumption is less as backup servers only come into play when the primary server fails.
- The time complexity of this is also less as there's no need for updating in all the node replicas, unlike active replication.

Disadvantages:

- If some failure occurs, the response time is delayed.

Q3. What is consistency? Compare Eventual and Strong-Consistency in Distributed Databases. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: Consistency here means that a read request for an entity made to any of the nodes of the database should return the same data.

Eventual consistency makes sure that data of each node of the database gets consistent eventually. Time taken by the nodes of the database to get consistent may or may not be defined.

1. Eventual Consistency : Eventual consistency is a consistency model that enables the data store to be highly available. It is also known as optimistic replication & is key to distributed systems. So, how exactly does it work? Let's Understand this with the help of a use case. Real World Use Case :

- Think of a popular microblogging site deployed across the world in different geographical regions like Asia, America, and Europe. Moreover, each geographical region has multiple data center zones: North, East, West, and South.

- Furthermore, each zone has multiple clusters which have multiple server nodes running. So, we have many datastore nodes spread across the world that micro-blogging sites use for persisting data. Since there are so many nodes running, there is no single point of failure.
- The data store service is highly available. Even if a few nodes go down, persistence service is still up. Let's say a celebrity makes a post on the website that everybody starts liking around the world.
- At a point in time, a user in Japan likes a post which increases the "Like" count of the post from say 100 to 101. At the same point in time, a user in America, in a different geographical zone, clicks on the post, and he sees "Like" count as 100, not 101.

Reason for the above Use case :

- Simply, because the new updated value of the Post "Like" counter needs some time to move from Japan to America and update server nodes running there. Though the value of the counter at that point in time was 101, the user in America sees old inconsistent values.
- But when he refreshes his web page after a few seconds, the "Like" counter value shows as 101. So, data was initially inconsistent but eventually got consistent across server nodes deployed around the world. This is what eventual consistency is.

2. Strong Consistency: Strong Consistency simply means the data must be strongly consistent at all times. All the server nodes across the world should contain the same value as an entity at any point in time. And the only way to implement this behavior is by locking down the nodes when being updated. Real World Use Case :

- Let's continue the same Eventual Consistency example from the previous lesson. To ensure Strong Consistency in the system, when

a user in Japan likes posts, all nodes across different geographical zones must be locked down to prevent any concurrent updates.

- This means at one point in time, only one user can update the post "Like" counter value. So, once a user in Japan updates the "Like" counter from 100 to 101. The value gets replicated globally across all nodes. Once all nodes reach consensus, locks get lifted. Now, other users can Like posts.
- If the nodes take a while to reach a consensus, they must wait until then. Well, this is surely not desired in the case of social applications. But think of a stock market application where the users are seeing different prices of the same stock at one point in time and updating it concurrently. This would create chaos. Therefore, to avoid this confusion we need our systems to be Strongly Consistent.
- The nodes must be locked down for updates. Queuing all requests is one good way of making a system Strongly Consistent. The strong Consistency model hits the capability of the system to be Highly Available & perform concurrent updates. This is how strongly consistent ACID transactions are implemented.

Q4. What is the consistency model? Explain Data-Centric and Client-Centric Consistency Models. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: A consistency model is a contract between a distributed data store and processes, in which the processes agree to obey certain rules in contrast the store promises to work correctly.

- A consistency model basically refers to the degree of consistency that should be maintained for the shared memory data.
- If a system supports the stronger consistency model, then the weaker consistency model is automatically supported but the converse is not true.

- The types of consistency models are Data-Centric and client centric consistency models.

1.Data-Centric Consistency Models

A data store may be physically distributed across multiple machines. Each process that can access data from the store is assumed to have a local or nearby copy available of the entire store.

i.Strict Consistency model

- Any read on a data item X returns a value corresponding to the result of the most recent write on X
- This is the strongest form of memory coherence which has the most stringent consistency requirement.
- Strict consistency is the ideal model but it is impossible to implement in a distributed system. It is based on absolute global time or a global agreement on commitment of changes.

ii.Sequential Consistency

- Sequential consistency is an important data-centric consistency model which is a slightly weaker consistency model than strict consistency.
- A data store is said to be sequentially consistent if the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process should appear in this sequence in a specified order.
- Example: Assume three operations read(R1), write(W1), read(R2) performed in an order on a memory address. Then (R1,W1,R2),(R1,R2,W1),(W1,R1,R2)(R2,W1,R1) are acceptable provided all processes see the same ordering.

iii.Linearizability

- It is weaker than strict consistency, but stronger than sequential consistency.
- A data store is said to be linearizable when each operation is timestamped and the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order
- The operations of each individual process appear in sequence order specified by its program.
- If tsOP1(x)< tsOP2(y), then operation OP1(x) should precede OP2(y) in this sequence.

iv.Causal Consistency

- It is a weaker model than sequential consistency.
- In Causal Consistency all processes see only those memory reference operations in the correct order that are potentially causally related.
- Memory reference operations which are not related may be seen by different processes in different order.
- A memory reference operation is said to be causally related to another memory reference operation if the first operation is influenced by the second operation.
- If a write(w2) operation is causally related to another write (w1) the acceptable order is (w1, w2).

v.FIFO Consistency

- It is weaker than causal consistency.
- This model ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed, like a single process in a pipeline.
- This model is simple and easy to implement, having good performance because processes are ready in the pipeline.

- Implementation is done by sequencing write operations performed at each node independently of the operations performed on other nodes.
- Example: If (w11) and (w12) are write operations performed by p1 in that order and (w21),(w22) by p2. A process p3 can see them as [(w11,w12),(w21,w2)] while p4 can view them as [(w21,w2),(w11,w12)].

vi.Weak consistency

- The basic idea behind the weak consistency model is enforcing consistency on a group of memory reference operations rather than individual operations.
- A Distributed Shared Memory system that supports the weak consistency model uses a special variable called a synchronization variable which is used to synchronize memory.
- When a process accesses a synchronization variable, the entire memory is synchronized by making visible the changes made to the memory to all other processes.

vii.Release Consistency

- Release consistency model tells whether a process is entering or exiting from a critical section so that the system performs either of the operations when a synchronization variable is accessed by a process.
- Two synchronization variables acquire and release are used instead of a single synchronization variable. Acquire is used when a process enters a critical section and release is when it exits a critical section.
- Release consistency can be viewed as a synchronization mechanism based on barriers instead of critical sections.

viii.Entry Consistency

- In entry consistency every shared data item is associated with a synchronization variable.

- In order to access consistent data, each synchronization variable must be explicitly acquired.
- Release consistency affects all shared data but entry consistency affects only those shared data associated with a synchronization variable.

2.Client-Centric Consistency Models

- Client-centric consistency models aim at providing a system wide view on a data store.
- This model concentrates on consistency from the perspective of a single mobile client.
- Client-centric consistency models are generally used for applications that lack simultaneous updates where most operations involve reading data.

i.Eventual Consistency

- In Systems that tolerate a high degree of inconsistency, if no updates take place for a long time all replicas will gradually and eventually become consistent. This form of consistency is called eventual consistency.
- Eventual consistency only requires those updates that guarantee propagation to all replicas.
- Eventual consistent data stores work fine as long as clients always access the same replica.
- Write conflicts are often relatively easy to solve when assuming that only a small group of processes can perform updates. Eventual consistency is therefore often cheap to implement.

ii.Monotonic Reads Consistency

- A data store is said to provide monotonic-read consistency if a process reads the value of a data item x, any successive read operation on x by that process will always return that same value or a more recent value.

- A process has seen a value of x at time t, it will never see an older version of x at a later time.
- Example: A user can read incoming mail while moving. Each time the user connects to a different email server, that server fetches all the updates from the server that the user previously visited. Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

iii.Monotonic Writes

- A data store is said to be monotonic-write consistent if a write operation by a process on a data item x is completed before any successive write operation on X by the same process.
- A write operation on a copy of data item x is performed only if that copy has been brought up to date by means of any preceding write operations, which may have taken place on other copies of x.
- Example: Monotonic-write consistency guarantees that if an update is performed on a copy of Server S, all preceding updates will be performed first. The resulting server will then indeed become the most recent version and will include all updates that have led to previous versions of the server.

iv.Read Your Writes

- A data store is said to provide read-your-writes consistency if the effect of a write operation by a process on data item x will always be a successive read operation on x by the same process.
- A write operation is always completed before a successive read operation by the same process no matter where that read operation takes place.
- Example: Updating a Web page and guaranteeing that the Web browser shows the newest version instead of its cached copy.

v.Writes Follow Reads

- A data store is said to provide writes-follow-reads consistency if a process has write operation on a data item x following a previous read operation on x then it is guaranteed to take place on the same or a more recent value of x that was read.
- Any successive write operation by a process on a data item x will be performed on a copy of x that is up to date with the value most recently read by that process.
- Example: Suppose a user first reads an article A then posts a response B. By requiring writes-follow-reads consistency, B will be written to any copy only after A has been written.

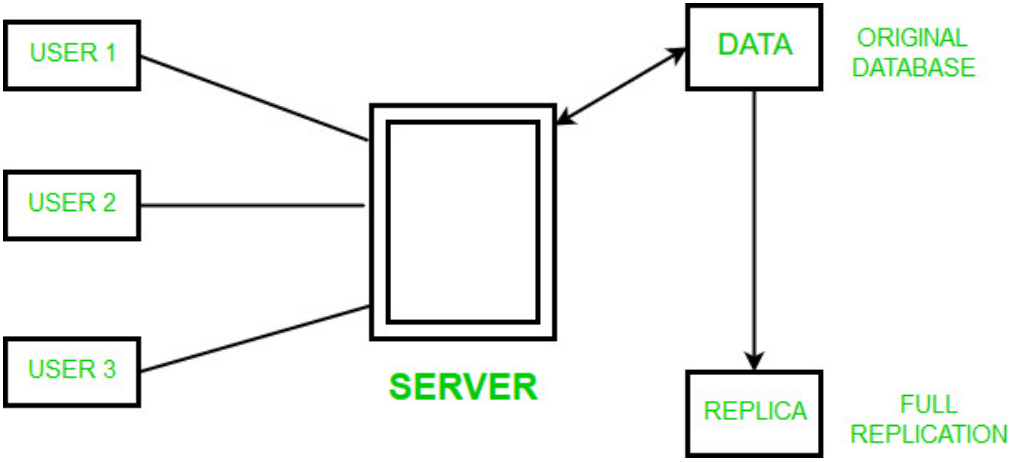
Q5. Short note on Replica Management. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: Replication is the key factor in improving the availability of data in distributed systems. Replicated data is stored at multiple sites so that it can be accessed by the user even when some of the copies are not available due to site failures. A major restriction to using replication is that replicated copies must behave like a single copy, i.e., mutual consistency as well as internal consistency must be preserved. Synchronization techniques for replicated data in distributed database systems have been studied in order to increase the degree of concurrency and to reduce the possibility of transaction rollback.

However in data replication data is available at different locations, but a particular relation has to reside at only one location. There can be full replication, in which the whole database is stored at every site. There can also be partial replication, in which some frequently used fragments of the database are replicated and others are not replicated. Types of Data Replication –

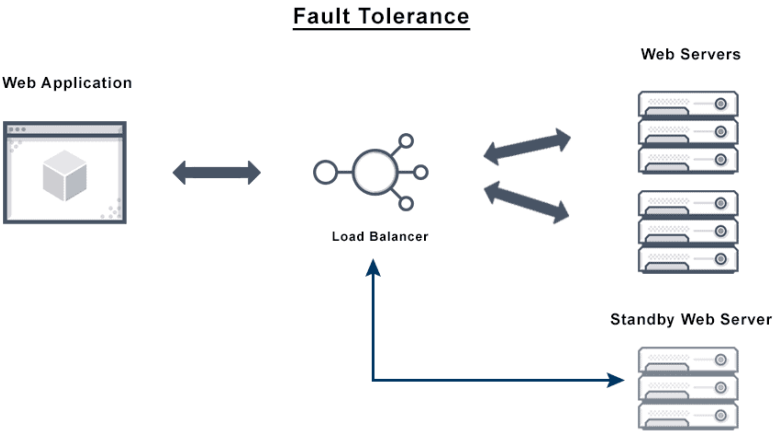
1. Transactional Replication – In Transactional replication users receive full initial copies of the database and then receive updates as data changes. Data is copied in real time from the publisher to the receiving database(subscriber) in the same order as they occur with the publisher therefore in this type of replication, transactional consistency is guaranteed. Transactional replication is typically used in server-to-server environments. It does not simply copy the data changes, but rather consistently and accurately replicates each change.
2. Snapshot Replication – Snapshot replication distributes data exactly as it appears at a specific moment in time and does not monitor for updates to the data. The entire snapshot is generated and sent to Users. Snapshot replication is generally used when data changes are infrequent. It is a bit slower than transactional because on each attempt it moves multiple records from one end to the other end. Snapshot replication is a good way to perform initial synchronization between the publisher and the subscriber.
3. Merge Replication – Data from two or more databases is combined into a single database. Merge replication is the most complex type of replication because it allows both publisher and subscriber to independently make changes to the database. Merge replication is typically used in server-to-client environments. It allows changes to be sent from one publisher to multiple subscribers.

Replication Schemes – 1. Full Replication – The most extreme case is replication of the whole database at every site in the distributed system. This will improve the availability of the system because the system can continue to operate as long as atleast one site is up.



Q6. Explain Fault Tolerance Mechanism. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: Fault Tolerance simply means a system's ability to continue operating uninterrupted despite the failure of one or more of its components. This is true whether it is a computer system, a cloud cluster, a network, or something else. In other words, fault tolerance refers to how an operating system (OS) responds to and allows for software or hardware malfunctions and failures.



- An OS's ability to recover and tolerate faults without failing can be handled by hardware, software, or a combined solution leveraging load balancers(see more below). Some computer systems use multiple duplicate fault tolerant systems to handle faults gracefully. This is called a fault tolerant network.

Fault tolerant computing may include several levels of tolerance:

- At the lowest level, the ability to respond to a power failure, for example.
- A step up: during a system failure, the ability to use a backup system immediately.
- Enhanced fault tolerance: a disk fails, and mirrored disks take over for it immediately. This provides functionality despite partial system failure, or graceful degradation, rather than an immediate breakdown and loss of function.
- High level fault tolerant computing: multiple processors collaborate to scan data and output to detect errors, and then immediately correct them.

Fault tolerance software may be part of the OS interface, allowing the programmer to check critical data at specific points during a transaction.

- Fault-tolerant systems ensure no break in service by using backup components that take the place of failed components automatically.

These may include:

- Hardware systems with identical or equivalent backup operating systems. For example, a server with an identical fault tolerant server mirroring all operations in backup, running in parallel, is fault tolerant.
- By eliminating single points of failure, hardware fault tolerance in the form of redundancy can make any component or system far safer and more reliable.

- Software systems backed up by other instances of software. For example, if you replicate your customer database continuously, operations in the primary database can be automatically redirected to the second database if the first goes down.
- Redundant power sources can help avoid a system fault if alternative sources can take over automatically during power failures, ensuring no loss of service.

Q7. What is Recovery in a Distributed System? (P4 - Appeared 1 Time) (5-10 Marks)

Ans: Recovery from an error is essential to fault tolerance, and error is a component of a system that could result in failure. The whole idea of error recovery is to replace an erroneous state with an error-free state. Error recovery can be broadly divided into two categories.

1. Backward Recovery:
    - Moving the system from its current state back into a formerly accurate condition from an incorrect one is the main challenge in backward recovery. It will be required to accomplish this by periodically recording the system's state and restoring it when something goes wrong. A checkpoint is deemed to have been reached each time (part of) the system's current state is noted.
2. Forward Recovery:
    - Instead of returning the system to a previous, checkpointed state in this instance when it has entered an incorrect state, an effort is made to place the system in a correct new state from which it can continue to operate. The fundamental issue with forward error recovery techniques is that potential errors must be anticipated in advance. Only then is it possible to change those mistakes and transfer to a new state.

- These two types of possible recoveries are done in fault tolerance in distributed systems.

Stable Storage :

- Stable storage, which can resist anything but major disasters like floods and earthquakes, is another option. A pair of regular discs can be used to implement stable storage. Each block on drive 2 is a duplicate of the corresponding block on drive 1, with no differences. The block on drive 1 is updated and confirmed first whenever a block is updated. then the identical block on drive 2 is finished.
- Suppose that the system crashes after drive 1 is updated but before the update on drive 2. Upon recovery, the disk can be compared with blocks. Since drive 1 is always updated before drive 2, the new block is copied from drive 1 to drive 2 whenever two comparable blocks differ, it is safe to believe that drive 1 is the correct one. Both drives will be identical once the recovery process is finished.
- Each process periodically saves its state to a locally accessible stable storage in backward error recovery techniques. We must create a stable global state from these local states in order to recover from a process or system failure. Recovery to the most current distributed snapshot, also known as a recovery line, is recommended in particular.

Coordinated Checkpointing :

- As the name suggests, coordinated checkpointing synchronizes all processes to write their state to local stable storage at the same time. Coordinated checkpointing's key benefit is that the saved state is automatically globally consistent, preventing cascading rollbacks that could cause a domino effect.

Message Logging :

- The core principle of message logging is that we can still obtain a globally consistent state even if the transmission of messages can

be replayed, but without having to restore that state from stable storage. Instead, any communications that have been sent since the last checkpoint are simply retransmitted and treated appropriately.

MODULE-6

Q1. What Are The Features of DFS. Explain Any one in detail. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: A distributed file system (DFS) is a file system that is distributed on various file servers and locations. It permits programs to access and store isolated data in the same method as in the local files. It also permits the user to access files from any system. It allows network users to share information and files in a regulated and permitted manner. Although, the servers have complete control over the data and provide users access control.

A good distributed file system should have the following features:

1. Transparency

- Transparency refers to hiding details from a user. The following types of transparency are desirable.
- i. Structure transparency: Multiple file servers are used to provide better performance, scalability, and reliability. The multiplicity of file servers should be transparent to the client of a distributed file system. Clients should not know the number or locations of file servers or the storage devices instead it should look like a conventional file system offered by a centralized, time sharing operating system.
- iii. Naming transparency: The name of the file should not reveal the location of the file. The name of the file must not be changed while moving from one node to another.
- iv. Replication transparency: The existence of multiple copies and their locations should be hidden from the clients where files are replicated on multiple nodes.

2. User mobility
    - The user should not be forced to work on a specific node but should have the flexibility to work on different nodes at different times. This can be achieved by automatically bringing the user's environment to the node where the user logs in.
3. Performance
    - Performance is measured as the average amount of time needed to satisfy client requests, which includes CPU time plus the time for accessing secondary storage along with network access time. Explicit file placement decisions should not be needed to increase the performance of a distributed file system.
4. Simplicity and ease of use
5. User interface to the file system should be simple and the number of commands should be as small as possible. A DFS should be able to support the whole range of applications.
6. Scalability
    - A good DFS should cope with an increase of nodes and not cause any disruption of service. Scalability also includes the system to withstand high service load, accommodate growth of users and integration of resources.
7. High availability
    - A distributed file system should continue to function even in partial failures such as a link failure, a node failure, or a storage device crash. Replicating files at multiple servers can help achieve availability.
8. High reliability
    - Probability of loss of stored data should be minimized. System should automatically generate backup copies of critical files in event of loss.

9. Data integrity
    - Concurrent access requests from multiple users who are competing to access the file must be properly synchronized by the use of some form of concurrency control mechanism. Atomic transactions can also be provided to users by a file system for data integrity.

10. Security
    - A distributed file system must secure data so that its users are confident of their privacy. File system should implement mechanisms to protect data that is stored within.

11. Heterogeneity
    - Distributed file systems should allow various types of workstations to participate in sharing files via distributed file systems. Integration of a new type of workstation or storage media should be designed by a DFS.

Q2. Write classification of File models. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: In Distributed File Systems (DFS), multiple machines are used to provide the file system's facility. Different file systems often employ different conceptual models. The models based on structure and mobility are commonly used for the modeling of files.

There are two types of file models:

- Unstructured and Structured Files
- Mutable and Immutable Files

Based on the *structure criteria*, file models are of two types:

1. Unstructured Files: It is the simplest and most commonly used model. A file is a collection of an unstructured sequence of data in

the unstructured model. There is no substructure associated with it. The data and structure of each file available in the file system is an uninterpreted sequence of bytes as it relies on the application used like UNIX or DOS. Most modern OS prefer to use the unstructured file model instead of the structured file model because of sharing of files by different applications. It follows no structure so different applications can interpret in different ways.

2. Structured Files: The rarely used file model now is the Structured file model. Here in the structured file model, the file system sees a file consisting of a collection of a sequence of records in order. Files exhibit different types, different sizes, and different properties. It can also be possible that records of different files belonging to the same file system are of variant sizes. Files possess different properties despite belonging to the same file system. The smallest unit of data that can be retrieved is termed a record. The read or write operations are performed on a set of records. In a structured files system, there are various "File Attributes" available, which describe the file. Each attribute consists of a name with its value. File attributes rely on the file system used. It contains information regarding files, file size, file owner, date of last modification, date of file creation, access permission, and date of last access. The Directory Service facility is used to maintain file attributes because of the varying access permissions.

The structured files further consist of two types:

- Files with Non-Indexed records: In files with non-indexed records, the retrieving of records is performed concerning a position in the file. For example, the third record from the beginning, the third record from the last/end.

- Files with Indexed records: In files with indexed records, one or more key fields exist in each record, each of which can be addressed by providing its value. To locate records fast, a file is maintained as a B-tree or other equivalent data structure or hash table.

Based on the modifiability criteria, file models are of two types:

3. Mutable Files: The mutable file model is used by the existing OS. The existing contents of a file get overwritten by the new contents after file updating. As the same file gets updated again and again after writing new contents, a file is described as a single sequence of records.

4. Immutable Files: Cedar File System uses the Immutable file model. In the immutable file model, the file cannot be changed once it has been created. The file can only be deleted after its creation. To implement file updates, multiple versions are created of the same file. Every time a new version of the file is created when a file is updated. There is consistent sharing in this file model because of the sharing of only immutable files.

Q3. What Are the File-Caching Schemes?(P4 - Appeared 1 Time) (5-10 Marks)

Ans: A file-caching scheme for a distributed file system contributes to its scalability and reliability as it is possible to cache remotely located data on a client node. Every distributed file system uses some form of file caching. The following can be used:

1.Cache Location

Cache location is the place where the cached data is stored. There can be three possible cache locations

i.Servers main memory:

A cache located in the server's main memory eliminates the disk access cost on a cache hit which increases performance compared to no caching.

The reason for keeping locating cache in server's main memory-

- Easy to implement
- Totally transparent to clients
- Easy to keep the original file and the cached data consistent.

ii.Clients disk:

If cache is located in clients disk it eliminates network access cost but requires disk access cost on a cache hit. This is slower than having the cache in servers main memory. Having the cache in the server's main memory is also simpler.

Advantages:

- Provides reliability.
- Large storage capacity.
- Contributes to scalability and reliability.

Disadvantages:

- Does not work if the system is to support diskless workstations.
- Access time is considerably large.

iii.Clients main memory

A cache located in a client's main memory eliminates both network access cost and disk access cost. This technique is not preferred to a client's disk cache when large cache size and increased reliability of cached data are desired.

Advantages:

- Maximum performance gain.
- Permits workstations to be diskless.

- Contributes to reliability and scalability.

2.Modification propagation

When the cache is located on a client's nodes, a file's data may simultaneously be cached on multiple nodes. It is possible for caches to become inconsistent when the file data is changed by one of the clients and the corresponding data cached at other nodes are not changed or discarded.

The modification propagation scheme used has a critical effect on the systems performance and reliability.

Techniques used include –

i.Write-through scheme

When a cache entry is modified, the new value is immediately sent to the server for updating the master copy of the file.

Advantage:

1. High degree of reliability and suitability for UNIX-like semantics.
2. The risk of updated data getting lost in the event of a client crash is low.

Disadvantage:

1. Poor Write performance.

ii.Delayed-write scheme

- To reduce network traffic for writes the delayed-write scheme is used. New data value is only written to the cache when an entry is modified and all updated cache entries are sent to the server at a later time.

There are three commonly used delayed-write approaches:

- Write on ejection from cache:
    - Modified data in cache is sent to the server only when the cache-replacement policy has decided to eject it from the client's cache. This can result in good performance but there

can be a reliability problem since some server data may be outdated for a long time.

- Periodic write:
    - The cache is scanned periodically and any cached data that has been modified since the last scan is sent to the server.
- Write on close:
    - Modification to cached data is sent to the server when the client closes the file. This does not help much in reducing network traffic for those files that are open for very short periods or are rarely modified.

Advantages:

1. Write accesses complete more quickly that result in a performance gain.

Disadvantage:

1. Reliability can be a problem.

3. Cache validation schemes

- The modification propagation policy only specifies when the master copy of a file on the server node is updated upon modification of a cache entry. It does not tell anything about when the file data residing in the cache of other nodes is updated. A file data may simultaneously reside in the cache of multiple nodes. A client's cache entry becomes stale as soon as some other client modifies the data corresponding to the cache entry in the master copy of the file on the server. It becomes necessary to verify if the data cached at a client node is consistent with the master copy. If not, the cached data must be invalidated and the updated version of the data must be fetched again from the server.

There are two approaches to verify the validity of cached data:

i.Client-initiated approach

The client contacts the server and checks whether its locally cached data is consistent with the master copy.

- Checking before every access- This defeats the purpose of caching because the server needs to be contacted on every access.
- Periodic checking- A check is initiated every fixed interval of time.
- Check on file open- Cache entry is validated on a file open operation.

ii.Server-initiated approach

- A client informs the file server when opening a file, indicating whether a file is being opened for reading, writing, or both. The file server keeps a record of which client has which file open and in what mode. The server monitors file usage modes being used by different clients and reacts whenever it detects a potential for inconsistency. E.g. if a file is open for reading, other clients may be allowed to open it for reading, but opening it for writing cannot be allowed. So also, a new client cannot open a file in any mode if the file is open for writing.
- When a client closes a file, it sends intimation to the server along with any modifications made to the file. Then the server updates its record of which client has which file open in which mode.
- When a new client makes a request to open an already open file and if the server finds that the new open mode conflicts with the already open mode, the server can deny the request, queue the request, or disable caching by asking all clients having the file open to remove that file from their caches.

Q4. Write a short note on File Replication. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: File Replication Service (FRS) is a feature in Microsoft Windows Server which is a successor to the LAN Manager Replication service of Windows NT Server. It is used for the replication of the system policies and script by the Windows Server. This data is stored in the SYSVOL, or the system volume, of the server.

- It is stored in the controllers of the domain, and can be accessed by the client servers of the network. Distributed File System Replication Service is now quickly replacing File Replication Service.
- FRS is a service which allows the sharing of Group Policies and logon scripts to the domain controllers, from where they may be accessed by the users through the client servers. The executable file running the service is NTFRS.exe. This service can also be used to replicate files and synchronize the data of its domain controllers using a DFS. It is also able to keep data on multiple servers at once.
- The synchronization process is quick and complete. As it initiates very important scripts and processes which are required for logon, the services have to be quick, efficient and dependable. This service fits all the requirements because it backs up all the data on different servers while replicating them. The sync service is very fast and any changes in the policies are instantaneously changed in the client's data.

Q5. Write a short note on File Accessing models. (P4 - Appeared 1 Time) (5-10 Marks)

Ans: The specific client's request for accessing a particular file is serviced on the basis of the file accessing model used by the distributed file system. The file accessing model basically depends on

- The unit of data access and
- The method used for accessing remote files.

On the basis of the unit of data access, following file access models might be used in order to access the specified file.

1. File-level transfer model
2. Block-level transfer model
3. Byte-level transfer model
4. Record-level transfer model

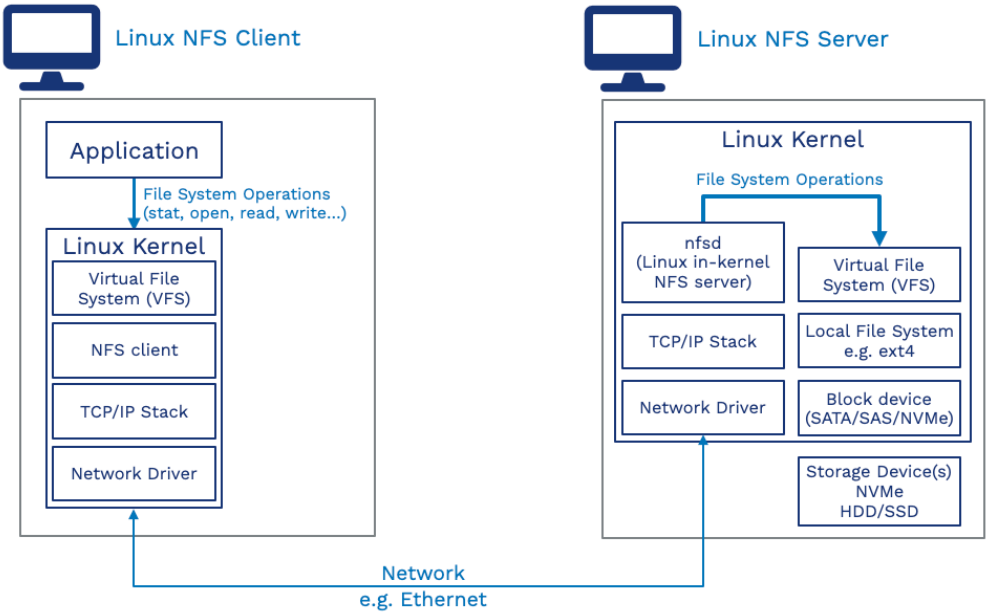
- File-level transfer model: In file-level transfer model, the complete file is moved while a particular operation necessitates the file data to be transmitted all the way through the distributed computing network amongst client and server. This model has better scalability and is efficient.
- Block-level transfer model: In block-level transfer model, file data transfers through the network amongst clients and a server is accomplished in units of file blocks. In short, the unit of data transfer in the block-level transfer model is file blocks. The block-level transfer model might be used in a distributed computing environment comprising several diskless workstations.
- Byte-level transfer model: In byte-level transfer model, file data transfers the network amongst clients and a server is accomplished in units of bytes. In short, the unit of data transfer in the byte-level transfer model is bytes. The byte-level transfer model offers more flexibility in comparison to the other file transfer models since it allows retrieval and storage of an arbitrary sequential subrange of a file. The major disadvantage of the byte-level transfer model is the trouble in cache management because of the variable-length data for different access requests.

- .Record-level transfer model: The record-level file transfer model might be used in the file models where the file contents are structured in the form of records. In the record-level transfer model, file data transfers through the network amongst clients and a server is accomplished in units of records. The unit of data transfer in the record-level transfer model is record.

Q6. What is the Network File System (NFS) and how does it work? (P4 - Appeared 1 Time) (5-10 Marks)

Ans: NFS is an abbreviation of the Network File System. It is a protocol of a distributed file system. This protocol was developed by Sun Microsystems in the year of 1984.

- It is an architecture of the client/server, which contains a client program, server program, and a protocol that helps for communication between the client and server.
- It is that protocol which allows the users to access the data and files remotely over the network. Any user can easily implement the NFS protocol because it is an open standard. Any user can manipulate files as if they were on like other protocols. This protocol is also built on the ONC RPC system.
- This protocol is mainly implemented in those computing environments where the centralized management of resources and data is critical. It uses the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) for accessing and delivering the data and files.
- Network File System is a protocol that works on all the networks of IP-based. It is implemented in that client/server application in which the server of NFS manages the authorization, authentication, and clients. This protocol is used with Apple Mac OS, Unix, and Unix-like operating systems such as Solaris, Linux, FreeBSD, AIX.



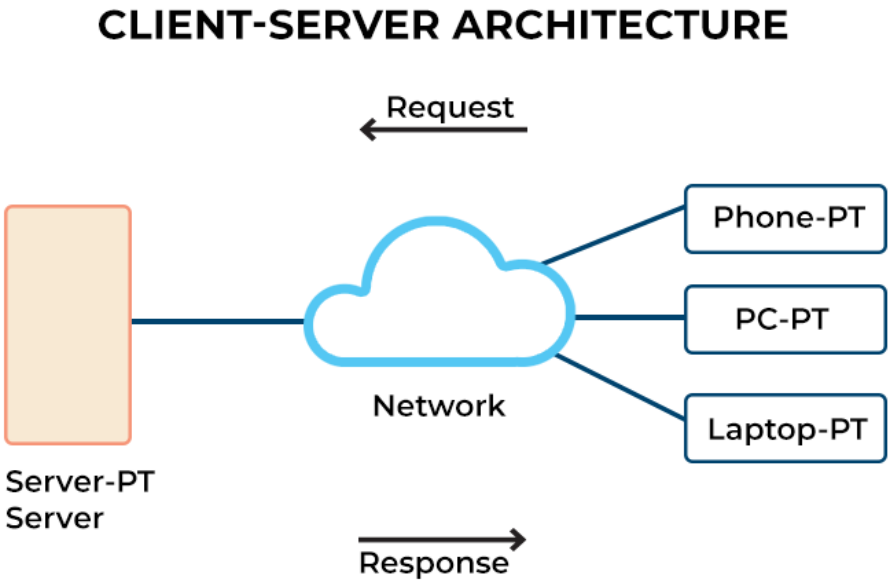
Q7. Explain The Architecture System of Distributed Systems.

Ans: In distributed architecture, components are presented on different platforms and several components can cooperate with one another over a communication network in order to achieve a specific objective or goal.

- System-level architecture focuses on the entire system and the placement of components of a distributed system across multiple machines.
- The client-server architecture and peer-to-peer architecture are the two major system-level architectures that hold significance today. An example would be an ecommerce system that contains a service layer, a database, and a web front.

i) Client-server architecture : As the name suggests, client-server architecture consists of a client and a server. The server is where all the work processes are, while the client is where the user interacts with the

service and other resources (remote server). The client can then request from the server, and the server will respond accordingly. Typically, only one server handles the remote side; however, using multiple servers ensures total safety.

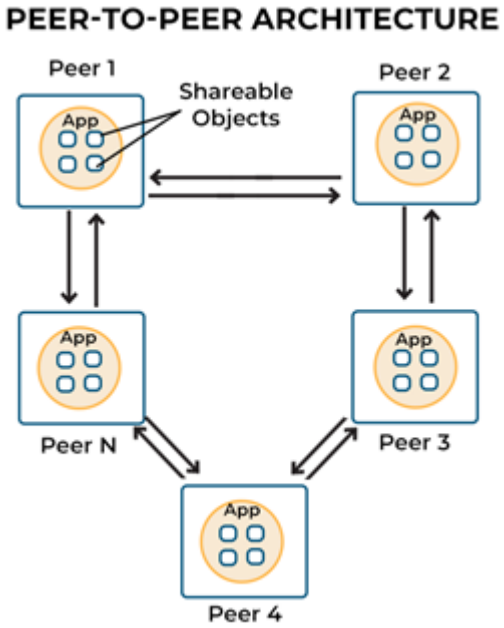


Client-server architecture has one standard design feature: centralized security. Data such as usernames and passwords are stored in a secure database for any server user to have access to this information. This makes it more stable and secure than peer-to-peer.

This stability comes from client-server architecture, where the security database can allow resource usage in a more meaningful way. The system is much more stable and secure, even though it isn't as fast as a server. The disadvantages of a distributed system are its single point of failure and not being as scalable as a server.

ii) Peer-to-peer (P2P) architecture : A peer-to-peer network, also called a (P2P) network, works on the concept of no central control in a distributed

system. A node can either act as a client or server at any given time once it joins the network. A node that requests something is called a client, and one that provides something is called a server. In general, each node is called a peer.



- If a new node wishes to provide services, it can do so in two ways. One way is to register with a centralized lookup server, which will then direct the node to the service provider.
- The other way is for the node to broadcast its service request to every other node in the network, and whichever node responds will provide the requested service.

P2P networks of today have three separate sections:

- Structured P2P: The nodes in structured P2P follow a predefined distributed data structure.

- Unstructured P2P: The nodes in unstructured P2P randomly select their neighbors.
- Hybrid P2P: In a hybrid P2P, some nodes have unique functions appointed to them in an orderly manner.
