1. ___ and ___ are used to hide the access and location of the system.
Option A: access transparency, location transparency.

2. The two popular remote object invocation models are
Option B: CORBA and RMI

3. In distributed systems, a logical clock is associated with _____
Option C: each process

4. Process transfer policy in Load-balancing algorithms is_____
Option B: Determines to which node the transferable process should be sent

5. Client centric consistency model useful in applications where ____
Option B: One client always updates data store

6. In distributed file system, filename does not reveal the file's _____
Option D: Physical storage location

7. The Ricart & Agrawala distributed mutual exclusion algorithm is:
Option D: Less efficient and less fault tolerant than a centralized algorithm.

8. The kernel is _____ of user threads.
Option C: unaware of

9. What is stub?
Option A: transmits the message to the server where the server side stub receives the message and invokes procedure on the server side

10. In a distributed file system, _____ is mapping between logical and physical objects.
Option B: Naming

11. RPC is an example of ----------
Option A: synchronous communication

12. What is a remote object reference?
Option D: An identifier for a remote object that is valid throughout a distributed system

13. In a distributed file system, _____ is mapping between logical and physical objects.
Option B: Naming

14. Concurrency transparency is
Option C: Hide that an object may be shared by several independent users

15. Client centric consistency model useful in applications where____
Option B: One client always updates data store

16. The ring election algorithm works by
Option D: Building a list of all live nodes and choosing the largest numbered node in the list

17. What is a stateless file server?
Option B: It maintains internally no state information at all

18. In which file model, a new version of the file is created each time a change is made to the file contents and the old version is retained unchanged
Option C: Immutable files

19. The Ricart Agrawala distributed mutual exclusion algorithm is:
Option D: Less efficient and less fault tolerant than a centralized algorithm.

20. Which of the following is NOT a technique for achieving scalability
Option A: Centralization

21. A layer which lies between an operating system and the applications running on it is called as
Option D: Middleware

22. Goals of Distributed system does not include
Option C: Sharing memory space

23. Which of the following is not the commonly used semantics for ordered delivery of multicast messages
Option B: Persistent ordering

24. The type of transparency that enables resources to be moved while in use without being noticed by users and application is
Option B: Migration Transparency

25. A paradigm of multiple autonomous computers, having a private memory, communicating through a computer network, is known as
Option A: Distributed computing

26. Following is not the common mode of communication in Distributed system
Option D: Shared memory

27. Following is not the physical clock synchronization algorithm:
Option D: Network time protocol

28. Distributed Mutual Exclusion Algorithm does not use-
Option C: Logical clock for event ordering

29. Vector Timestamp Ordering Algorithm is an example of
Option D: Logical Clock Synchronization

30. What is fault tolerance in distributed Computing?
Option A: Ability of the system to continue functioning in the event of a complete failure.

31. In Task Assignment Approach, we have to
Option A: Minimize IPC cost

32. Backward error recovery requiresOption A: Grouping
Option C: Check pointing

33. Which of these consistency models does not use synchronization operations?
Option A: Sequential

34. Which is not possible in a distributed file system?
Option B: Migration

35. X.500 is a-
Option A: Directory services

36. A DFS is executed as a part of
Option B: Operating system

37. Processes on the remote systems are identified by
Option C: Host name and identifier

38. The function of load-balancing algorithm is
Option A: It tries to balance the total system load by transparently transferring the workload from heavily loaded nodes to lightly loaded

39. A Multi-threaded Server has following threads
Option B: Client Thread

40 Maekawa's Mutual Exclusion Algorithm is based on-
Option C: Voting

PHYSICAL MODEL:

1. A physical model is a representation of a distributed system's underlying hardware parts.
2. It describes the hardware components of a system in terms of computers and other devices, as well as the network that connects them.
3. Physical Models are divided into three generations: early distributed systems, Internet-scale distributed systems, and contemporary distributed systems.

I) Early distributed systems:

- These systems arose in the late 1970s and early 1980s as a result of the widespread adoption of local area networking technology.
- The system was generally made up of 10 to 100 nodes connected via a LAN, with limited Internet access and services.
- For instance, a shared local printer and file servers

II) Internet-scale distributed systems:

- These systems arose in the 1990s as the Internet grew in popularity.
- It connects a vast number of nodes from different organizations.
- Heterogeneity has risen.

III) Contemporary distributed systems:

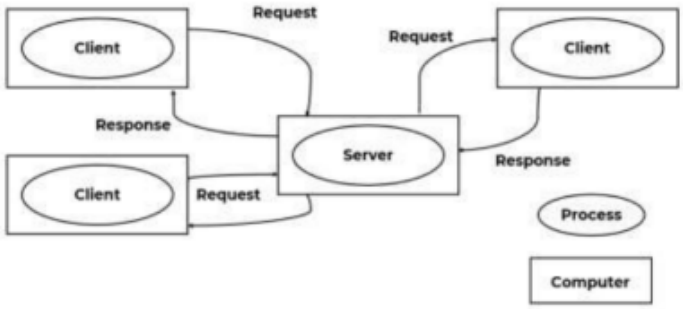
- The rise of mobile computing has resulted in location-independent nodes.
- Cloud computing and ubiquitous computing are on the rise.
- The scale is enormous.

ARCHITECTURAL MODEL:

1. The architectural model describes how the system's components interact with one another and how they are translated into an underlying network of computers.
2. It abstracts and simplifies the functioning of individual distributed system components.
3. It explains how duties are allocated across system components and where they are located.
4. Examples: Client Server Model and Peer to Peer Model.

I) Client Server Model:

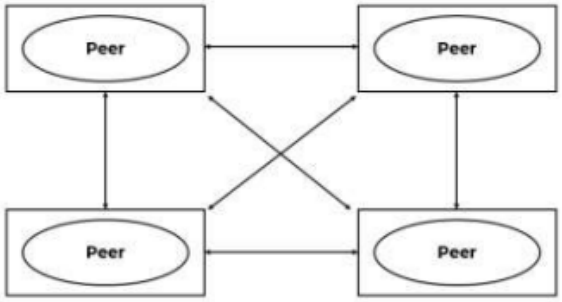
- Processes in a distributed system are separated into two categories in the basic client-server model: server and client.
- A server implements a particular service, such as a file system or database service.
- A client sends a request to a server and then waits for the server to respond.
- A basic request/reply protocol is commonly used in the client-server concept.
- A server can request services from other servers, thereby becoming the server a client in this new relationship.



II) Peer to Peer Model:

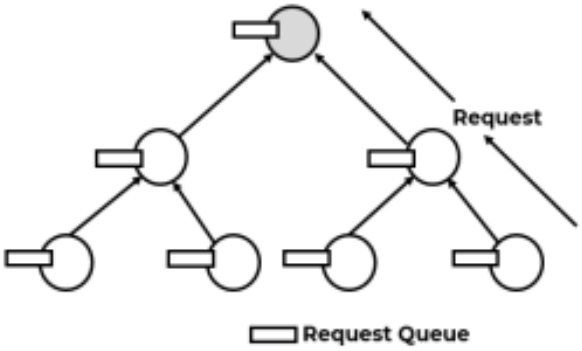
- The main concept behind the peer to peer model is that in a distributed system, there is no central control.
- The fundamental idea is that at any one time, any node can be either a client or a server.
- When a node requests something, it is referred to as a client, and when a node provides something, it is referred to as a server.

- Each node is referred to as a peer in general and this is the most general and flexible model.
- Any new node in this network must first join the network.
- They may either seek or provide services when they join this network
- All processes in this model have the same capabilities and responsibilities.

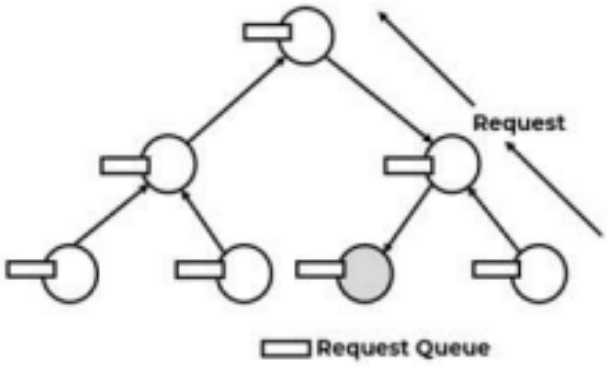


RAYMOND'S TREE BASED ALGORITHM:

1. Raymond's Tree Based Algorithm is a token based algorithm.
2. In Raymond's Tree Based Algorithm, a directed tree is formed.
3. Nodes are arranged as a tree.
4. Every node has a parent pointer.
5. Each node has a FIFO queue of requests.
6. There is one token with the root and the owner of the token can enter the critical section.
7. The figure shows the example of Raymond's Tree Based Algorithm.



8. As the token moves across nodes, the parent pointers change.
9. They always point towards the holder of the token as shown in figure below.
10. It is possible to reach the token by following parent pointers.



Requesting a Token:

1. The node adds "self" in its request queue.
2. Forwards the request to the parent.
3. The parent adds the request to its request queue.
4. If the parent does not hold the token and it has not sent any requests to get the token, it sends a request to its parent for the request.
5. This process continues till we reach the root (holder of the token).

Releasing a Token:

1. Ultimately a request will reach the token holder.
2. The token holder will wait till it is done with the critical section.
3. It will forward the token to the node at the head of its request queue.
   a. It removes the entry.
   b. It updates its parent pointer.
4. Any subsequent node will do the following:
   c. Dequeue the head of the queue.
   d. If "self" was at the head of its request queue, then it will enter the critical section.
   e. Otherwise, it forwards the token to the dequeued entry.
5. After forwarding the entry, a process needs to make a fresh request for the token, if it has outstanding entries in its request queue.

SUZUKI-KASAMI'S BROADCAST ALGORITHM:

1 Suzuki-Kasami's Broadcast Algorithm is a token based algorithm.

2. It is used for achieving mutual exclusion in distributed systems.

3. This is a modification to the Ricart-Agrawala algorithm.

4. In this algorithm a REQUEST and REPLY message are used for attaining the critical section.

5. The process holding the token is the only process able to enter its critical section.

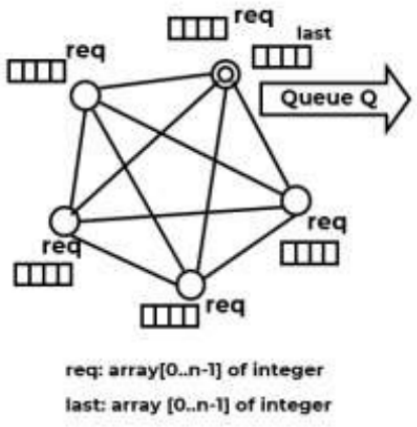
6. Each process maintains an array request.

- Req[j] denotes the sequence no of the latest request from process j

7. Additionally, the holder of the token maintains an array last.

- Last[j] denotes the sequence number of the latest visit to Critical Section for process j.

8. The figure shows an example of Suzuki-Kasami's Broadcast Algorithm.



ALGORITHM:

Requesting the critical section (CS):

1. If the site does not have the token, then it increases its sequence number Reqi [i] and sends a request (i, sn) message to all other sites (sn= Reqi [i])
2. When a site Sj receives this message, it sets Reqi [i] to Max (Reqi [i], sn). If Sj has the idle token, then it sends the token to Si, if Reqj [i] = Last [i] + 1

Releasing the CS:

1. When done with the CS, site Si sets Last [i] = Reqi [i]
2. For every site Sj whose ID is not in the token queue, it appends its ID to the token queue if Reqi [j] =Last[j] + 1
3. If the queue is not empty, it extracts the ID at the head of the queue and sends the token to that site.

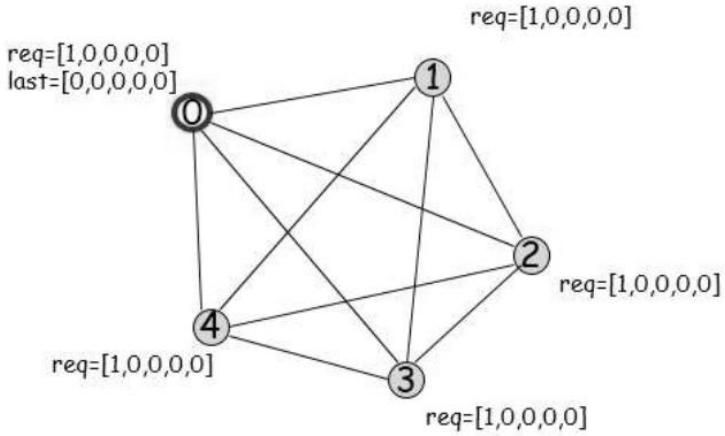
Executing the CS:

- Site Si executes the CS when it has received the token.

EXAMPLE:

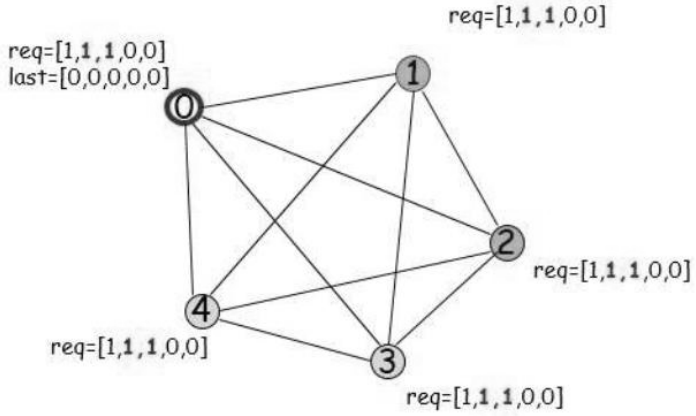
Step 1:

Initial state: Process 0 has sent a request to all, and grabbed the token.



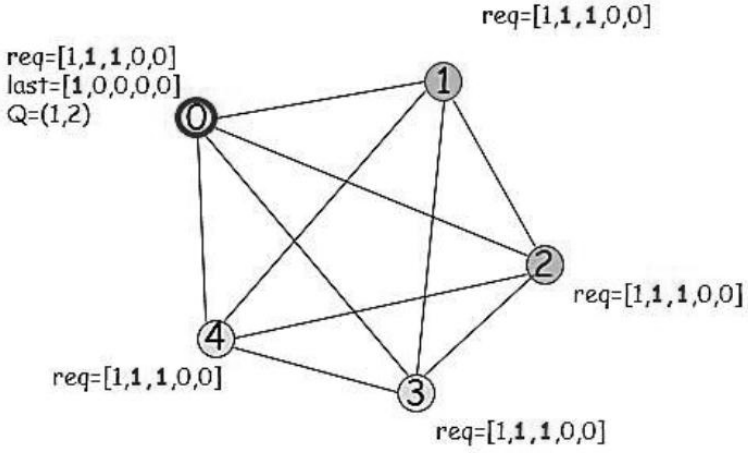
Step 2:

Now 1 & 2 send requests to enter CS



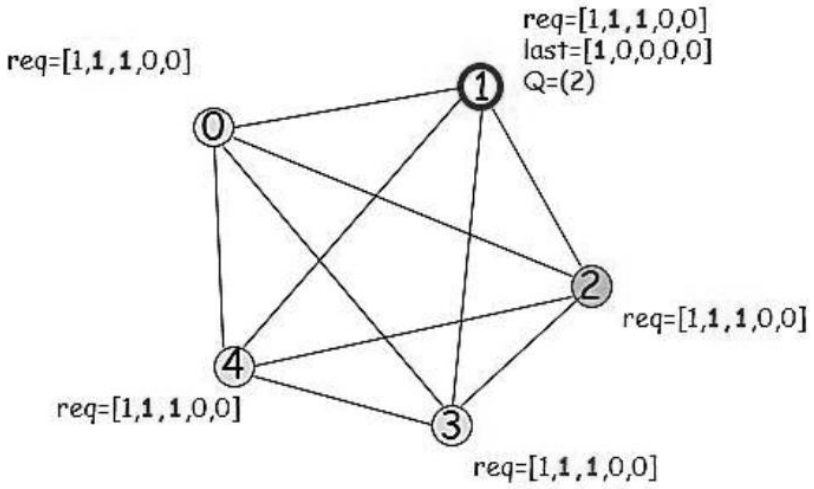
Step 3:

Now 0 prepares to exit CS



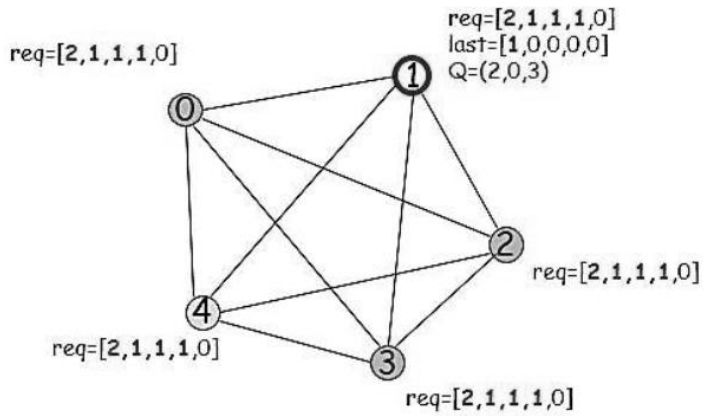
Step 4:

Now 0 passes token (Q and last) to 1



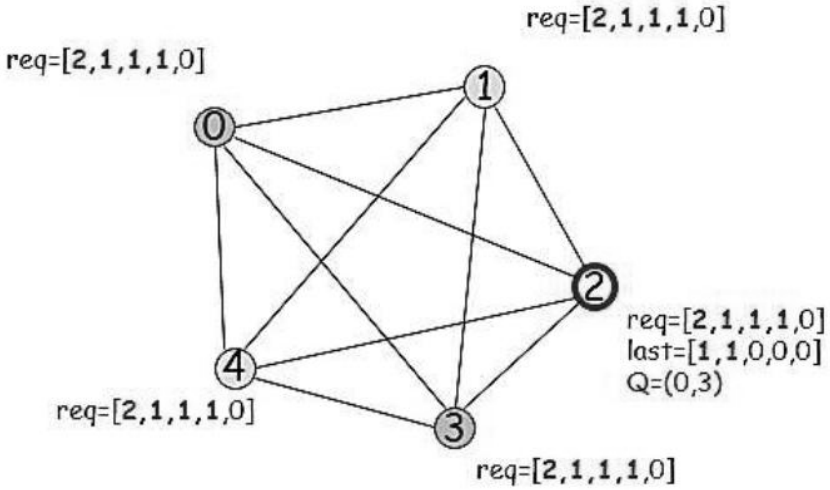
Step 5:

Now 0 and 3 send requests



Step 6:

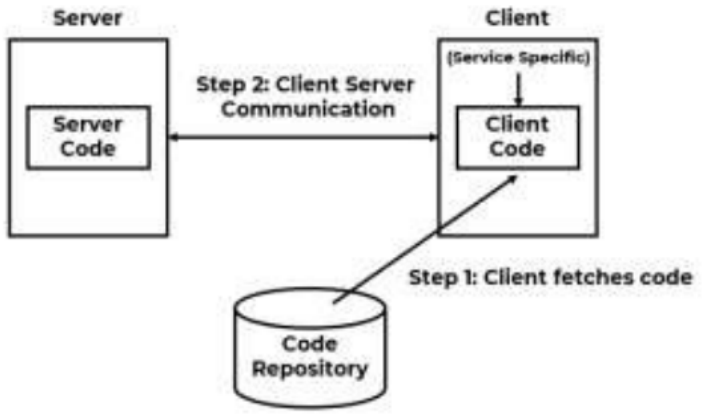
Finally 1 sends token to 2.



CODE MIGRATION:

1. In process migration, an entire process has to be moved from one machine to another.
2. But this may be a crucial task, but it has good overall performance.
3. In some cases, a code needs to be migrated rather than the process.
4. Code Migration refers to transfer of a program from one node to another.
5. For example: Consider a client server system, in which the server has to manage a big database.
6. The client application has to perform many database operations.
7. In such a situation, it is better to move part of the client application to the server and the result is sent across the network.
8. Thus code migration is a better option.
9. Code Migration is used to improve overall performance of the system by exploiting parallelism.
10. The advantage of this approach is that the client does not need to install all the required software. The software can be moved in as when necessary and discarded when it is not needed.

Example:



- Here the server is responsible to provide the client's implementation, when the client binds to the server.

CODE MIGRATION ISSUES:

1. Communication in distributed systems is concerned with exchanging data between processes.
2. Code migration in the broadest sense which deals with moving programs between machines, with the intention to have those programs be executed at the target.
3. In code migration framework, a process consists of 3 segments i.e. code segment, resource segment and execution segment.
4. Code segment contains the actual code.
5. Resource segment contains references to resources needed by the process.
6. Execution segment stores the execution state of a process.
7. Mobile agents move from site to site.
8. Code-on-demand is used to migrate part of the server to the client.
9. Remote evaluation is used to perform database operations involving large amounts of data.

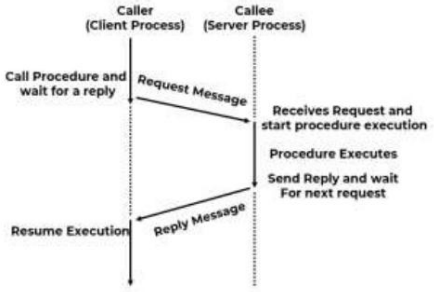
Reasons for Migrating Code:

- Performance is improved by moving code from heavily loaded machines to lightly loaded machines, although it is a costly operation.
- If a server manages a huge database and client application performs operations on this database that involves a large quantity of data.
- In this case, it is better to transfer part of the client application to the server machine.
- In other cases, instead of sending requests for data validation to the server it is better to carry out client side validation.
- Hence, to transfer part of the server application to the client machine.
- Code migration also helps to improve performance by exploiting parallelism where there is no need to consider details of parallel programming.
- Consider the example of searching for information on the Web, a search query which is a small mobile program (mobile agent) moves from site to site.
- Each copy of this mobile agent can be transferred to different sites to achieve a linear speedup compared to using just a single program instance.
- Code migration also helps to configure a distributed system dynamically.
- In other cases, there is no need to keep all software at client.
- Whenever the client binds with the server, it can dynamically download the required software from the web server.
- Once work is done, all this software can be discarded.
- No need to preinstall client side software.

REMOTE PROCEDURE CALL:

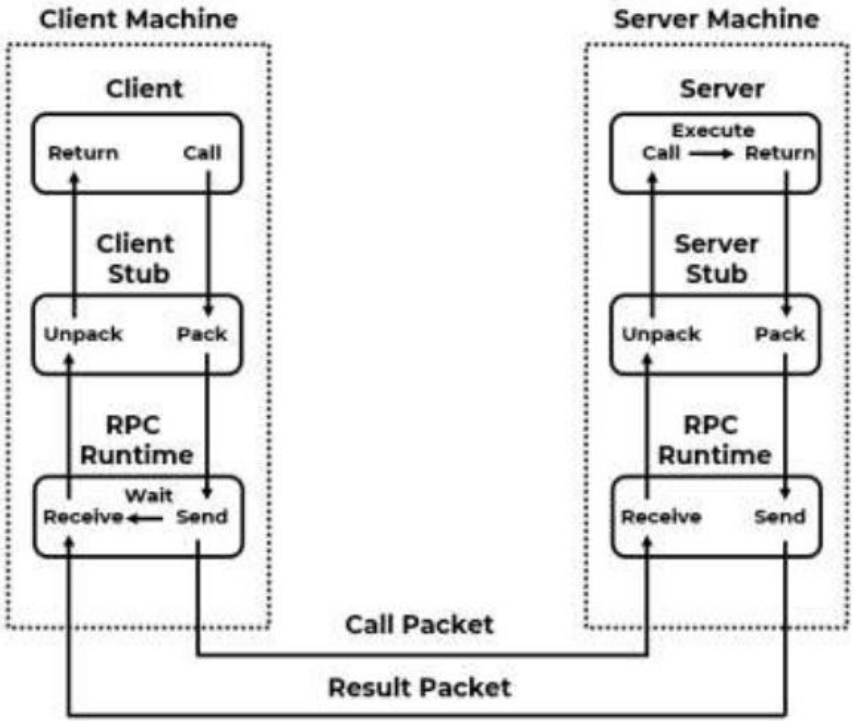
1. Remote Procedure Call is as known as Remote Function Call or a Remote Subroutine Call.
2. A remote procedure call is an Interprocess communication technique that is used for client-server based applications.
3. RPC provides a general IPC protocol that can be used for designing several distributed applications.
4. Features of a RPC for distributed applications are:
   a. Simple call syntax.
   b. Familiar semantics.
   c. Ease of use.
   d. Generality.
   e. Efficiency.

GENERIC RPC MODEL:



1. Figure above shows the generic RPC Model.
2. The basic idea of RPC is to make a remote procedure call look transparent.
3. The calling procedure should not be aware that the called procedure is executing on a different machine.
4. RPC achieves transparency in an analogous way.
5. The caller (client process) sends a call (request) message to the callee (server process) and waits for the reply.
6. The request message contains the remote procedures parameters, among other things.
7. The server process executes the procedure and then returns the result of procedure execution in a reply message to the client process.
8. Once the reply message is received, the result of procedure execution is extracted, and the caller's execution is resumed.
9. It has no idea that work was done remotely instead of the local operating system.
10. In this way transparency is achieved.

IMPLEMENTATION OF RPC:



1. To achieve the goal of semantic transparency the implementation of an RPC mechanism is based on the concept of STUBS.
2. Stubs provide a local procedure call abstraction.
3. The implementation of an RPC mechanism usually involves the following five elements.
4. Client: The user process that initiates a remote procedure call.
5. Client Stub: It is responsible for:
   a. On receipt of a call request from the client, it packs a specification of the target procedure and the arguments into the message.
   b. On receipt of the result of procedure execution, it unpacks the result and passes it to the client.
6. RPC Runtime: It handles transmission of messages across the network between client and server machines including encryption, routing acknowledgement.
7. Server Stub: The job of the server stub is very similar to client stubs.
8. Server: The server executes the appropriate procedure and returns the result.

ADVANTAGES OF REMOTE PROCEDURE CALL

1. Remote procedure calls support process oriented and thread oriented models.
2. The internal message passing mechanism of RPC is hidden from the user.
3. The effort to re-write and re-develop the code is minimal in remote procedure calls.
4. Remote procedure calls can be used in distributed environments as well as the local environment.

DISADVANTAGES OF REMOTE PROCEDURE CALL

1. The remote procedure call is a concept that can be implemented in different ways. It is not a standard.
2. There is no flexibility in RPC for hardware architecture. It is only interaction based.
3. There is an increase in costs because of remote procedure calls.

DATA CENTRIC CONSISTENCY MODEL:

1. Data-centric consistency models aim to provide system wide consistent view of the data store.
2. A data store may be physically distributed across multiple machines.
3. Each process that can access data from the store is assumed to have a local or nearby copy available of the entire store.
4. Figure shows the example of a data centric consistency model.

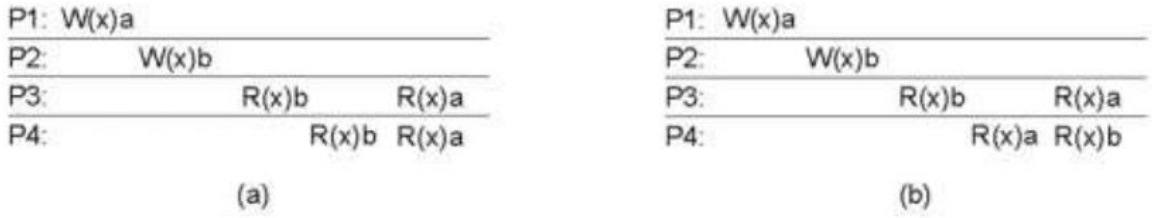
I) Strict Consistency Model:

1. Any read on a data item X returns a value corresponding to the result of the most recent write on X.
2. This is the strongest form of memory coherence which has the most stringent consistency requirement.
3. Behavior of two processes, operating on the same data item.

a. A strictly consistent store.
b. A store that is not strictly consistent.

II) Sequential Consistency:

1. Sequential consistency is an important data-centric consistency model which is a slightly weaker
consistency model than strict consistency.
2. 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.



(a) A sequentially consistent data store.
(b) A data store that is not sequentially consistent.

III) Linearizability:

1. It is weaker than strict consistency, but stronger than sequential consistency.
2. 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.
3. The operations of each individual process appear in sequence order specified by its program.
4. If tSopi(x) < tSoP2(y), then operation OP1(x) should precede OP2(y) in this sequence.

IV) Causal Consistency:

1. It is a weaker model than sequential consistency.
2. In Causal Consistency all processes see only those memory reference operations in the correct order that are potentially causally related.
3. Memory reference operations which are not related may be seen by different processes in different order.
4. 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.
5. If a write (w2) operation is causally related to another write (wl) the acceptable order is (wl, w2).

V) FIFO Consistency:

1. It is weaker than causal consistency.
2. 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.
3. This model is simple and easy to implement, having good performance because processes are ready in the pipeline.
4. Implementation is done by sequencing write operations performed at each node independently of the operations performed on other nodes.

P1: W(x)a						
P2:	R(x)a	W(x)b	W(x)c			
P3:				R(x)b	R(x)a	R(x)c
P4:				R(x)a	R(x)b	R(x)c

VI) Weak Consistency:

1. The basic idea behind the weak consistency model is enforcing consistency on a group of memory reference operations rather than individual operations.
2. 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.
3. 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:

1. 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.
2. 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.
3. Release consistency can be viewed as a synchronization mechanism based on barriers instead of critical sections.

VIII) Entry Consistency:

1. In entry consistency every shared data item is associated with a synchronization variable.
2. In order to access consistent data, each synchronization variable must be explicitly acquired.
3. Release consistency affects all shared data but entry consistency affects only those shared data associated with a synchronization variable.

ELECTION ALGORITHM:

1. Many algorithms used in distributed systems require a coordinator.
2. In general, all processes in the distributed system are equally suitable for the role.
3. Election algorithms are designed to choose a coordinator.
4. Any process can serve as coordinator.
5. Any process can "call an election".
6. There is no harm in having multiple concurrent elections.
7. Elections may be needed when the system is initialized, or if the coordinator crashes or retires.
8. Example of election algorithm is the Bully Algorithm.

Assumptions and Requirement of Election Algorithm:

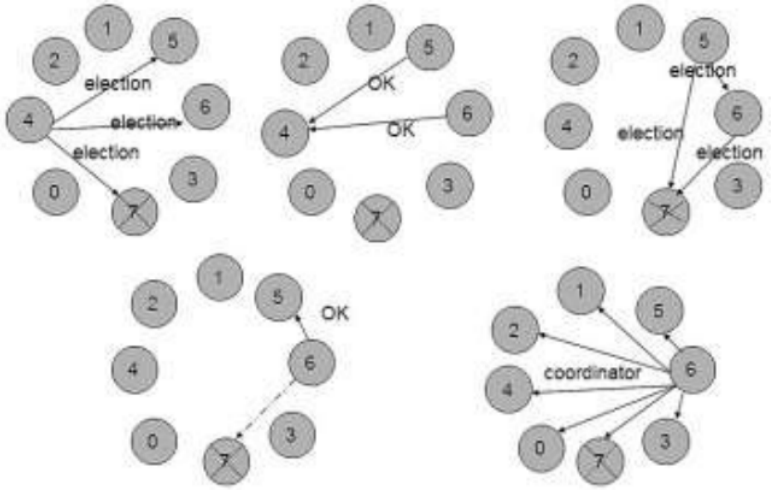
1. Every process/site has a unique ID.
2. Example: The network address or a process number.
3. Every process in the system should know the values in the set of ID numbers, although not which processors are up or down.
4. The process with the highest ID number will be the new coordinator.
5. Process groups (as with ISIS toolkit or MPI) satisfy these requirements.
6. When the election algorithm terminates a single process has been selected and every process knows its identity.
7. Formalize: every process pi has a variable e, to hold the coordinator's process number. vi, e_i = undefined or e; = P, where P is the non-crashed process with highest id.
   - All processes (that have not crashed) eventually set e; = P.

BULLY ALGORITHM:

1. The Bully Algorithm was proposed by Garcia-Molina.
2. This algorithm is planned on assumptions that:
   a. Each process involved within a scenario has a process number that can be used for unique identification.
   b. Each process should know the process numbers of all the remaining processes.
   c. Scenario is synchronous in nature.
   d. The process with the highest process number is elected as a coordinator.
3. Process 'p' calls an election when it notices that the coordinator is no longer responding.

4. Process 'p' sends an election message to all higher-numbered processes in the system.
5. If no process response, then p becomes the coordinator.
6. If a higher-level process (q) responds, it sends 'p' a message that terminates p's role in the algorithm.
7. The process 'q' now calls an election (if it has not already done so).
8. Repeat until no higher-level process responds.
9. The last process to call an election "wins" the election.
10. The winner sends a message to other processes announcing itself as the new coordinator.

Example:



1. Process (4) calls an election when it notices that the coordinator is no longer responding.
2. Process (4) sends an election message to all higher-numbered processes in the system.
3. If there is no process response, then Process (4) becomes the coordinator.
4. If a higher-level process (5, 6) responds, it sends process (4) a message that terminates (4)'s role in the algorithm.
5. Now process (5) now calls an election (if it has not already done so).
6. Repeat until no higher-level process responds.
7. The last process to call an election "wins" the election.
8. The winner sends a message to other processes announcing itself as the new coordinator.

Advantages:

- The advantages of the Bully algorithm are that this algorithm is a distributed method with simple implementation.
- This method requires at most five stages, and the probability of detecting a crashed process during the execution of the algorithm is lowered in contrast to other algorithms.
- Therefore other algorithms impose heavy traffic in the network in contrast to the Bully algorithm.
- The processes with higher priority number respect to the priority number of processes that detect the crash coordinator will be involved in election, not all processes are involved.

Disadvantages:

- The number of stages to decide the new leader
- Huge number of messages exchanged due to the broadcasting of elections and OK messages.

FAULT TOLERANCE:

1. A system fails when it does not match its promises.
2. An error in a system can lead to a failure.
3. The cause of an error is called a fault.
4. Faults are generally transient, intermittent or permanent.
5. The ability of the system to continue functioning in the event of partial failure is known as fault tolerance.
6. Fault tolerance is the important issue in designing a Distributed file system.
7. There are various types of fault which harms the integrity of a system.
8. For example: If the processor loses the contents of its main memory in the event of a crash it leads to logically complete but physically incomplete operations, making the data inconsistent.
9. Decay of disk storage devices may result in the loss or corruption of data stored by a file system.
10. Decay is when the portion of the device is irretrievable.

FILE PROPERTIES WHICH INFLUENCE THE ABILITY OF DFS TO TOLERATE THE FAULTS ARE:

1) Availability:

1. This refers to the fraction of time for which the file is available for use.
2. This property depends on the location of the clients and location of files.
3. Example: If a network is partitioned due to a communication failure, a link may be available to the clients on some nodes but may not be available to clients on other nodes.
4. Replication is the primary mechanism for improving the availability.

II) Robustness:

1. This refers to power to survive crashes of the storage files and storage decays of the storage medium on which it is stored.
2. Storage devices implemented using redundancy techniques are often used to store robust files.
3. A robust file may not be available until the faulty component has been removed.
4. Robustness is independent of either the location of the file or the location of clients.

III) Recoverability:

1. This refers to the ability to roll back to the previous stable and consistent state when an operation on a file is aborted by the client.
2. In this atomic update techniques such as transaction mechanisms are used.

Types:

Omission Fault:

- When a process or channel fails to do something that it is expected to, it is termed an omission failure.
- There are two categories of omission fault i.e. process and communication omission fault.

Process omission failures:

- A process makes an omission failure when it crashes-it is assumed that a crashed process will make no further progress on its program.
- A crash is considered to be clean if the process either functions correctly or has halted.
- A crash is termed a fail-stop if other processes can detect with certainty that the process has crashed.

Communication omission failures:

- Communication omission failures may occur in the sending process (send-omission failures), the receiving process (receive omission failures) or the channel (channel omission failures).

Arbitrary Failures

- In this type of failure process or communication channel behaves arbitrarily. In this failure, the responding process may return wrong values or it may set wrong values in the data item. Processes may arbitrarily omit intentional processing steps or carry out unintentional processing steps.
- Arbitrary failure also occurs with respect to communication channels. The examples of these failures are : message content may change, repeated delivery of the same messages or non-existent messages may be delivered. These failures occur rarely and can be recognized by communication software.

Timing Fault:

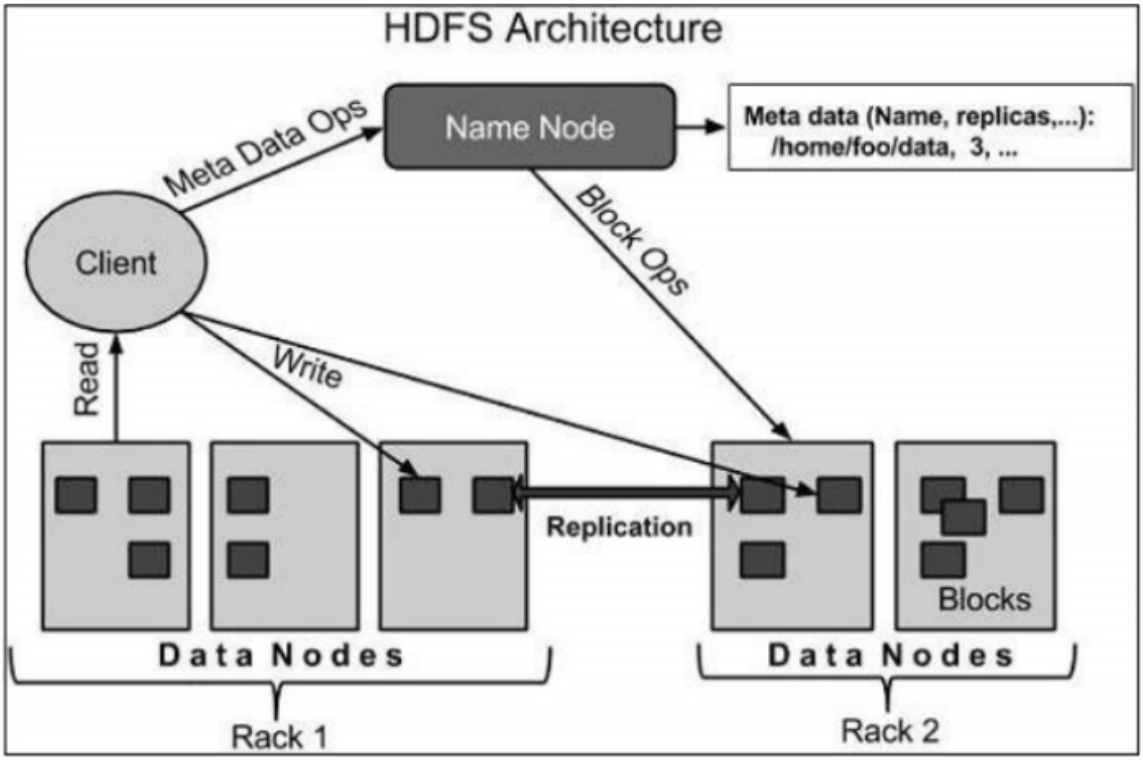
- Synchronous distributed system is one in which each of the following are defined:
  - Upper and lower bounds on the time to execute each step of a process;
  - A bound on the transmission time of each message over a channel;
  - A bound on the drift rate of the local clock of each process.
- If one of these bounds is not satisfied in a synchronous system then a timing failure is said to have occurred.

1. Hadoop File System was developed using distributed file system design that is run on commodity hardware.
2. Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.
3. HDFS holds a very large amount of data and provides easier access.
4. To store such huge data, the files are stored across multiple machines.
5. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure.
6. HDFS also makes applications available for parallel processing.

Features of HDFS:

- It is suitable for distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of the cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

HDFS Architecture



I) Namenode

1. The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware.
2. The system having the namenode acts as the master server and it does the following tasks -
    - Manages the file system namespace.
    - Regulates client's access to files.
    - It also executes file system operations such as renaming, closing, and opening files and directories.

II) Datanode

1. The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode.
2. These nodes manage the data storage of their system.
    - Datanodes perform read-write operations on the file systems, as per client request.
    - They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

III) Block

1. Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes.
2. These file segments are called blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block.
3. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Message-queuing systems:

1. Message-queuing systems or Message-oriented Middleware (MOM) supports persistent asynchronous communication.
2. In this type of communication, sender or receiver of the message need not be active during message transmission. Message is stored in intermediate storage.

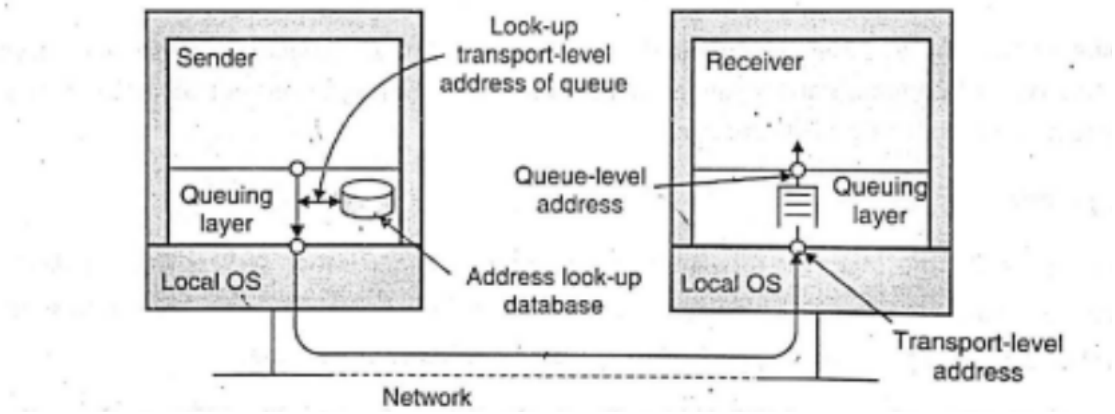
Message-Queuing Model

1. In a message-queuing system, messages are forwarded through many communication servers.
2. Each application has its own private queue to which other applications send the message.
3. In this system, guarantee is given to the sender that its sent message will be delivered to the recipient queue.
4. Messages can be delivered at any time.
5. In this system, the receiver need not be executing when a message arrives in its queue from the sender.
6. Also, sender need not be executed after its message is delivered to the receiver.
7. In exchange of messages between sender and receiver, message is delivered to receiver with following execution modes.
    - Sender and receiver both are running
    - Sender running but receiver passives
    - Sender passive but receiver running
    - Both sender and receiver are passive.

General Architecture of a Message-Queuing System:

1. In the message-queuing system, source queue is present either on the local machine of the sender or on the same LAN Messages can be read from the local queue.
2. The message put in the queue for transmission contains the specification of the destination queue.
3. Message-queuing system provides queues to senders and receivers of the messages.
4. Message-queuing systems keep mapping of queue names to network locations.
5. A queue manager manages queues and interacts with applications which send and receive the messages.

6. Special queue managers work as routers or relays which forward the messages to other queue managers.
7. Relays are more suitable as in several message-queuing systems, dynamically mapping of queue-to-location is not available.
8. Hence, each queue manager should have a copy of queue-to-location mapping. For larger size message-queuing systems, this approach leads to network management problems.
9. To solve these problems, only routers need to be updated after adding or removing the queues.
10. Relays thus provide help in construction of scalable message-queuing systems.



ANDREW FILE SYSTEM (AFS):

1. The Andrew File System (AFS) is a distributed file system.
2. It was developed by Carnegie Mellon University as part of the Andrew Project.
3. Originally named "Vice", AFS is named after Andrew Carnegie and Andrew Mellon.
4. Its primary use is in distributed computing.
5. It uses session semantics.
6. AFS supports information sharing on large scale.

FEATURES:

Andrew File System is designed to:

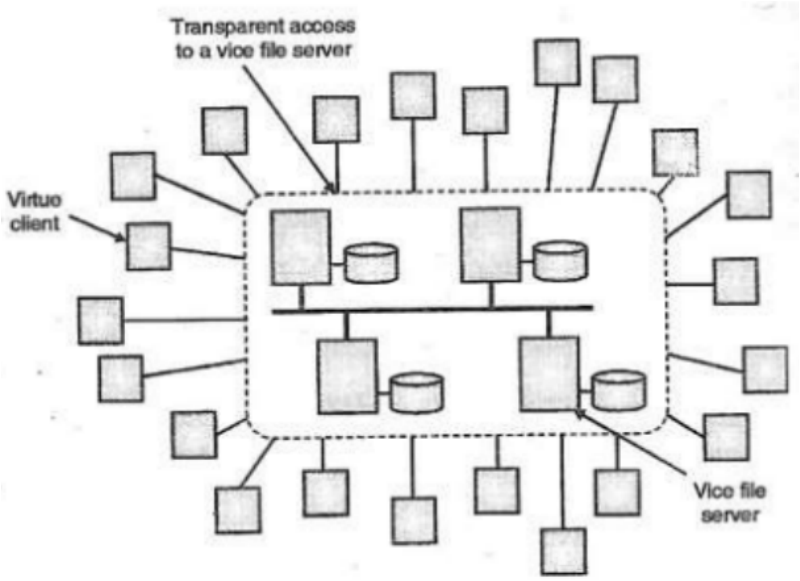
1. Handle terabytes of data.
2. Handle thousands of users.
3. Working in a WAN Environment.

LIMITATION OF AFS:

1. Non-availability of services when servers and network components fail.
2. Scaling problem.

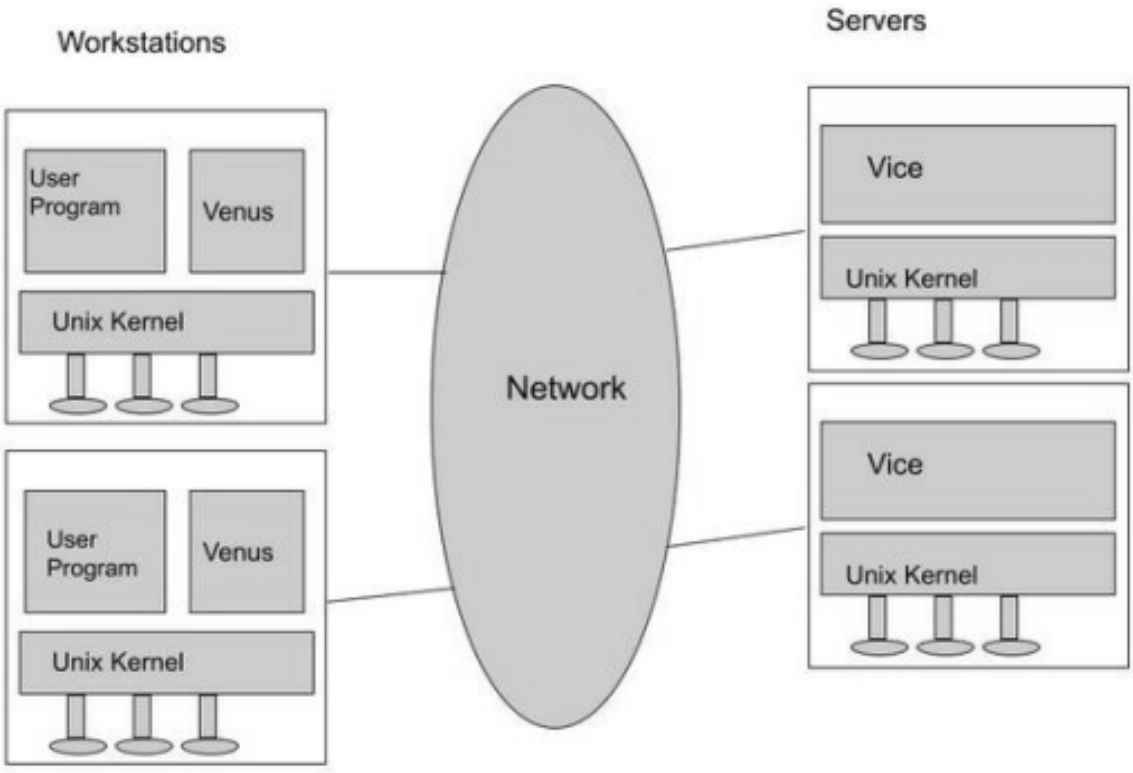
AFS ARCHITECTURE:

1. AFS is divided into two types of nodes:
    a. Vice Node: It is a dedicated file server.
    b. Virtue Node: It is a Client Machines.
2. In VFS, the entire file is copied to the Virtue Node (local machine) from the Vice Node (Server) when opened.
3. When the file is changed - it is copied to the server when closed.



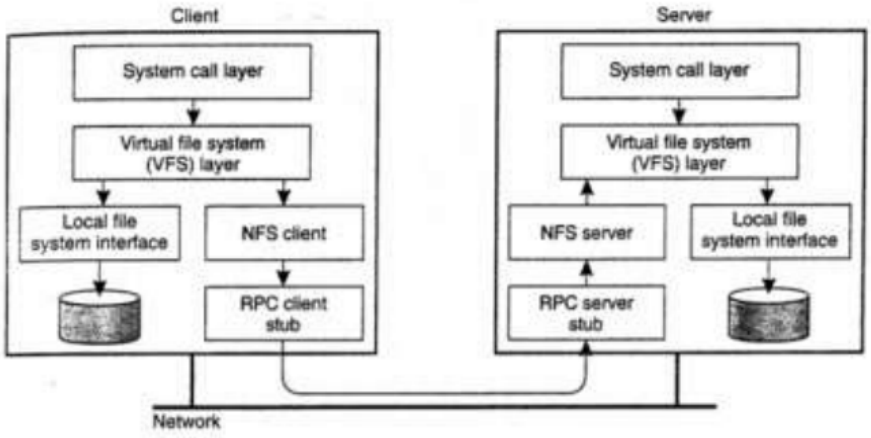
IMPLEMENTATION of AFS:

1. Afs implementation contains 2 softwares components vice and virtue.
2. Vice processes run on server machines and Virtue processes run on client machines both as User level UNIX processes.
3. At client side, local and shared files are available. Shared files are stored on the server and cached by clients in disk cache.
4. At client side, one of the file partitions is used as cache storing files, cached copies of files from shared space. The management of cache is carried out by virtue process.
5. It removes Least recently used files so that new files accessed from the server get space. The size of the disk cache at the client is large.
6. Vice server implements the flat file service. Virtue processes at client side implements hierarchic directory structure that is needed by user programs.



NFS:

1. NFS stands for Network File System.
2. NFS is a platform independent remote file system technology created by Sun Microsystems (Sun) in 1984.
3. It is a client/server application that provides shared file storage for clients across a network.
4. It was designed to simplify the sharing of file systems resources in a network of non-homogeneous machines.
5. It is implemented using the RPC protocol and the files are available through the network via a Virtual File System (VFS), an interface that runs on top of the TCP/IP layer.
6. Allows an application to access files on remote hosts in the same way it access local files.
7. NFS is generally implemented following the layered architecture



NFS SERVERS: COMPUTERS THAT SHARE FILES

1. During the late 1980s and early 1990s, it was customary to set up an NFS Server on a powerful workstation with thousands of local drives and frequently without a graphical display.
2. "Thin," diskless workstations would then mount the NFS Servers' distant file systems and utilize them transparently as if they were local files.

NFS Simplifies management:

1. NFS offers a single copy of a directory such as /usr/local that is shared by all systems on the network, rather than duplicating it on each system.
2. Simplify backup operations - With NFS, a sysadm just has to backup the server's discs, rather than the local contents of each workstation (for example, /home).

NFS Clients: Computers that access shared files

1. On the client side, NFS uses a combination of kernel support and user-space daemons.
2. Users can share data by mounting the same remote file system on several clients.
3. Mounting can be done during the boot process.
4. An NFS client:
    a. Mounts a remote file system onto the client's local file system namespace.
    b. It Provides an interface so that access to the files in the remote file system is done as if they were local files.

GOALS OF NFS DESIGN:

1. Compatibility:
    a. NFS should provide the same semantics as a local unix file system.
    b. Programs should not need or be able to tell whether a file is remote or local.
2. Easy deployable:
    a. Implementation should be easily incorporated into existing systems remote files should be made available for local programs without these having to be modified or relinked.
3. Machine and OS independence:
    a. NFS Clients should run in non-unix platforms Simple protocols that could be easily implemented in other platforms.
4. Efficiently:
    a. NFS should be good enough to satisfy users, but did not have to be as fast as local FS.
    b. Clients and Servers should be able to easily recover from machine crashes and network problems.

NFS PROTOCOL:

1. Uses Remote Procedure Call (RPC) mechanisms
2. RPCs are synchronous.
3. NFS uses a stateless protocol. This simplifies crash recovery. All that is needed to resubmit the last request.
4. In this way, the client cannot differentiate between a server that crashed and recovered and one that is just slow.

LOAD BALANCING APPROACH:

1. Load Balancing is used in Process Management.
2. Scheduling Algorithms that use a Load Balancing Approach are called Load Balancing Algorithms.
3. The main goal of load balancing algorithms is to balance the workload on all the nodes of the system.
4. Load balancing aims to:
   a. Optimize resource use.
   b. Maximize throughput
   c. Minimize response time.
   d. Avoid overload of any single resource.

DESIGNING ISSUES:

1) Load Estimation Policy:

1. The first issue in a load balancing algorithm is to decide which method to use to estimate the workload of a particular node.
2. Estimation of the workload of a particular node is a difficult problem for which no completely satisfactory solution exists.
3. A nodes workload can be estimated based on some measurable parameters below:
   a. Total number of processes on the node.
   b. Resource demands of these processes.
   c. Instruction mixes of these processes.
   d. Architecture and speed of the node's processor.

II) Process Transfer Policy:

1. The idea of using this policy is to transfer processes from heavily loaded nodes to lightly loaded nodes.
2. Most of the algorithms use the threshold policy to decide whether the node is lightly loaded or heavily loaded.
3. Threshold value is a limiting value of the workload of node which can be determined by:
   a. Static Policy: Each node has a predefined threshold value.
   b.Dynamic Policy: The threshold value for a node is dynamically decided.
4. Below the threshold value, node accepts processes to execute.
5. Above threshold value node tries to transfer processes to a lightly loaded node.
6. Single threshold policy may lead to unstable algorithms.

7. To reduce instability a double threshold policy has been proposed which is also known as high low policy.

III) Location Policy:

1. When a transfer policy decides that a process is to be transferred from one node to any other lightly loaded node, the challenge is to find the destination node.
2. Location policy determines this destination node.
3. It includes following:
    a. Threshold: This policy chooses a random node and examines whether it is capable of receiving the process before transferring it. If a node refuses the transfer, a new node is chosen at random. This process is repeated until the probe limit is reached.
    b. Shortest method: In this 'm' distinct nodes are chosen at random and each is polled to determine its load with minimum value.
    c. Bidding method: In this method, nodes contain managers to send processes and contractors to receive processes.
    d. Pairing: It is used to reduce the variance of load only between nodes of the system.

IV) State Information Exchange Policy:

1. It is used for frequent exchange of state information within the nodes.

2. It includes:

a. Periodic Broadcast: Each node broadcasts its state information after the elapse of T units of time.
b. Broadcast when state changes: Each node broadcasts its state information only when a process arrives or departures.
c. On demand exchanges: Each node broadcasts its state information request message when its state switches from normal to either under load or overload.
d. Exchanges by polling: The partner node is searched by polling the other nodes one by one until the poll limit is reached.

V) Priority Assignment Policy:

1. The priority of the execution of local and remote processes at a particular node should be planned.

2. It includes:
    a. Selfish priority assignment rule: In this local process are given higher priority than remote processes.
    b. Altruistic priority assignment rule: Remote processes are given higher priority than local processes.
    c. Intermediate priority assignment rule: Priority in this rule is decided depending upon the number of local and remote processes on a node.

VI) Migration Limiting Policy:

1. This policy determines the total number of times a process can migrate from one node to another.
2. It includes:
    a. Uncontrolled Policy: The remote process is treated as a local process. Hence, it can be migrated any number of times.
    b. Controlled Policy: Migration count is used for remote processes.

MUTUAL EXCLUSION:

1. There may be some resources like printers that must be restricted to single process at a time.
2. Therefore, exclusive accesses to such shared resources by a process have to be ensured.
3. This exclusiveness of access is called mutual exclusion.
4. The algorithm implementing mutual exclusion must satisfy two conditions:
   a. Mutual Exclusion: Only one process at a time can access a particular resource.
   b. No starvation: Every request must be eventually granted.
5. It includes a centralized and distributed approach.

Centralized algorithms for Mutual Exclusion in Distributed Systems.:

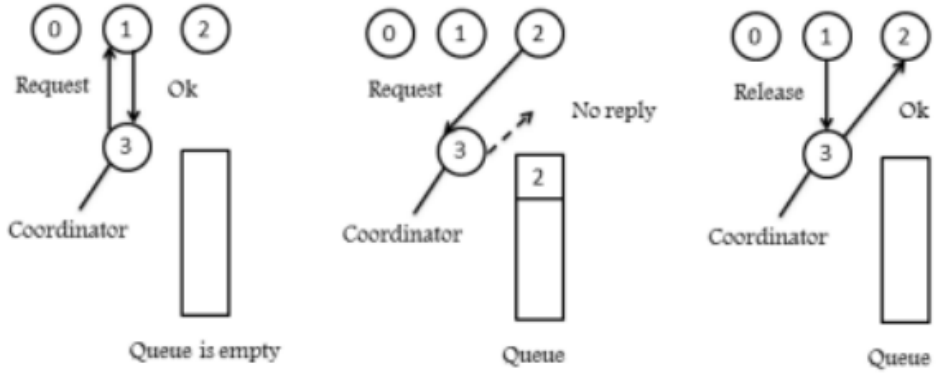


Figure (a) Figure (b) Figure (c)

1. In a centralized algorithm one process is elected as the coordinator.
2. Coordinator may be the machine with the highest network address.
3. Whenever a process wants to enter a critical region, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission.
4. If no other process is currently in that critical region, the coordinator sends back a reply granting permission as shown in figure A.
5. When the reply arrives, the requesting process enters the critical region.
6. Suppose another process 2 shown in figure B, that asks for permission to enter the same critical region.
7. Now the coordinator knows that a different process is already in the critical region, so it cannot grant permission.
8. The coordinator just refrains from replying, thus blocking process 2, which is waiting for a reply or it could send a reply saying 'permission denied'.

9. When process 1 exits the critical region, it sends a message to the coordinator releasing its exclusive access shown in figure C.
10. The coordinator takes the first item off the queue of deferred requests and sends that process a grant Message.
11. If the process is still blocked it unlocks and enters the critical region.
12. If an explicit message has already been sent denying permission, the process will have to poll for incoming traffic or block later.
13. When it sees the grant, it can enter the critical region.

Advantages:

1. No starvation problem.
2. Easy to implement & use for general resources allocation.

Disadvantages:

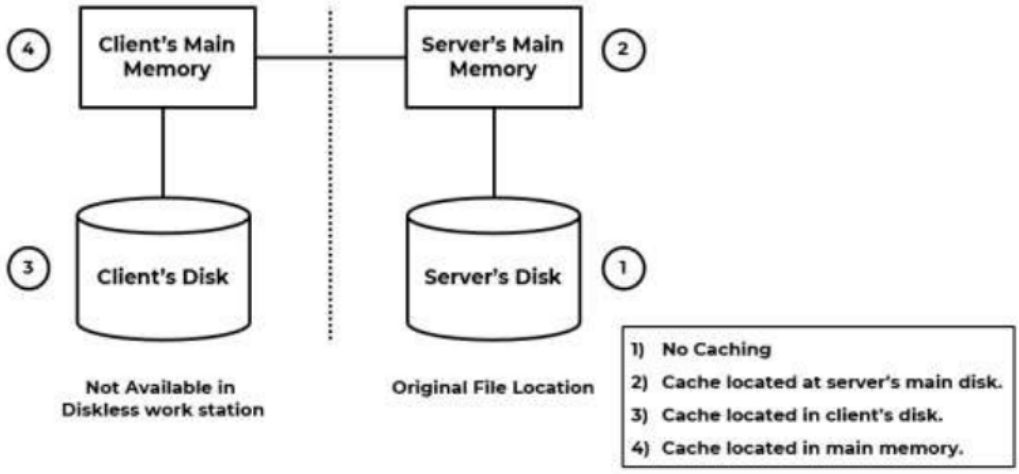
1. If the coordinator crashes, the entire system will fail. In a large system a single coordinator can become a performance bottleneck.

FILE CACHING:

1. In order to improve the file I/O performance in centralized time sharing systems, file caching has been implemented and executed in numerous distributed file systems.
2. Main goal of the file caching scheme is to retain recently accessed data in main memory.
3. So that repeated access to the same information can be efficiently handled.
4. In implementing a file-caching scheme, one has to make several key decisions like granularity of cached data.
5. Three design issues are involved in file caching which are:

I) Cache location:

1. Cache location refers to the place where the cached data is stored.
2. There exists three possible cache locations in servers DFS i.e. server's main memory, client's disk and client's main memory as shown in figure.



Server's main memory:

1. In this the file must be first transferred to the server's main memory and then across the network
2. Hence the disk access cost is reduced.

Client's disks: The cache located in the client's disk eliminates network access cost but requires disk access cost.

Client's main memory: The cache located in a client's main memory eliminated both network access cost
and disk access cost.

II) Modification propagation:

1. When the cache is located on clients' node; a file's data may simultaneously be cached on multiple nodes.
2. 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.
3. A variety of approaches is used to handle these issues:
    a. When to propagate modifications made to a cached data to corresponding file server?
    b. How to verify the validity of cached data?

4. It consists of the following techniques:

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

Delayed-write scheme:

1. In this scheme when a cache entry is modified the new value is written only to the cache.
2. The client just makes a note of the cache entry that has been updated.
3. All the cache entries are collected together and sent to the server later on.
4. Its approaches are as follows:
    a. Write on ejection from cache:
        - Modified data in the cache is sent to the server when the cache replacement policy has decided to eject it from the client's cache.
    b. Periodic write:
        - Here cache is scanned at regular intervals.
        - And cached data is modified right from the last scan and is sent to the server.
    c. Write on close:
        - Modification made to a cached data by a client is sent to the server when the corresponding file is closed by the client.

III) Cache Validation Schemes:

1. A file data resides in the cache of multiple nodes.
2. It becomes necessary to check if the data cached at the client node is consistent with the master node.
3. If it is not then the cached data must be invalidated and the updated version must be fetched from the server.
4. Validation is done in two ways:

Client initiated approach:

1. In this method the client contacts the server to check if locally cached data is consistent with the master copy.
2. File sharing semantics depends on the frequency of the validity check and can be use below approaches:
   a. Check before every access: The main purpose of caching is defeated in this process because the server has to be contacted on every access.
   b. Periodic check: For this method a check is initialized after a fixed interval of time.
   c. Check on file open: A client cache entry is validated only when the client opens a corresponding file for use.

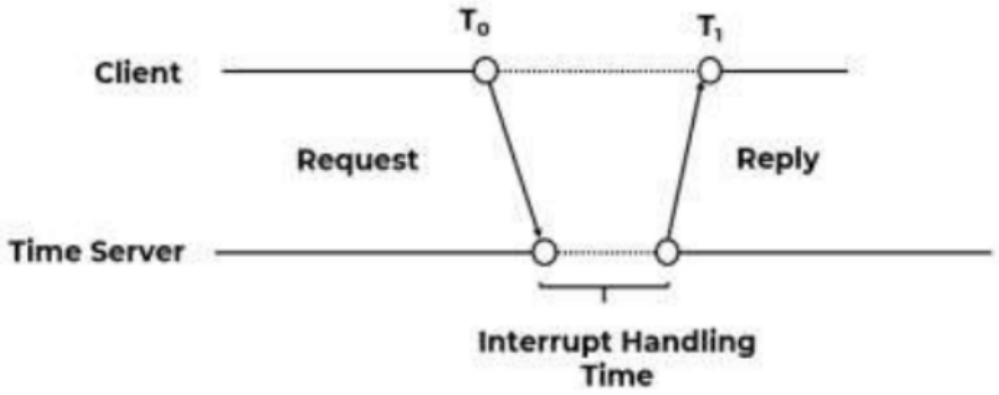
Server initiated approach:

1. In this method a client informs the file server when it opens a file indicating whether the file is being opened for reading, writing or both.
2. The file server keeps a record of which client has which file opened and in what mode.
3. Thus the server keeps a track of the file usage modes being used by different clients and reacts in case of inconsistency.
4. Inconsistency occurs when more clients try to open a file in conflicting modes.
5. Example: If a file is open for reading, other clients may be able to open it to read a file without any problem, but not for writing.
6. Similarly, a new client must not be allowed to open a file which is already open for writing.
7. When a client closes a file it sends intimation to the server along with any modification made to the file.
8. On receiving intimation the server updates its record of which client has which file open in what mode.

Open Distributed System:

1. Openness is the important goal of the distributed system.
2. An open distributed system provides services as per standard rules that tell the syntax and semantics of those services.
3. In distributed systems, services are usually specified through interfaces, which are expressed in an Interface Definition Language (IDL).
4. The definitions of the interfaces written in an IDL almost capture only the syntax of these services.
5. The semantics of interfaces means specification of what these interfaces can perform.
6. Processes make use of interfaces to communicate with each other.
7. There can be different implementations of these interfaces leading to different distributed systems that function in the same manner.
8. Appropriate specifications are complete and neutral.
9. But, many interface definitions do not obey completeness.
10. So that it is required for a developer to put in implementation-specific details. If specifications do not impose what an implementation should look like they should be neutral.
11. Completeness and neutrality are significant for interoperability and portability.
12. An open distributed system should allow configuring the system out of different components from different developers. It means, an open distributed system should also be extensible.
13. Monolithic approach should be avoided for building the system.
14. There should be separation between policy and mechanism.
15. For example, apart from storing the documents by browser, users should be able to make a decision which documents are stored and for how much duration.
16. User should be able to implement his own policy as a component that can be plugged into the browser.
17. Open distributed systems provide a uniform communication mechanism and it is also based on published interfaces in order to access the common resources.

Cristian’s algorithm for physical clock synchronization



1. Cristian's Algorithm is a clock synchronization algorithm used to synchronize time with a time server by client processes.
2. This algorithm works well with low-latency networks where Round Trip Time is short as compared to accuracy.
3. Here Round Trip Time refers to the time duration between start of a Request and end of corresponding Response.
4. In this method, each node periodically sends a message to the server.
5. When the time server receives the message it responds with a message T, where T is the current time of the server node.
6. Assume the clock time of client be To when it sends the message and T, when it receives the message from the server.
7. To and Ti are measured using same clock so best estimate of time for propagation is $(T_1\text{-To})/2$.
8. When the reply is received at the client's node, its clock is readjusted to T+ (Ti-To)/2.
9. Advantage: It assumes that no additional information is available.
10. Disadvantage: It restricts the number of measurements for estimating the value.

STREAM ORIENTED COMMUNICATION:

1. RPC and Message Oriented Communication are based on the exchange of discrete messages.
2. Timing might affect performance, but not correctness.
3. In Stream Oriented Communication the message content must be delivered at a certain rate, as well as correctly.

Example: Music or video.

4. Stream Oriented Communication supports continuous media communication.

TRANSMISSION MODES IN STREAM ORIENTED COMMUNICATION:

1. Asynchronous Mode: No restrictions with respect to when data is to be delivered
2. Synchronous Mode: Define a maximum end-to-end delay for individual data packets.

3. Isochronous Mode: Define a maximum end-to-end delay and maximum delay variance.

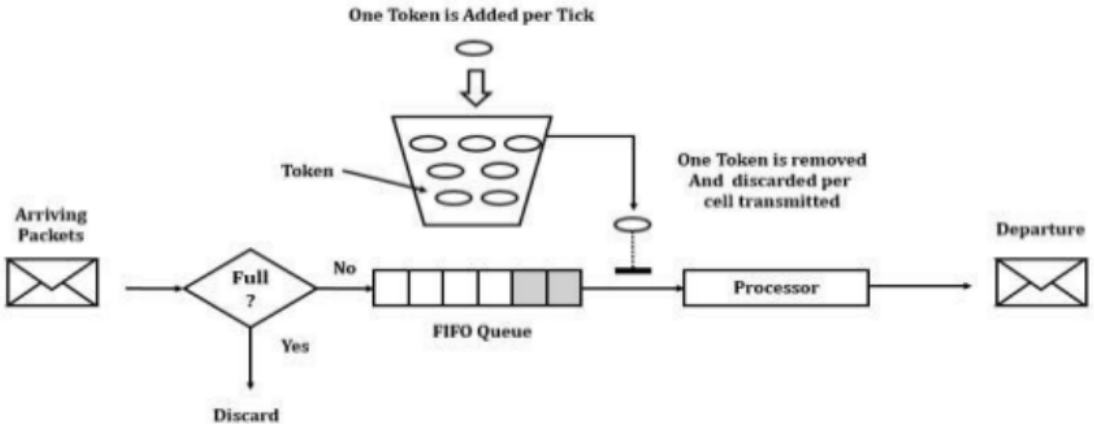
STREAM:

1. A data stream is a connection-oriented communication facility.
2. It supports isochronous data transmission.
3. Some common stream characteristics:
    a. Streams are unidirectional.
    b. There is generally a single source.
4. Stream types:
    a. Simple: It consists of a single flow of data
    (e.g., audio or video)
    b. Complex: It consists of multiple data flows
    (e.g., stereo audio or combination audio/video)

EXAMPLE:

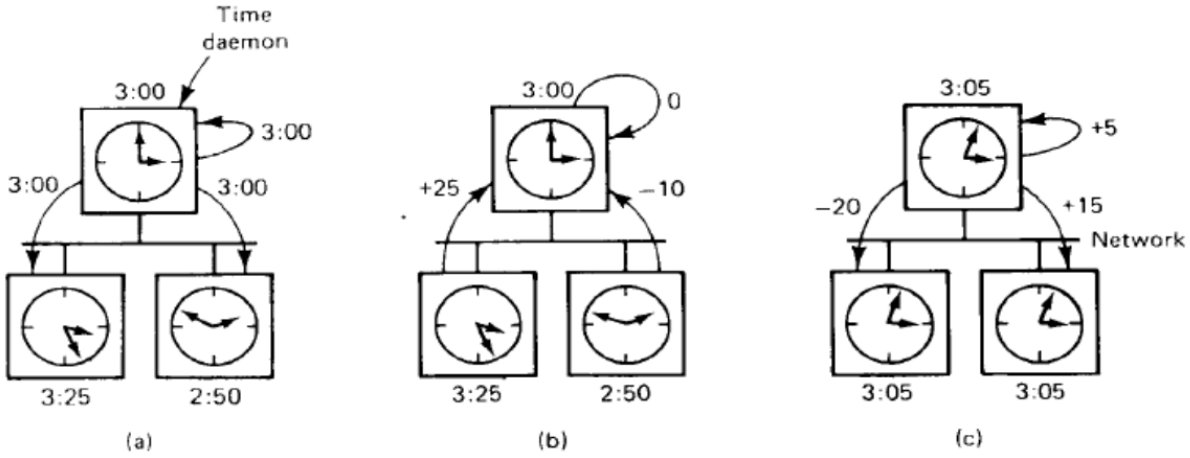
Token Bucket Algorithm:

1. Figure 2.9 shows the implementation of Token Bucket.
2. The token bucket can be easily implemented with a counter.
3. The token is initialized to zero.
4. Each time a token is added, the counter is incremented by 1.
5. Each time a unit of data is dispatched, the counter is decremented by 1.
6. If the counter contains zero, the host cannot send any data.



Berkeley Algorithm:

1. The Berkeley algorithm is a method of clock synchronization in distributed computing which assumes no machine has an accurate time source.
2. It was developed by Gusella and Zatti at the University of California, Berkeley in 1989
3. This is an active time server approach.
4. In this approach, the time server periodically sends a message to all the computers in the group of computers.
5. When this message is received each computer sends back its own clock value to the time server.
6. The time server has a prior knowledge of the approximate time required for propagation of a message which is used to readjust the clock values.
7. It then takes the average of clock values of all the computers.
8. The calculated average is the current time to which all clocks should be readjusted.
9. The time server readjusts its own clock to this value and instead of sending the current time to other computers it sends the amount of time each computer needs for readjustment.
10. This can be a positive or negative value.



(a) The time daemon asks all the other machines for their clock values. (b) The machines answer. (c) The time daemon tells everyone how to adjust their clock.

Maekawa's Algorithm:

1. Maekawa's Algorithm is a quorum based approach to ensure mutual exclusion in distributed systems.
2. In a quorum based approach, a site does not request permission from every other site but from a subset of sites which is called quorum.
3. In this algorithm three types of messages (REQUEST, REPLY and RELEASE) are used.
    a. 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.
    b. A site sends a REPLY message to a requesting site to give its permission to enter the critical section.
    c. 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:

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

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

2. $\forall i : 1 \leq i \leq N :: Si \in Ri$
3. $\forall i : 1 \leq i \leq N :: |Ri| = K$
4. Any site Si is contained in exactly K sets.
5. $N = K(K - 1) + 1$ and $|Ri| = \sqrt{N}$

Algorithm:

1. 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.

2. 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

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

Message Complexity:

- Maekawa's Algorithm requires invocation of 3√N messages per critical section execution as the size of a request set is √N. These 3√N messages involve.
- √N request messages
- √N reply messages
- √N release messages

Drawbacks of Maekawa's Algorithm:

- This algorithm is deadlock prone because a site is exclusively locked by other sites and requests are not prioritized by their timestamp.

Performance:

- Synchronization delay is equal to twice the message propagation delay time
- It requires 3√n messages per critical section execution.

GROUP COMMUNICATION:

1. A group is the collection of users sharing some common interest.
2. In group communication, Messages sent to a group of processes will deliver to all members of the group.
3. There are three types of group communication i.e. one to many, many to one and many to many communication.

ONE-TO-MANY COMMUNICATIONS:

1. There exist single sender and multiple receivers.
2. It is also called multicast communication.
3. There exists a special case of multicast communication as broadcast communication in which a message is sent to all processors connected to the network.
4. One to many communication includes:

a. Group Management: There are two types of groups: open group and closed group.
   - A closed group is the one in which only the members of the group can send a message to the group as a whole, although it may send a message to an individual.
   - An Open Group is the one in which any processes in the system can send a message to the group as a whole.

b. Group Addressing:
   - It is possible to create a special network address to which multiple machines can listen.

c. Buffered and Unbuffered Multicast:
   - For Buffered Multicast the message is buffered for receiving processes so each process of the multicast group will eventually receive the message.
   - For Unbuffered Multiplication, the message may be lost if the receiving process is not in the state ready to receive it.

d. Send to all and Bulletin Semantics:
   - Send-to-all semantics: A copy of message is sent to each process for multicast group and the message is buffered until it is accepted by the process.
   - Bulletin-board Semantics: A Message to be multicast is addressed to a channel instead of being sent to every individual process of the multicast group.

e. Flexible Reliability in Multicast Communication:

- 0 reliable: No response is expected by the sender.
- 1 reliable: The sender expects at least one acknowledgement.
- M out of N reliable: The sender decides as to how many acknowledgements it expects out of N.

MANY - TO - ONE COMMUNICATION:

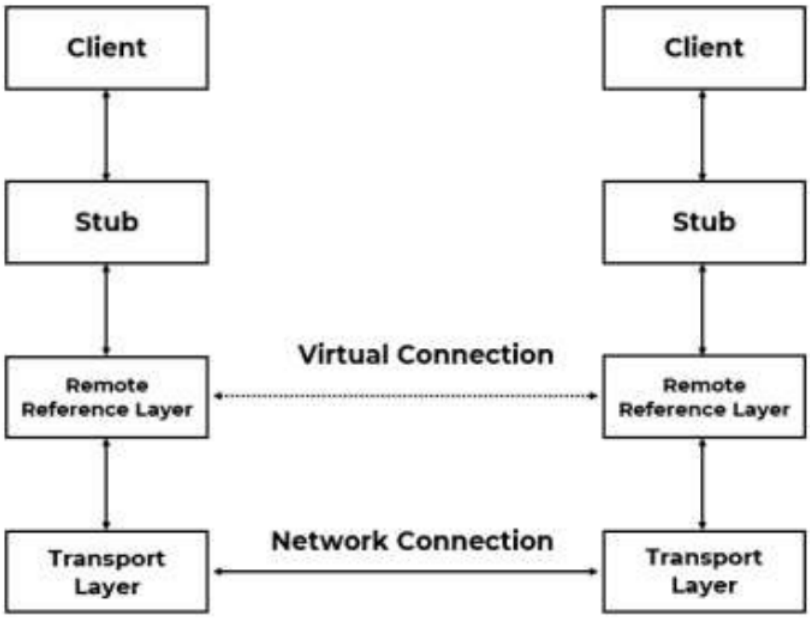
1. There is only one receiver at any given time.
2. The receiver can be selective or nonselective.
3. Selective Receiver: Specifies the unique sender. A message exchange takes place only if that sender sends a message.
4. Nonselective Receiver: The receiver is ready to accept a message from any sender who is present in the system.

MANY - TO - MANY COMMUNICATIONS:

1. In this scheme multiple senders send messages to multiple receivers.
2. An important issue with many to many communications is ordered message delivery.
3. It includes:
   a. No ordering:
      In many-to-many communication, a message sent from one place to another may arrive at the destination of one sender before that of the other. The commonly used semantics for ordered delivery of Multicast messages is absolute ordering, consistent ordering and causal ordering.
   b. Absolute ordering:
      This semantics ensures that all messages are delivered to all receiver processes in the exact order in which they were sent.
   c. Consistent ordering:
      This ensures that all messages are delivered to all receiver processes in the same order as they reach the senders.
   d. Casual Ordering:
      This semantics ensures that if the event of the sending one message is causally related to the event of sending another message, the two messages are delivered to all receivers in the correct order.

Remote Method Invocation:

1. In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).
2. Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
3. The client program requests the remote objects on the server and tries to invoke its methods.
4. Architecture of an RMI.



It includes:

a. Transport Layer: This layer connects the client and the server. It manages the existing connection and also sets up new connections.

b. Stub: A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.

c. Skeleton: This is the object which resides on the server side. Stub communicates with this skeleton to pass requests to the remote object.

d. Remote Reference Layer: It is the layer which manages the references made by the client to the remote object.

WORKING OF RMI:

1. When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the Remote Reference Layer (RRL).
2. When the client-side RRL receives the request, it invokes a method of the object. It passes the request to the RRL on the server side.
3. 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.
4. The result is passed all the way back to the client.

MARSHALLING AND UNMARSHALLING

1. Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network.
2. These parameters may be of primitive type or objects.
3. In case of primitive type, the parameters are put together and a header is attached to it.
4. In case the parameters are objects, then they are serialized.
5. This process is known as marshalling.
6. At the server side, the packed parameters are unbundled and then the required method is invoked.
7. This process is known as unmarshalling.

GOALS OF RMI

1. To minimize the complexity of the application.
2. To preserve type safety.
3. Distributed garbage collection.
4. Minimize the difference between working with local and remote objects.

1. In replication, a replica is created at the server, whereas a cached copy is associated with clients.
2. A replica is more persistent as compared to a cached copy.
3. Replica is widely known, secure, accurate, available and complete.
4. Cached copy existence depends on locality in access patterns.
5. Existence of replicas depends on availability and performance.
6. Cache copy is dependent on replicas.
7. It needs to be periodically verified with replicas then it becomes useful.

Advantages of Replication

1. Increased Availability: Replication ensures the system's availability. Even if there is a failure, the system remains functioning and accessible to users. The vital information can be duplicated over many servers. If the primary copy fails, another server can still access it.
2. Increased Reliability: Recovery in the event of a failure is feasible since various copies of the files are available on separate servers. Data that has been lost permanently due to a catastrophic failure is easily recoverable from other mirrored copies. As a result, replication ensures consistency.
3. Improved Response Time: Data can be retrieved locally or from a node whose access time is smaller than the access time of the primary copy.
4. Reduced Network Traffic: Client access requests are handled locally if a replica of the file is present on the file server on the client computer. As a result, network traffic is minimized.
5. Improved System Throughput : Because various clients' requests are handled by separate servers simultaneously, throughput is enhanced.
6. Better Scalability: If a file is duplicated over many file servers, all client requests for that file will not be sent to a single file server. As a result, load is divided and various client requests are handled by separate servers. It boosts scalability.
7. Autonomous Operation: All files required by clients for a limited time period can be duplicated on a client-side file server. This guarantees that client nodes can operate autonomously for a limited time.

PHYSICAL MODEL:

1. A physical model is a representation of the underlying hardware elements of a distributed system.
2. It captures the hardware composition of a system in terms of computers and other devices and their interconnecting network.
3. Physical models can be categorized into three generations of distributed systems i.e. early distributed systems, Internet-scale distributed systems and Contemporary distributed systems.

ARCHITECTURAL MODEL:

1. Architectural model define the way in which the components of the system interact with each other and the way in which they are mapped onto an underlying network of computers.
2. It simplifies and abstracts the functionality of the individual components of a distributed system.
3. It describes responsibilities distributed between system components and how these components are placed.
4. Examples: Client Server Model and Peer to Peer Model.

FUNDAMENTAL MODEL:

1. The purpose of a fundamental model is to make explicit all relevant assumptions about the system we are modeling.
2. It includes interaction, fault and security models.

1) Interaction Model:

- Interaction model deal with the performance and are used for handling time in distributed systems i.e. for process execution, message delivery, clock drifts etc.
- The two variants of the interaction model are synchronous and asynchronous distributed systems.

II) Fault Model:

- Failures can occur both in processes and communication channels.
- The reason can be both software and hardware faults.
- The failure model defines how the failure occurs in the system and what its effects are.
- Fault models are needed in order to build systems with predictable behavior in case of faults (systems which are fault tolerant).

- There are three categories of fault - omission fault, arbitrary fault and timing fault.

III) Security Model:

- Security model is based on establishing the trustworthiness and role of each component such as trusted users, trusted servers, trusted administrators and clients.

Security model is used for:

- Protecting access to objects - Access rights and authentication of clients
- Protecting processes and interactions.
- Protecting communication channels.

NEED OF CLIENT CENTRIC CONSISTENCY MODELS:

1. It is one type of consistency model in a distributed system.
2. System wide consistency is hard, so usually a client centric consistency model is more preferable.
3. It is easier to deal with inconsistencies.
4. Using a client centric consistency model many inconsistencies can be hidden in a cheap way.
5. It aims at providing a consistent view on a database.

CLIENT CENTRIC CONSISTENCY MODELS:

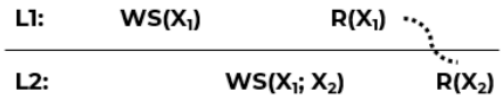
I) Eventual Consistency:

1. An eventual consistency is a weak consistency model.
2. It lacks simultaneous updates.
3. It defines that if updates do not occur for a long period of time, all replicas will gradually become consistent
4. Eventual Consistency is a lot inexpensive to implement.
5. Requirements for Eventual Consistency are:
   a. Few read/write conflicts.
   b. No write/write conflicts.
   c. Clients can accept temporary inconsistency.
6. Example: WWW.

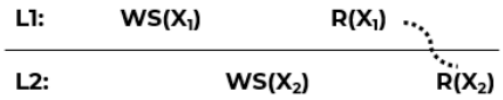
II) Monotonic Reads:

1. A data store is said to offer monotonic read consistency if the following condition holds: "If process P reads the value of data item x, any successive read operation on x by that process will always return the same value or a more recent one."
2. Consider a Distributed e-mail database.
3. It has distributed and replicated user-mailboxes.
4. Emails can be inserted at any location.
5. However, updates are propagated in a lazy (i.e. on demand) fashion.
6. Suppose the end user reads his mail in Mumbai. Assume that only reading mail does not affect the mailbox.
7. Later when the end user moves to Pune and opens his mail box again, monotonic read consistency guarantees that the messages that were in the mailbox in Mumbai will also be in the mailbox when it is opened in Pune.

8. Figure 5.1 shows the read operations performed by a single process 'P' at two different local copies of the same data store.
9. Figure 5.1 (a) Demonstrate a monotonic-read consistent data store.
10. Figure 5.2 (b) Demonstrate a data store that does not provide monotonic reads.



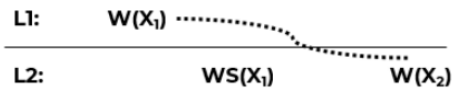
(a)



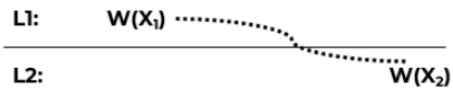
(b)

III) Monotonic Writes:

1. A data store is said to offer monotonic write consistency if the following condition holds: "A write operation by process P on data item x is completed before any successive write operation on x by the same process P can take place".
2. Consider a software library example.
3. In several cases, updating a library is accomplished by replacing one or more functions.
4. With monotonic write consistency, assurance is given that if an update is performed on a copy of the library, all previous or earlier updates will be accomplished primarily.
5. Figure 5.2 shows the write operations performed by a single process 'P' at two different local copies of the same data store.
6. Figure 5.2 (a) Demonstrate a monotonic-write consistent data store.
7. Figure 5.2 (b) Demonstrate a data store that does not provide monotonic-write consistency.



(a)

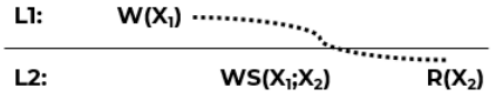


(b)

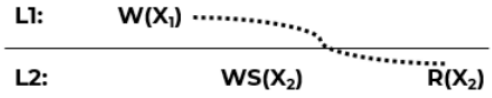
IV) Read Your Writes:

1. A data store is said to offer read your write consistency if the following condition holds.
   "The effect of a write operation by a process P on a data item x at a location L will always be seen by a successive read operation by the same process."

2. Example: Updating a Web page and guaranteeing that the Web browser shows the newest version instead of its cached copy.
3. Figure 5.3 (a) Demonstrate a data store that delivers read your writes consistency.
4. Figure 5.3 (b) Demonstrate a data store that does not delivers read your writes consistency.



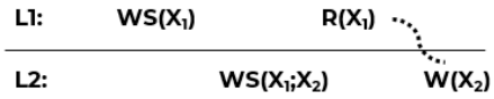
(a)



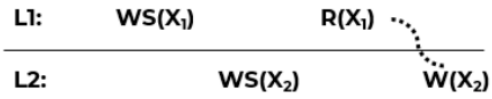
(b)

V Writes Follow Reads:

1. A data store is said to offer write follow read consistency if the following condition holds "A write operation by a process P on a data item x following a previous read by the same process, is guaranteed to take place on the same or even a more recent value of x, than the one having been read before."
2. 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.
3. Figure 5.4 (a) Demonstrates a data store that delivers writes follow reads consistency.
4. Figure 5.4 (b) Demonstrates a data store that does not deliver writes follow reads consistency.



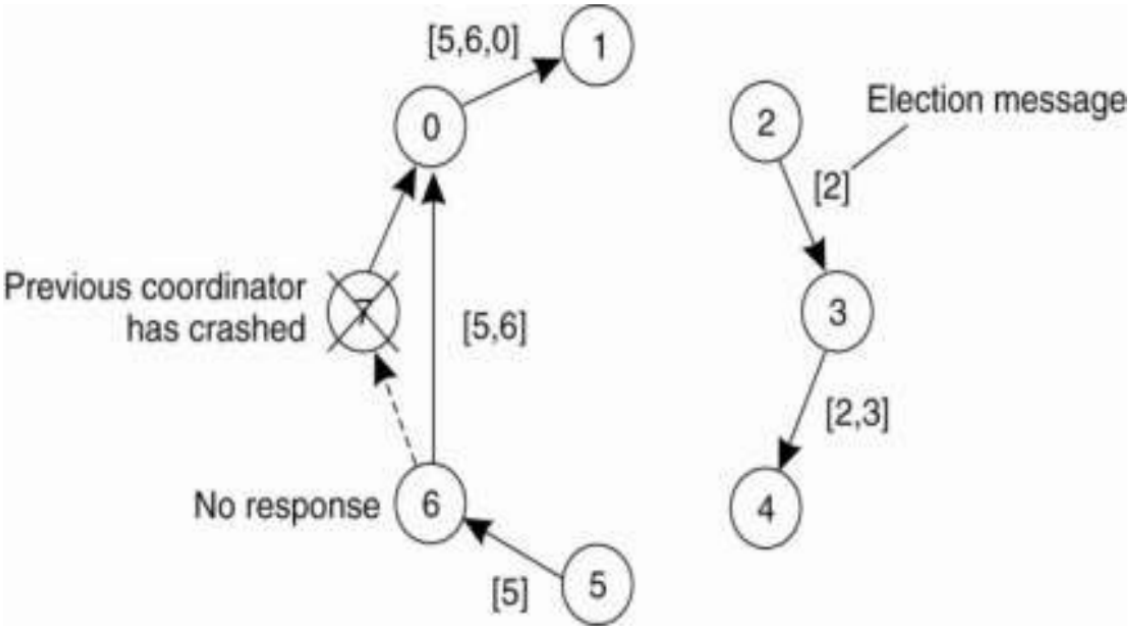
(a)



(b)

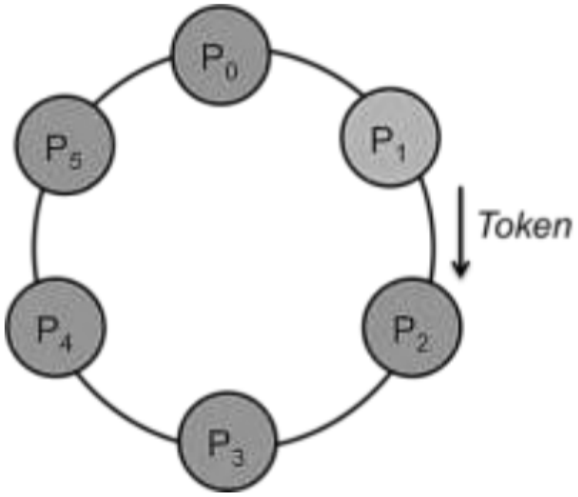
RING ALGORITHM:

1. The ring algorithm assumes that the processes are arranged in a logical ring and each process knows the order of the ring of processes.
2. Processes are able to "skip" faulty systems: Instead of sending to process j, send toj + 1.
3. Faulty systems are those that don't respond in a fixed amount of time.
4. Process 'P' thinks the coordinator has crashed; builds an ELECTION message which contains its own ID number.
5. Sends to first live successor
6. Each process adds its own number and forwards to next.
7. OK to have two elections at once.
8. When the message returns to p, it sees its own process ID in the list and knows that the circuit is complete.
9. P circulates a COORDINATOR message with the new high number.
10. Here, both 2 and 5 elect 6:
    [5,6,0,1,2,3,4]
    [2,3,4,5,6,0,1]



TOKEN RING ALGORITHM:

1. Token ring approach is used to achieve mutual exclusion in a distributed system.
2. Here we have a bus network (for e.g. Ethernet).
3. A logical ring is constructed in which each process is assigned a position in the ring.
4. The ring positions may be allocated in numerical order of network addresses or some other means.
5. It does not matter what the ordering is.
6. All that matters is that each process knows who is next in line after itself.
7. When the ring is initialized, process P1 (say) is given a token.
8. The token circulates around the ring.
9. It is passed from process k to process k+1 in point-to-point messages.
10. When a process acquires the token from its neighbor, it checks to see if it is attempting to enter a Critical Section (CS).
11. If so, the process enters the CS, does all the work it needs to, and leaves the section.
12. After it has exited, it passes the token along the ring. It is not permitted to enter a second CS using the same token.
13. If a process is handed the token by its neighbor and is not interested in entering a CS, it just passes it along.
14. As a consequence, when no processes want to enter any CS, the token just circulates at high speed around the ring.



Advantage:

No starvation problem.

Disadvantage:

Detecting the lost token and regeneration is difficult.

LAMPORT'S DISTRIBUTED MUTUAL ALGORITHM:

1. Lamport's Distributed Mutual Exclusion Algorithm is a contention-based algorithm for mutual exclusion on a distributed system.
2. It is a non-token based computer algorithm.
3. Lamport was the first to give a distributed mutual exclusion algorithm as an illustration of his clock synchronization scheme
4. It is intended to improve the safety in the usage of shared resources.
5. For synchronization of logical clocks, Lamport established a relation termed the happen-before relation.
6. Let Ri be the request set of site Si, i.e. the set of sites from which Si needs permission when it wants to enter a critical section (CS).
7. In Lamport's algorithm, Vi: 1 ≤ i ≤N: Ri= (S1, S2.,..,SN).
8. Every site Si keeps a queue, request_queue i, which contains mutual exclusion requests ordered by their timestamps.
9. This algorithm requires messages to be delivered in the FIFO order between every pair of sites.
10. Each process maintains a request queue.
11. Queue contains mutual exclusion requests.

Requesting the critical section:

1. Process Pi sends requests (i, Ti) to all nodes.
2. It then places requests on its own queue.
3. When a process Pj receives a request, it returns a timestamped ack.

Entering the critical section:

1. Pi received a message (ack or release) from every other process with a timestamp larger than Ti.
2. Pi's request has the earliest timestamp in its queue

Releasing the critical section:

1. It removes the request from its own queue.
2. It then sends a timestamped release message.
3. When a process receives a release message:

a. It removes the request for that process from its queue.
b. This may cause its own entry to have the earliest timestamp in the queue, enabling it to access the critical section.

RESOURCE MANAGEMENT:

1. Resource management is the efficient and effective deployment of an organization's resources when they are needed.
2. Such resources may include financial resources, inventory, human skills, production resources, or information technology.
3. A resource manager schedules the processes in a distributed system.
4. This is done to make use of the system resources in such a manner that resource usage, response time, network congestion are optimized.
5. The techniques are:

I) Task Assignment Approach:

Each process submitted by a user for processing is viewed as a collection of related tasks and these tasks are scheduled to suitable nodes so as to improve performance.

Task assignment approach should fulfill the following Goals:

- Minimization of IPC costs.
- Quick turnaround time for the complete process.
- A high degree of parallelism.
- Efficient utilization of system resources in general.

II) Load Balancing Approach:

All the processes submitted by the users are distributed among the nodes of the system so as to equalize the workload among the nodes.

III) Load Sharing Approach:

Make sure that no node is idle and share the resources accordingly.

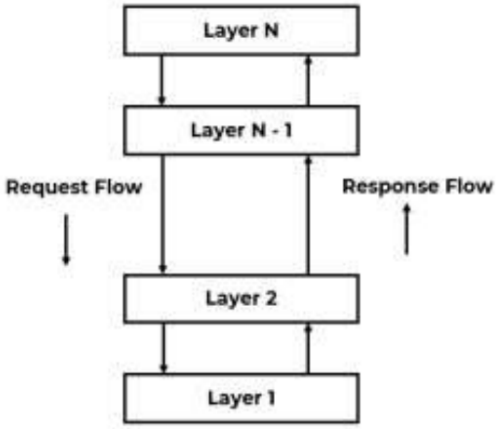
DISTRIBUTED SYSTEM:

1. A distributed system is a collection of independent computers that appear to the users of the system as a single computer.
2. Example is the World Wide Web.

ARCHITECTURAL STYLES:

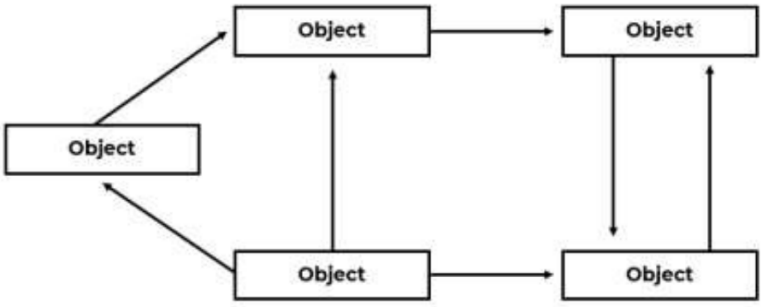
I) Layered Architecture:

1. The basic idea for the layered architecture is that all the components are organized in a layered manner.
2. Where a component at any layer is allowed to call components at its underlying layer.
3. A fundamental observation is that control generally flows from layer to layer, i.e., requests go down the hierarchy whereas the results flow upward.



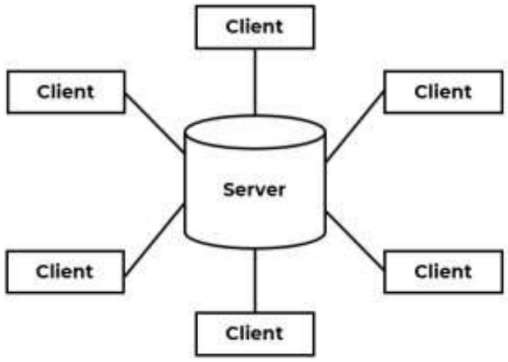
II) Object-based Architecture:

1. Object based architecture uses Remote procedure call mechanism for communication.
2. In the object based architecture, each component or object is connected through a remote procedure call mechanism.
3. Thus, any object can call to any other object in the system and hence the called object can return data or information to the calling object.



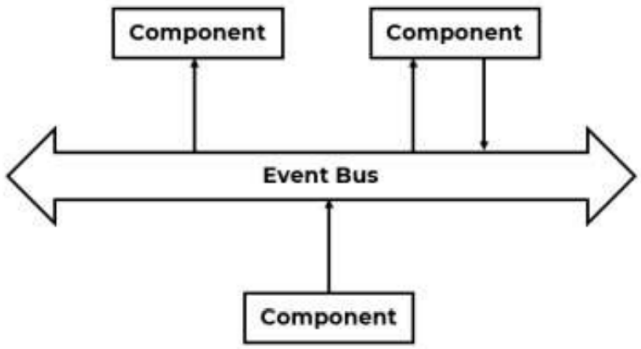
III) Data-Centered Architecture:

1. In data-centered architecture, a server or database lies at the center of the architecture.
2. Clients are placed around the server.
3. Thus the centered server provides data or information to different clients of the system.



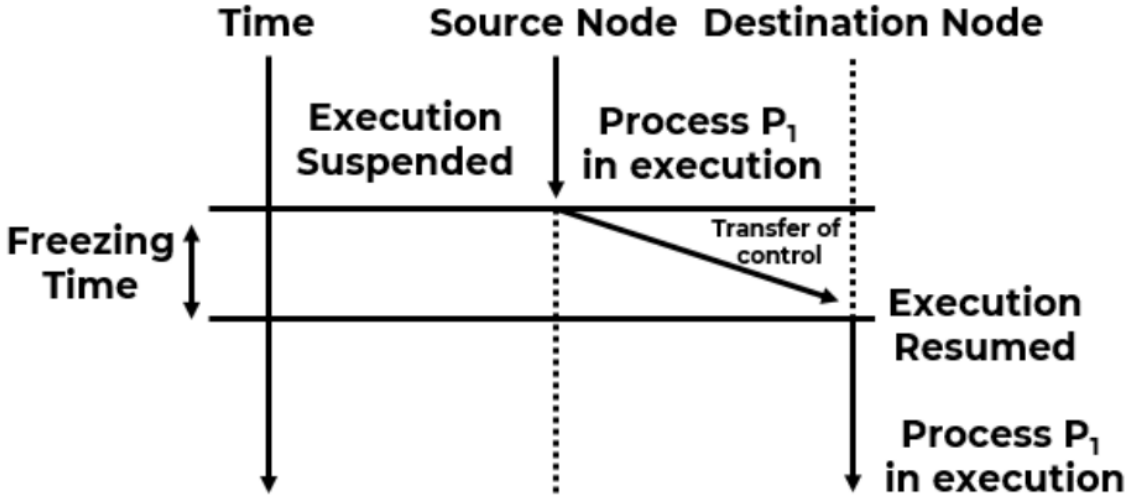
IV) Event-Based Architecture:

1. The entire communication in this kind of a system happens through events.
2. When an event is generated, it will be sent to the bus system.
3. With this, everyone else will be notified telling that such an event has occurred.
4. So, if anyone is interested, that node can pull the event from the bus and use it.
5. Sometimes these events could be data, or even URLS to resources.
6. So the receiver can access whatever the information is given in the event and process accordingly.
7. Processes communicate through the propagation of events.
8. These events occasionally carry data.
9. The main advantage of event-based distributed systems is that processes are loosely coupled or they are simply distributed.



Process Migration:

1. To Move processes from one computing environment to another, process migration is required.
2. Process migration is a specialized form of process management.
3. The most common application of process migration is in computer clusters where processes are moved from machine to machine.
4. Process migration is the relocation of a process from its source node to another destination node.
5. 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.
6. Preemptive process migration is more costly compared to non-preemptive because the process environment must accompany the process to its new node.
7. Steps:
    a. Process is selected for migration.
    b. Selection of destination nodes for the process to be migrated.
    c. Transfer of selected process to the destination node.
8. Migration policy is responsible for the first two steps while the third step is handled by the migration mechanism.
9. Migration of a process is complex that involves handling various activities to meet the requirements for a good process migration.



10. The sub activities involved are:
    a. Freezing the process on its source node and restarting it on its destination node.
    b. Transferring the process's address space from its source node to its destination node.
    c. Forwarding messages meant for the migrant process.
    d. Handling communication between cooperating processes that have been separated as a result of process migration.
11. A preemptive process migration facility allows the transfer of an executing process from one node to another.
12. On the other hand, in a system supporting only non-preemptive migration facilities, a process can only be transferred prior to beginning its execution.
13. 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.

FEATURES OF PROCESS MIGRATION:

1. Transparency.
2. Minimal Interference.
3. Efficiency.
4. Robustness.

Software concept in distributed systems includes:

1) Distributed Operating System:

1. Distributed Operating System are referred as Loosely Coupled Systems.
2. Distributed Operating System is a model where distributed applications are running on multiple computers linked by communications.
3. A distributed operating system is an extension of the network operating system.
4. It supports higher levels of communication and integration of the machines on the network.
5. Goal is to hide and to manage hardware resources.

Distributed Operating Systems provide the following advantages:

1. Sharing of resources.
2. Reliability.
3. Communication.
4. Computation speedup.

II) Network Operating System:

1. Network Operating Systems are referred to as Tightly Coupled Systems.
2. An operating system that provides connectivity among a number of autonomous computers is called a Network Operating System.
3. It includes special functions for connecting computers and devices into a local-area network (LAN) or Inter-network.
4. Some popular network operating systems are Windows NT/2000, Linux, Sun Solaris, UNIX, and IBM OS/2.
5. Goal is to offer local services to remote clients.

Network Operating Systems provide the following advantages:

1. Centralized servers are highly stable.
2. Security is server managed.
3. Communication.
4. Remote access to servers is possible from different locations.

III) Middleware:

1. Middleware is an additional layer at the top of Network Operating System.
2. It is used to implement general purpose services.
3. Goal is to provide distribution transparency.

Middleware Services:

1. Communication Services.
2. Information System Services.
3. Control Services.
4. Security Services.

Stream Oriented Communications	Message Oriented Communications
It is Connection – Oriented Communication.	It is Connection - less Communication.
1 – to – 1 Communication.	Many – to – Many Communication.
Sender transfers a sequence of individual bytes.	Sender transfers a sequence of discrete messages.
It send data byte-by-byte.	It sends data as input stream
It does not keep track of the message boundaries.	It keep track of the message boundaries.
Arbitrary length transfer.	Each message limited to 64 KB.
It is used by most applications.	It is used for multimedia applications.
It runs over TCP.	It runs over UDP.
Timing might affect performance.	Timing does not affect performance.
Example of the byte stream downloading a song or a movie.	Example Of message stream is sequence of pages

Load Sharing	Load Balancing
Load sharing is the ability to distribute outgoing traffic over multiple paths	Load balancing is the ability to split the load toward the same destination over multiple paths.
Traffic is only shared across different links.	Traffic is balanced across different links.
Load Sharing is more efficient.	Load Balancing is less efficient as compared to load sharing.
Proper utilization of resources is required.	Proper utilization of resources is not required.
It is necessary to prevent the nodes from being idle.	It is not necessary to prevent the nodes from being idle.
It does not uses round robin fashion.	It uses round robin fashion.
It cannot performed on packet-by-packet basis.	It can performed on packet-by-packet basis.
It is a static concept.	It is a dynamic concept.
It is difficult to setup and maintain.	It is easy to setup and maintain.
Example of Protocol: eBGP.	**Example of Protocol:** IGP.

Parameters	NOS	DOS
Definition:	It is a computer operating system that is designed primarily to support workstation, personal computer which is connected on a LAN.	It is an operating system which manages a number of computers and hardware devices which make up a distributed system.
Goal:	Goal is to offer local service to remote client.	Goal is to hide and manage hardware resources.
OS type:	It is tightly coupled operating system.	It is loosely coupled operating system.
Used for:	Used for Heterogeneous multi computers.	Used for multi-processor and homogeneous multi computers.
Architecture:	Follows '2' tier client server architecture.	Follows 'n' tier client server architecture.
Types:	Multicomputer and multiprocessor operating system.	Peer to peer and client server architecture.
Communication mode:	It uses file for communication.	It uses messages for communication.
Transparency:	Degree of transparency is low.	Degree of transparency is high.
Reliability:	Low.	High.
Example:	Novell NetWare.	Microsoft distributed component object model.

Strict Consistency Model	Sequential Consistency Model
It was proposed by R. Denvik.	It was proposed by Lamport.
In this the value returned by a read operation on a memory address is always the same as the value written by the most recent write operation to that address.	In this all the processes see the same order of all memory access operations on the shared memory.
Consistency requirement is stronger than sequential model.	Consistency requirement is weaker than strict model.
Implementation of the strict consistency model requires the existence of an absolute global time.	Implementation of the sequential consistency model does not require the existence of an absolute global time.
In the absence of explicit synchronization operations, running a program twice gives same results.	In the absence of explicit synchronization operations, running a program twice may not give same results.

Data Centric Consistency	Client Centric Consistency
It is used for all client in a system.	It is used for individual client.
It does not have lack of simultaneous updates.	It has lack of simultaneous updates.
It is data specific.	It is client specific.
Globally accessible.	Not globally accessible.
It aims to provide system wide consistent view on a database	It aims to provide client specific consistent view on a database.

Examples: ●

- Data Centric Consistency
  - Strict Consistency Model ○
  - Sequential Consistency
  - Linearizability
  - Causal Consistency
  - FIFO Consistency
  - Weak Consistency
  - Release Consistency
  - Entry Consistency
- Client Centric Consistency
  - Eventual consistency
  - Monotonic reads
  - Monotonic writes
  - Read your writes
  - Writes follow reads