1) Explain about the Distributed Computing Systems & Distributed Information System?

- **Distributed Computing Systems:**

Distributed computing is a computing concept that refers to multiple computer systems working on a single problem.

The single problem is divided into many parts, and each part is solved by different computers. As long as the computers are networked, they can communicate with each other to solve the problem.

The most important functions of distributed computing are: Resource sharing, Openness, Concurrency, Scalability, Fault tolerance, Transparency.

Types Of Distributed Computing Systems:

1.Cluster Computing : Collection of similar workstations/PCs, closely

connected by means of a high-speed LAN.

Benefits:

Each node runs the same OS

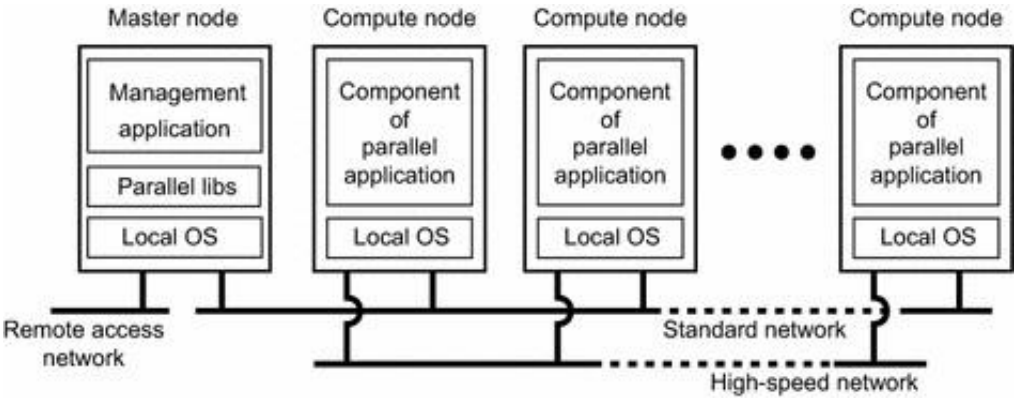
Homogeneous environment

Can serve as a supercomputer

Excellent for parallel programming

Examples: Linux-based Beowulf clusters,MOSIX (from Hebrew University).

Architecture for Cluster Computing System:



2.Grid Computing: Collection of computer resources, usually owned by multiple parties and in multiple locations, connected together such that users can share access to their combined power.

Benefits:

Easier to collaborate with other organizations.

Make better use of existing hardware.

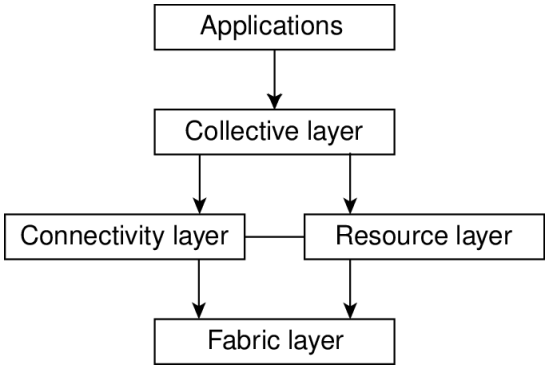
Can solve larger, more complex problems in a shorter time.

Grid environments are much more modular and don't have single points of failure. If one of the servers/desktops within the grid fails there are plenty of other resources able to pick the load.

Highly powerful systems can be used and the computing power can be scaled as needed.

Examples: EGEE - Enabling Grids for E-SciencE(Europe), Open Science Grid (USA).

Architecture for Grid Computing Systems:



3.Cloud computing: Collection of computer resources, usually owned by a single entity, connected together such that users can lease access to a share of their combined power.

Benefits:

Location independence: the user can access the desired service from anywhere in the world, using any device with any (supported) system.

Cost-effectiveness: the whole infrastructure is owned by the provider and requires no capital outlay by the user.

Reliability: You'll receive continuous updates and upgrades at no additional cost, because your cloud-provider will need to ensure its services are up-to-date.

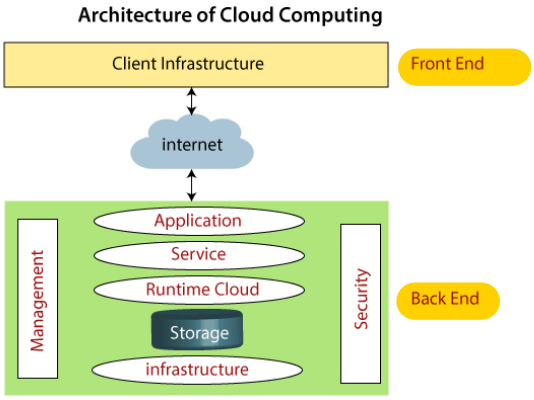
Scalability: user needs can be tailored to available resources as demand dictates – cost benefit is obvious.

Security: low risk of data loss thanks to centralization.

Examples: Amazon EC2 (Elastic Compute Cloud), Google App Engine, IBM Enterprise Data Center,

MS Windows Azure, SUN Cloud Computing.

Architecture for Cloud Computing Systems:



- **Distributed Information System:**

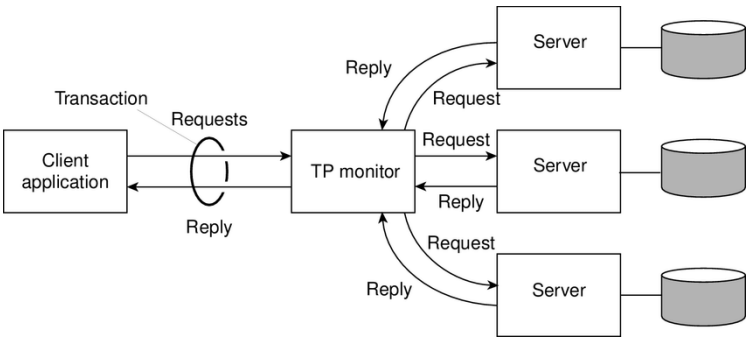
A set of **information systems** physically **distributed** over multiple sites, which are connected with some kind of communication network.

Types Of Distributed Information Systems:

1.

Transaction Processing Systems

- Provide a highly structured client-server approach for database applications
- Transactions are the communication model
- Obey the ACID properties:
  - Atomic: all or nothing. Each transaction either happen completely, or not at all. And if it happens it happens in a single indivisible, instantaneous action.
  - Consistent: invariants are preserved. Transaction does not violate system invariants
  - Isolated (serializable) concurrent transactions do not interfere with each other
  - Durable: committed operations can't be undone



The role of a TP monitor in distributed systems

2.

Enterprise Application Integration (EAI)

- Less structured than transaction-based systems
- EA components communicate directly
  - Enterprise applications are things like HR data, inventory programs, ...
  - May use different OSs, different DBs but need to interoperate sometimes.
- Communication mechanisms to support this include CORBA, Remote Procedure Call (RPC) and Remote Method Invocation (RMI)

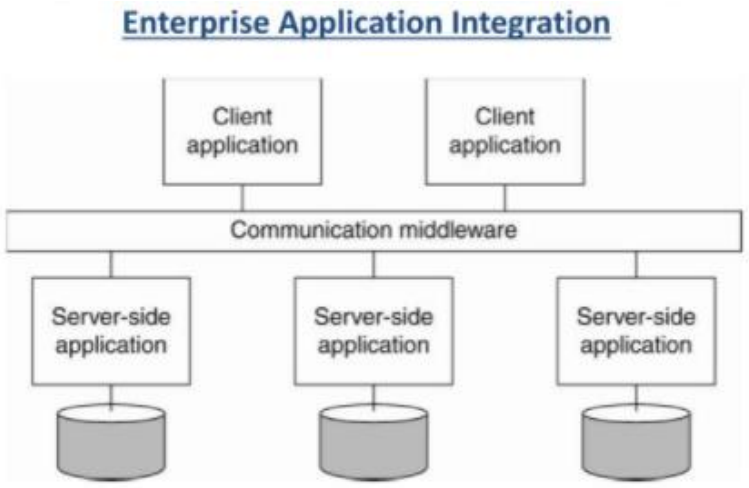


Figure: Middleware as a communication facilitator in enterprise application integration.

2) Explain about Centralised & Decentralised architectures?

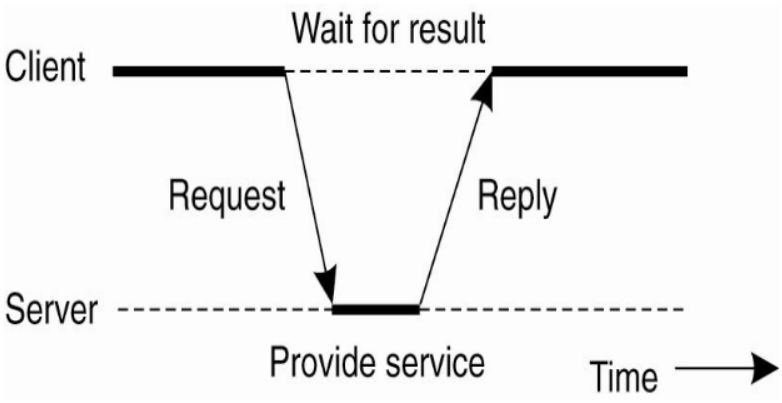
1. Centralised Architectures: **(Client Server Architecture)**

The client server architecture has two major components: the client and the server.

The Server is where all the processing, computing and data handling is happening, whereas the Client is where the user can access the services and resources given by the Server (Remote Server).

The clients can make requests from the Server, and the Server will respond accordingly.

Generally, there is only one server that handles the remote side. But to be on the safe side, we do use multiple servers will load balancing techniques.



As one common design feature, the Client Server architecture has a centralized security database. This database contains security details like credentials and access details. Users can't log in to a server, without the security credentials.

So, it makes this architecture a bit more stable and secure than Peer to Peer. The system might get low, as the server only can handle a limited amount of workload at a given time.

2. Decentralised Architectures: (Peer to Peer (P2P))

The basic idea is that, each node can either be a client or a server at a given time.

If the node is requesting something, it can be known as a client, and if some node is providing something, it can be known as a server. In general, each node is referred to as a Peer.

In this network, any new node has to first join the network. After joining in, they can either request a service or provide a service.

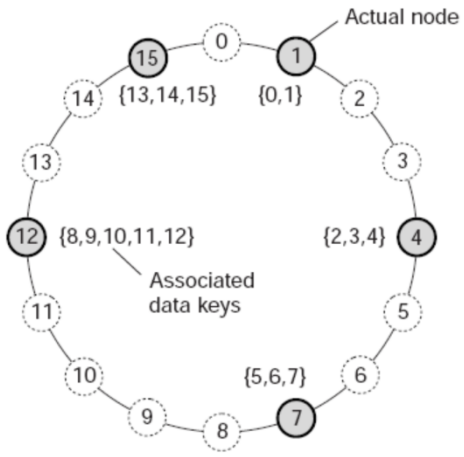
In general, the peer to peer systems can be separated into three unique sections:

(a) Structured P2P Architecture:

The meaning of the word structured is that the system already has a predefined structure that other nodes will follow.

Every structured network inherently suffers from poor scalability, due to the need for structure maintenance.

In general, the nodes in a structured overlay network are formed in a logical ring, with nodes being connected to this ring. In this ring, certain nodes are responsible for certain services.



A common approach that can be used to tackle the coordination between nodes is to use distributed hash tables (DHTs).

(b) Unstructured P2P Architecture:

There is no specific structure in these systems, hence the name "unstructured networks".

Due to this reason, the scalability of the unstructured p2p systems is very high. These systems rely on randomized algorithms for constructing an overlay network.

Every unstructured system tried to maintain a random path. Due to this reason, the search of a certain file or node is never guaranteed in unstructured systems.

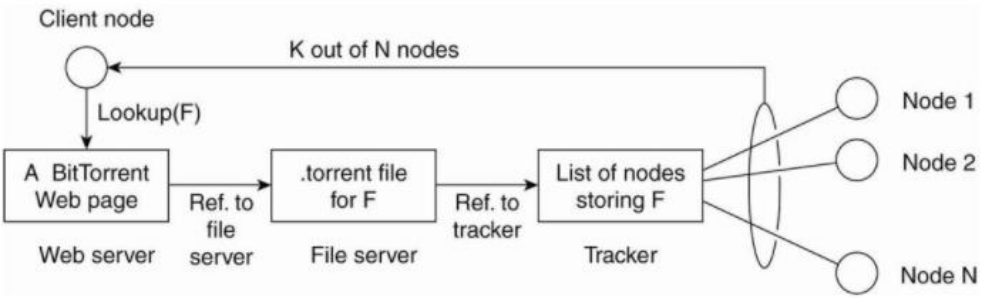
The basic principle is that each node is required to randomly select another node, and contact it.

(c) Hybrid P2P Systems:

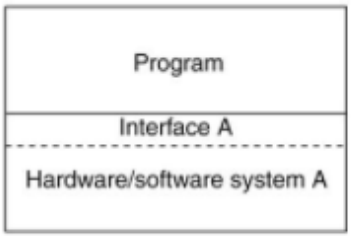
Hybrid systems are often based on both client server architectures and p2p networks.

Eg: Collaborative Distributed System:

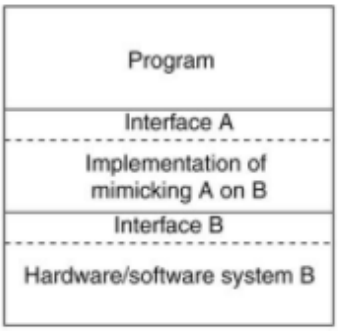
A famous example is Bit torrent, which we use every day. It is a peer-to-peer file downloading system. Here when a user wants a file he downloads chunks of it from other user until all the download chunks are combined to obtain a complete file. The following figure shows the operation of bit torrent:



3) Explain Virtualization in detail?



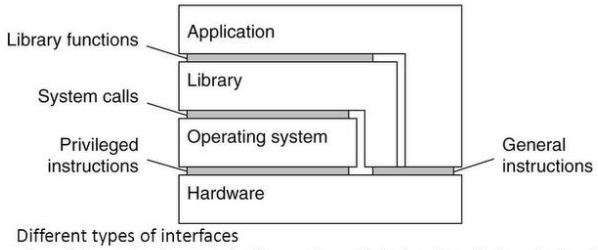
(a)



(b)

Virtualization means extending or replacing an existing interface to mimic the behaviour of another system.

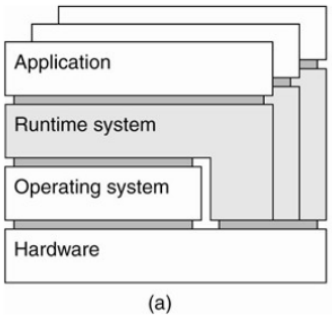
Types of Interfaces



- Different types of interfaces
  1. Between hardware and software: Assembly instructions that can be invoked by any program.
  2. Between OS and hardware: Assembly instructions by privileged programs
  3. System calls: Offered by OS
  4. Library functions: APIs
- Depending on what is replaced /mimicked, we obtain different forms of virtualization

Types of Virtualization
1. Process Virtual Machines

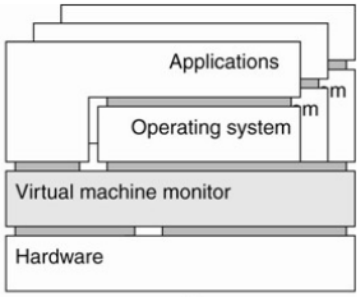
- Runtime system that essentially provides an abstract instruction set that is to be used for executing applications.
- Instructions can be interpreted but could also be emulated as is done for running Windows applications on Unix platforms.
  - In this case, emulator also mimic the system calls.
- Ex. **JVM**



(a)

Types of Virtualization
2. Virtual Machine Monitor VMM

- Typical examples are VMware and Xen
- Virtualization is implemented as a layer **completely shielding hardware**
- Offering the complete instruction sets
- Can be offered simultaneously to different programs at the same time
- It is possible to have multiple, and different operating systems run independently and concurrently on the same platform.



(b)

4) Explain about implementation of RPC & RMI?

(A)RPC (REMOTE PROCEDURE CALL):

Remote procedure call is a method that allows programs to call procedures from other machines.

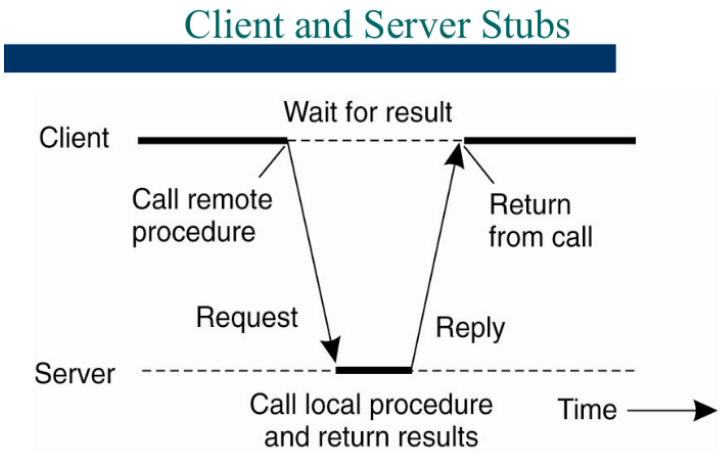
RPC IMPLEMENTATION:

The main idea behind working of RPC is to make a remote procedure call look like a local call. In this way transparency is achieved.

The procedure is split into two parts:

The CLIENT "stub" – implements the interface on the local machine through which the remote functionality can be invoked.

The SERVER "stub" – implements the actual functionality, i.e., does the real work!

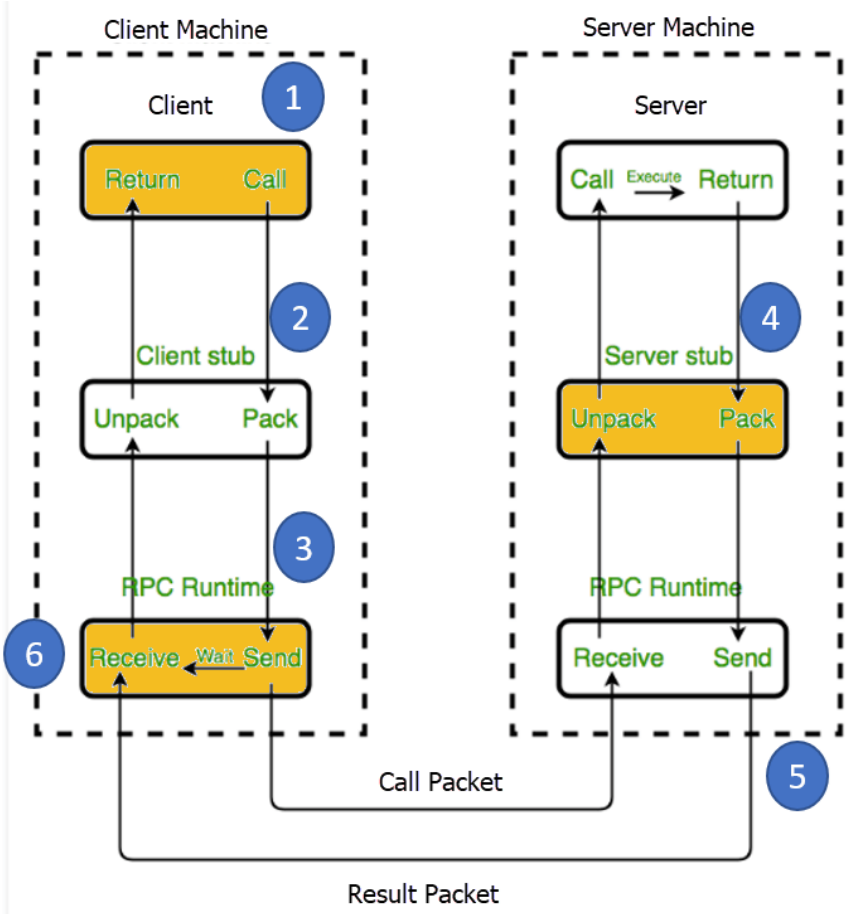


The 10 Steps of a RPC:

An RPC is performed in the following steps:

1. Initially a client procedure calls a client stub
2. Then the client stub develops a message and call a local OS
3. The OS of the client transmits the message to remote OS
4. The remote OS sends the message to the server stub
5. The server stub then unpacks the parameters.

6. After this, the server stub calls the server
7. The server executes and returns the result to the stub
8. The server stub packs the result in a messages and calls its local OS
9. The OS of the server transmits the messages to the OS of the client
10. The OS of the client sends the message to the client stub
11. Finally the server stub unpacks the result and returns it to the client



(B) RMI (REMOTE METHOD INVOCATION):

Remote method invocation (RMI) refers to calling a method on a remote object.

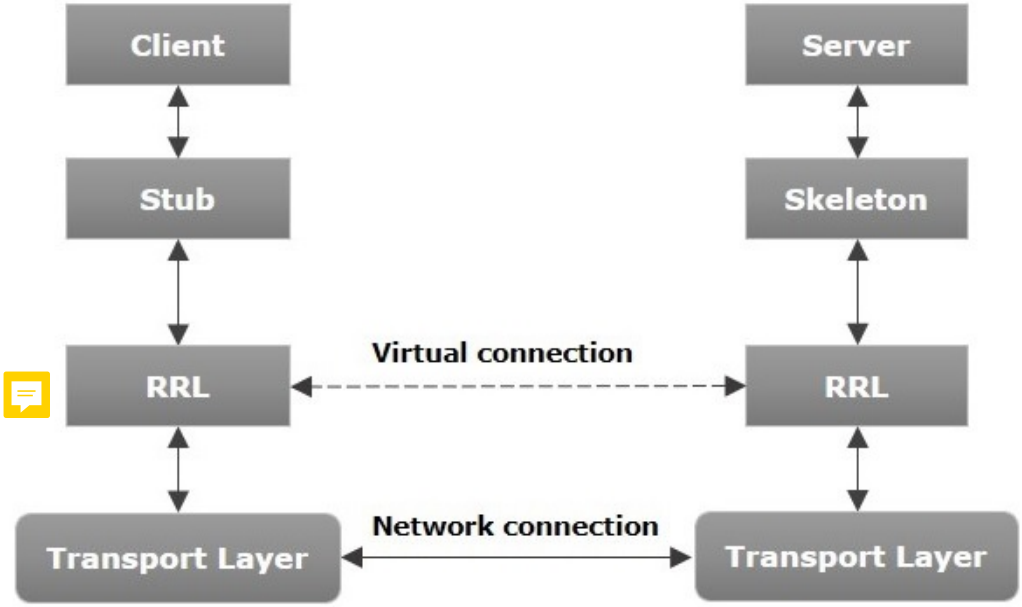
RMI allows us to distribute our objects on various machines, and invoke methods on the objects located on remote sites.

The key feature of an object is that it encapsulates data, called the state, and the operations on those data, called the methods. Methods are made available through an interface.

An object may implement multiple interfaces. Likewise, given an interface definition, there may be several objects that offer an implementation for it.

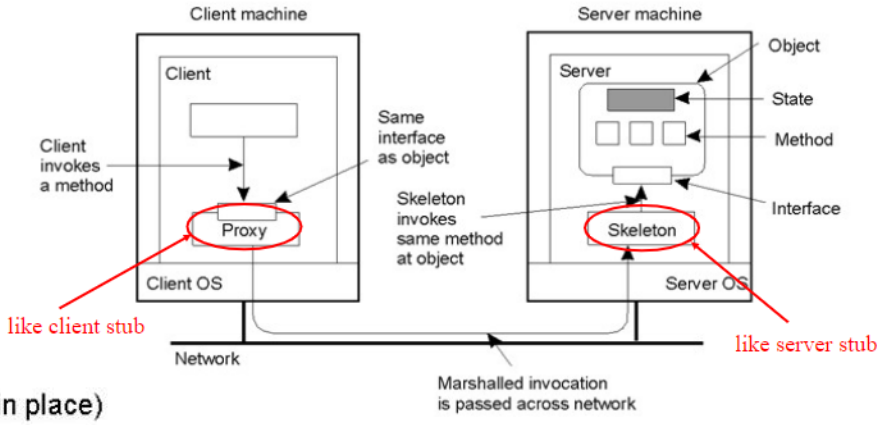
This separation between interfaces and the objects implementing these interfaces is crucial for distributed systems.

A strict separation allows us to place an interface at one machine, while the object itself resides on another machine. This organization, which is shown in Figure, is commonly referred to as a distributed object.



ed Objects

ote object with client-side proxy.



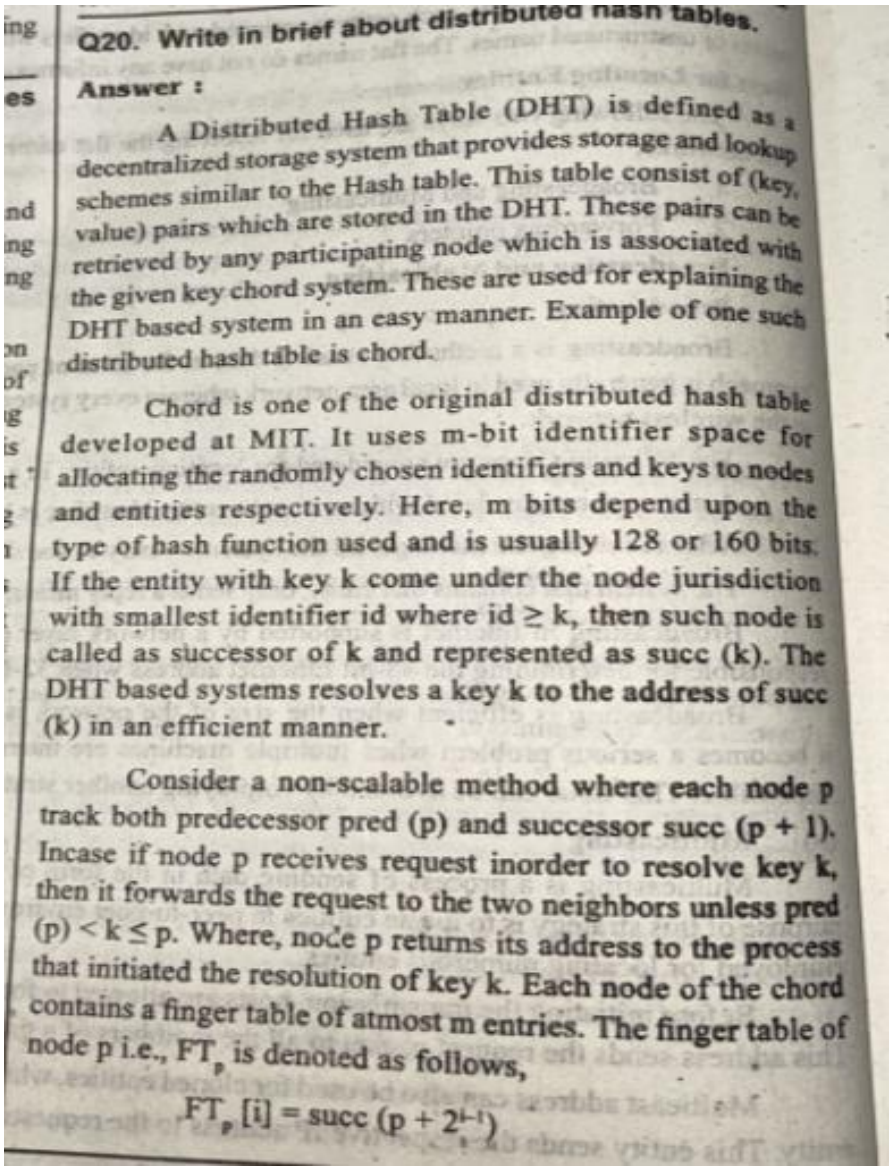
Basics: (Assume client stub and server skeleton are in place)

- Client invokes method at stub
- Stub marshals request and sends it to server
- Server ensures referenced object is active:
  - Create separate process to hold object
  - Load the object into server process
- Request is unmarshaled by object's skeleton, and referenced method is invoked
- If request contained an object reference, invocation is applied recursively (i.e., server acts as client)
- Result is marshaled and passed back to client
- Client stub unmarshals reply and passes result to client application

5) Explain about Distributed Hash Tables & Directory Services?

Distributed Hash Tables:

(make points and study from the fig)



Q20. Write in brief about distributed hash tables.

Answer :

A Distributed Hash Table (DHT) is defined as a decentralized storage system that provides storage and lookup schemes similar to the Hash table. This table consist of (key, value) pairs which are stored in the DHT. These pairs can be retrieved by any participating node which is associated with the given key chord system. These are used for explaining the DHT based system in an easy manner. Example of one such distributed hash table is chord.

Chord is one of the original distributed hash table developed at MIT. It uses m-bit identifier space for allocating the randomly chosen identifiers and keys to nodes and entities respectively. Here, m bits depend upon the type of hash function used and is usually 128 or 160 bits. If the entity with key k come under the node jurisdiction with smallest identifier id where id $\geq$ k, then such node is called as successor of k and represented as succ (k). The DHT based systems resolves a key k to the address of succ (k) in an efficient manner.

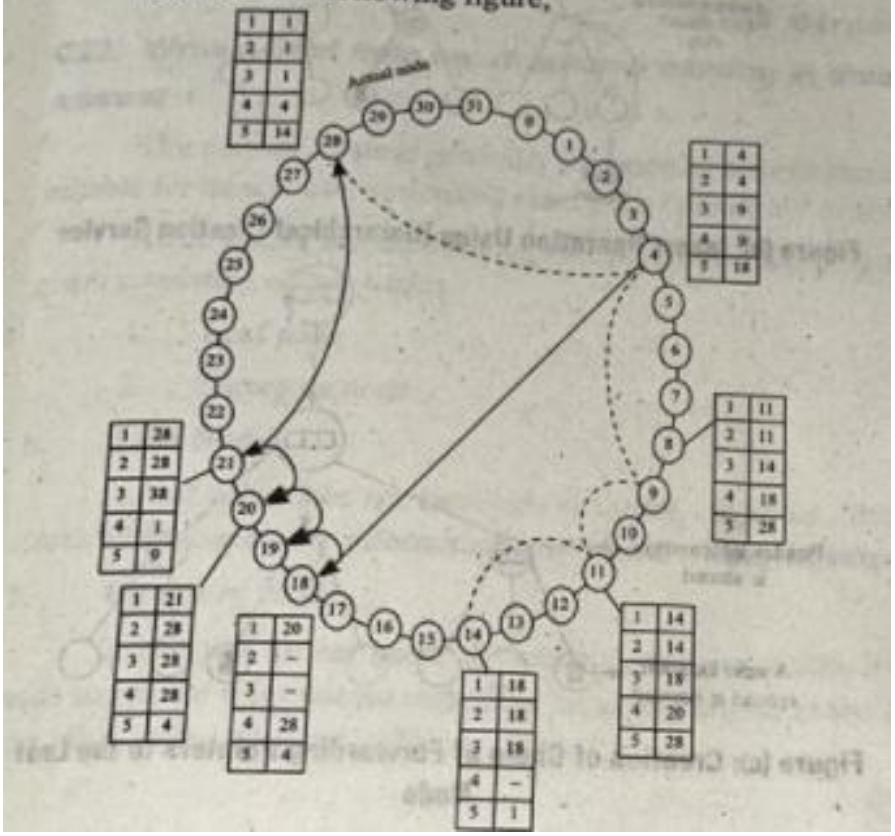
Consider a non-scalable method where each node p track both predecessor pred (p) and successor succ (p + 1). Incase if node p receives request inorder to resolve key k, then it forwards the request to the two neighbors unless pred (p) < k $\leq$ p. Where, node p returns its address to the process that initiated the resolution of key k. Each node of the chord contains a finger table of atmost m entries. The finger table of node p i.e., FT_p is denoted as follows,

$$FT_p[i] = succ(p + 2^{i-1})$$

The i^{th} entry points to the first node succeeding p by 2^{i-1}. Here, these references are considered as shortcuts to the existing nodes in identifier space. The shortcut distance from the node p increases as the index of the finger table increases. The node p forwards the request to node q to look up a key k with index j in the finger table of node p i.e.,

$$q = FT_p[j] \leq k < FT_p[j+1]$$

Consider the following figure,



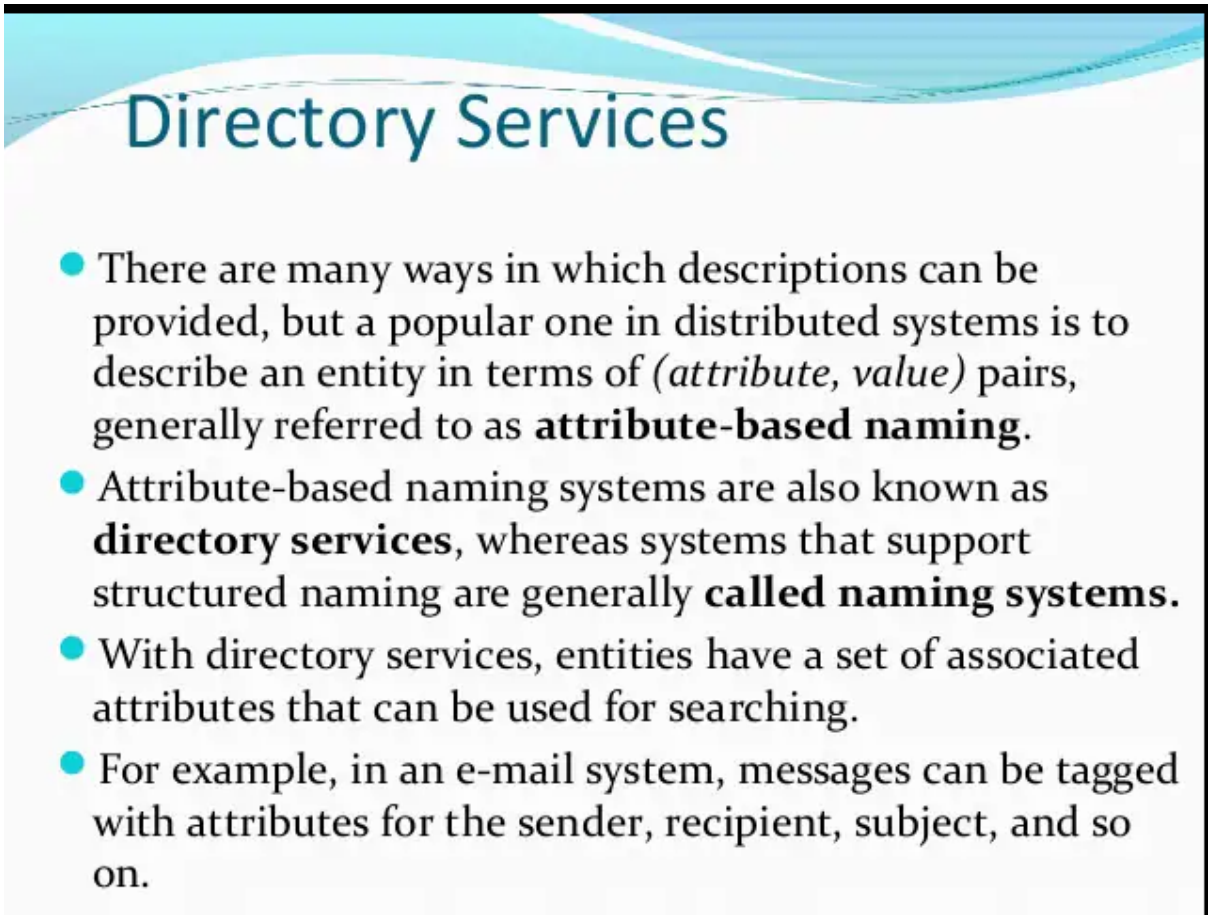
In the above figure, k = 26 from node 1. Initially, node 1 look up for k = 26 in the finger table for determining that this value is greater than $FT_1[5]$ i,e., the request will be send to node $18 = FT_1[5]$. Next, the node 18 chooses node 20 as $FT_{18}[2] < k \leq FT_{18}[3]$. Then, the request is sent from node 20 to node 21 and to node 28 [as node 28 is responsible for k = 20]. Here, the address of node 28 is again returned to node 1 and the key is resolved. If node 28 is requested for resolving k = 12, then the request is routed in dashed line as shown in the above figure. The lookup requires O(log(N)) steps where N is the number of nodes present in the system.

The group of participating nodes in large distributed systems can be changed over the time. The nodes leave, join and fail to recover again.

If node p wants to join the system, it contacts an arbitrary node and requests for a lookup "succ (P + 1) in the existing system. The node p is inserted into the ring only if the node is identified. The complexity in this scenario arises by updating the finger table. For each node q, $FT_q[1]$ is true and this entry indicates to the next node in the ring i.e., successor of q + 1. Each node q runs a simple method that contacts succ(q + 1) and requests to return pred (succ(q + 1)). When q = pred (succ(q + 1)) then q knows that the data is consistent with successor. The new node p enters the system with $q < p \leq$ succ (q + 1) when the successor of q updated its predecessor.

Thus, the node q adjust with $FT_p[1]$ to p. It also checks whether node p recorded node q as its predecessor. Otherwise, another adjustment is needed for $FT_q[1]$. In the same way, to update the finger table, the node q requires the successor for $k = q + 2^{i-1}$ for each entry i. This can be done by getting a request to resolve the successor of k such kind of requests are issued in a p background process in chord system.

Similarly, each node q checks whether its predecessor is alive or not when predecessor fails, node q records the fact and sets the pred (q) to 'unknown'. And if node is updating its link to next known node in the ring and knows that the predecessor of succ (q + 1) that it suspects to be predecessor. The chord system is ensured to be consistent with exception of few nodes.



6) Explain Iterative and Recursive Name Resolution?

(A) Recursive Name Resolution:

When a client sends a recursive request to the DNS server, the server responds back with the answer if it has information sought. If it doesn't, the server takes the responsibility for finding the answer by becoming a client on behalf of the original client and sending new request to another server. The original client sends only one request, and eventually gets the information it wants.

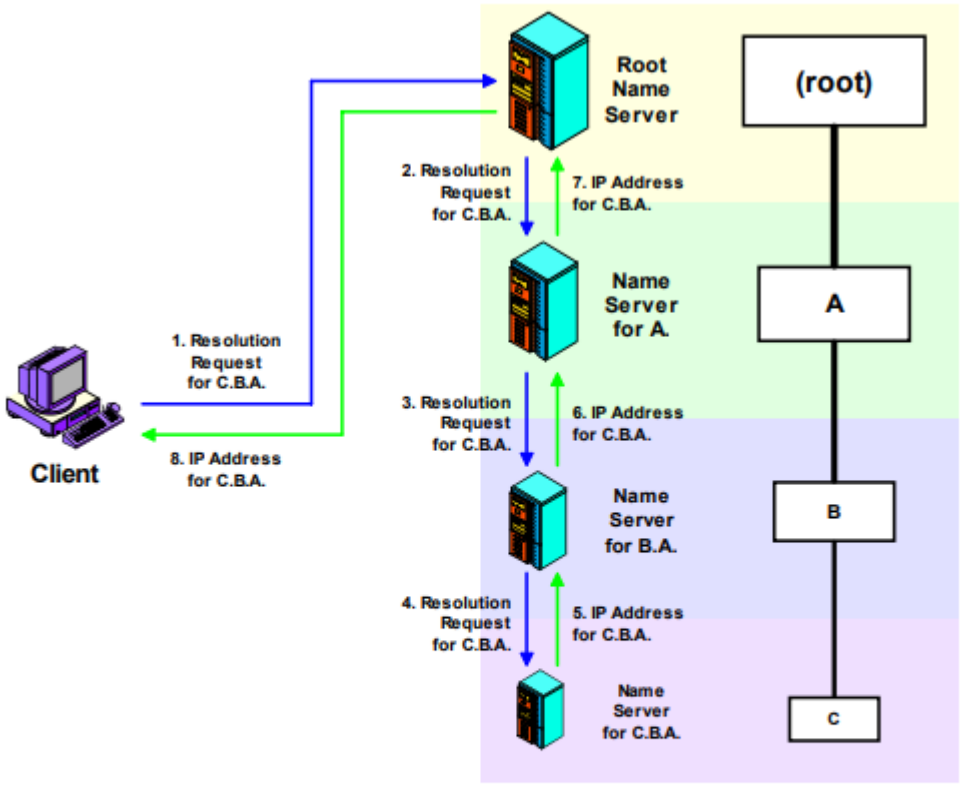


Figure 244: Recursive DNS Name Resolution

This is the same theoretical DNS resolution that I showed in Figure 243, but this time, the client asks for the name servers to perform recursive resolution and they agree to do so. As in the iterative case, the client sends its initial request to the root name server. That server doesn't have the address of "C.B.A.", but instead of merely returning to the client the address of the name server for "A.", it sends a request to that server itself. That name server sends a request to the server for "B.A.", which in turn sends a request to the server for "C.B.A.". The address of "C.B.A." is then carried back up the chain of requests, from the server of "C.B.A." to that of "B.A.", then "A.", then the root, and then finally, back to the client.

(B) Iterative Name Resolution:

When a client sends an iterative request to the DNS server, the server responds back with either the answer to the request (or) name of another server that has the information.

The original client must then iterate by sending new request to this referred server, which again may either answer it (or) provide another server name.

The process is continued until right server is found.

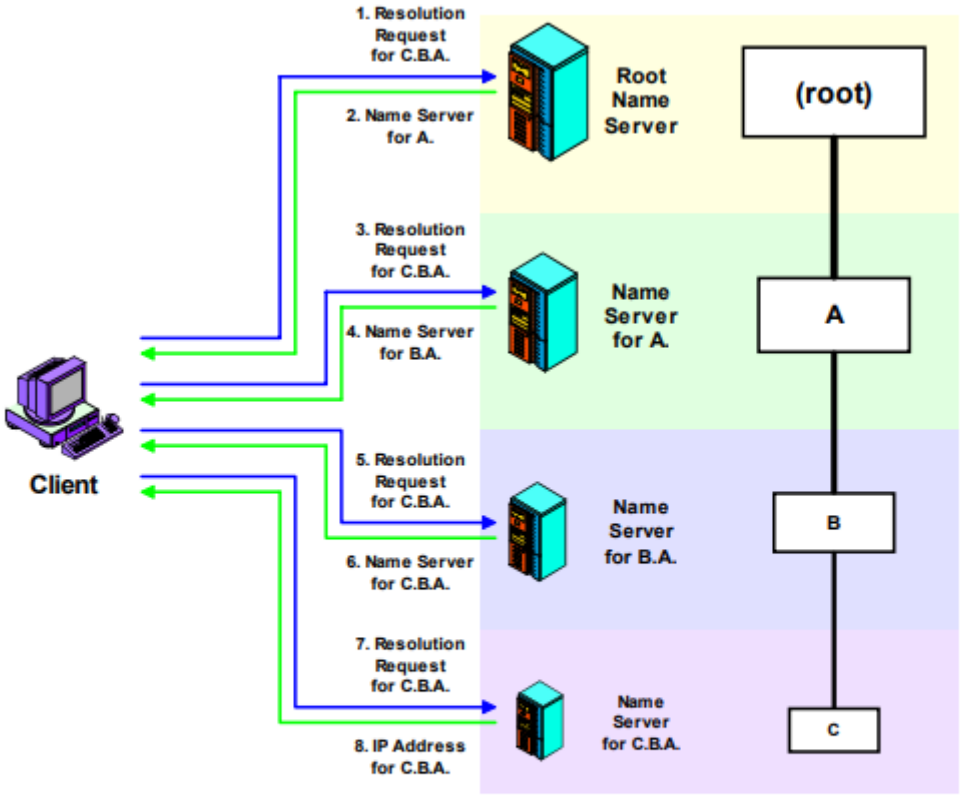


Figure 243: Iterative DNS Name Resolution

In this example, the client is performing a name resolution for "C.B.A." using strictly iterative resolution. It is thus responsible for forming all DNS requests and processing all replies. It starts by sending a request to the root name server for this mythical hierarchy. That server doesn't have the address of "C.B.A.", so it instead returns the address of the name server for "A.". The client then sends its query to that name server, which points the client to the server for "B.A.". That name server refers the client to the name server that actually has the address for "C.B.A.", which returns it to the client. Contrast to Figure 244.

7) Explain any two Mutual Exclusion Algorithms?

1. Centralized Algorithm :

As its name indicates, there is one coordinator which handles all the requests to access the shared resource/data.

Every process takes permission to the coordinator, if coordinator will give permission only then a particular process can enter into CS. Coordinator will maintain a queue to keep all the requests in order.

Algorithm

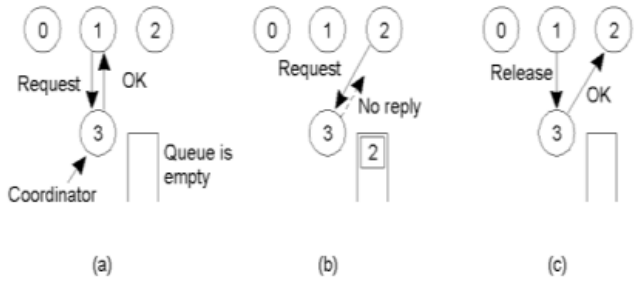


Fig.2.1: Working of Centralized algorithm

a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted because queue is empty and no pending request is there so coordinator will give permission.

b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply because P1 is not exited from CS till now.

c) When process 1 exits the critical region, it tells the coordinator, after that Coordinator will give permission to P2.

Advantages:

- Fair algorithm; follow FIFO order for to give permission.
- Easy to implement
- Scheme can be used for general resource allocation.

Shortcomings:

- Single point of failure, No fault tolerance.
- Confusion between No-reply and permission denied.
- Performance bottleneck because of single coordinator in a large system

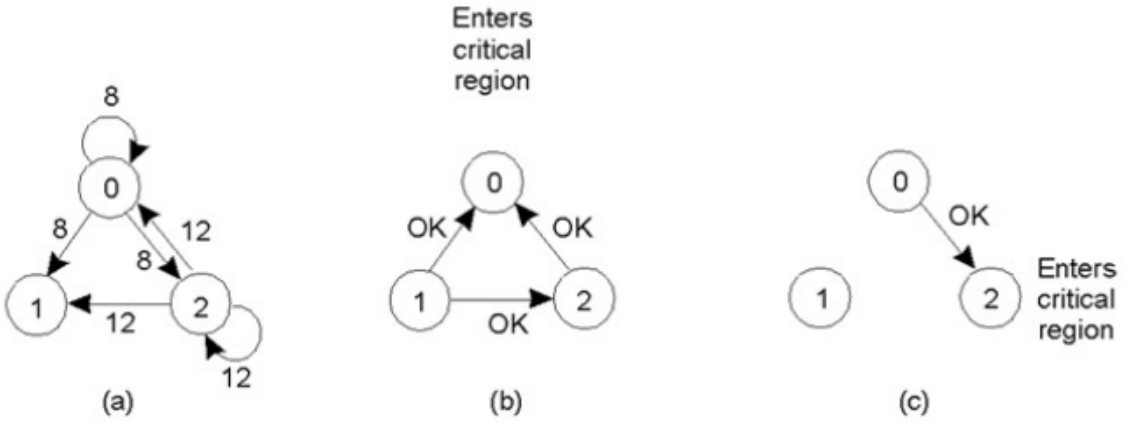
2. Distributed Algorithm :

In this scheme, there is no coordinator. Every process asks to other process for permission to enter into CS.

A Distributed Algorithm

- Assuming there is a total ordering of all events in the system
  1. When a process wants to enter a critical region, it builds a message containing the name of the critical region it wants to enter, its process number, and the current time. Then it sends the message to all other processes.
  2. When a process receives a request,
     1. If the receiver is not in the critical region, and does not want to enter it, it sends back an OK message to the sender.
     2. If the receiver is already in the critical region, it does not reply. It queues the request.
     3. If the receiver wants to enter the critical region, but has not yet done so, it compares the timestamp in the incoming message with the one contained in the message that it has sent to everyone. The lowest one wins. If the incoming message is lower, the receiver sends back an OK message. Otherwise, it queues the message.
  3. As soon as all permissions are in, it may enter the critical region. When it exits the critical region, it sends OK message to all processes on its queue and deletes them all from the queue.

A Distributed Algorithm



a) Two processes want to enter the same critical region at the same moment.

b) Process 0 has the lowest timestamp, so it wins.

c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

- Advantage
  - No deadlock or starvation
  - Number of message per entry is *2(n-1)*
  - No single point of failure
- Disadvantage
  - *n* points of failure
    - If any process crashes, the failure to respond to requests will be interpreted as denial of permission
    - Patch: When a request comes in, the receiver always sends a reply, either granting or denying. Whenever a request or reply is lost, the sender times out and keep trying until a reply comes back or the sender concludes that the receiver is dead. After a request is denied, the sender should block waiting for a subsequent OK message.
  - group communication support needed
  - If one process is unable to handle the load, it is unlikely that forcing everyone to do exactly the same thing in parallel is going to help much.
    - Modify the algorithm so that when a process has collected permission from majority of the other processes.

5

8) Explain any two Election Algorithms?

We often need one process to act as a coordinator. The selection of this process can be performed automatically by an "election algorithm".

For simplicity, we assume the following:

i. Processes each have a unique, positive identifier.

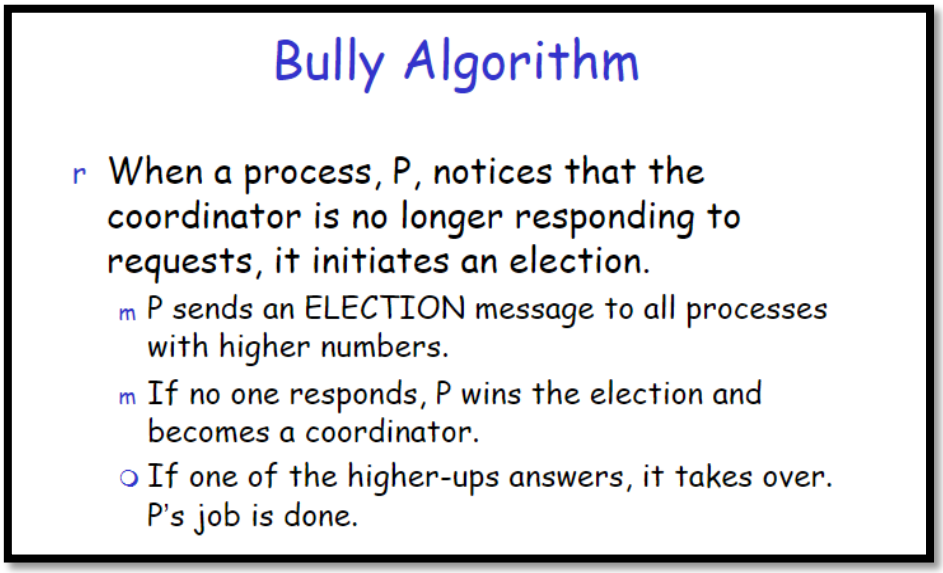
ii. All processes know all other process identifiers.

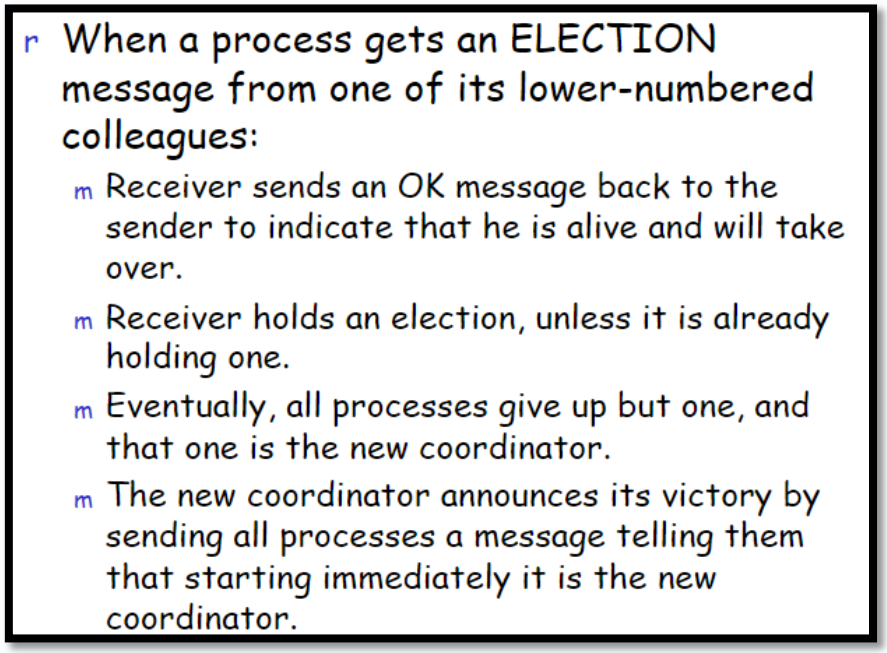
iii. The process with the highest valued identifier is duly elected coordinator.

When an election "concludes", a coordinator has been chosen and is known to all processes.

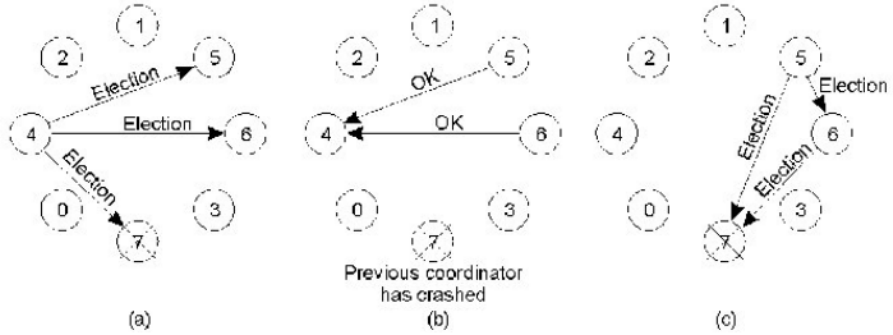
There are two types of election algorithm:

1. **BULLY ALGORITHM:**



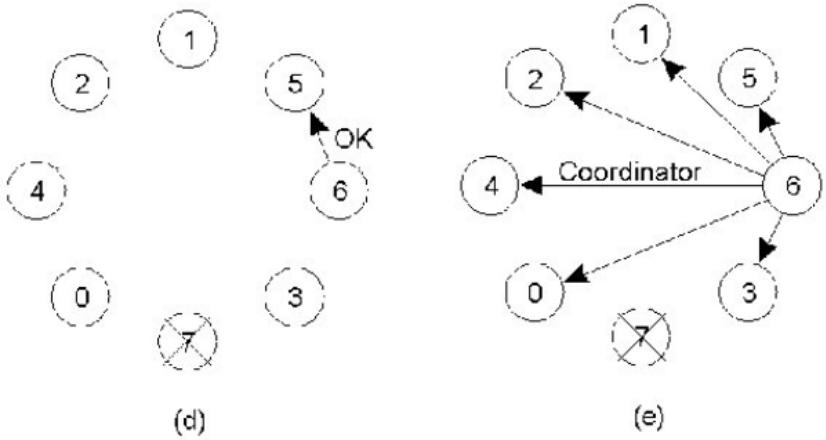


- If a process that was previously down comes back:
  - It holds an election.
    - If it happens to be the highest process currently running, it will win the election and take over the coordinator's job.
- "Biggest guy" always wins and hence the name "bully" algorithm.



The bully election algorithm

- Process 4 holds an election
- Process 5 and 6 respond, telling 4 to stop
- Now 5 and 6 each hold an election



d) Process 6 tells 5 to stop

e) Process 6 wins and tells everyone

2. RING ALGORITHM:

Ring Algorithm

- Use a ring (processes are physically or logically ordered, so that each process knows who its successor is).
- Algorithm
  - When a process notices that coordinator is not functioning:
    - Builds an ELECTION message (containing its own process number)
    - Sends the message to its successor (if successor is down, sender skips over it and goes to the next member along the ring, or the one after that, until a running process is located).
    - At each step, sender adds its own process number to the list in the message.

  - When the message gets back to the process that started it all:
    - Process recognizes the message that contains its own process number
    - Changes message type to COORDINATOR
    - Circulates message once again to inform everyone else: Who the new coordinator is (list member with highest number); Who the members of the new ring are.
    - When message has circulated once, it is removed.
- Even if two ELECTIONS started at once, everyone will pick same leader since node with highest identifier is picked.

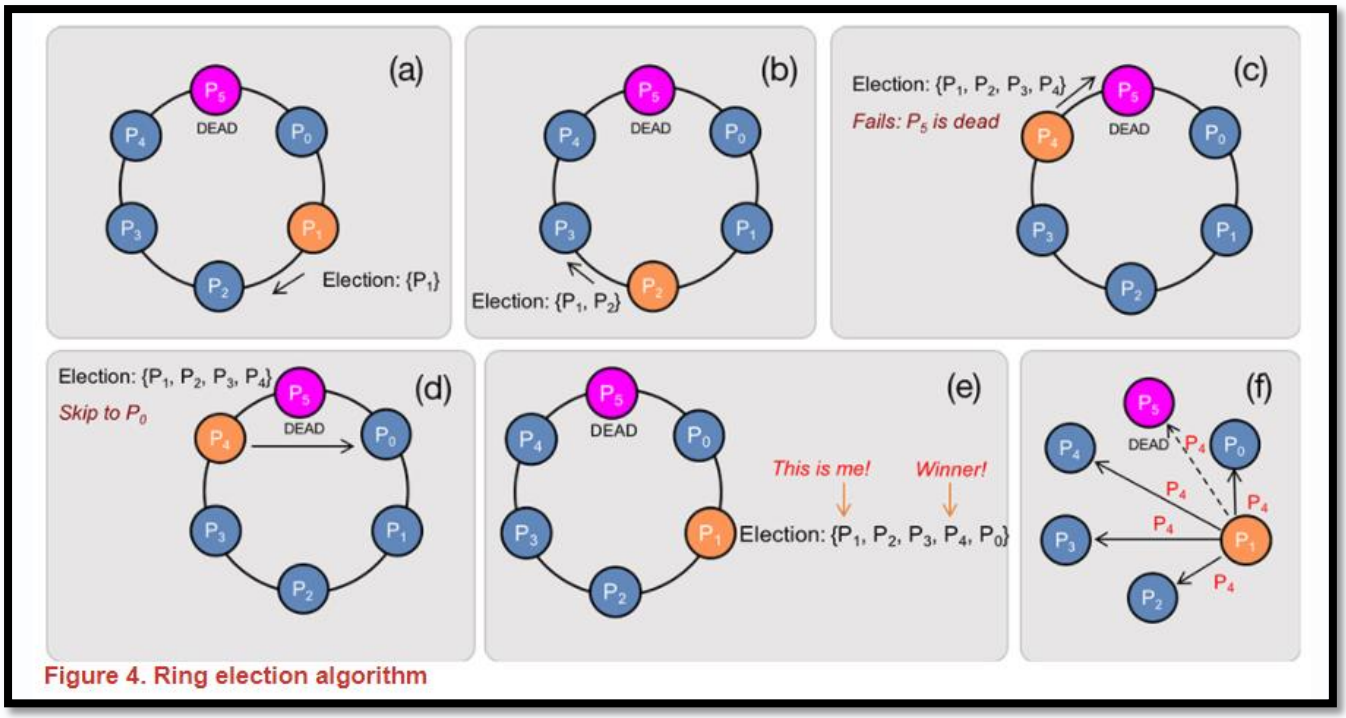


Figure 4. Ring election algorithm

(not mandatory to write the explanation below, just for the understanding purpose I have given it)

Figure 4 shows a ring of six processes (0–5). Process 1 detects that the coordinator, Process 5, is dead. It starts an election by sending an *election* message containing its process ID to its neighbour, Process 2 (Figure 4a).

Process 2 receives an *election* message and sends an *election* message to its neighbour with the ID of Process 2 suffixed to the list (Figure 4b).

The same sequence of events occurs with Process 3, which adds its ID to the list it received from Process 2 and sends an *election* message to process 4. Process 4 then tries to send an *election* message to Process 5 (Figure 4c).

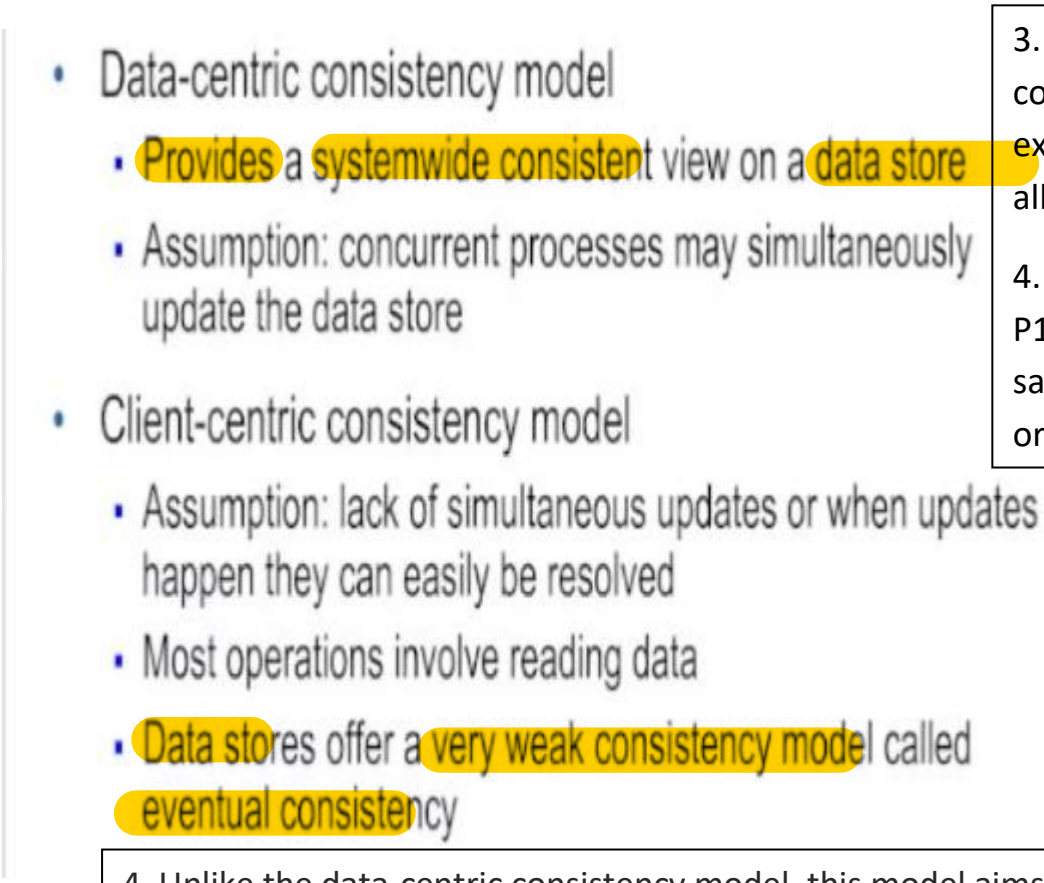
Since process 5 is dead and the message is not delivered, Process 4 tries again to its neighbour once removed: Process 0 (Figure 4d).

Because Process 5 never received the *election* message, its ID does not appear in the list that Process 0 receives. Eventually, the message is received by

Process 1, the originator of the election (Figure 4e). Process 1 recognizes that it is the initiator of the election because its ID is the first in the list of processes.

It then picks a leader. In this implementation, it chooses the highest-numbered process ID in the list, which is that of process 4. It then informs the rest of the group of the new coordinator (Figure 4f).

9) What is the difference between the data centric consistency models & client centric consistency models?



3. Describe the consistency experienced by all clients

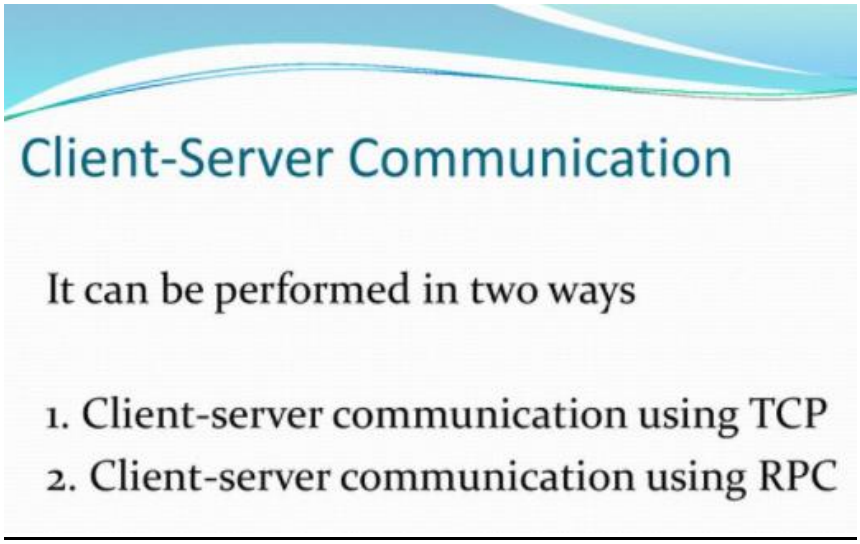
4. Clients P1,P2,P3… see same kinds of orderings.

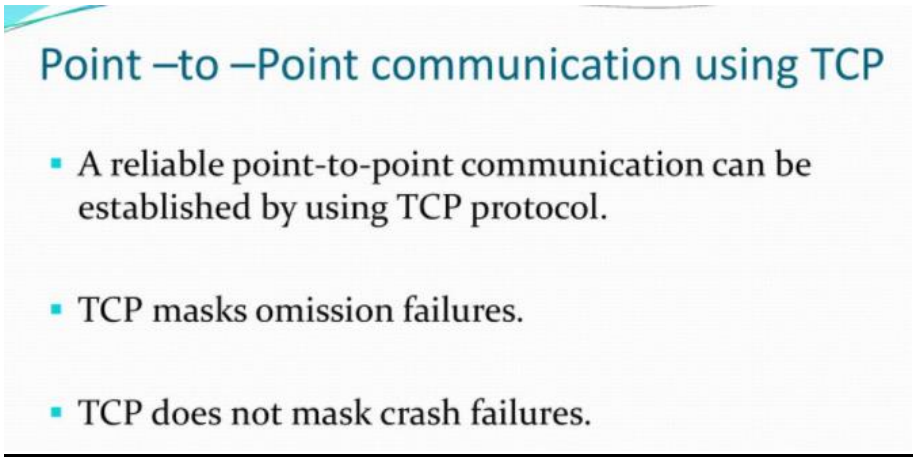
4. Unlike the data-centric consistency model, this model aims to guarantee consistency for a single client.

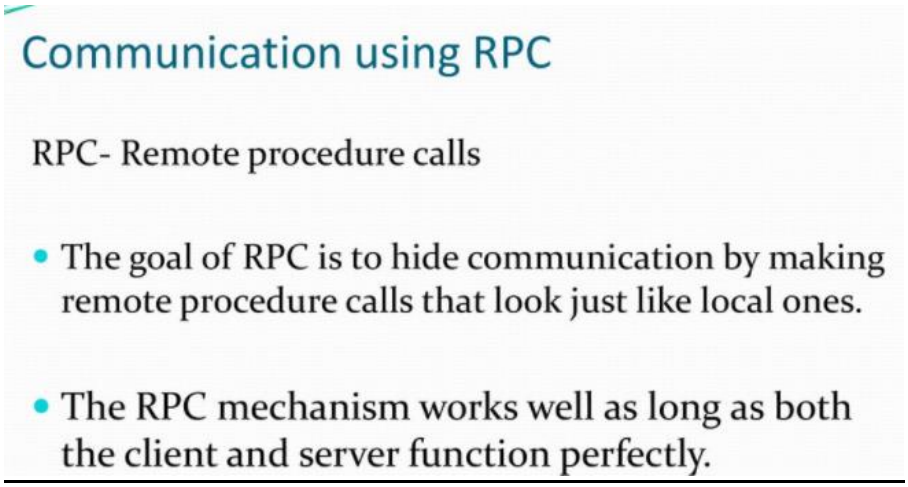
5. Clients P1,P2,P3… may see same different of orderings.

10) Explain reliable client server communication & reliable server communication?

1. Reliable client server communication:







There are 5 classes of failures in RPC

1. The client unable to locate server. (so no request can be sent.)
2. The request message from client to server is lost.(so no response is returned by the server to the waiting client.)
3. The reply message from the server to the client is lost.(the service has completed, but the results never arrive at the client)
4. The server crashes after receiving a request.(and the service request is left acknowledged, but undone.)
5. The client crashes after sending a request.(and the server sends a reply to a newly-restarted client that may not be expecting it.)

1. Client unable to locate server

- Problem- When server goes down

  The RPC system informs the client of the failure

- Solution- error raise an exception
  Java- division by zero
  C- signal handlers

2. Request message from client to server is lost.

- The operating system starts a timer when the stub is generated and sends a request. If response is not received before timer expires, then a new request is sent.
- Lost message – works fine on retransmission.
- If request is not lost, we should make sure server knows that its a retransmission.

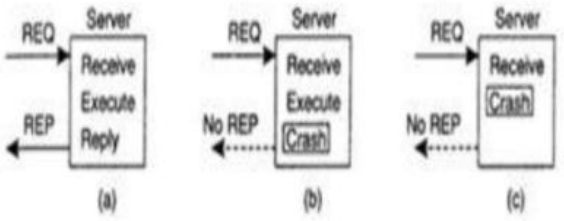
3.Reply message from server to the client is lost.

- Some messages can be retransmitted any number of times without any loss.
- Some retransmissions cause severe loss.

- Solution- client assigns sequence number on requests made by client. So server has to maintain the sequence number while sending the reply.

4. Server crashes after receiving request

- The client cannot tell if the crash occurred before or after the request is carried out



Methods involved

- Remote operation: print some text and (when done) send a completion message.
- Three events that can happen at the server:

1. Send the completion message (M),
2. Print the text (P),
3. Crash (C).

- These three events can occur in six different orderings:

1. M →P →C: A crash occurs after sending the completion message and printing the text.
2. M →C (→P): A crash happens after sending the completion message, but before the text could be printed.
3. P →M →C: A crash occurs after sending the completion message and printing the text.
4. P→C(→M): The text printed, after which a crash occurs before the completion message could be sent.
5. C (→P →M): A crash happens before the server could do anything.
6. C (→M →P): A crash happens before the server could do anything.

Strategies to be followed

- There are four strategies
- 1. Never- client will never issue a request at the risk of text not being printed.
- 2. Always- It can reissue a request but results in printing twice.
- 3. When not ack - client decides to reissue request to server when there is no acknowledgement that the request is sent.
- 4. When ack - request is retransmitted. Another technique is the inclusion of additional bits in a retransmission to identify it as such to the server.

5 . Client crashes after sending request

- When a client crashes, and when an 'old' reply arrives, such a reply is known as an *orphan*.
- Four orphan solutions have been proposed:
  1. *extermination* (the orphan is simply killed-off).
  2. *reincarnation* (each client session has an *epoch* associated with it, making orphans easy to spot).
  3. *gentle reincarnation* (when a new epoch is identified, an attempt is made to locate a requests owner, otherwise the orphan is killed).
  4. *expiration* (if the RPC cannot be completed within a standard amount of time, it is assumed to have expired).

1. Reliable group communication: (Make points and study)

Q20. Write about reliable group communication?

Model Paper-III, Q13(a)

Answer :

The reliable multicasting services are important as they make sure that messages are delivered to each and every member that are present in a process group.

Reliable Multicasting Schemes

There are several transport layers that are capable of providing reliable point-to-point channels, but cannot provide reliable communication between a group of processes. A simple solution is to provide a point-to-point connection between each process that needs to communicate. This will lead to wastage of network bandwidth, but when there are less number of processes in a group, this does not leads an issue thus the reliability can be achieved easily.

Reliable-Multicasting corresponds to the provision of a message to each and every process present in a process group. There may be possibility that a process present is process group will crash or a new process may join the process group. Thus, a reliable communication can be distinguished as,

1. Reliable communication when faulty processes are present.
2. Reliable communication, assuming processes operate correctly.

When all the processes receive the message except those which are faulty, the multicasting is said to be a reliable multicasting. If it is assumed that the process do not crash or join or leave the process group, then the reliable multicasting will be achieved when the message is delivered to each current process in a process group.

Example

Consider a situation when a single sender multicasts a message to five different receivers. If this communication system provides an unreliable multicasting, then the solution is to assign a sequence number for each and every message that is multicasted.

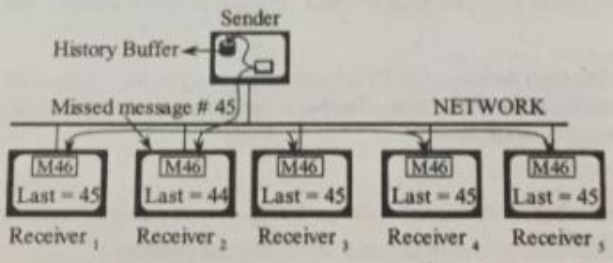


Figure: Message Transmission

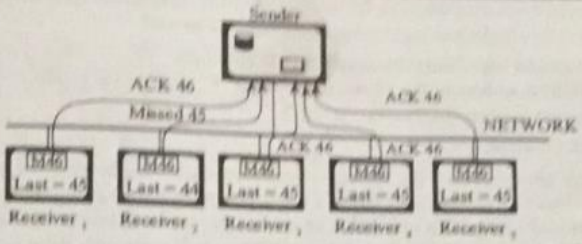


Figure: Reporting Feedback

Let us assume that the sender has multicasted messages with sequence numbers 1 to 46. Recently, it multicasted message #46, which is now stored in the history buffer of the sender. After transmitting each message, the sender receives the acknowledgment from the receivers. If any of the receivers missed any message then the sender will be informed when the acknowledgments are received from each receiver. Let receiver$_3$ miss the message #45, then the sender will know about this and retransmit this message. Thus, with this, the multicasting is made reliable.

Scalability Issue in Reliable Multicasting

When there are limited number of receivers, the above scheme is favourable to be used. But, as the number of receivers increases to N, the number of acknowledgments sent by each receiver to the sender will also increase to N. This will make the sender flooded with acknowledgements each time it sends a message. To overcome this problem, the receivers can inform the sender only if they miss any message. This will give rise to another problem as the sender will have to save each message within its history buffer, assuming it may receive any message from receiver informing that it missed a message. The two different approaches that are used to achieve scalable reliable multicasting are

1. Non-hierarchical feedback control
2. Hierarchical feedback contro

1. Non-hierarchical Feedback Control

To make a reliable multicasting to be scalable across a wide area network, the number of feedbacks must be reduced. The commonly used model for this purpose is "Feedback Suppression". This model plays a central role in achieving "Scalable Reliable Multicasting". (SRM) protocol.

In this approach, the positive acknowledgements are not forwarded to the sender. Instead, only the negative acknowledgments are forwarded as feedback to the remaining members of the group. This multicasting of feedback will suppress the feedback from the other member of the group which missed the same message. It is assumed that the sender will retransmit the missed message to all the members of the group. Thus, only a single feedback will be reached to sender when a single message is missed by more than one member of a group but that message will be retransmitted to all the members of that group. This scheme can be diagrammatically shown as:

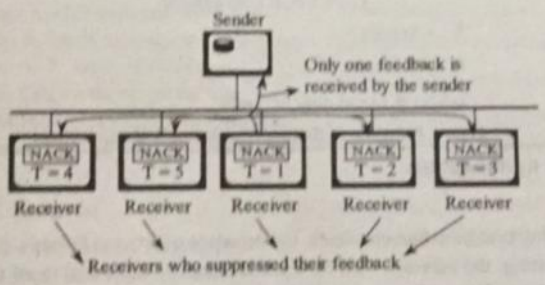


Figure: Feedback Suppression

Problems that are faced when feedback suppression scheme is used:

1. Feedback messages must be scheduled accurately at all the receivers, or else the receivers may forward the feedback at the same time. Maintaining timers for each process present in a process group which is distributed across a wide-area network is very difficult.
2. The processes that have successfully received the messages will be interrupted when the feedbacks are multicasted.

The second problem can be overcome, when all the receivers that have missed the same message, group themselves and share a multicast channel that transfers feedbacks and the retransmitted message as well.

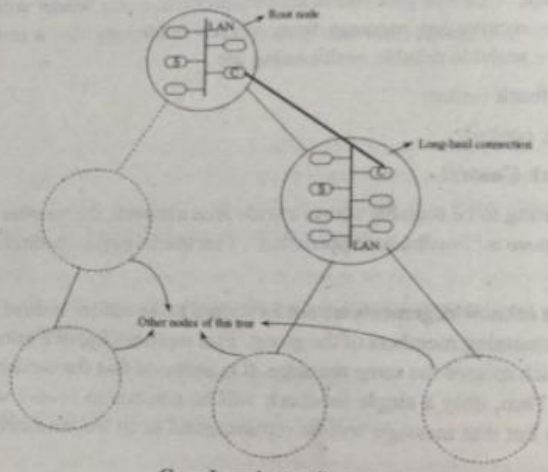
The scalability of SRM can be improved if local recovery is employed. That is, when a receiver 'A' multicasts a feedback that, it missed a message to all the members of the group, the receiver 'C' that has successfully received a message can respond to this feedback by forwarding that message to receiver 'A'.

2. Hierarchical Feedback Control

When there are very large group of receivers, the hierarchical approach must be used to scale them up. Assume that a single sender multicasts a message to numerous receivers present in a single group. This large group will be divided into several subgroups. These subgroups are then organized in the form of a tree. The root of this tree will be a subgroup that contains the sender. Each subgroup can employ any of the reliable multicasting schemes. Within each subgroup, there will be a local coordinator, which receives the message from the sender and stores it in its history buffer. The local coordinator will then multicast the message to all the members of that subgroup. If any of the members misses a message it gives a feedback to the local coordinator and not the sender. The local coordinator will then be responsible for retransmitting that message. If the local coordinator itself misses a message, then it will inform the local coordinator of its parent subgroup to retransmit that message. It also sends an acknowledgment to the local coordinator of its parent subgroup when it receives the message successfully. The history buffer present in a local coordinator will delete a message from it only if the local coordinator receives acknowledgements from all the members of its subgroup.

Problems Faced When Hierarchical Approach is Used

In hierarchical approach, a dynamic construction of tree is required. Due to which, a multicast tree which is already present in the network is being used. Here, the multicast router must be improved in such a way that it works as a local coordinator as required in this approach.



C → Local coordinator

S → Sender

R → Receiver

LAN → Local area network

Figure: Principle of Hierarchical Reliable Multicasting

21. Explain about atomic multicast.

11) What is two phase commit protocol? Explain in detail?

(Make points and study for the below answer)

Q23. Explain in detail about two-phase commit protocol.

Answer :

Model Paper-I, Q13(b)

Two-phase Commit Protocol

To overcome the drawback of one-phase commit protocol, Gray designed a new distributed commit protocol called two-phase commit protocol. As the name suggests, the working of this protocol is based on two phases.

(a) Voting phase

(b) Decision phase.

The description of each phase is given below.

(a) Voting Phase

This phase consists of two stages.

Stage 1

Suppose the participants have finished the transaction. Here, the coordinator in order to know whether the participants want to abort or commit the transaction, transmits a VOTE_REQUEST message.

Stage 2

After this message is received by all the clients, depending on the operation to be performed by each of them, they send one of the following vote messages to the coordinator in response.

- If some of the participants want to commit the transaction, they transmit VOTE_COMMIT message.
- If some of the participants want to abort a transaction, they send VOTE_ABORT message.

(b) Decision Phase

Like voting phase, this phase also comprises of two stages.

Stage 1

In this stage based on the majority of particular vote message sent by each client, coordinator transmits either of the following two messages to the clients.

- If most of the clients are interested in aborting the transaction, then the coordinator transmits GLOBAL_ABORT message.
- If most of the clients are willing to commit the transaction, coordinator sends a GLOBAL_COMMIT message.

However, an interesting point to note here is that coordinator not always send one of the above messages based on the majority of a particular vote message. Even if one participant wants to commit or abort a transaction, it still sends GLOBAL_COMMIT or GLOBAL_ABORT messages.

Stage 2

Here, the participants based on the vote message sent by them, wait for one of the above messages to be transmitted by the coordinator.

- Those clients who wanted to commit a transaction, wait for GLOBAL_COMMIT message. The moment they receive this message, the transaction is committed.
- Similarly those clients who were interested in aborting a transaction, expect GLOBAL_ABORT message from the coordinator. As they receive it, the transaction is successfully aborted.

The above two phases can be explained in the form of finite state machines. To be more precise, these machines can

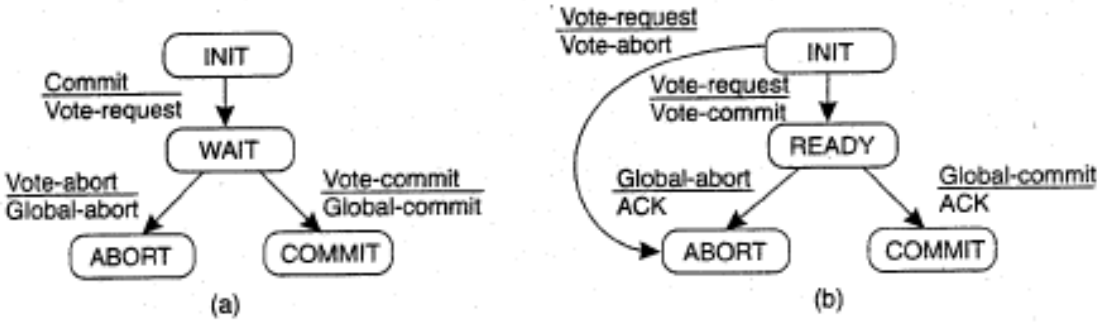


Figure 8-18. (a) The finite state machine for the coordinator in ;2PC. (b) The finite state machine for a participant..

- COMMIT state

Here, INIT state is common to both coordinator and the participants.

Finite state machine for a coordinator and participants can be explained as follows.

By default coordinator will be in INIT state. Here, the coordinator sends VOTE_REQUEST message to determine whether the participants want to commit or abort a transaction. A specific time limit will be given to the clients in which they should respond.

On the other hand, every participant will also be in INIT state waiting for the VOTE_REQUEST message from the coordinator.

There are two possibilities that might take place;

(a) If response from coordinator is not on time, clients by default will transmit a VOTE_ABORT message to the coordinator that they want to abort the problem.

(b) If the response from clients are not on time, the coordinator will enter in WAIT state wherein it will send either GLOBAL_ABORT/GLOBAL_COMMIT message and hence will be in either COMMIT or ABORT state.

Participants (client) will be in READY state now as they will be waiting for either GLOBAL_COMMIT or GLOBAL_ABORT from the coordinator. Here, if the client does not get the particular global vote message on time, it cannot directly abort the transaction. In this case one client (say A) must ask another client (say B) because there might be a possibility that coordinator might have crashed multicasting this message and also B might have received the message and not A. Hence, based on the state of (B), (A) can perform a particular operation.

For example, if A wanted to abort the transaction and it sees B in ABORT state, then it comes to know that B may have received GLOBAL_ABORT message and it aborts. Similarly, if A is interested in committing the transaction and sees B in COMMIT state, then A can successfully commit the transaction as A can guess that B could have got GLOBAL_COMMIT message.

Suppose coordinator might have crashed while transmitting a VOTE_REQUEST message and (B) is in INIT state. There is a possibility that (A) may have got the message and not (B). In this situation, (A) and (B) can enter ABORT state i.e., they can abort the transaction.

The real problem comes when all the participants are in READY state i.e., every participant waiting for a particular global vote message. In this case, until the coordinator doesn't send any global vote message, the participants cannot make a decision even though they want to either abort or commit a transaction. In other words, the protocol is blocked, till the coordinator recover from the crash. For this reason, two-phase commit protocol is also known as blocking commit protocol.

The diagrammatic representation of two finite state machines is given below.

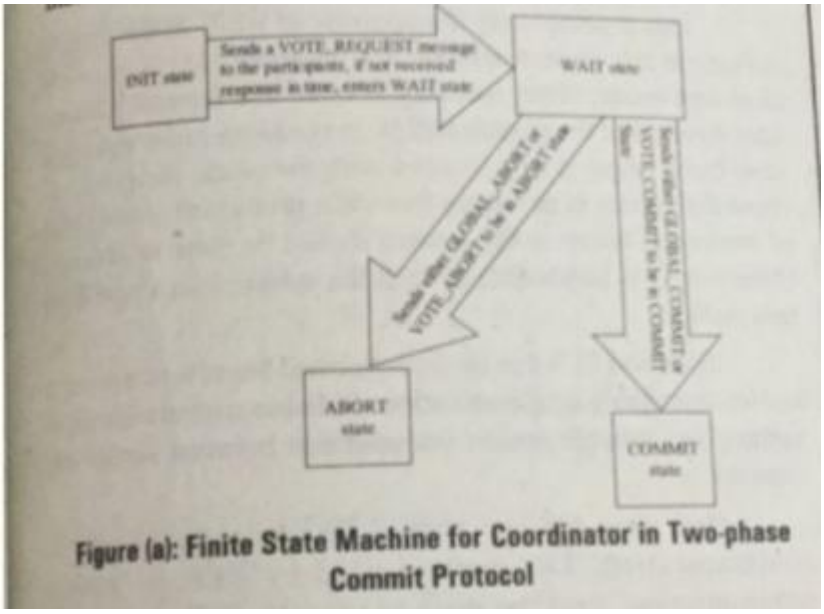


Figure (a): Finite State Machine for Coordinator in Two-phase Commit Protocol

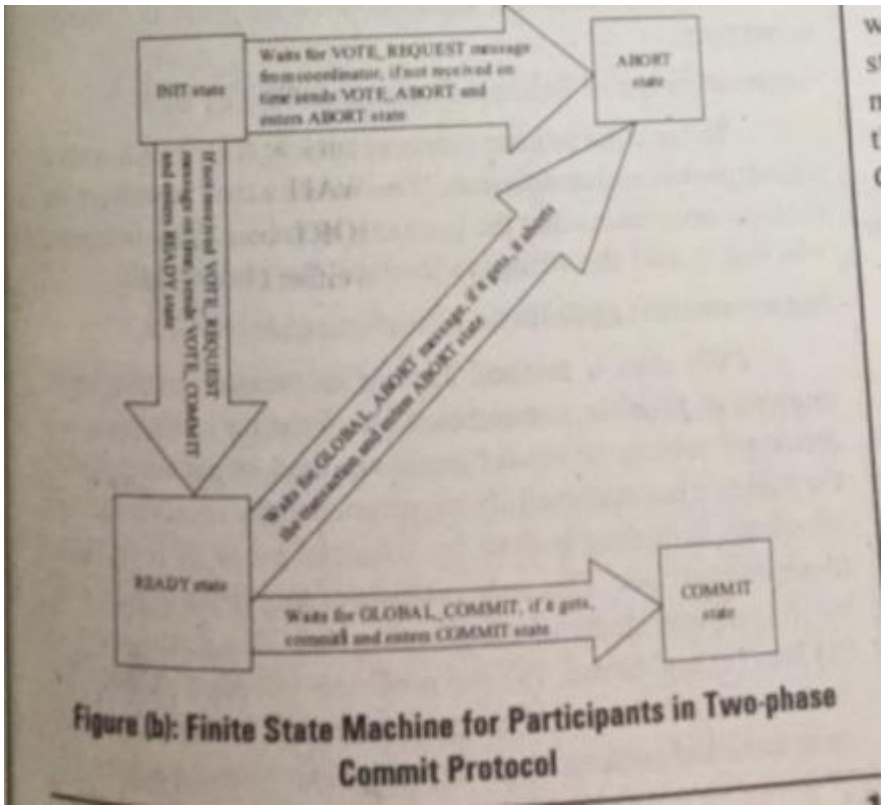
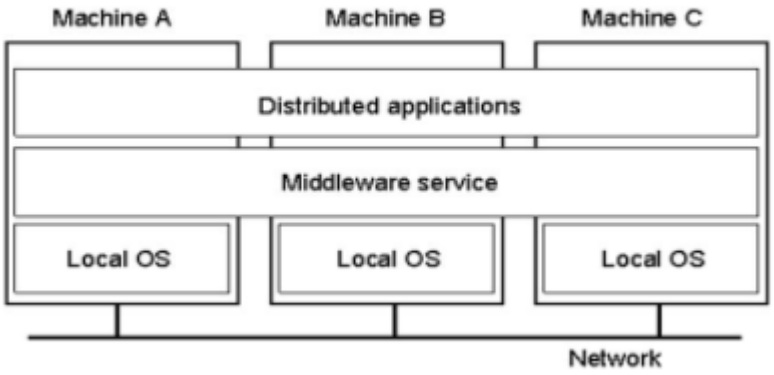
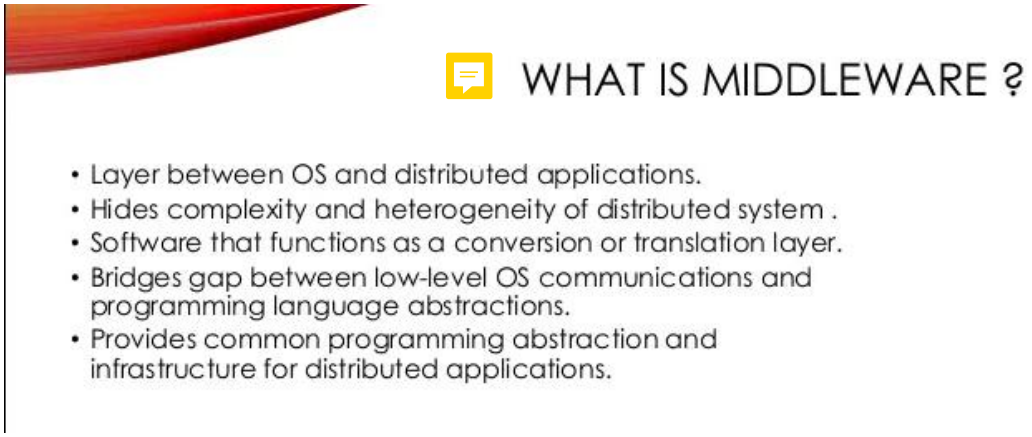


Figure (b): Finite State Machine for Participants in Two-phase Commit Protocol

12) What is middleware? What are the goals of distributed systems?

Middleware:





Goals of distributed systems:

1. Making Resources Accessible:

The main goal of a distributed system is to make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way.

Examples include things like printers, computers, storage facilities, data, files, Web pages, and networks, etc.

There are many reasons for wanting to share resources. One obvious reason is that of economics. For example, it is cheaper to let a printer be shared by several users in a small office than having to buy and maintain a separate printer for each user.

2. Transparency:

It is important for a distributed system to hide the location of its process and resource. A distributed system that can portray itself as a single system is said to be transparent.

The various transparencies need to be considered are access, location, migration, relocation, replication, concurrency, failure and persistence.

Aiming for distributed transparency should be considered along with performance issues.

3. Openness:

Openness is an important goal of distributed system in which it offers services according to standard rules that describe the syntax and semantics of those services.

Open distributed system must be flexible making it easy to configure and add new components without affecting existing components.

An open distributed system must also be extensible.

4. Scalable:

Scalability is one of the most important goals which are measured along three different dimensions.

First, a system can be scalable with respect to its size which can add more user and resources to a system.

Second, users and resources can be geographically apart.

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