Module 1

Introduction to Distributed systems

Distributed system

- Also known as distributed computing
- It is a collection of independent computers that appear to the users of the system as a single computer.
- Example, World Wide Web

Characteristics of Distributed systems

1. Resource sharing
2. Transparency
3. Openness
4. Scalability
5. Heterogeneity
6. Security
7. Fault Tolerance
8. Concurrency
9. Quality of service

Advantages of Distributed System

- All the nodes in distributed system are connected to each other. so nodes can easily share data with other nodes.
- More nodes can easily be added. i.e. it can be scaled as required.
- Failure of one node does not lead to the failure of the entire distributed system. Other nodes can still communicate with each other.

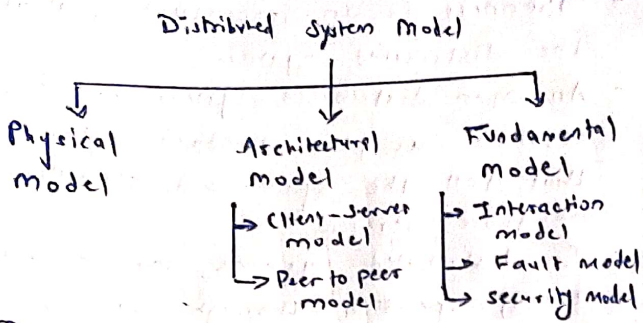
Disadvantages of Distributed system

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

Open distributed system (Openness)

- Openness is an important goal of the distributed system.
- An open distributed system provides services as per standard rules that tell the syntax and semantics of those services.
- In distributed systems, services are usually specified through interfaces, which are expressed in an IDL i.e. Interface Definition Language.
- Processes make use of interfaces to communicate with each other.
- There can be different implementations of these interfaces leading to different distributed system that function in the same manner.
- Appropriate specifications are complete and neutral. Complete specifications specify the whole thing that is essential to make an implementation.
- An open distributed system should allow configuring the system out of different components from different developers. Also, it should be trouble-free to put in new components or replace existing ones without disturbing those components that stay in place. It means an open distributed system should also be extensible. Open system interfaces are published.
- Monolithic approach should be avoided for building the system. There should be separation between policy and mechanism.
- Open distributed system provides uniform communication mechanism and it is also based on published interfaces in order to access the common resources.

Distributed System Models



(i) Physical Model

- It is a representation of the underlying hardware elements of a distributed system.
- It captures the hardware composition of a system in terms of computers and other devices and their interconnecting network.
- Physical model can be categorized in three generations of distributed systems.
  - (a) Early distributed system
  - (b) Internet-scale distributed system
  - (c) Contemporary distributed system.

(2) Architectural model

- It defines the way in which the components of the system interact with each other and the way in which they are mapped on to an underlying network of computers.
- It simplifies and abstracts the functionality of the individual components of a distributed system.
- It describes responsibilities distributed between system components and how these components are placed.
- Example,
  - (a) Client-server model
  - (b) Peer-to-peer model

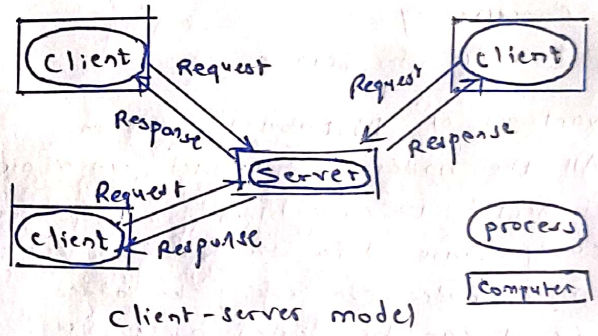
(3) Fundamental model

- The propose of fundamental model is to make explicit all relevant assumptions about the system we are modelling.
- It includes interaction, fault and security model.
  - (a) Interaction model deals with performance and are used for handling time.
  - (b) Failure model defines how the failure occurs in the system and what are its effects.
  - (c) Security model is based on establishing trust.

Architectural model

(1) Client server model

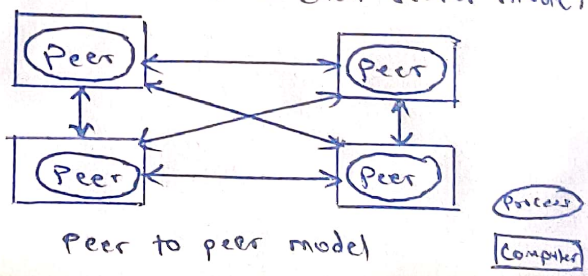
- In the basic client-server model, processes in a distributed system are divided into two groups, i.e. server and client.
- A server is a process implementing a specific service. Eg., File system service, Database service, etc.
- A client is a process that requests a service from a server by sending it a request and subsequently waiting for a server's reply.
- Client-server model is usually based on a simple request/reply protocol.
- A server can itself request services from other servers; thus, in this new relation, server itself acts like a client.



Client-server model

(2) Peer to peer model

- General idea behind this model is that there is no central control in a distributed system.
- Basic idea is that each node can either be a client or a server at a given time.
- If a node is requesting something, it can be known as a client.
- If a node is providing something, it can be known as a server.
- In general, each node is referred to as a peer.
- This is the most general & flexible model.
- In this model, any new node have to first join the network.
- After joining in, they can either request or provide a service.
- In this model, all the processes possess same capabilities and responsibilities.
- It tries to solve some problems that are associated with client-server model.



Peer to peer model

Content	Distributed OS		Network OS	Middleware OS
	Multiprocedure	Multi Computer		
Degree of Transparency	very High	High	Low	High
Same OS on all nodes	Yes	Yes	NO	NO
No. of copies of OS	One	multiple	multiple	multiple
Basis of Communication	shared memory	message	Files	Model specific
Resource management	Global, Central	Global, Distributed	Per Node	Per Node
Scalability	No	Moderate	Yes	Varies
Openness	Closed	Closed	Open	Open

Module 2
Communication

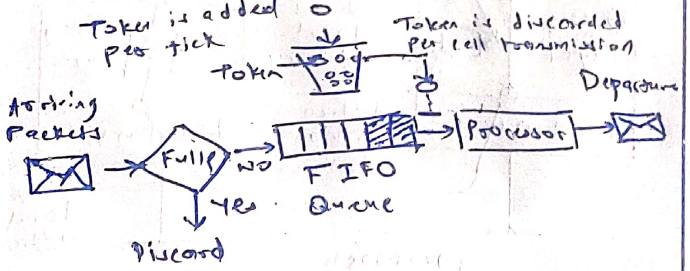
Stream - oriented Communication

- In stream oriented communication, the message content must be delivered at a certain rate as well as correctly. Ex; Music or video
- Stream - oriented communication supports continuous media communication.
- It is used by TCP.
- Data is sent by with no particular structure.
- It is reliable as data acknowledged
- Transmission speed is slow
- Connection must be established before communication.
- Suitable for applications like email where data must be persistent.

Transmission Modes

1. Asynchronous mode
   - No restriction 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 oriented communication uses token bucket algorithm as it overcomes drawbacks of leaky bucket algorithm.
- Example, Token Bucket Algorithm
- Token bucket can easily be implemented with a counter.
- The token is initialized to zero
- Each time a token is added, the counter is incremented by 1.
- Each time a unit data is dispatched, counter is decremented by 1.
- If the counter contains zero, the host cannot send any data.

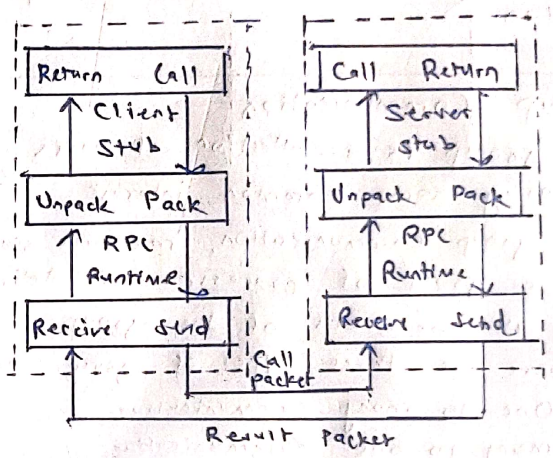


Implementation of Token Bucket.

Remote Procedure Call (RPC)

- RPC is a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network details.
- Also known as Remote Function Call
- RPC is an interprocess communication technique that is used for client server based applications.

RPC model

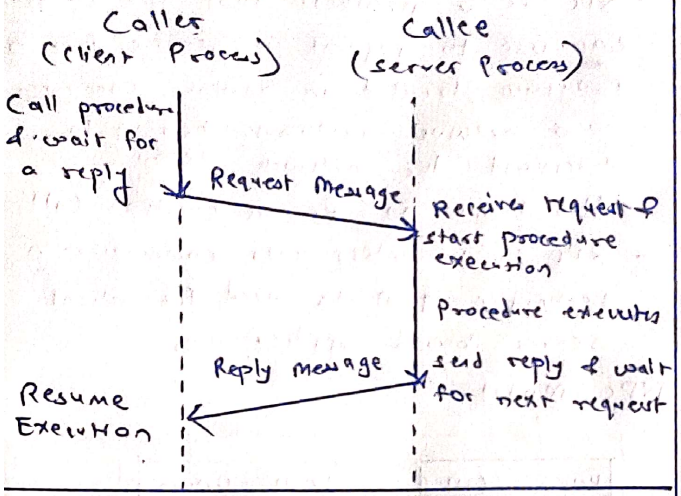


Implementation of RPC

- To achieve the goal of semantic transparency the implementation of a RPC mechanism is based on the concept of stubs.
- stubs provide a local procedure call abstraction.
- Implementation of RPC usually involves 5 elements.

1. Client - User process that initiates a remote procedure call.
2. Client stub - It is responsible for
   a. On receipt of a call request from the client, it packs a specification of target procedure & arguments into message.
   b. On receipt of the result of procedure execution, it unpacks the result & passes it to the client.
3. RPC Runtime - It handles transmission of message across the network between client & server including encryption, routing acknowledgement.
4. Server stub - IIS job is similar to client stub.
5. Server - Server executes appropriate procedure and returns the result.

Working of RPC



Group Communication

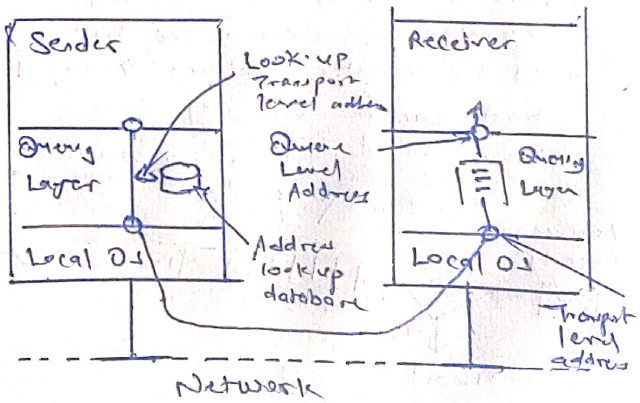
- A group is a collection of users sharing some common interest.
- In group communication, message sent to a group of processes will deliver to all members of the group.
- There are three types of group comm.
  1. One to many communication
  2. many to one communication
  3. many to many communication

1. One to many
   - There exists single sender & multiple receivers.
   - Also called as multicast communication.
   - There exists a special case of multicast communication in which message is sent to all processors connected to the network
2. Many to one communication
   - There is only one receiver at any given time
   - The receivers can be selective or non selective
   - a. Selective receiver
     - Specifies the unique sender.
     - A message exchange takes place only if that sender sends a message
   - b. Non selective receiver
     - The receiver is ready to accept a message from any sender that who is present in the system
3. Many to many communication
   - In this, multiple senders send message to multiple receivers.
   - An important issue with many to many communication is ordered message delivery
   - It includes
     - a. No ordering
     - b. Absolute ordering
     - c. Consistent Ordering
     - d. Casual Ordering

Message Queuing System

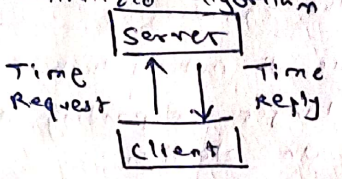
- In message queuing system, messages are forwarded through many communication servers. Each application has its own private queue to which other application sends the message. In this system, guarantee is given to sender that its sent message will be delivered to recipient queue. message can be delivered at any time.
- 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 local queue. The message put in queue for transmission contains specification of destination queue. Message queuing system provides queues to senders & receivers.
- message queuing system keeps mapping of queue names to network locations. A queue manager manages queues and interacts with application which sends and receives message. Special queue manager work as router or relay which forwards the message to other queue managers.
- Each queue manager should have copy of queue-to-location mapping. For larger size message queuing system this approach leads to network management problem. To solve this problem, only routers need to be updated after adding or removal of the queues.

General Architecture of message queuing system

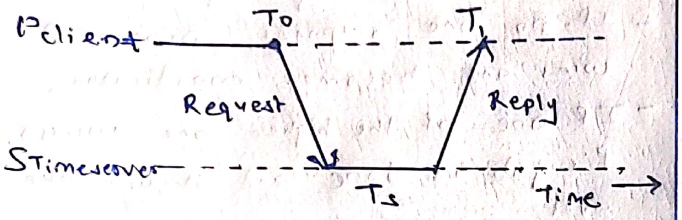


Module 3
Synchronization

Cristian's Algorithm
- In cristian algorithm, client approaches server.
- It is centralized algorithm



$T_{new} = T_{server} + \frac{T_1 - T_0}{2}$



Algorithm:
- Let S be the time server and T_s be its time.
- Process P requests the time from S.
- After receiving the request from P, S prepares response and append the time T_s from its own clock and then send it back to P.

Example:

Send Request → 8:08:18:100 (T_0)

Receive Response → 8:08:18:900 (T_1)

Response → 8:09:28:300 (T_{server})

$\therefore T_{NEW} = T_{server} + \frac{T_1 - T_0}{2}$

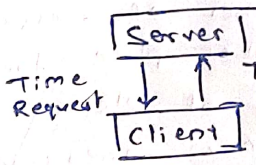
$= 8:09:28:300 + \frac{900 - 100}{2}$

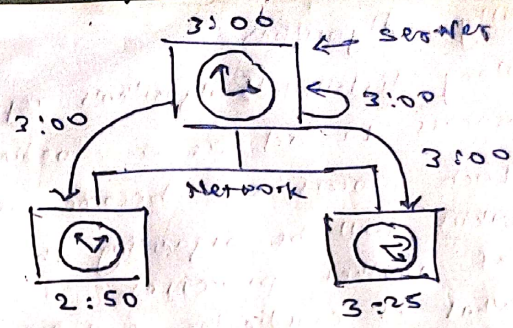
$= 8:09:28:300 + \frac{800}{2}$

$= 8:09:28:300 + 400$

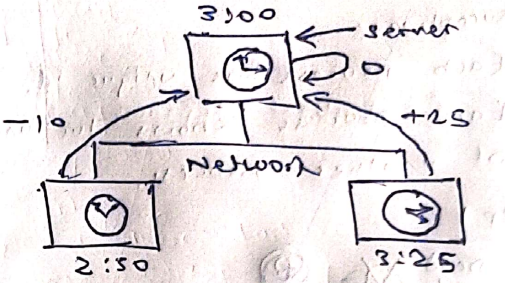
$= 8:09:28:700$

Berkeley Algorithm
- In Berkeley algorithm, server approaches client.
- It is centralized algorithm.

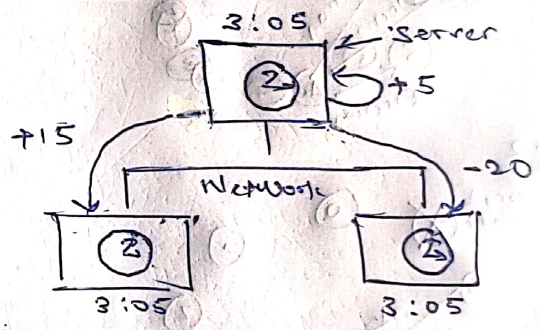




(a) server asks all the other machines for their clock values.



(b) machines (clients) answer



(c) Time server tells everyone how to adjust their clock.

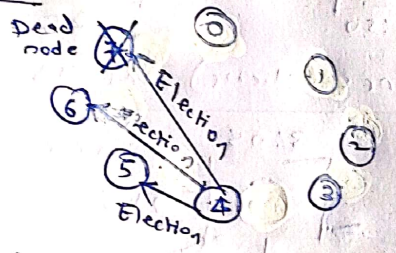
Algorithm:
- It is a method of clock synchronization in distributed computing which assumes no machine has an accurate time source.
- It is an active time server approach.
- In this approach, the time server periodically sends a message to all the computers in a group of computers.
- When this message is received each computer send backs its own clock value to the time server.
- Time server has prior knowledge of readjusting clock values. Time server takes average clock value of all computers.
- The calculated average is the current time to which all clocks should be readjusted.
- Time server itself readjusts its own clock to this average value. Instead of sending current time, it sends amount of time each computer needs to readjust their clock. This can be positive or negative.

Bully Algorithm

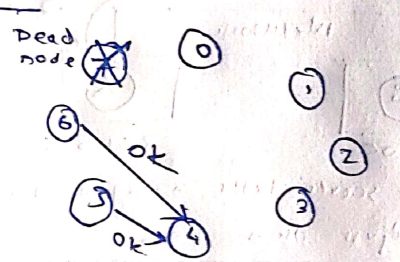
- Many distributed algorithms require a process to act as a coordinator.
- Each process can become coordinator which will be organizing the actions of another processes.
- What if coordinator fails? How the new coordinator be elected?

Assumptions:
- Each process has unique id
- Each process knows the uniqueid of another process.

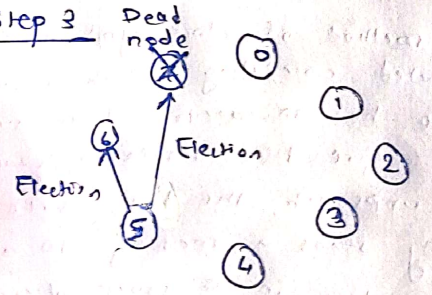
Step 1



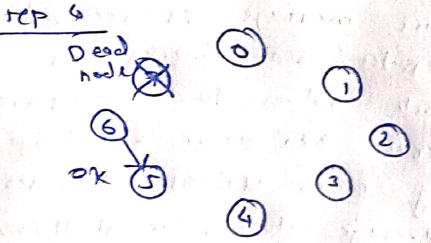
Step 2



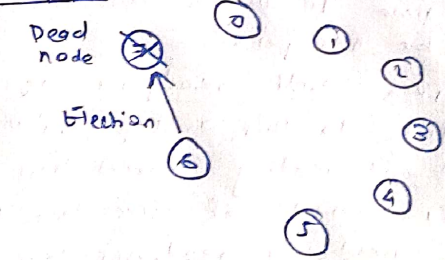
Step 3



Step 4

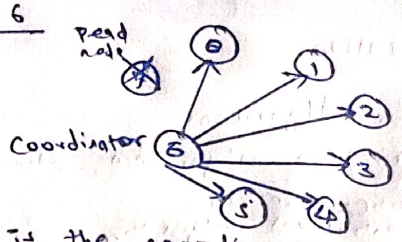


Step 5



Here, node 6 does not receive any acknowledgement from node 7. So it declares itself as the coordinator

Step 6



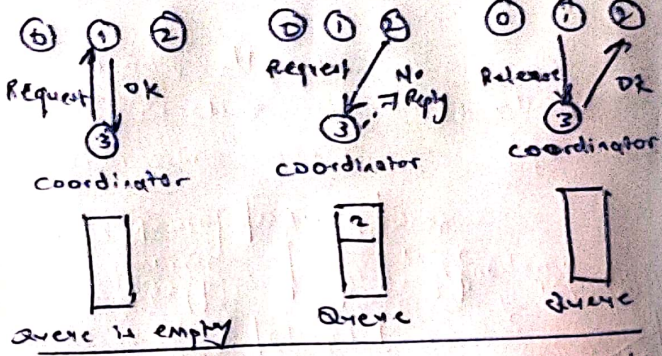
∴ 6 is the coordinator

- A process P notices that coordinator is no longer responding, it will initiate an election.
- It will send election to all other process with higher process id.
- If no one responds, process P wins and becomes coordinator.
- If one of the higher one answers, P's job is done and it will take over.
- When process P gets a message from one of its lower id, receiver sends an ok message as ACK to sender that it will take over and P's job is done.
- Eventually all process will give up apart from one, that one is the coordinator.
- The coordinator finally wins and announces its victory by sending coordinator message to all the process.

Advantages
- It is a distributed method with simple implementation.
- Only the processes with higher priority number respect to priority number of process that detects the crash coordinator will be involved in an election, not all processes are involved.

Disadvantages
- It takes number of stages to decide the new coordinator.
- Huge number of message exchange takes place due to broadcasting of election and acknowledgement message.

Centralized Approach for mutual exclusion



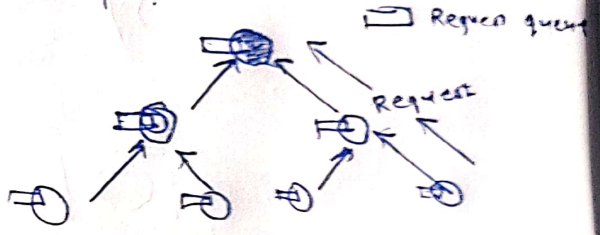
- In Centralized algorithm, one process is elected as the coordinator.
- Coordinator may be the machine with highest network address.
- Whenever a process wants to enter a critical section, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission.
- Suppose another process asks for permission to enter the same critical section.
- Now the coordinator knows that a process is already in the critical section, so it cannot grant permission.
- The coordinator just refrains from replying, thus blocking message 2, which is waiting for a reply or it could send a reply saying 'permission denied'.
- When process 1 exits the critical section it sends a message to coordinator releasing its exclusive access.
- The coordinator takes first item off the queue of deferred requests and sends that process a grant message.
- If the process was still blocked it unblocks and enters the critical section.

Advantages
- Easy to implement.
- No starvation problem

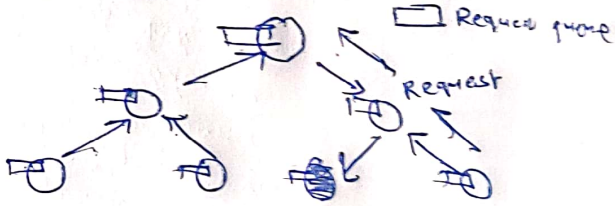
Disadvantages
- If coordinator crashes, entire system will fail.

Raymond's tree based algorithm

- It is a token based algorithm.
- In raymond's tree based algorithm, a direct tree is formed.
- Nodes are arranged as a tree
- Every node has a parent pointer.
- Each node has a FIFO queue of requests.
- There is one token with root. Only owner of token can enter the critical section.
- Example:



- As the token moves across nodes, the parent pointers change.
- They always point towards the holder of the token.
- It is thus possible to reach the token by following parent pointer.



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

Releasing a token
- Ultimately a request will reach the token holder.
- Token holder will wait till it is done with critical section.
- It will forward the token to the node at the head of its request queue
- After forwarding, a process needs to make a fresh request for token, if it has outstanding entries in its request queue.

Module 4

Resource and Process Management

Load Balancing Approach

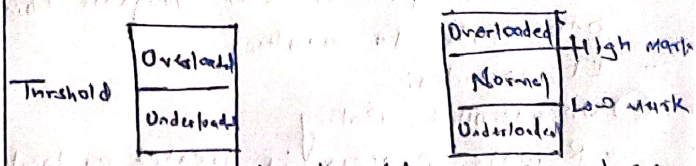
- It is used in process management
- Scheduling algorithm that uses load balancing approach are called as load balancing algorithms.
- The main goal of load balancing algorithms is to balance the workload on all the nodes of the system.
- Load balancing aims to:
  1. Optimize resource use
  2. Maximize throughput
  3. Minimize response time
  4. Avoid overload of any single resource

Designing issues

- Designing a load balancer is a critical task. It includes,

1. Load Estimation Policy
   - First issue in load balancing is to decide which method to use to estimate the workload of a particular node.
   - Estimation of the workload of a particular node is a difficult problem for which no completely satisfactory solution exists.
   - A nodes workload can be estimated based on
     a. Total no. of processes on the node
     b. Resource demands of these processes
     c. Instruction mixes of these processes
     d. Architecture & speed of node's processor

2. Process Transfer Policy
   - The idea of using this policy is to transfer processes from heavily loaded nodes to lightly loaded nodes.
   - Most of the algorithms use the threshold policy to decide whether the node is lightly loaded or heavily loaded.
   - Threshold value is a limiting value of a workload & can be determined by
     a. static Policy - Each node has a predefined threshold value.
     b. Dynamic Policy - Threshold value for a node is dynamically decided.
   - Below threshold value, node accepts processes to execute.
   - Above threshold value, node tries to transfer processes to a lightly loaded node
   - To reduce instability, double threshold policy has been proposed. Also known as high low policy



(a) Single threshold policy (b) Double Threshold Policy.

3. Location Policy
   - 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 load.
   - Location policy determines this destination node.

4. State Information Exchange Policy
   - It is used for frequent exchange of state information within the nodes
   - It includes:
     a. Periodic Broadcast
        - Each node broadcasts its state info after elapse of T units of time
     b. Broadcast when state changes
        - Each node broadcasts its state info when a process arrives or departures
     c. On demand exchanges
        - Each node broadcasts its state info when its state switches from normal to either underload or Overload
     d. Exchanges by polling
        - The partner node is searched by polling the other nodes one by one until poll limit is reached.

5. Priority assignment policy
   - The priority of the execution of local and remote processes at a particular node should be planned
   - It includes:
     a. Selfish priority assignment rule
        - Local process are given higher priority than remote process
     b. Altruistic priority assignment rule
        - Remote processes are given higher priority than local process.
     c. Intermediate priority assignment rule.
        - Priority is decided depending upon number of local & remote process on a node.

⑥ Migration Limiting Policy

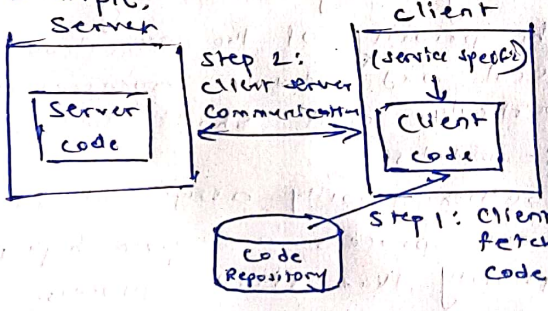
- Policy determines the total no. of times a process can migrate from one node to another.
- It includes:

ⓐ Uncontrolled Policy
- Remote process is treated as local process. Hence, it can be migrated any number of times.

ⓑ Controlled Policy
- Migration count is used for remote processes.

Code Migration

- In process migration, an entire process has to be moved from one machine to another. But this may be a crucial task.
- In some cases, code needs to be migrated rather than the process.
- Code migration refers to transfer of program from one node to another.
- For eg, Consider a client server system, in which a server has to manage a big database. The client application has to perform many database operations. In such situation, it is better to move part of the client application to the server and the result is send across the network. Thus code migration is better option.
- Code migration is used to improve overall performance of the system by exploiting pallellism.
- Example,



- Advantage of this approach is that, the client does not need to install all the required software. software can be moved in as when necessary and discarded when it is not needed.

Need for Code migration

① Performance is improved by moving code from heavily loaded machines to lightly loaded machine.

② No need to install heavy softwares softwares can be moved in or discard as needed.

Code Migration Issues

- Communication in distributed system is concerned with exchanging data between processes.
- Code migration deals with moving programs between machines with the intention to have those programs executed at target.
- In code migration framework, a process consists of 3 segments

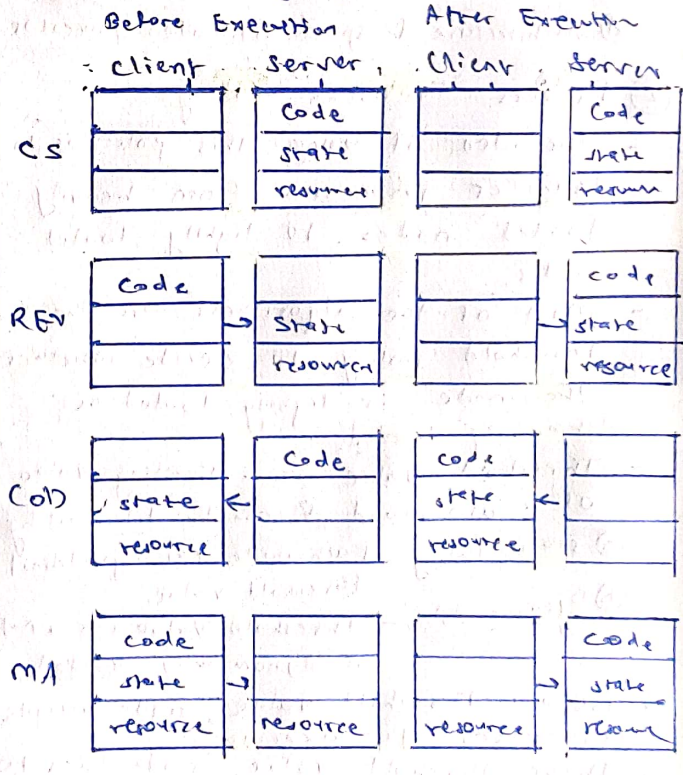
① Code segment
- Contains actual code

② Resource segment
- Contains references to resources needed by the process.

③ Execution segment
- Stores the execution state of a process

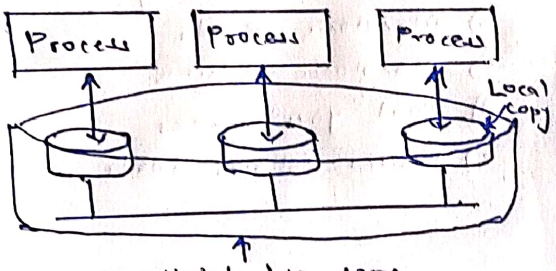
- Code migration

ⓐ CS: Client-Server
ⓑ REV: Remote Evaluation
ⓒ COD: Code on Demand
ⓓ MA: Mobile Agents



Module 5
Consistency, Replication and Fault Tolerance

Data Centric Consistency Models
- It aims to provide system wide consistent view of the data store.
- A data store may be physically distributed across multiple machines.
- Each process that can access data from the store is assumed to have a local copy available of the entire system.



. Example of data centric consistency

① Strict consistency model
- Any read on a data item x returns a value corresponding to the result of the most recent write on X.
- This is the strongest form of memory coherence which has the most stringest consistency requirement.

P1: W(x)a	
P2:	R(x)a

(a)

A strictly consistent store

P1: W(x)a	
P2:	R(x)NIL R(x)a

(b)

A store that is not consistent

② Sequential consistency
- It is an important data centric consistency model which is slightly weaker consistency model than strict consistency
- A data store is said to be sequentially consistent if the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and operations of each individual process should appear in this sequence in a specified order.

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

(a)

A sequentially consistent data store

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

(b)

A data store that is not sequentially consistent.

③ Linearizability
- It is weaker than strict consistency but stronger than sequential consistency
- A data store is said to be linearizable when each operation is timestamped and the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order.

④ Causal Consistency
- It is a weaker model than sequential consistency
- In causal consistency all processes see only those memory reference operations in the correct order that are potentially causally related.
- Memory reference operations which are not related may be seen by different processes in different order.

⑤ FIFO Consistency
- It is weaker than causal consistency
- This model ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed like a single process in a pipeline.
- This model is simple and easy to implement having good performance because processes are ready in the pipeline.
- Implementation is done by sequencing write operations performed at each node independently of 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

⑥ Weak Consistency
- Basic idea is enforcing consistency on a group of memory reference operations rather than individual operations.
- A distributed shared memory system that supports weak consistency model uses a special variable which is used to synchronize memory

Data Centric Consistency

1. It is used for all clients in a system
2. It does not have lack of simultaneous updates.
3. It is data specific
4. Globally accessible
5. It aims to provide system wide consistent view on a database
6. Examples:
   a) Strict consistency model
   b) Sequential consistency
   c) Linearizability
   d) Causal consistency
   e) FIFO consistency
   f) Weak consistency
   g) Release consistency
   h) Entry consistency

Client Centric Consistency

1. It is used for individual client
2. It has lack of simultaneous update
3. It is client specific.
4. Not globally accessible
5. It aims to provide client specific consistent view on a database.
6. Examples
   a) Eventual consistency
   b) Monotonic Reads
   c) Monotonic Writes
   d) Read your Writes
   e) Writes follows reads

Replication

- In replication, replica is created at server, whereas cached copy is associated with clients.
- A replica is more persistent as compared to cached copy.
- Replica is widely known, secure, accurate, available and complete.

Advantages of Replication

1. Increased availability
   - Replication offers availability of the system. Although failure occurs, the system remains operational and available to users.
   - The critical data can be replicated on several servers.
2. Increased reliability
   - As several copies of files are available on different servers, recovery in case of failure is possible. Hence, replication offers reliability.
3. Improved response time
   - Replication allows data to be accessed locally or from the node whose access time is less than that of primary copy access time.
4. Reduced network traffic
   - If replica of file is available with file server that resides on client machine then client access request is serviced locally. Hence, it results in reduced network traffic.
5. Improved system throughput
   - As different client's requests are serviced by different servers in parallel, it results in improved throughput.
6. Better scalability
   - A file is replicated at multiple file servers then all the client's requests for that file would not arrive at one file server. In this way, load gets distributed and different client's requests are serviced by different servers. It improves scalability
7. Autonomous operation.
   - All the file needed by clients for limited time period can be replicated on file server that resides on client machine. This ensures temporary autonomous operation of client node.

Fault Tolerance

- A system fails when it does not match its promise.
- An error in a system can lead to failure.
- The cause of an error is called a fault.
- Faults are generally transient, intermittent, or permanent.
- The ability of a system to continue functioning in the event of partial failure is known as fault tolerance.
- Fault tolerance is an important issue in designing a distributed file system.
- There are various types of fault which harms the integrity of a system.
- For example, If the processor loses the content of its main memory in the event of a crash it leads to logically complete but physically incomplete operation, making the data inconsistent.
- Decay of disk storage services may result in the loss or corruption of data stored by a file system.
- Decay is when the portion of device is irretrievable.

Types of fault

① Omission Fault

- When a process fails to do something that is expected to it is termed an omission failure.
- Two types of omission fault
  - (a) Process omission failure
  - (b) Communication omission failure

② Arbitrary fault

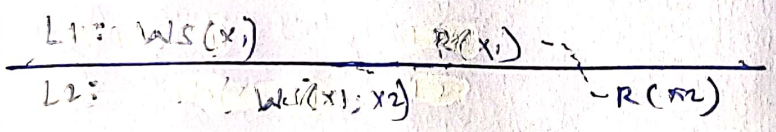
- In this failure, the responding process may return wrong values or it may set wrong value in data item.
- Examples of these failures:
  - (a) message content may change
  - (b) Repeated delivery of the same message
  - (c) Non-existent message may be delivered

③ Timing Fault

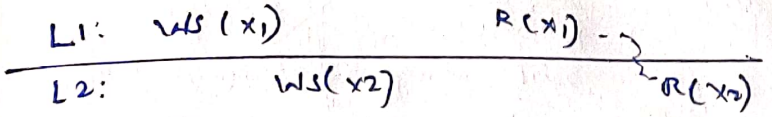
- In timing failure, clock failure affects process as its local clock may drift from perfect time.

Monotonic Reads

- A data store is said to offer monotonic read consistency if:
- "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".
- Consider a distributed email database.
- It has distributed and replicated user mailboxes.
- Emails can be inserted at any location.
- However, updates are propogated in a lazy fashion.
- Suppose the end user reads his mail in Mumbai. Assume that only reading mail does not affect the mailbox.
- Later when the end user moves to Pune and opens his mailbox again, monotonic read consistency assurances that the message that were in the mailbox in Mumbai will also be in the mailbox when it is opened in Pune.
- Example, read operation performed by a single process "P" at two different local copies of the same data store



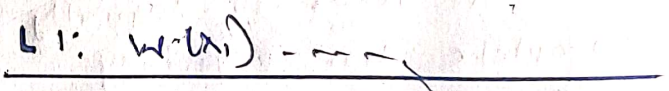
(a) Demonstrate a monotonic-read consistent data store



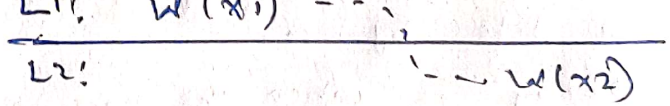
(b) Demonstrate a data store that does not provide monotic reads

Monotonic Writes

- A data store is said to offer monotonic write consistency if:
- "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".
- Consider a software library exam
- In several cases, updating a library is accomplished by replacing one or more functions
- With monotonic write consistency, assurance is given that if an update is performed on the copy of a library, all previous or earlier updates will be accomplished primarily
- Example, Write operation performed by single process P at two different local copies of the same data store



(a) Demonstrate a monotonic write consistent data store



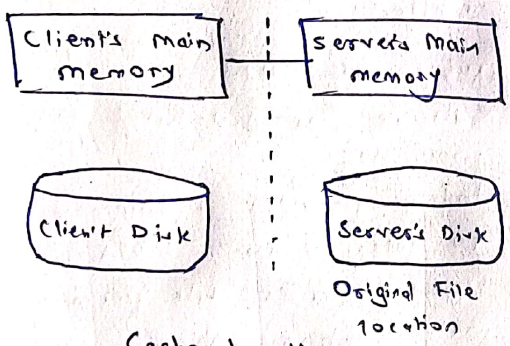
(b) Demonstrate a data store that does not provide monotonic write consistency

File Caching Schemes

- 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.
- Main goal of file caching system is to retain recently accessed data in main memory so that repeated access to the same information can be efficiently handled.
- Three design issues are involved in file caching schemes
  1. Cache location
  2. Modification propagation
  3. Cache validation schemes.

1. Cache location

- Refers to the place where cached data is stored.
- There exists three possible cache locations in servers DFS.



Cache location.

a. Server's main memory
b. Client's main memory
c. Client's Disk

2. Modification propagation

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

3. Cache Validation Schemes

- A file data resides in the cache of multiple nodes.
- It becomes necessary to check consistency in cached data.
- If cached data is not consistent then it must be updated to latest version.

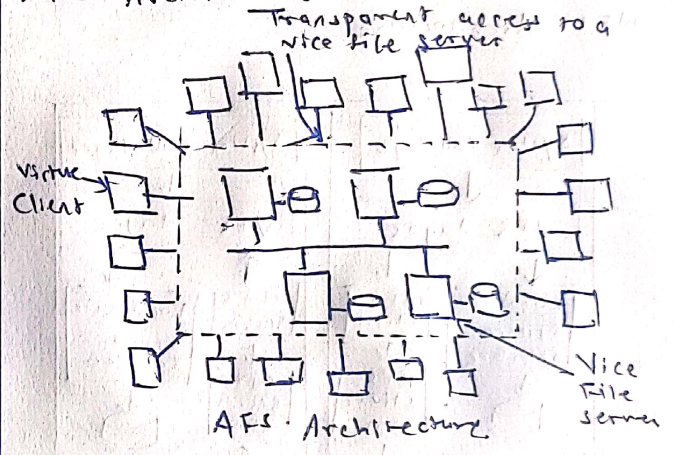
Cache validation is done in 2 ways.
1. Client initiated approach
2. Server initiated approach

Andrew File System (AFS)

- AFS is a distributed file system.
- It was developed by Carnegie Mellon University as part of the Andrew project.
- Originally named "Vice", AFS is named after Andrew Carnegie & Andrew Mellon.
- AFS supports information sharing at large scale

Features:
1. Handles Terabytes of data.
2. Handles Thousands of users.
3. Working in WAN environment.

AFS Architecture



AFS Architecture

- AFS is divided into 2 types of nodes
  1. Vice node – It is dedicated to file server.
  2. Virtue node – It is client machines.
- In VFS, an entire file is copied to the virtue node (local machine) from the vice node (server) when opened
- When the file is changed – It is copied to the server when closed.

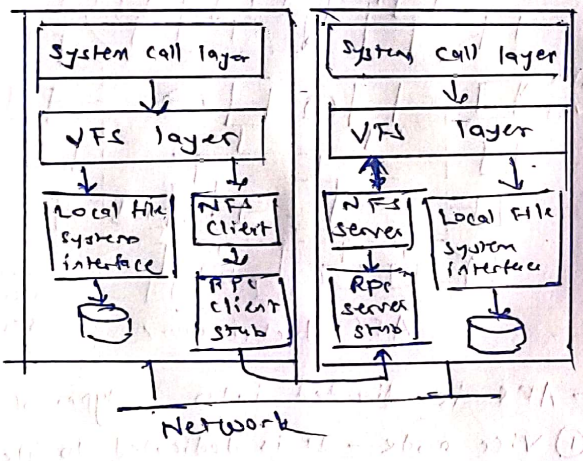
Limitations of AFS

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

Network File System (NFS)

- NFS is a platform independent remote file system technology created by Sun microsystems in 1984.
- It is client-server application that provides shared file storage for clients across a network.
- It was designed to simplify the sharing of file systems resources in a network of non-homogeneous machines.
- It was implemented using the RPC protocol & files are available through the network via VFS (virtual file system).
- Allows an application to access files on remote hosts in same way it access local files.

NFS Architecture



NFS servers: Computers that shares files

- During late 1980s & 1990s, a common configuration was to configure a powerful workstation with lots of local disk & often without graphical display to be a NFS server.
- "Thin" diskless workstations would then mount the remote file systems provided by NFS servers and transparently use them as if they were local.

NFS simplifies management.

- Instead of duplicating common directories, NFS provides a single copy of directly that is shared by all systems of the network.
- Simplify backup procedure - Instead of setting up a backup for local disk, NFS takes backup of server's disk.

Goals of NFS design

1. Compatability
2. Easy deployable
3. Machine and os independence
4. Efficiency

Hadoop Distributed File System (HDFS)

- It was developed using distributed file system design.
- It runs on commodity hardware
- HDFS is highly fault tolerant & designed using low cost hardware.
- HDFS holds very large amount of data & provides easier access.
- To store such huge data, files are stored across multiple machines.
- Files are stored in redundant fashion to rescue the system from possible data losses in case of failure.
- HDFS also makes applications available to parallel processing.
- HDFS follows a master-slave architecture.

Features of HDFS.

1. Suitable for distributed storage and processing
2. Hadoop provides a command interface to interact with HDFS.
3. HDFS provides file permission & authentication.
4. Streaming access to file system data