CBCS SCHEME

USN | | | | | | | | | | |

BCS515D

Fifth Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026
Distributed Systems

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Justify how concurrency , No global clock , Independent failures become the consequences of Distributed system.	6	L3	CO1
	b.	Explain the challenges in designing a scalable distributed system and techniques for dealing failures.	10	L2	CO1
	c.	What is the prime motivation for constructing and using distributed system? Describe it.	4	L2	CO1
		OR			
Q.2	a.	Illustrate the request – reply protocol with an example.	6	L3	CO2
	b.	What is Remote Procedure call and explain the design issues of Remote Procedure Call (RPC).	8	L2	CO2
	c.	With the help of a diagram, explain the implementation of Remote Method Invocation (RMI).	6	L2	CO2
		Module – 2			
Q.3	a.	Give the abstract model of basic Distributed file systems and structure of the file attribute record.	4	L2	CO3
	b.	Analyse the key requirements of distributed file system.	8	L3	CO3
	c.	Examine the case studies of two distributed file system of SUN NFS and Andrew File System.	8	L3	CO3
		OR			
Q.4	a.	Explain the concepts of : i) Uniform Resource Identifier ii) Uniform Resource Locators in the context of Name Services.	6	L2	CO3
	b.	Compare flat naming and structured naming schemes in distributed system.	5	L2	CO3
	c.	What is Navigation in the context of Name Servers? Explain the types of navigation used in Name servers.	9	L2	CO3
		Module – 3			
Q.5	a.	Discuss the following : i) Clock skew ii) Clock Drift iii) Co-ordinated Universal Time	6	L2	CO4
	b.	Explain the techniques for synchronization of physical clocks.	14	L3	CO4

		OR			
Q.6	**a.**	Compare physical and logical clocks in distributed system.	**4**	**L2**	**CO4**
	b.	Define Logical clocks and explain the Lamport timestamp algorithm with an example diagram.	**10**	**L2**	**CO4**
	c.	Describe the role of global states in debugging distributed systems.	**6**	**L2**	**CO4**
		Module – 4			
Q.7	**a.**	Discuss the following algorithms for mutual exclusion in Distributed System : i) Central Server algorithm ii) Ring – based algorithm iii) Multicast and Logical clocks	**12**	**L3**	**CO5**
	b.	Narrate the significance of fault tolerance in consensus algorithms.	**4**	**L2**	**CO5**
	c.	Discuss the challenges in achieving consensus in distributed system.	**4**	**L2**	**CO5**
		OR			
Q.8	**a.**	What is an Election algorithm? What are its requirements?	**10**	**L1**	**CO5**
	b.	Discuss how distributed systems handle failure during elections using an example.	**10**	**L2**	**CO5**
		Module – 5			
Q.9	**a.**	Differentiate between flat and nested transactions with an example diagram.	**8**	**L3**	**CO6**
	b.	Explain the two – phase commit protocol in distributed transactions.	**12**	**L2**	**CO6**
		OR			
Q.10	**a.**	Discuss the various methods of concurrency control in distributed transactions.	**10**	**L2**	**CO6**
	b.	Explain the following approaches used in the file recovery in distributed system : i) Logging ii) Shadow version	**10**	**L2**	**CO6**

* * * * *

Module - 1

Q.1 a. Justify how concurrency, no global clock, and independent failures become the consequences of Distributed Systems. (6 Marks)

The design of a distributed system naturally leads to three unavoidable consequences:

1. Concurrency: In a distributed system, multiple processes execute on different computers simultaneously.
    - Because these processes share resources (like databases or files), they must be coordinated to ensure data consistency.
    - Unlike a single-processor system, where the OS controls everything, here, coordination happens over a network, making simultaneous access a constant challenge.
2. No Global Clock: There is no single "master clock" that all computers in the system can read.
    - Each computer has its own internal crystal oscillator, which "drifts" at a different rate.
    - This makes it difficult to determine the exact order of events (e.g., who sent a message first?) without using specialized logical clock algorithms.
3. Independent Failures: Components in a distributed system fail independently.
    - A network link may fail, or one server may crash while others remain healthy.
    - The system must be designed to continue functioning despite these partial failures, as other components often do not immediately detect a specific node's failure.

Q.1 b. Explain the challenges in designing a scalable distributed system and techniques for dealing with failures. (10 Marks)

1. Challenges in Scalability

Scalability is the ability of a system to remain effective as the number of resources and users increases significantly.

- Controlling the Cost of Physical Resources: As demand grows, the cost of adding new resources should increase proportionally. If n users require n^2 resources, the system is not scalable.
- Controlling Performance Loss: As the system size increases, the response time should not degrade significantly. Centralized algorithms or data structures (like a single central directory) must be avoided to prevent bottlenecks.
- Preventing Software Bottlenecks: Many systems fail to scale because of software limits, such as using 32-bit IP addresses which eventually run out.
- Transparency: As a system scales, it should continue to appear as a single unit to the user, even if data is moved or replicated across new nodes.

2. Techniques for Dealing with Failures

Reliability is achieved through several key strategies:

- Failure Detection: Using "heartbeat" messages or checksums to identify when a component has stopped working or data has been corrupted.
- Failure Masking: Hiding failures from the user. For example, if a message is lost, the system can automatically retransmit it.
- Failure Tolerance: Designing software that can "gracefully degrade." If a search engine's index server fails, it might still return partial results rather than failing completely[31].
- Redundancy: Maintaining multiple copies of data or services. If one replica fails, another can take over immediately[33].

Q.1 c. What is the prime motivation for constructing and using distributed systems? Describe it. (4 Marks)
The fundamental motivation is Resource Sharing.

- Sharing Hardware and Data: Users seek to share expensive hardware (e.g., high-end printers) and, more importantly, data (e.g., a shared database or a web page).
- Collaboration: Distributed systems allow people to work together on the same data from different geographic locations.
- Performance/Speed: By distributing tasks across many computers, complex problems can be solved faster than on a single machine.
- Reliability: If one machine fails, the data remains accessible from another machine, which is a major motivation for businesses.

Q.2 a. Illustrate the request-reply protocol with an example. (6 Marks)

The Request-Reply protocol is designed to support client-server communication in a distributed system, typically providing a synchronous message exchange.

- Communication Pattern:
    1. Request: The client sends a message to the server containing the operation to be performed (e.g., read, write) and the necessary arguments.
    2. Reply: The server processes the request and returns a message indicating the operation result or an error code.
- Key Primitives: It often uses primitives such as doOperation (client-side) and getRequest and sendReply (server-side).
- Example: A DNS Query. When your browser needs the IP address for a domain:
    1. Request: The browser (client) sends a "Request" message, including the domain name, to a DNS server.
    2. Reply: The DNS server looks up the address in its database and sends a "Reply" message containing the IP address back to the browser.

Q.2 b. What is Remote Procedure Call (RPC) and explain the design issues of RPC. (8 Marks)

Remote Procedure Call (RPC) is a mechanism that enables a program to execute a procedure on a different computer (server) as if it were a local call within the client's address space. It abstracts the network details from the programmer.

Design Issues of RPC:

- Interface Definition: Since the client and server may be written in different languages, a common Interface Definition Language (IDL) is required to define the procedures and data types.
- Marshalling and Unmarshalling: The process of packing arguments into a message format suitable for transmission (marshalling) and unpacking them at the receiver (unmarshalling). It must handle differences in data representation, such as "Big-endian" vs "Little-endian".
- Binding: The client must be able to locate the server. This involves "Service Discovery," in which the server registers its service with a binder/directory, and the client queries the directory to locate the server.
- Failure Semantics: Unlike local calls, RPC can fail due to network issues or server crashes. Designers must choose a delivery guarantee:
    - Maybe: The request is sent, but there is no guarantee of execution.
    - At least once: The request is retried until a reply is received, which may result in duplicate execution.
    - At-most-once: The system ensures that the procedure is executed at most once, avoiding duplicate side effects.

Q.2 c. With the help of a diagram, explain the implementation of Remote Method Invocation (RMI). (6 Marks)

Remote Method Invocation (RMI) is the object-oriented extension of RPC. It allows an object residing in one process to invoke methods on an object residing in another process.

Implementation Components:

1. Proxy (Stub): This resides on the client side. It implements the same interface as the remote object and acts as a local surrogate. It handles the marshalling of arguments.
2. Dispatcher: This resides on the server side. It receives the request from the communication module and selects the appropriate method of the remote object to invoke.
3. Skeleton: The server-side counterpart to the proxy. It unmarshals the arguments, calls the actual method on the Servant (the real object), and marshals the result for return.

Module 2

Q.3 a. Abstract Model of Distributed File Systems & File Attribute Record (4 Marks)

The abstract model of a DFS separates data storage from the logical organization of files.

Abstract Model Components

- Flat File Service: The lowest level of the service, which performs operations on file contents. It identifies files using Unique File Identifiers (UFIDs) rather than text names.
- Directory Service: This service provides a mapping between text-based names and the UFIDs used by the Flat File Service. It supports operations such as creating directories and searching for files.
- Client Module: This runs on the user's workstation and provides a single, consistent API for applications, thereby concealing the distribution of files across servers.

File Attribute Record Structure

A file attribute record contains metadata that describes the file's properties. Typical structures include:

- File Size: The current length of the file in bytes.
- Timestamps: Records for creation time, last modification, and last access.
- Ownership: The User ID (UID) of the file creator/owner.
- Access Control: A list of permissions (Read, Write, Execute) for different users or groups.

Q.3 b. Analyse the Key Requirements of Distributed File Systems (8 Marks)

Designing a DFS requires balancing several conflicting technical requirements to ensure a seamless user experience.

- Transparency: The system must provide Location Transparency (the user doesn't need to know where the file is) and Migration Transparency (files can move between servers without changing their names).
- Concurrent Updates: The DFS must handle multiple clients writing to the same file concurrently. This usually requires locking mechanisms or atomic update protocols.
- Fault Tolerance: The service must remain operational even if a server crashes. This is often achieved through Stateless Server designs or data replication.
- Consistency: When a file is updated, all users should see the latest version. Maintaining this across different cached copies is a major challenge.
- Efficiency: The system should perform as well as a local file system. This is typically managed through caching (storing recently used data locally).

Q.3 c. Examine Case Studies: SUN NFS and Andrew File System (8 Marks)

These two systems represent different philosophies in distributed storage.

SUN NFS (Network File System)

- Architecture: It follows a Remote Access Model. When a file is accessed, the client requests specific blocks of data from the server.
- Server State: It is Stateless. The server does not keep track of which clients have which files open. This facilitates recovery—if the server crashes, the client simply retries the request.
- Performance: NFS relies heavily on client-side caching of file blocks to reduce network traffic.

Andrew File System (AFS)

- Architecture: It follows a Whole-file Caching Model. When a client opens a file, the *entire* file is downloaded to the local disk.
- Scalability: AFS is more scalable than NFS because it reduces server load; once a file is downloaded, all subsequent reads and writes are local.
- Consistency: It uses Callbacks. The server maintains a record of which clients have cached a file and notifies them if the file is modified by another client.

Q.4 a. Explain the concepts of: i) Uniform Resource Identifier (URI) ii) Uniform Resource Locators (URL) in the context of Name Services. (6 Marks)
In distributed systems, naming services use these identifiers to locate and access resources across a network.

1. Uniform Resource Identifier (URI):
    - Concept: A URI is a generalized string of characters used to identify a logical or physical resource. It is the "parent" category for both URLs and URNs.
    - Function: It identifies a resource either by location, by name, or both. It provides a uniform syntax for resource identification.
2. Uniform Resource Locators (URL):
    - Concept: A URL is a specific type of URI that identifies a resource by its access mechanism (e.g., http, ftp) and its network location.
    - Structure: It typically consists of a protocol (scheme), a domain name (or IP address), a port number, and a path to the specific resource.
    - Key Difference: While a URI identifies *a resource, a URL specifies its location and how to access* it.

Q.4 b. Compare flat naming and structured naming schemes in distributed systems. (5 Marks)

Feature	Flat Naming	Structured Naming
Identifier Type	Unstructured bit strings (e.g., GUIDs, 128-bit IDs).	Human-readable, hierarchical names (e.g., /etc/passwd).
Readability	Machines can process them easily, but humans cannot.	Easy for humans to remember and use.
Location	Does not contain any hint about where the resource is.	Often reflects the logical organization (e.g., sales.company.com).
Scalability	Hard to resolve globally; requires broadcasting or large tables.	Highly scalable through a decentralized hierarchy (like DNS).
Usage	Used internally by systems for unique identification.	Used for user-facing services and file systems.

Q.4 c. Discuss the types of navigation in Name Services. (9 Marks)
Navigation is the process of resolving a name when the naming service is distributed across multiple servers. To find the address associated with a name, the client must "navigate" through these servers.

1. Iterative Navigation:
    - The client sends a request to the first Name Server (NS1).
    - If NS1 cannot resolve the name, it returns the address of the next server (NS2) to the client.
    - The client then contacts NS2 directly. This repeats until the name is resolved.
    - Pro: It places less load on the servers. Con: The client does more work and network traffic is higher for the client.
2. Recursive Navigation:
    - The client sends a request to NS1.

    - If NS1 cannot resolve it, NS1 acts as a client and contacts NS2 on behalf of the original user.
    - The result is passed back through the chain to the client.
    - Pro: Very efficient for the client; allows for server-side caching of results. Con: High demand on server resources.
3. Non-Recursive Server Navigation (Multicast):
    - This is often used in local networks. A client multicasts a "Where is [Name]?" message to all servers.
    - Only the server that holds the name responds.
    - Pro: Fast in small networks. Con: Does not scale to large distributed systems like the Internet.

Q.5 a. Explain the following terms: i) Clock Skew ii) Clock Drift iii) Co-ordinated Universal Time (UTC). (6 Marks)
In distributed systems, physical clocks on different machines do not naturally stay in sync.

1. Clock Skew:
    - Definition: Clock skew is the instantaneous difference between the time readings of two different clocks.
    - Example: If Computer A shows 10:00:00 and Computer B shows 10:00:05, the clock skew is 5 seconds.
2. Clock Drift:
    - Definition: Physical clocks are based on quartz crystals that vibrate at a specific frequency. However, due to temperature changes or manufacturing variations, they vibrate at slightly different rates.
    - Impact: This causes the clocks to gradually gain or lose time relative to a perfect reference. The rate at which a clock gets out of sync is called the drift rate.
3. Co-ordinated Universal Time (UTC):
    - Definition: UTC is the international standard for time, based on atomic clocks (International Atomic Time) with corrections for the Earth's rotation.
    - Usage: Distributed systems use UTC as an external reference to ensure that machines worldwide can synchronize to a single, absolute time.

Q.5 b. Explain the techniques for synchronization of physical clocks. (14 Marks)

Cristian’s Algorithm (External Synchronization)

- Mechanism: Designed for systems where one node has an accurate UTC receiver (Time Server).

- Process: The client sends a request for the time to the server. The server responds with its current time T.
- Correction: The client cannot just set its clock to *T* because the message took time to travel. The client calculates the Round-Trip Time (*RTT*) and sets its clock to:
  Time_new = T_server + RTT/2
- Limitation: If the RTT is high or variable, the accuracy drops.

Berkeley Algorithm (Internal Synchronization)

I. Mechanism: Used when no external UTC source is available. It synchronizes all clocks in a group to an average time.
II. Process:
    A. An elected Master polls all Slaves for their current times.
    B. The Master computes a fault-tolerant average (ignoring clocks that are too far off from the others).
    C. Instead of sending the "correct time" (which might cause clocks to jump backward), the Master sends an offset to each slave, telling it to speed up or slow down its clock.

Network Time Protocol (NTP)

- Mechanism: A highly scalable, hierarchical system used to synchronize clocks across the global Internet.
- Hierarchy (Strata):
  - Stratum 0: High-precision atomic clocks or GPS receivers.
  - Stratum 1: Servers directly connected to Stratum 0 devices.
  - Stratum 2: Servers that synchronize with Stratum 1, and so on.
- Operation: NTP uses complex statistical algorithms to filter out "noise" (network jitter) and can synchronize clocks to within milliseconds over public networks.

Q.6 a. Compare physical and logical clocks in distributed system. (6 Marks)
In distributed systems, the choice between physical and logical clocks depends on whether the application needs "real-time" accuracy or just a "sequence" of events.

Feature	Physical Clocks	Logical Clocks
Basis	Based on hardware oscillators (quartz crystals).	Based on software counters.

Purpose	Measures the actual time of day (e.g., 10:30 AM).	Measures the relative ordering of events.
Synchronization	Requires external sources (UTC/NTP/GPS).	Synchronizes via message passing between processes.
Error Factor	Suffers from physical clock drift and skew.	No drift; values only change when an event occurs.
Constraint	Hard to keep perfectly synchronized across nodes.	Guaranteed to preserve causality (Happened-before).
Use Case	Financial transactions, logging, timeouts.	Version control, mutual exclusion, and debugging.

Q.6 b. Define Logical clocks and explain the Lamport timestamp algorithm with an example diagram. (10 Marks)

Definition: A Logical Clock is a mechanism for capturing the chronological and causal relationships among events in a distributed system. It provides a way to assign a number (timestamp) to each event such that, if event A causes event B, the timestamp of A is less than that of B.

Lamport's Algorithm Rules:

I. Local Events: Each process P_i maintains a local counter L_i. Before an event occurs at P_i, it increments L_i:
   L_i = L_i + 1.
II. Sending a Message: When P_i sends a message m, it attaches its current clock value t = L_i to the message.
III. Receiving a Message: When process P_j receives message m with timestamp t, it updates its own counter:
   L_j = \max(L_j, t) + 1.

Example Explanation:

Imagine Process P_1 sends a message at clock 3. When P_2 receives it, if P_2's current clock is 2, it must jump to 4 (3+1). This ensures the "receive" event happens logically after the "send" event, even if P_2's physical clock was slower.

Q.6 c. Describe the role of global states in debugging distributed systems. (4 Marks)
A Global State is the combined state of all individual processes and the communication channels at a specific moment in time.

- Detecting Invariants: Debugging requires checking if "invariants" (conditions that should always be true) are broken. For example, in a distributed banking system, the total sum of money across all accounts and "in-flight" messages should remain constant.
- Detecting Deadlocks: By capturing a global state (a "snapshot"), developers can analyze if a set of processes are stuck waiting for each other in a cycle.
- Termination Detection: Global states help determine whether a distributed computation has completed or if there are still messages circulating in the network.
- Consistent Cuts: Because there is no global clock, debugging relies on identifying a "consistent cut"—a set of events that represents a state that could have occurred.

Q.7 a. Discuss the following algorithms for mutual exclusion in Distributed Systems. (12 Marks)

The goal of mutual exclusion is to ensure that in a group of processes, only one can access a shared resource (the Critical Section) at any given time.

i) Central Server Algorithm
This is the simplest approach, conceptually similar to a "ticket counter."

- Mechanism: One process is designated as the Coordinator.
- Protocol:
    1. Request: A process sends a request message to the coordinator to enter the critical section.
    2. Wait/Grant: If the critical section is free, the coordinator replies with a "Grant" message. If it is occupied, the coordinator queues the request and remains silent.
    3. Release: When the process finishes, it sends a "Release" message. The coordinator then takes the next request from the queue and sends a "Grant."
- Performance: Requires 3 messages per entry (Request, Grant, Release).
- Pros/Cons: It is easy to implement and fair. However, the coordinator is a single point of failure and a performance bottleneck.

ii) Ring-based Algorithm

This algorithm uses a logical network structure where each process is assigned a position in a circle.

- Mechanism: A special message, called a Token, is circulated around the ring.
- Protocol:
    1. A process can only enter the critical section if it is currently holding the token.
    2. If a process receives the token and does not need to enter the critical section, it immediately passes it to its neighbor.
    3. If it enters the section, it holds the token until finished, then passes it on.
- Performance: Very efficient under high load, but under low load, the token circulates pointlessly, consuming bandwidth.
- Pros/Cons: It is starvation-free. However, if any process crashes, the ring is broken, and a new ring must be reconstructed.

iii) Multicast and Logical Clocks (Ricart-Agrawala)

This is a fully decentralized approach that uses Lamport's timestamps to order requests.

I. Mechanism: There is no central authority; processes reach a consensus on who goes next.
II. Protocol:
    A. Request: A process P_i wanting to enter the section multicasts a message containing its ID and a timestamp (T, P_i) to all other processes.
    B. Decision: When a process P_j receives a request:
        1. If P_j is not interested in the section, it replies "OK" immediately.
        2. If P_j is already in the section, it queues the request and doesn't reply.
        3. If P_j also wants to enter, it compares timestamps. The lowest timestamp wins (earlier request).
    C. Entry: P_i enters only after receiving "OK" from every other process.
III. Pros/Cons: No single point of failure. However, it requires $2(N-1)$ messages, and the failure of any single node stops the entire system from progressing.

Q.7 b. Narrate the significance of fault tolerance in consensus algorithms. (4 Marks)

Consensus is the process of getting all non-faulty nodes to agree on a single value (e.g., who is the leader or whether to commit a transaction).

- Handling Partial Failures: In distributed systems, it is common for one node to crash while others remain active. A consensus algorithm without fault tolerance would "hang" forever waiting for the crashed node.
- Maintaining Consistency: Fault tolerance ensures that even if some nodes fail, the remaining nodes do not reach different conclusions (preventing "Split-Brain" scenarios).

- Quorum Systems: Most modern consensus protocols (like Raft or Paxos) use a Majority Quorum. As long as more than 50% of the nodes remain alive, the system remains operational, providing high availability.

Q.7 c. Discuss the challenges in achieving consensus in distributed system. (4 Marks)
Achieving 100% agreement is mathematically difficult due to these factors:

I. Asynchronous Uncertainty (FLP Impossibility): It has been proven (the FLP result) that in an asynchronous system where messages can be delayed indefinitely, it is impossible to guarantee consensus if even one process fails.
II. Unreliable Communication: Networks drop, duplicate, or reorder messages. Distinguishing between a "crashed" server and a "slow" network link is nearly impossible.
III. Byzantine Failures: A node might not just crash; it might behave maliciously or send conflicting data to different nodes (Byzantine Faults), making it extremely hard to reach a true consensus.
IV. Scalability: As the number of nodes (N) increases, the number of messages required to reach agreement grows exponentially, significantly slowing down the system.

Q.8 a. What is an Election algorithm? What are its requirements? (10 Marks)
Definition:

An election algorithm is a protocol used by a group of distributed processes to elect a unique coordinator (or leader). This is necessary when a system starts up or when the previous coordinator crashes. The newly elected leader is then responsible for managing shared resources or coordinating tasks.

Requirements for Election Algorithms:

To be considered correct and effective, an election algorithm must satisfy two primary properties and several operational requirements:

1. Safety (Agreement): Only one process should be elected as the leader at any given time.
    - All non-faulty processes must eventually agree on the same leader's identity. This prevents a "Split-Brain" scenario in which two nodes believe they are in charge.
2. Liveness (Termination): The algorithm must eventually finish. In a system where a leader has crashed, the election must conclude to allow the system to resume normal operations.
3. Efficiency:

    - Message Complexity: The number of messages exchanged to reach a decision should be minimized to avoid network congestion.
    - Turnaround Time: The time from the start of the election to the proclamation of the new leader should be as short as possible.
4. Fault Tolerance: The algorithm itself must be robust. If a process fails *during* the election, the remaining processes must still be able to complete the election.

Q.8 b. Discuss how distributed systems handle failure during elections using an example. (10 Marks)
Handling failures during an election is complex because the nodes responsible for picking a leader might themselves crash. Distributed systems handle this through timeouts and re-initiation.

Example: The Bully Algorithm

The Bully Algorithm is the classic example of how a system handles failures during the election process based on process IDs (the higher the ID, the "stronger" the process).

1. How the Election Starts:

When a process *P* notices that the coordinator is no longer responding, it initiates an election by sending an ELECTION message to all processes with higher IDs than itself.

2. Handling Failures of Higher-ID Processes:

- If P receives no responses within a specific timeout period, it assumes that all processes with higher IDs have failed (or are unreachable).
- P then "bullies" its way into leadership by sending a COORDINATOR message to all lower-ID processes.

3. If a Process Fails During the Election:

- Suppose a higher-ID process Q receives the ELECTION message from P. Q sends an OK response to P to tell it to step down, and Q then starts its own election by messaging nodes with IDs even higher than itself.
- Failure Scenario: If Q crashes immediately after sending OK to P, P will be waiting for a COORDINATOR message from Q that will never come.
- The Solution: P sets another timeout. If it doesn't hear a "Coordinator" announcement within that time, it simply restarts the entire election process from scratch.

4. Recovery Failure (The "Bully" Returns):

If a process that was previously down (and has a high ID) recovers, it doesn't ask who the current leader is. It immediately starts an election. Because it has a higher ID, it will take over leadership from any existing lower-ID leader.

Q.9 a. Differentiate between flat and nested transactions with an example diagram. (8 Marks)

In distributed systems, transactions ensure that a series of operations are executed reliably. The structure of these transactions determines how failures are handled.

1. Flat Transactions

- Concept: A flat transaction is a single, continuous unit of work. It starts with a Begin and ends with either a Commit (success) or Abort (failure).
- Failure Handling: It follows the All-or-Nothing principle. If any single operation within the transaction fails (e.g., a network timeout or lack of funds), the entire transaction is rolled back to its original state.
- Limitation: They are "brittle." In long-running distributed processes, failing at the very last step causes all previous successful work to be undone.

2. Nested Transactions

- Concept: A nested transaction is composed of a "Top-level" parent transaction that contains one or more "Sub-transactions" (children).
- Failure Handling: They allow for partial recovery. If a sub-transaction fails, the parent can catch the error and either retry that specific sub-transaction or try an alternative path without rolling back the entire top-level transaction.
- Hierarchy Rules:
    - A sub-transaction can only commit if its parent is still active.
    - The "Commit" of a sub-transaction is provisional; it only becomes permanent when the top-level parent transaction commits.

Example Case: Imagine booking a holiday.

- Flat: If the hotel booking is cancelled, the flight booking is automatically cancelled.
- Nested: If the first hotel fails, the system can try to book a *different* hotel sub-transaction while keeping the flight booking intact.

Q.9 b. Explain the two-phase commit (2PC) protocol in distributed transactions. (12 Marks)

The Two-Phase Commit (2PC) protocol is a distributed algorithm used to ensure that all participating nodes in a distributed transaction either commit or abort their changes, thereby maintaining atomicity.

The protocol involves a Coordinator (the leader) and several Participants (the servers holding the data).

Phase 1: The Voting Phase (Prepare Phase)

1. Request: The Coordinator sends a PREPARE (or "CanCommit?") message to all Participants involved in the transaction.

2. Vote: Each Participant checks if it can successfully complete the transaction (e.g., checks if the record is locked and data is valid).
    - If Yes: The Participant records the update in its permanent log (Undo/Redo logs) and sends a VOTE_COMMIT message to the coordinator. It then waits for the final word.
    - If No: The Participant sends a VOTE_ABORT message.

Phase 2: The Completion Phase (Commit Phase)

1. Decision: The Coordinator collects all the votes.
    - Global Commit: If ALL participants voted "Yes," the Coordinator sends a GLOBAL_COMMIT message.
    - Global Abort: If ANY participant voted "No" (or if a timeout occurred), the Coordinator sends a GLOBAL_ABORT message.
2. Acknowledgement: Participants receive the decision. If it's a commit, they make the changes permanent; if it's an abort, they use their logs to undo any temporary changes. They then send an ACK (Acknowledgement) to the coordinator to finish the cycle.

Q.10 a. Discuss methods of concurrency control in distributed transactions. (10 Marks)

Concurrency control ensures that database integrity is maintained when multiple transactions are executed simultaneously across different nodes.

1. Locking Methods (Distributed 2PL):
    - Mechanism: Each data item has a lock (Read or Write). In the Growing Phase, the transaction acquires all necessary locks. In the Shrinking Phase, it releases them.
    - Distributed Aspect: Since data is on different servers, a Lock Manager at each site tracks local locks. A transaction must get a "grant" from all involved Lock Managers before proceeding.
    - Challenge: This can lead to Distributed Deadlock, where Server A is waiting for a lock held by Server B, and vice versa.
2. Timestamp Ordering:
    - Mechanism: Every transaction is assigned a unique global timestamp when it starts.
    - Rule: For any two conflicting operations, the one belonging to the transaction with the earlier timestamp must be executed first.
    - Conflict Handling: If a "younger" transaction tries to access data that an "older" transaction is already using, the younger one is aborted and restarted with a new timestamp.
3. Optimistic Concurrency Control:
    - Mechanism: Based on the assumption that conflicts are rare. Transactions work on local "shadow" copies of data without locking anything.

- Phases: 1. Working Phase: Execute using local copies. 2. Validation Phase: Check if any other transaction committed changes that conflict with your work. 3. Update Phase: If no conflict, make the changes permanent.
- Benefit: Excellent for read-heavy systems as it eliminates the overhead of managing locks.

Q.10 b. Explain the role of the Transaction Coordinator. (10 Marks)

In a distributed system, a transaction involves multiple "Participants" (servers). The Transaction Coordinator (TC) is the central entity responsible for managing the transaction lifecycle.

Key Roles and Responsibilities:

1. Transaction Identification: The TC assigns a globally unique Transaction ID (TID) to enable tracking of the transaction across servers.
2. Participant Management maintains a list of all servers (Resource Managers) that are performing work on the transaction.
3. Executing Two-Phase Commit (2PC): The TC initiates the Voting Phase (asking if everyone is ready).
    - Based on the votes, the TC makes the final decision and initiates the Completion Phase (telling everyone to either Commit or Abort).
4. Failure Recovery & Logging: The TC maintains a Transaction Log on stable storage.
    - If a participant crashes and restarts, it contacts the TC to find out the final status of the transaction.
    - If the TC itself crashes, it uses its log to resume the 2PC process from where it left off.
5. Timeout Handling: If a participant fails to respond within a specific window, the TC makes a safety-first decision (usually to Abort) to prevent the system from hanging indefinitely.

Q.10 c. Approaches in file recovery: Logging and Shadow version. (10 Marks)

1. Logging (Write-Ahead Logging - WAL):
    - Concept: Before any change is made to the actual file, a record of the change (an "intent") is written to a sequential Log File on stable storage.
    - Redo/Undo: If the system crashes, the recovery manager reads the log. It performs a Redo for committed but not yet written-to-disk transactions, and an Undo for transactions that were active during the crash to remove partial changes.
    - Pros: Highly efficient for high-frequency updates.
2. Shadow Version (Shadow Paging):

- Concept: The system maintains two-page tables. When a transaction modifies a file, it does not overwrite the original data. Instead, it creates a "shadow" copy of the affected pages.
- The Switch: The "current" pointer is updated to the new shadow version only when the transaction is fully successful.
- Recovery: Recovery is trivial. If a crash occurs mid-transaction, the system simply ignores the unfinished shadow pages and continues using the original (old) version.
- Pros: Near-instant recovery; no need for complex undo/redo logic.