USN | | | | | | | | | |

Internal Assessment Test II – December 2025 – Answer Key

Sub:	Distributed Systems				Sub Code:	BCS515D	Branch:	ISE
Date:	02-12-2025	Duration:	90 min's	Max Marks: 50	Sem/Sec:	V / A, B & C		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	a. What is time consistency? Give an example	4	CO4	L2
	b. Explain the importance of maintaining time consistency in distributed systems.	6		

1 a. What is time consistency? Give an example

A distributed system is time-consistent if:

- Events are ordered in a way that respects causality
- Time-based decisions (like timestamps, leases, or timeouts) are interpreted uniformly across nodes
- No node observes an event as occurring "before" its cause

This is usually achieved through:

- Logical clocks (Lamport clocks)
- Vector clocks
- Clock synchronization protocols (e.g., NTP, Cristian's algorithm)

Example:
Consider a distributed database with two servers: S1 and S2.

- Client A writes x = 10 to S1 at time t1
- S1 sends an update message to S2
- Due to network delay, S2 receives the update later
- Meanwhile, Client B reads x from S2

Without time consistency:

- S2 might think its local time is earlier than t1
- Client B may read an old value (x = 5)
- The system violates the expected ordering of events

With time consistency:

- S1 timestamps the write using a logical or synchronized clock
- S2 applies the update only if the timestamp is newer
- Client B reads the correct latest value (x = 10)

b. Explain the importance of maintaining time consistency in distributed systems.

Maintaining time consistency in distributed systems is important because it ensures that events are interpreted in the correct order across multiple nodes, even though there is no global clock. Its importance can be explained through the following points:

1. Correct ordering of events

Time consistency preserves the causal relationship between events (cause happens before effect). This prevents situations where a node observes an effect before its cause, which could lead to incorrect system behavior.

2. Data consistency

In distributed databases and file systems, time consistency ensures that read and write operations are applied in the correct sequence, preventing stale reads, lost updates, and conflicting writes.

3. Coordination and synchronization

Many distributed algorithms (leader election, mutual exclusion, transactions) rely on time or event ordering. Time consistency allows nodes to coordinate actions reliably without ambiguity.

	4. Fault detection and recovery Time-based mechanisms such as timeouts, leases, and heartbeat messages depend on consistent time. Without time consistency, a healthy node may be wrongly suspected as failed or vice versa. **5. Concurrency control** Time consistency helps in resolving concurrent operations by determining which operation happened first, enabling safe conflict resolution in concurrent environments. **6. System reliability and predictability** A consistent notion of time across nodes makes system behavior deterministic and predictable, which is crucial for debugging, auditing, and ensuring correctness.			
2	Discuss the followings w.r.t Network Time Protocol: (i) Design aims and features (ii) Modes of NTP server synchronization. (i) Design Aims of NTP **Accurate time synchronization -** To synchronize clocks of computers over a network to UTC (Coordinated Universal Time) with high accuracy, despite network delays. **Scalability -** To support synchronization across large, global networks ranging from small LANs to the entire Internet. **Fault tolerance and reliability -** To continue providing correct time even when some time sources or network paths fail. **Efficiency -** To minimize network and processing overhead while maintaining accurate synchronization. **Robustness to variable delays -** To handle network latency, jitter, and asymmetric delays without significant loss of accuracy. **Security and trust management -** To prevent incorrect or malicious time updates through authentication and sanity checks. **Features:** **Hierarchical time distribution (Stratum levels)** - Stratum 0: Reference clocks (atomic clocks, GPS) - Stratum 1: Servers directly connected to stratum 0 - Stratum 2, 3, …: Servers synchronized from higher strata This hierarchy ensures scalability and structured synchronization. **Clock offset and delay calculation** NTP uses round-trip delay and clock offset estimation to accurately adjust local clocks. **Clock discipline algorithms** Gradually adjusts clocks using slewing (small adjustments) instead of abrupt changes to maintain stability. **Redundancy and filtering** Synchronizes with multiple servers, applies statistical filtering, and selects the most reliable time source. (ii) Modes of NTP server synchronization **1. Client–Server Mode** One system acts as an NTP client and synchronizes its clock with an NTP server. The client periodically requests time updates; the server responds. Most commonly used mode. **Example:** A workstation synchronizing with an Internet NTP server. **2. Symmetric Active Mode** Both systems act as peers and can synchronize with each other. Used when both systems are equally reliable time sources. Each peer can initiate synchronization.	5 5	**CO4**	**L2**

	Example: Two primary time servers in different data centers. **3. Symmetric Passive Mode** Used in conjunction with symmetric active mode. A system enters passive mode when it responds to a symmetric active peer. It does not initiate synchronization but participates once contacted. **Example:** Backup or secondary time server. **4. Broadcast Mode** A server broadcasts time information periodically to multiple clients. Clients do not send requests; they simply listen. Reduces network traffic. Used mainly in local area networks (LANs). **5. Multicast Mode** Similar to broadcast mode, but uses multicast addresses. Time updates are sent only to interested clients. More efficient than broadcast for large networks. **6. Anycast Mode** Clients send requests to an anycast address. Routing automatically directs requests to the nearest available NTP server. Improves scalability and reliability in large networks.			

3	a. What is a failure detector? Give an example
	A **failure detector** is a mechanism in a distributed system that **identifies whether a process or node has failed (crashed) or is unreachable**. Since nodes communicate over unreliable networks and there is no global clock, failure detection is inherently uncertain.
	Example:
	Consider a distributed system with nodes **A, B, and C**.
	1. Each node periodically sends a **heartbeat message** to others (or to a monitoring node).
	2. If node **B** fails or becomes unreachable, it stops sending heartbeats.
	3. If node **A** does not receive a heartbeat from **B** within a specified **timeout**, it **suspects B has failed**.
	4. Node A may then:
	    o Inform other nodes
	    o Trigger leader election
	    o Redirect requests away from B
	
	b. Briefly discuss the two types of failure detectors.
	
	1. Unreliable Failure Detector
	• May **incorrectly suspect** a process as failed due to network delays or temporary slowdowns.
	• Cannot always distinguish between a **crashed node** and a **slow node**.
	• Commonly used in **asynchronous distributed systems**.
	• **Example:** Timeout-based heartbeat detector.
	2. Reliable Failure Detector
	• **Never makes false failure detections**.
	• Accurately detects crashed processes.
	• Assumes bounded message delays and synchronized clocks.
	• Used in **synchronous distributed systems**.
		4
		6
			CO5	**L3**

4	Briefly explain the following election algorithms with suitable examples: (i) Bully (ii) Ring. (i) Bully Algorithm 1. **Detection of failure** A process detects that the current coordinator has failed.	5 5	**CO5**	**L2**

2. Election initiation
The detecting process sends an **ELECTION** message to all processes with **higher IDs**.
3. Response from higher-ID processes

- If no higher-ID process responds, the initiator **becomes the coordinator**.
- If a higher-ID process responds with an **OK** message, the initiator **withdraws**.

4. Recursive election
The higher-ID process that replied now starts its **own election** by sending ELECTION messages to processes with even higher IDs.
5. Coordinator announcement
The process with the **highest active ID** wins and broadcasts a **COORDINATOR** message to all other processes.

Example:
Processes: **P1, P2, P3, P4, P5**
(IDs: 1 < 2 < 3 < 4 < 5)

- Current coordinator **P5** crashes.
- **P2** detects the failure and sends ELECTION to **P3, P4, P5**.
- **P3 and P4** reply with OK.
- **P4** starts a new election and sends ELECTION to **P5**.
- No response from **P5**.
- **P4** becomes the new coordinator and sends COORDINATOR message to all.

Advantages:

- Simple and easy to implement.
- Guarantees selection of the **highest-ID active process**.

Disadvantages:

- High message overhead: **$O(n^2)$** messages in the worst case.
- Not efficient for large systems.
- Multiple elections may occur simultaneously.

(ii) Ring Algorithm

1. **Election initiation**
   Any process that detects the failure of the coordinator initiates an election by sending an **ELECTION** message to its **successor**.
2. **Passing the election message**
   - The ELECTION message contains the **ID of the initiator**.
   - Each process that receives the message:
     - Adds its own ID to the message (or replaces it if larger)
     - Forwards the message to the next process in the ring
3. **Selection of leader**
   - When the ELECTION message completes a full round and returns to the initiator, the **highest ID** in the message is selected as the new coordinator.
4. **Coordinator announcement**
   The initiator sends a **COORDINATOR** message around the ring, informing all processes of the new leader.

Processes in ring: **P1 → P2 → P3 → P4 → P5 → P1**
IDs: 1 < 2 < 3 < 4 < 5

- Coordinator **P5** crashes.
- **P2** initiates election and sends ELECTION(2) to **P3**.
- Each process forwards the message, updating the maximum ID.
- When it returns to **P2**, the highest ID is **P4**.
- **P4** is elected coordinator.
- COORDINATOR(4) message is circulated to all.

Advantages:

- Lower message overhead compared to Bully Algorithm.

	• Simple and orderly message passing. • Suitable for systems with logical ring topology. **Disadvantages:** • Election takes longer due to sequential message passing. • Failure of multiple consecutive nodes can disrupt the ring. • Requires maintenance of the logical ring structure.			
5	a. Discuss the challenges in achieving consensus in distributed systems. **1. Absence of a Global Clock** Distributed systems lack a single, perfectly synchronized clock. • Nodes cannot reliably determine the exact order of events. • This makes it hard to decide which proposal or update came first. **2. Process and Node Failures** Nodes may **crash, restart, or behave unpredictably**. • Consensus must tolerate failures while still reaching agreement. • Determining whether a node is failed or just slow is difficult. **3. Unreliable Communication** Networks may experience **message loss, duplication, delay, or reordering**. • Messages may arrive late or not at all. • This uncertainty complicates coordination among nodes. **4. Network Partitions** The network may split into isolated groups. • Each partition may independently try to reach consensus. • Ensuring consistency across partitions is challenging. **5. Asynchronous Nature of Systems** In asynchronous systems, there are **no bounds on message delays or process speeds**. • This leads to the **FLP impossibility result**, which states that deterministic consensus is impossible if even one node may fail. **6. Trade-off Between Consistency and Availability** According to the **CAP theorem**, during network partitions: • A system must choose between **consistency** and **availability**. • Consensus protocols often sacrifice availability to preserve correctness. **7. Scalability and Performance** As the number of nodes increases: • Message overhead and latency increase. • Maintaining fast and efficient consensus becomes harder. **8. Security and Faulty Behavior** Nodes may act **maliciously or arbitrarily** (Byzantine faults). • Byzantine fault tolerance requires complex algorithms and high overhead. • Achieving consensus securely is significantly more difficult. b. List the properties to be satisfied by a reliable multicast. 1. **Validity** • If a correct process multicasts a message, **it eventually delivers that message** to itself. 2. **Agreement** • If a correct process delivers a message, **all correct processes in the multicast group eventually deliver the same message**. 3. **Integrity** • A message is **delivered at most once** to each process. • A message is delivered **only if it was actually multicast** by some process. 4. **No Creation** • No spurious messages are delivered; messages must originate from a legitimate sender. 5. **No Duplication**	5 5	CO6	L3

	• Each message is delivered **only once**, even if retransmissions occur.			
6	a. What are the two phases of two-phase commit protocol?	5 5	**CO6**	**L4**

1. Voting Phase (Prepare Phase)

- The **coordinator** sends a **PREPARE** (or *VOTE-REQUEST*) message to all participating processes.
- Each participant:
  - Checks whether it can commit the transaction.
  - Writes the decision to stable storage.
  - Responds with **VOTE-COMMIT** (Yes) or **VOTE-ABORT** (No).

2. Commit Phase (Decision Phase)

- Based on the votes:
  - If **all participants vote COMMIT**, the coordinator sends a **GLOBAL-COMMIT** message.
  - If **any participant votes ABORT**, the coordinator sends a **GLOBAL-ABORT** message.
- Participants carry out the decision and acknowledge completion.

b. Discuss the best case and worst case of the protocol.

Best Case of Two-Phase Commit
Scenario:

- No node failures
- No message loss
- All participants are ready to commit

Behaviour:

1. Coordinator sends **PREPARE** messages.
2. All participants reply with **VOTE-COMMIT**.
3. Coordinator sends **GLOBAL-COMMIT**.
4. Participants commit and acknowledge.

Characteristics:

- Minimum message exchanges
- Transaction completes quickly
- No blocking occurs
- Consistent commit at all sites

Result: High efficiency and correctness when the system is fault-free.

Worst Case of Two-Phase Commit
Scenario:

- Coordinator crashes after participants vote COMMIT
- Network delays or partitions occur

Behaviour:

1. Participants vote **COMMIT** and enter the **uncertain (blocked) state**.
2. Coordinator fails before sending **GLOBAL-COMMIT / GLOBAL-ABORT**.
3. Participants **cannot decide independently**.
4. They must wait until the coordinator recovers.

Characteristics:

- **Blocking problem**
- Participants hold locks and resources
- System throughput decreases
- Risk of long delays and deadlocks

Result: The protocol may block indefinitely, reducing availability.