Q1. Illustrate the working of a synchronous single-initiator spanning tree algorithm with an example network of 5 nodes.

Answer:

- In a **synchronous single-initiator spanning tree algorithm**, one node initiates the construction of the spanning tree by sending messages to its neighbors in synchronized rounds.
- **Steps:**
    1. The initiator (say node A) sends a "search" message to all its neighbors in Round 1.
    2. Each neighbor, upon receiving the message for the first time, sets its parent to the sender and forwards the message to all its other neighbors in the next round.
    3. Nodes ignore duplicate "search" messages if they already have a parent.
    4. The process continues until all nodes have parents.
    5. The collection of parent-child relationships forms the spanning tree.

Example (5 nodes network):

- **Round 1:** A → sends to B, C
- **Round 2:** B → sends to D ; C → sends to E
- **Round 3:** D and E have no new neighbors, algorithm stops.

Result: Spanning Tree edges = {A-B, A-C, B-D, C-E}

Q2. Apply the concept of message passing and shared memory to explain how you would design a distributed chat application using both models.

Answer:

- **Message Passing Model:**
    - Each user is represented as a process.
    - Messages (chat text) are sent using primitives like send(user, msg) and receive(user).
    - Example: Alice sends "Hi" → system delivers to Bob's process inbox.
    - Pros: Explicit communication, natural for distributed systems.
    - Cons: Higher latency, need for reliable delivery.

- **Shared Memory Model:**
  - A shared space (e.g., distributed database or shared memory abstraction) stores chat logs.
  - Processes write messages to shared memory → others read from it.
  - Example: Alice writes "Hi" into a shared chat buffer → Bob's client reads it.
  - Pros: Easier to program, resembles local memory.
  - Cons: Hard to implement in physically distributed systems (consistency issues).

Q3. A team is developing a multiplayer online game hosted across several servers. Explain the relation of this system to a parallel multiprocessor and a multicomputer system.

Answer:

- **Parallel Multiprocessor System:**
  - Multiple CPUs share a common memory.
  - In games: Would be like one big server with multiple processors handling tasks (graphics, physics, networking).
  - Advantage: Low communication latency, shared state consistency.
- **Multicomputer (Distributed System):**
  - Independent machines with their own memory, communicating over a network.
  - In games: Multiple geographically distributed servers hosting different regions of the virtual world.
  - Advantage: Scales to millions of players, fault tolerance.
  - Disadvantage: Network latency, consistency management.

Relation:
The game uses a **multicomputer model** (servers across locations) but internally each server may use **multiprocessors** to handle computations.

Q4. Demonstrate how communication latency affects the performance of a distributed system compared to a parallel multiprocessor system.

Answer:

- **Parallel Multiprocessor:**
  - Communication is via shared memory → nanoseconds.
  - Example: Thread A updates score, Thread B reads instantly.
- **Distributed System:**
  - Communication over a network → milliseconds.
  - Example: Player's action on Server A must be transmitted to Server B → delay visible as lag.

Impact:
Distributed systems suffer more due to network delays, packet loss, and synchronization overhead, while multiprocessors are faster due to direct memory access.

Q5. Use a simple graph scenario to illustrate how a distributed depth-first search algorithm proceeds through message passing.

Answer:

- **Graph:** A–B–C–D
- **DFS Steps via Message Passing:**
    1. Initiator A sends "DFS-visit" to B.
    2. B sets parent = A, forwards to C.
    3. C sets parent = B, forwards to D.
    4. D has no new neighbor → sends "backtrack" to C.
    5. Backtracking continues till A.

Result: DFS tree edges: {A-B, B-C, C-D}

Q6. Estimate the total number of rounds and messages required to build a spanning tree in both synchronous and asynchronous approaches.

Answer:

- **Synchronous (Single Initiator):**
    - Rounds = Diameter of network (longest shortest path).
    - Messages = $2 \times (n-1) \approx O(n)$ (one forward + one acknowledgment per edge).
- **Asynchronous:**
    - No strict round notion; depends on message delays.
    - Messages = $O(m)$ (where m = number of edges) since flooding can cover all edges.

Q7. Demonstrate the formation of a depth-first spanning tree using the asynchronous concurrent-initiator algorithm on a given undirected graph.

Answer:

- Suppose nodes A, B, C, D connected in a cycle.
- **Steps:**
    1. Multiple nodes may initiate DFS simultaneously.
    2. Each node maintains parent pointer (first sender).
    3. A initiates → visits B → then C → then D.

4. If B also initiates, it merges into A's DFS when it encounters already-visited nodes.

- **Result:** A consistent DFS spanning tree is formed with arbitrary initiator winning locally.

Q8. Apply Lamport's scalar time rules to a given sequence of events in a distributed system and assign timestamps.

Answer:
Rules:

1. Each process increments its clock before each event.
2. On message send, attach timestamp.
3. On receive, set clock = max(local, received)+1.

Example:

- P1: e1 → send m → e2
- P2: e3 (receive m) → e4
- Timestamps:
  - P1: e1(1), send(2), e2(3)
  - P2: e3(receive)(max(0,2)+1=3), e4(4)

Q9. A bank uses synchronous communication between branches to settle transactions. Compare the advantages and disadvantages of this versus asynchronous communication.

Answer:

- **Synchronous:**
  - Advantages: Strong consistency, predictable ordering.
  - Disadvantages: Slow, blocked if one branch delays.
- **Asynchronous:**
  - Advantages: Faster, non-blocking, tolerant of delays.
  - Disadvantages: May cause temporary inconsistencies, harder reconciliation.

Q10. Imagine a social media application where likes and comments are recorded across servers. Explain how scalar logical clocks help maintain the correct ordering of events in such a distributed system.

Answer:

- Each like/comment is an event with a timestamp.

- If two events are concurrent, timestamps provide an ordering to detect causality.
- Example: If Alice likes at T=5 and Bob comments at T=6, servers merge logs in correct order.
- Helps maintain consistent feed ordering without central clock.

Q11. Distinguish the correctness and convergence properties of the asynchronous concurrent-initiator spanning tree algorithm.

Answer:

- **Correctness:** Guarantees that a spanning tree (no cycles, all nodes connected) will be built despite multiple initiators.
- **Convergence:** Algorithm terminates when all nodes have a parent and no new messages are generated. Multiple initiators eventually merge trees.

Q12. Why is the program structure of distributed graph algorithms important in designing fault-tolerant applications?

Answer:

- Clear structure ensures that partial failures can be detected and recovered.
- Example: If a DFS algorithm crashes mid-way, structured state (parent pointers, visited flags) helps recovery.
- Fault-tolerant applications rely on deterministic, modular graph algorithm design.

Q13. Discuss the advantages and disadvantages of synchronous vs asynchronous execution in distributed environments with practical scenarios.

Answer:

- **Synchronous:**
  - Advantages: Simpler reasoning, predictable rounds.
  - Disadvantages: Requires global clock, slow if one node lags.
  - Example: Bank settlement (safe, but slower).
- **Asynchronous:**
  - Advantages: Scalable, fault-tolerant, realistic.
  - Disadvantages: Harder to debug, potential inconsistencies.
  - Example: Social media feed updates (fast, but eventual consistency).

Q14. Compare and contrast message passing systems and shared memory systems with appropriate use-case examples.

Answer:

- **Message Passing:**
  - Explicit communication with send/receive.
  - Example: Email system, chat apps.
  - Works naturally in distributed settings.
- **Shared Memory:**
  - Processes read/write common memory.
  - Example: Multi-core cache sharing.
  - Harder to implement in physically distributed systems.

Q15. A spanning tree must be constructed from a network with one node initiating the process. Explain how the single-initiator spanning tree algorithm using flooding would work in a synchronous system.

Answer:

- Initiator floods "search" message in synchronized rounds.
- Each node sets parent on first message, ignores later duplicates.
- Flooding ensures all nodes are reached.
- The parent-child relations form the spanning tree.

Q16. Compare the asynchronous concurrent-initiator flooding algorithm with the single-initiator version in terms of message complexity and reliability.

Answer:

- **Single-Initiator:**
  - Messages = O(m), controlled flooding.
  - Reliable, but requires designated root.
- **Concurrent-Initiator:**
  - Multiple initiators flood simultaneously.
  - Messages can be >O(m), more redundant.
  - More fault-tolerant (no single point of failure).

Q17. An IoT network uses depth-first traversal to update firmware. Explain how the asynchronous concurrent-initiator depth-first search spanning tree algorithm would operate.

Answer:

- Any node can initiate DFS.
- Each device forwards update to an unvisited neighbor.

- Backtracks when no unvisited neighbor remains.
- Multiple initiators may merge into one DFS tree.
- Ensures all devices get firmware with minimal redundancy.

Q18. A telecommunications provider wants to minimize the cost of linking cities via cables. Describe how a synchronous MST algorithm can be applied.

Answer:

- MST ensures minimum total edge (cable) cost while connecting all cities.
- Example Algorithm: Synchronous version of Prim's or Borůvka's.
- Rounds: Nodes exchange minimum-weight edge info synchronously.
- Result: MST gives cheapest network design with no cycles.

Q19. A power grid network seeks to optimize its transmission lines. Explain why minimum-weight spanning trees are important and how they might be computed synchronously.

Answer:

- Power grid must connect all stations with minimal line cost.
- MST avoids redundant connections, reduces costs, ensures efficiency.
- **Synchronous MST algorithm:**
    - Each substation selects min outgoing edge in each round.
    - Edges are added gradually until MST forms.
    - Ensures low-cost, cycle-free design.

Q1. A distributed database tracks operations using scalar timestamps. Explain how this helps with maintaining causal ordering.

Answer:

- **Scalar timestamps (Lamport clocks)** assign an integer value to each event in the system.
- Rules:
    1. Increment clock before each local event.
    2. Send timestamp with each message.
    3. On receiving, set local time = max(local, received)+1.

Use in a distributed database:

- If transaction T1 happens before T2, then **timestamp(T1) < timestamp(T2)**.
- This ensures causal ordering: if an update depends on a prior update, it will always be seen as later.

Example:

- Node A updates "Balance=500" at time 5.
- Node B reads at time 7.
- Since 5 < 7, the database knows the read must reflect the update.

Q2. Two events occur in different parts of a distributed system. Describe how vector clocks help determine the causality between them.

Answer:

- **Vector clocks** maintain an array of timestamps, one for each process.
- Rules:
  - Increment own entry before event.
  - Send entire vector on messages.
  - On receive, take elementwise max.

Causality check:

- Event e1 → e2 if VC(e1) < VC(e2) (all elements ≤, and at least one <).
- If neither VC(e1) < VC(e2) nor VC(e2) < VC(e1), events are **concurrent**.

Example:

- P1: [2,0,0] ; P2: [1,3,0]
- Neither vector dominates → events are concurrent.

Q3. Compare scalar time and vector time using a scenario involving three communicating nodes with conflicting updates.

Answer:

- **Scenario:**
  - Node A updates "X=1" → scalar time 5.
  - Node B updates "X=2" → scalar time 6.
  - No message exchanged (updates are concurrent).
- **Scalar Time Problem:**
  - Only timestamps 5 and 6 → system thinks B's update happened "after" A's.
  - But actually they were concurrent, so causality is misrepresented.
- **Vector Time Solution:**
  - A's vector: [2,0,0] ; B's vector: [0,2,0].

- Incomparable vectors → concurrency correctly detected.

Conclusion: Vector clocks capture concurrency, scalar clocks do not.

Q4. A collaborative document editor must track edits across sessions. Which logical clock would best help ensure correct version control? Why?

Answer:

- **Best choice: Vector clocks.**
- Reason:
  - Concurrent edits must be detected (e.g., Alice and Bob editing same paragraph).
  - Scalar clocks would impose a false total order, potentially overwriting one update.
  - Vector clocks detect concurrency, so a merge policy can resolve conflicts.

Q5. A military distributed command system tracks causality between many agents. Explain how matrix time extends vector clocks and improves traceability.

Answer:

- **Matrix clocks:** each process maintains an n×n matrix, where row i contains process i's knowledge of all others.
- Extends vector clocks by tracking not just "what I know," but also "what I know about what others know."

Benefit for military system:

- If Agent A knows that Agent B is aware of C's command, this prevents redundant re-transmission.
- Provides deeper causality tracking for **multi-level communication**.

Q6. Give an example where scalar and vector clocks fail to identify the complete causal relationship, but matrix time succeeds.

Answer:

- Suppose P1 sends a message to P2, and P2 forwards it to P3.
- **Scalar clocks:** P3 cannot tell if info came from P1 indirectly.
- **Vector clocks:** P3 sees P2's knowledge but not whether P2 learned from P1.

- **Matrix clocks:** By maintaining second-order knowledge, P3 can infer the original source (P1).

Thus, matrix clocks capture **"knowledge of knowledge"**, unlike scalar/vector.

Q7. A distributed stock trading platform relies on accurate timestamps for transactions. Explain how NTP (Network Time Protocol) helps maintain synchronization.

Answer:

- **NTP (Network Time Protocol):**
  - Synchronizes clocks of distributed machines with reference servers.
  - Uses offset and delay calculation from round-trip messages.
- **For stock trading:**
  - Ensures buy/sell orders are timestamped consistently.
  - Prevents disputes ("who traded first").
  - Helps in auditing and fraud detection.

Q8. Describe a scenario where physical clock drift leads to inconsistencies in a distributed system. How does NTP correct it?

Answer:

- **Scenario:**
  - Server A's clock drifts 2 seconds ahead.
  - Transaction recorded as "executed at 10:00:02" before another transaction at "10:00:01" on Server B.
  - Leads to **wrong ordering**.
- **NTP correction:**
  - Periodically polls servers.
  - Adjusts drift using small corrections (slewing instead of abrupt resets).
  - Keeps all nodes within milliseconds of accuracy.

Q9. A hospital management system has distributed departments and needs to share patient data securely and efficiently. Discuss the type of distributed system, communication, and synchronization involved.

Answer:

- **Type:** Distributed information system.
- **Communication:** Likely message passing over secure protocols (HTTPS, gRPC).
- **Synchronization:**

    - Logical clocks or database transaction ordering.
    - Ensures patient updates (e.g., diagnosis, prescriptions) are consistent across departments.
- **Security:** Encryption and authentication are critical due to sensitive health data.

Q10. In a real-time multiplayer game, players are located across the globe. Discuss how clock synchronization and logical time can improve fairness.

Answer:

- **Problem:** Latency makes events arrive in different orders at different servers.
- **Solution:**
    - **Clock synchronization (NTP/Precision Time Protocol):** keeps physical clocks close.
    - **Logical clocks:** ensure causality of game actions.
- **Result:**
    - Fair ordering of attacks, moves, and scores.
    - Avoids "who shot first?" disputes.

Q11. A blockchain system involves decentralized verification and transaction recording. Analyze its execution model (synchronous/asynchronous), message passing style, and use of logical time.

Answer:

- **Execution model:** Mostly **asynchronous** (nodes are geographically distributed, unpredictable delays).
- **Message passing:** Gossip protocol for block/transaction propagation.
- **Logical time:** Blocks ordered by cryptographic hash chains and consensus rules (like Lamport’s "happens-before").
- **Result:** Eventual consistency across nodes.

Q12. A ride-sharing application uses multiple backend services distributed across different data centers. Describe how this setup qualifies as a distributed system and the challenges it might face.

Answer:

- **Qualifies as distributed system:**
    - Independent components (driver service, passenger service, payments) communicate over a network.
- **Challenges:**
    - Network latency (driver location updates).

- Fault tolerance (one data center failure).
- Data consistency (booking vs payment).
- Synchronization (avoiding double-booking).

Q13. A distributed file storage system needs to replicate data across multiple machines. Compare how this would be handled in a parallel multiprocessor system versus a multicomputer system.

Answer:

- **Parallel Multiprocessor (shared memory):**
  - Replication is easier → all processors share memory.
  - Consistency enforced by hardware cache coherence.
- **Multicomputer (distributed memory):**
  - Replication requires explicit message passing (e.g., Hadoop HDFS).
  - Must implement consistency protocols (e.g., quorum, Paxos, Raft).

Q14. A real-time video processing system uses GPUs and CPUs working together. Analyze whether shared memory or message passing is more appropriate for inter-device communication, and justify your answer.

Answer:

- **Shared memory** (via CUDA unified memory, OpenCL buffers) is most efficient.
- Reason: GPUs and CPUs work on large data blocks (video frames). Copying via messages would be too costly.
- Shared memory avoids overhead of serialization/deserialization.

Q15. A distributed weather monitoring system collects data from sensors and forwards it to a central server. Identify and describe two communication primitives used in this system.

Answer:

1. **Send/Receive:** Sensors send readings, server receives them.
2. **Broadcast/Multicast:** Central server may broadcast alerts (e.g., cyclone warning) to all sensors.

Q16. A university's online exam system must coordinate submission across thousands of students. Propose a communication strategy using primitives that ensures reliable and ordered delivery.

Answer:

- **Primitives used:**
  - Reliable send/receive with acknowledgments.
  - FIFO channels to ensure order of submissions.
- **Strategy:**
  - Each student sends encrypted submission → server acknowledges receipt.
  - Retransmission if ACK not received.
  - Ordered delivery ensures timestamps reflect fairness.

Q17. A factory automation system coordinates robots for assembly tasks. Justify why synchronous execution is critical for this use case, and what could go wrong if asynchronous execution were used instead.

Answer:

- **Synchronous execution:**
  - Ensures robots act in lockstep (e.g., robot arm places part, another welds it).
- **If asynchronous:**
  - Robot B may weld before Robot A places part → malfunction, safety hazard.
- **Thus:** Synchronous execution guarantees safety and correctness in real-time industrial systems.

Q18. A distributed social media app allows users to post, like, and comment simultaneously. Explain the pros and cons of asynchronous event processing in this scenario.

Answer:

- **Pros:**
  - High scalability, responsive UI.
  - Users can keep interacting without waiting for global consistency.
- **Cons:**
  - Temporary inconsistencies (e.g., like count showing differently across servers).
  - Need eventual consistency models.

Trade-off: Better user experience at the cost of consistency.

Q19. A drone swarm communication network needs to form a spanning tree to relay commands efficiently. Discuss the application of the asynchronous concurrent-initiator spanning tree algorithm in this setting.

Answer:

- **Asynchronous concurrent-initiator spanning tree:**
  - Multiple drones may initiate spanning tree construction simultaneously.
  - Each drone selects parent on first message, ignores duplicates.
  - Trees eventually merge into one consistent spanning tree.

Application in drone swarm:

- Robust against node failures (no single leader needed).
- Efficient command relay from control center.
- Scales well as drones dynamically join/leave the network.

DC ALM-2

Q1. Apply the concept of FIFO, causal, and total order message delivery to a group chat application. Describe how incorrect ordering can affect the application's consistency.

Answer:

- **FIFO Order:** Messages from each sender are delivered in the order sent.
  - Example: If Alice sends “Hi” then “How are you?”, Bob must see them in the same order.
- **Causal Order:** Messages that are causally related must be delivered respecting causality.
  - Example: Alice posts “Exam tomorrow”, Bob replies “Yes”, everyone must see Alice’s message before Bob’s reply.
- **Total Order:** All processes see all messages in exactly the same order.
  - Example: Even if two users send at the same time, the system imposes a single global order.

Incorrect ordering effect:

- Without causal order, replies may appear before the original message.
- Without FIFO, one person’s thoughts may appear jumbled.
- Without total order, different users may see different chat histories → inconsistency.

Q2. Demonstrate how propagation trees for multicast can be used to efficiently deliver messages to multiple recipients in a distributed group communication system.

Answer:

- **Propagation Tree:** A spanning tree rooted at the sender used to deliver multicast messages.
- **Steps:**
    1. Sender sends message to immediate children.
    2. Each child forwards to its own children.
    3. Process repeats until all recipients receive message.

Efficiency:

- Avoids redundant transmissions (no flooding).
- Each link carries the message at most once.

Example: In a group of 6 nodes, sender builds a tree → message flows along tree edges → total messages = $n-1$, not $O(n^2)$.

Q3. If Node A sends Message 1 and then Message 2 to Node B, Node B will receive Message 1 first, followed by Message 2. Analyze the above scenario using FIFO, causal, and total order message delivery.

Answer:

- **FIFO:** Guarantees M1 arrives before M2 at Node B.
- **Causal:** Same guarantee since M1 → M2 (causal chain).
- **Total Order:** All nodes (not just B) must agree on the sequence {M1, M2}.

Thus, in this case, all three paradigms coincide.

Q4. Distributed Banking System: multiple banks must agree on transaction order. Apply message ordering paradigms.

Answer:

- **FIFO:** Each bank sees its own outgoing transactions in order.
- **Causal:** Ensures dependent transactions respect dependencies (withdrawal before transfer).
- **Total Order:** All banks see the same transaction log, enabling reconciliation.

Conclusion: Banking requires **total order**, since partial orders can cause inconsistent balances.

Q5. Given a social media system with likes, shares, and comments, compare the impact of using causal vs. total order.

Answer:

- **Causal:**
  - A comment always follows the post it refers to.
  - A like appears after the original post.
  - Different users may still see posts in different overall orders.
- **Total Order:**
  - Everyone sees likes, shares, comments in the same global sequence.
  - Stronger consistency but higher overhead.

Trade-off: Causal = efficiency + logical correctness; Total = strict consistency.

Q6. In a sensor network, analyze trade-offs between FIFO and causal ordering.

Answer:

- **FIFO:** Simple, less message complexity.
  - Useful when only one sensor sends to server.
- **Causal:** Needed when sensor readings depend on each other (e.g., "temperature high" → "fan ON").
  - More complex (vector timestamps).

Trade-off: FIFO = low cost, Causal = correctness under dependencies.

Q7. Apply the Chandy–Lamport snapshot algorithm to capture a consistent global state in a distributed system with FIFO channels. Show the steps briefly.

Answer:

- **Steps:**
  1. Initiator records local state and sends **marker** on all outgoing channels.
  2. On receiving marker first time, a process records local state, then forwards marker.
  3. Messages arriving after local state but before marker are part of channel state.
  4. Collection of all local states + channel states = consistent global state.

Guarantee: Works with FIFO channels, ensures no message is "in transit twice."

Q8. Demonstrate how the Lai–Yang algorithm works for snapshot collection in a system with non-FIFO channels. Use the color marking process.

Answer:

- **Lai–Yang uses coloring:**
  - Processes initially white.
  - On snapshot initiation, initiator turns red and records state.
  - When a red process sends a message, it marks it red.
  - If a white process receives a red message, it records local state and turns red.

Benefit: Works without FIFO channels, since color ensures in-transit messages are tracked.

Q9. Design a propagation tree for multicast in a peer-to-peer video streaming service. Explain design choices.

Answer:

- Root = streaming source.
- Internal nodes = peers relaying video.
- Leaves = viewers.
- **Design choices:**
  - Balance depth to reduce delay.
  - Avoid redundant forwarding.
  - Assign strong peers as intermediaries.

This ensures scalability and efficient bandwidth use.

Q10. A chat app guarantees FIFO message delivery, but users still see out-of-order messages. Analyze using causal ordering.

Answer:

- **Reason:** FIFO guarantees per-sender order, but not inter-sender causality.
- Example: Alice posts “Hi” → Bob replies “Hello.”
- If messages travel differently, Bob’s “Hello” may arrive before Alice’s “Hi.”
- **Solution:** Use causal ordering so dependencies are preserved.

Q11. Two processes send and receive updates simultaneously. Construct a timeline using logical clocks and analyze causality.

Answer:

- Suppose P1 sends event e1, P2 sends e2 concurrently.
- Lamport clocks: e1=2, e2=2 → appear ordered but may be concurrent.
- Vector clocks:
    - e1 = [1,0] ; e2 = [0,1].
    - Incomparable → correctly recognized as concurrent.

Q12. In a collaborative document editing system, explain how out-of-order delivery affects the state of the document. Suggest a solution.

Answer:

- **Problem:** User A deletes line 3, User B edits line 3 concurrently.
- If order is inconsistent, some copies show deleted text, others show modified text.
- **Solution:** Use causal or total ordering (vector clocks + conflict resolution).

Q13. Consider a distributed banking system. Describe how global state helps detect consistent account balances.

Answer:

- Each branch records local balance.
- Messages represent inter-branch transfers.
- **Global snapshot** ensures:
    - No transfer is counted twice.
    - No transfer is lost.
- Result: Bank can prove total balance = constant.

Q14. Explain the system assumptions necessary for the Chandy–Lamport snapshot to work. Provide a scenario where the model breaks down.

Answer:

- **Assumptions:**
    - Reliable channels.
    - FIFO delivery.
    - No failures during snapshot.
- **Breakdown:**
    - If channels drop messages, snapshot may miss some in-transit states.
    - If non-FIFO, marker order breaks consistency.

Q15. A distributed alarm system uses FIFO channels. Justify the use of Chandy–Lamport snapshot in maintaining system-wide consistency.

Answer:

- Alarm events must be globally correlated (e.g., smoke + temperature rise).
- Chandy–Lamport ensures a consistent cut where related events are recorded together.
- FIFO assumption makes it feasible without extra overhead.

Q16. Analyze the purpose and timing of marker messages in the Chandy–Lamport algorithm using a 3-process communication example.

Answer:

- **Purpose:** Delimit "before snapshot" and "after snapshot" messages.
- **Example:** P1 initiates snapshot, sends marker → P2, P3.
- On receiving first marker, P2 records state.
- Messages arriving before marker = channel state; after marker = normal.

Thus, markers separate consistent global cut.

Q17. Global State – Evaluate how a consistent global state helps in deadlock detection using a distributed resource allocation scenario.

Answer:

- Each process holds locks and waits for others.
- Local views may not detect cycles.
- Global snapshot reveals **wait-for graph** across all nodes.
- If cycle exists globally → deadlock detected.

Q18. A monitoring system has non-FIFO communication channels. Simulate Lai–Yang's snapshot and explain how consistency is ensured.

Answer:

- Initiator turns red, records state.
- Sends red message.
- White process receives red message → records state → turns red.
- Ensures in-transit messages are captured via color.
- **Consistency preserved without FIFO assumption.**

Q19. Describe Helary's wave synchronization method and compare it with Chandy–Lamport.

Answer:

- **Helary's Wave Method:**
  - Uses a "wave" of control messages.
  - Coordinators collect acknowledgments from all processes.
  - Ensures synchronization without needing FIFO.
- **Comparison:**
  - Chandy–Lamport: lightweight, FIFO assumption.
  - Helary: more message overhead but stronger coordination.

Q20. Given a time-sensitive distributed application, analyze whether Chandy–Lamport or Helary's wave method is more suitable.

Answer:

- **Chandy–Lamport:** Good for low overhead, but assumes FIFO. Not ideal for time-critical apps with non-FIFO.
- **Helary's Wave:** Stronger, works under weaker assumptions. More reliable for real-time coordination, despite overhead.

Conclusion: For time-sensitive apps, **Helary's wave method** is more suitable.

DC-HA-2

Q1. Discuss the importance of color-coding in the Lai–Yang snapshot method. Give an example showing how it captures local state.

Answer:

- **Color-coding (White/Red):**
  - **White state:** before recording snapshot.
  - **Red state:** after recording snapshot.
- **How it works:**
  1. Initially, all processes and messages are **white**.
  2. Initiator records local state, turns **red**, and marks all subsequent messages red.
  3. Any process receiving a red message records local state and turns red.

Example:

- P1 initiates snapshot → turns red, records state.
- Sends message m (red) to P2.
- P2 (white) receives red m → records its state (white state) and turns red.
- This guarantees **all in-transit messages are captured** in snapshot.

Importance: Color-coding ensures **consistent global state** even without FIFO channels.

Q2. Compare the snapshot algorithms (Chandy–Lamport, Helary, Lai–Yang) based on their applicability in heterogeneous systems.

Answer:

- **Chandy–Lamport:**
  - Works on **FIFO channels**.
  - Lightweight, but limited if channels are non-FIFO.
- **Lai–Yang:**
  - Works on **non-FIFO channels** using color marking.
  - More general for heterogeneous systems.
- **Helary's Wave:**
  - Synchronizes processes via wave of control messages.
  - Suitable for **large-scale, heterogeneous systems** with unpredictable delays.

Comparison:

- For **FIFO, simple systems** → Chandy–Lamport.
- For **non-FIFO, dynamic environments** → Lai–Yang.
- For **complex heterogeneous systems** with high coordination → Helary.

Q3. What happens if causal ordering is violated in a collaborative app?

Answer:

- **Violation consequences:**
  - Replies may appear before original messages.
  - Edits may apply to outdated document versions.
  - Example: Alice posts *"Start exam at 9"*, Bob replies *"Okay"*. If causal order is broken, users may see Bob's reply without Alice's instruction.

Impact: Leads to **inconsistency, confusion, and incorrect document/chat states**.

Q4. Give an example where a global state snapshot results in inconsistency. Explain how to correct or avoid it.

Answer:

- **Example:**
  - P1 sends $10 transfer to P2.
  - Snapshot records P1 after sending, but P2 before receiving.
  - Global state: P1 = −10, P2 unchanged → total money reduced.

Correction:

- Use **Chandy–Lamport or Lai–Yang** snapshot to include in-transit messages.
- Ensures no money "lost" in snapshot.

Q5. How do snapshot algorithms ensure consistency across asynchronous systems? Support with a timing diagram.

Answer:

- **Mechanism:**
  - Marker (Chandy–Lamport) or color (Lai–Yang) identifies a boundary in event timeline.
  - Ensures all local states + in-transit messages are captured.

Timing diagram idea:

- P1 sends message → marker sent.
- P2 receives message before marker → counts in local state.
- If after marker → counts in channel state.

Thus, snapshot = **consistent cut**, no missing or double-counted events.

Q6. A blockchain system requires global snapshots of states. Discuss which snapshot algorithm best suits this use case and why.

Answer:

- Blockchain = **asynchronous, decentralized, non-FIFO channels**.
- **Best choice: Lai–Yang** (handles non-FIFO).
- **Why not Chandy–Lamport?** Relies on FIFO assumption, unsuitable.
- **Why not Helary?** Too heavy for frequent global snapshots.

Conclusion: Lai–Yang balances correctness with efficiency for blockchain networks.

Q7. Analyze the need for synchronizers in an asynchronous distributed system running synchronous algorithms.

Answer:

- Many algorithms assume **synchronous rounds**.
- Real systems are **asynchronous** (unknown message delays).
- **Synchronizer:** Virtual layer that simulates synchrony by grouping asynchronous events into logical rounds.

Need: Enables use of synchronous algorithms in real asynchronous settings.

Q8. A network with unknown delays must simulate synchronous behavior. Explain how a synchronizer achieves this and trade-offs.

Answer:

- **How:**
  - Each process waits for acknowledgments or control messages before moving to next round.
  - Guarantees all nodes complete one round before next.
- **Trade-offs:**
  -
    - Enables synchronous correctness.
  - − Adds message overhead, increases latency.

Q9. Analyze how the Chandy–Lamport snapshot algorithm ensures a consistent global state in a system with FIFO channels.

Answer:

- **FIFO property:** Marker divides messages into before/after snapshot.
- Messages before marker → part of local state.
- Messages after marker → recorded in channel state.
- Ensures no lost or duplicated events.

Thus, snapshot = consistent global cut.

Q10. In a distributed simulation of cellular automata, explain how synchronizers ensure correct progression of time steps.

Answer:

- Simulation requires **step-by-step progression** (t, t+1, …).
- Synchronizer ensures:
  - All nodes finish step t before starting t+1.
  - Prevents one node from advancing while others lag.
- Without synchronizer → inconsistent evolution of simulation grid.

Q11. A synchronizer fails in one node. Analyze consequences in distributed simulation and recovery.

Answer:

- **Consequences:**
  - Node may advance out of sync.
  - Simulation correctness compromised.
  - Possible deadlock if others wait indefinitely.
- **Recovery:**
  - Timeout + re-election of coordinator.
  - Restart synchronizer process at failed node.

Q12. A distributed file system needs a new leader after a failure. Simulate the bully algorithm and analyze message complexity.

Answer:

- **Bully algorithm steps:**
  1. Node with highest ID takes over as leader.
  2. On leader failure, a lower-ID node starts election.
  3. It contacts higher-ID nodes.
  4. Highest alive ID becomes new leader.
- **Message complexity:** $O(n^2)$ in worst case (each node may contact all higher ones).

Q13. Discuss a scenario where the bully algorithm performs poorly and suggest improvements.

Answer:

- **Poor performance:**
  - Large systems with frequent failures.
  - Many messages exchanged during elections → overhead.
- **Improvements:**
  - Use **Ring or Paxos-based election** (fewer messages).
  - Limit retries with timeouts.
  - Use **hierarchical election** in large networks.

Q14. Given a snapshot record of messages and process states, analyze whether the global state indicates termination.

Answer:

- **Termination detection:**
  - Check if all processes are passive (no local events).
  - Check if all channels are empty (no in-transit messages).
- If both hold → system terminated.
- If any active process or in-transit message exists → not terminated.

Q15. Suppose two nodes mistakenly have the same UID. Analyze how this affects correctness of the LCR algorithm.

Answer:

- **LCR algorithm:** Each node circulates UID; highest UID wins.
- **Problem:** Duplicate UIDs → two nodes think they are leaders.
- **Impact:** Violation of uniqueness → correctness fails.

Fix: Use tie-breaker (e.g., IP address) or randomized ID assignment.

Q16. In a dynamic IoT environment, which leader election method adapts best? Justify.

Answer:

- **Best choice: Ring-based election.**
- **Reason:**
  - Works well in dynamic, low-power networks.
  - Handles nodes joining/leaving efficiently.
  - Message complexity = O(n), better for resource-limited IoT.

Q17. After a leader is elected, another node with a higher ID joins. Analyze how each algorithm handles this.

Answer:

- **Bully:** New higher-ID node immediately initiates new election → becomes leader.
- **Ring:** Node joins ring, next election will consider it.
- **LCR:** Must re-run election across full ring → overhead.

Q18. In a cloud environment with multiple clusters, how can leader election algorithms be extended for hierarchical leadership?

Answer:

- **Hierarchical approach:**
  - Each cluster elects local leader (using Bully/Ring/LCR).
  - Local leaders form a higher-level ring or quorum.
  - Global leader elected among them.

Benefits:

- Scalability.
- Fault isolation.
- Reduces overhead compared to flat election across all nodes.