ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY
UNIVERSITY FOR INDUSTRY

College of Engineering

Department of Software Engineering

Distributed Systems

Individual Assignment - II

NAME : - ANTENEH GETNET

ID NO. :- ETS0200/14

SECTION A

Submitted to: Mr. Kassahun Admeke

Submitted Date :11 - 27 - 2025

2.0 Clock Synchronization

In a distributed environment, the notion of time is ill-defined because physical clocks drift and there is no universal time source accessible instantly by all nodes. This makes determining the exact order of events challenging. Clock synchronization techniques aim to provide a meaningful ordering of events across different machines.

2.1 Physical Clocks and UTC

Sometimes, the absolute, exact time is required. The global standard is Universal Coordinated Time (UTC), based on the transitions of the cesium-133 atom. UTC is broadcast via short-wave radio and satellites (achieving accuracy of about ±0.5 ms).

The goal of clock synchronization is defined by two metrics:

- **Precision (π):** Keeping the deviation between any two clocks within a specified bound:
  For all t, p, q: $|C_p(t) - C_q(t)| \leq \pi$ (internal synchronization).
- **Accuracy (α):** Keeping a clock bound to the actual UTC time:
  For all t, p: $|C_p(t) - t| \leq \alpha$ (external synchronization).

Hardware clocks have a maximum clock drift rate (ρ), where the oscillator frequency *F(t)* relates to the ideal frequency *F* such that:
$(1 - \rho) \leq F(t)/F \leq (1 + \rho)$.

2.2 Logical Clocks: Lamport Timestamps

Logical clocks, proposed by Leslie Lamport, provide a mechanism for ordering events based on the **happens-before** relation (→). This defines a partial ordering of events consistent with potential causality:

1. If *a* and *b* are events in the same process, and *a* occurs before *b*, then **a → b**.
2. If *a* is the sending of a message and *b* is the receipt of that message, then **a → b**.
3. Transitivity: if **a → b** and **b → c**, then **a → c**.

Lamport's algorithm uses a simple counter. A process increments its counter before every event. When sending a message *m*, process *Pi* includes its timestamp **Ci**.

The receiving process *Pj* updates its clock:

$Cj = \max(Cj, Ci) + 1$ before processing the message.

This ensures: if **$a \rightarrow b$**, then **$C(a) < C(b)$** (total ordering).
However, Lamport timestamps do *not* detect true causality; two events may be concurrent even if timestamps differ.

2.3 Vector Clocks

Vector clocks overcome Lamport's limitations by capturing full causality.
For a system with *N* processes, a vector clock is:
$V = [v_1, v_2, \ldots, v_n]$,
where v_i counts events at process *Pi*.

Rules:

- Before any event, **Pi increments $V_i[i]$**.
- When sending a message, *Pi* includes the entire vector **V_i**.
- The receiver *Pj* updates:
  $V_j[k] = \max(V_j[k], Vmessage[k])$ for all k,
  then increments **$V_j[j]$**.

Causality:

- **$A \rightarrow B$** iff **$VA < VB$** (every element ≤ and at least one <).
- Otherwise, A and B are **concurrent**.

Vector clocks are widely used in distributed storage systems for conflict detection.

3.0 Mutual Exclusion

Mutual exclusion ensures that only one process at a time accesses the **Critical Section (CS)**.
Requirements:

- **Safety:** at most one process in the CS
- **Liveness:** every requesting process eventually enters

- **Fairness:** bounded waiting

3.1 Centralized and Decentralized Algorithms

Centralized Algorithm:

A single coordinator grants CS access.

Simple but suffers from:

- single point of failure
- performance bottleneck

Decentralized Algorithm:

A process must obtain permission from a **majority (quorum)** of nodes.

Avoids single point of failure but may:

- deadlock if two processes get half-votes
- require complex recovery

3.2 Distributed Algorithm: Ricart–Agrawala

A fully distributed algorithm using logical timestamps to order requests.

Steps:

1. Process *Pi* broadcasts a REQUEST containing **timestamp Ti** and **process ID i**.
2. Receiver *Pj* replies **OK** if:
    - it is not requesting the CS, or
    - *(Ti, i)* has higher priority than *(Tj, j)*.
3. *Pj* **defers** the reply if it is in the CS or has higher-priority request.
4. *Pi* enters CS after receiving **OK from all N–1 processes**.

Requires **2(N−1)** messages per entry → high overhead but no single point of failure.

4. Election Algorithms

Election Algorithms are used to reliably select a new Coordinator or Leader when the current one fails or the system initializes. This is vital for algorithms like centralized mutual exclusion and consensus protocols.

4.1 The Bully Algorithm

The Bully Algorithm is designed for systems where processes are ordered (e.g., by ID). The process with the highest ID that is alive is always elected.

Mechanism:

1. A process P initiates an election by sending an ELECTION message to all higher-ID processes.
2. If no higher process responds (after a timeout), P wins and broadcasts a COORDINATOR message.
3. If a higher-ID process Q responds with an OK, P stops its election, and Q starts its own election attempt.

The process with the absolute highest active ID will eventually win the election and "bully" any lower-numbered initiators into submission.

4.2 The Ring Algorithm

The Ring Algorithm is used in systems logically organized in a ring.

Mechanism:

1. An initiator P creates an ELECTION message containing its ID and sends it to its successor.
2. The message circulates the ring, with each active process adding its own ID to the list of participants.
3. When the message returns to P, the initiator sees the complete list, chooses the highest ID as the new leader, and broadcasts a second COORDINATOR message to inform all nodes.

This method guarantees only one election occurs at a time but is dependent on the ring structure's integrity and can have high message latency for large rings.

5.0 Gossip-based Coordination

Gossip protocols (epidemic protocols) are decentralized, lightweight, and robust methods modeled after the spread of rumors. They achieve large-scale consistency and coordination by having nodes periodically select a few random neighbors and exchange state information.

5.1 Mechanism: Anti-Entropy vs. Rumor Mongering

Anti-Entropy (Reconciliation): Focuses on correcting system inconsistencies by exchanging full or partial state (e.g., file versions). Push/Pull is the most effective mode for fast state convergence.

Rumor Mongering (Dissemination): Focuses on quickly spreading new information ("rumors"). An "infected" node pushes the rumor to random neighbors. The key challenge is Termination, ensuring the rumor stops spreading after broad dissemination to conserve bandwidth.

5.2 Application Examples

Gossip protocols are crucial for systems requiring high availability:

- ✓ Failure Detection: Used in distributed storage systems (like Amazon's Dynamo) to quickly disseminate information about node failures and recoveries across thousands of servers.
- ✓ Membership Services: Maintains a consistent, eventually-up-to-date global view of which nodes are currently alive and participating in a cluster.
- ✓ Data Aggregation: Used for probabilistic aggregation of data, such as calculating average load or temperature across a cluster without relying on a central server.

6. Distributed Event Matching

Distributed Event Matching is the core component of Publish/Subscribe (Pub/Sub) systems, decoupling publishers from subscribers. Events are produced by publishers and consumed by subscribers, with the matching system handling the routing.

6.1 Publish/Subscribe Systems

- ✓ Topic-Based Pub/Sub: Events are routed based on abstract channels or topics (e.g., stock_quotes). Subscribers express interest in these specific topics. Example: Kafka.
- ✓ Content-Based Pub/Sub (CBR): Subscribers specify interest using complex logical expressions (predicates) over the event content (e.g., "all stock updates where ticker='GOOG' AND price > 1800"). The matching engine must evaluate the event against all active subscriptions.

6.2 Content-Based Routing (CBR)

In distributed CBR, the goal is efficient routing without global knowledge of every subscription. Routers employ Subscription Advertisement and Filter Composition. A router advertises an aggregate filter representing the combined interest of its downstream neighbors. When an event arrives, the router only forwards it to a neighbor if the event matches that neighbor's aggregate filter, effectively pruning the dissemination tree and conserving network bandwidth.

7. Location Systems

Location systems identify the physical or logical position of resources in distributed environments.

7.1 Physical and Logical Location

Physical location systems such as GPS determine real-world geographic position, while logical location systems such as DNS resolve symbolic names to network addresses. Logical systems allow users to access services without needing to know their physical location, providing location transparency. This transparency is vital in distributed systems, where resources may move or be replicated.

7.2 Design Considerations

Effective location systems must be scalable, support dynamic mobility, and provide fast lookup times. Systems such as DNS achieve this through hierarchical naming and caching, while DHT-based systems achieve scalability by distributing lookup responsibilities across many nodes.

7.3 Challenges

Maintaining consistent and up-to-date location information is difficult in systems where nodes move frequently or where workload fluctuates. Avoiding bottlenecks and ensuring availability of the location service are also significant design concerns.

8. Integrated Examples and Case Studies

Distributed systems typically integrate multiple coordination mechanisms for comprehensive fault tolerance and consistency.

8.1 Case Study: Leader-Based Consensus in Distributed Databases

A sharded NoSQL store relies on several coordination mechanisms to maintain state consistency across replicas:

- ✓ Election (Bully/Ring): Selects a primary (leader) node for a data shard.
- ✓ Mutual Exclusion (Primary Lock): The leader implicitly holds the lock for writing, serializing updates.
- ✓ Clock Synchronization (Vector Clocks): The leader includes Vector Clocks with replicated state updates. Followers use these clocks to ensure updates are applied in the correct causal order and to handle conflict resolution during failure recovery.
- ✓ Gossip (Failure Detection): A continuous, background gossip protocol quickly detects leader failure, triggering a new election.

This integration—Election for authority, Mutual Exclusion via the leader, Clocks for consistency, and Gossip for fault tolerance—enables reliable large-scale distributed data management.

8.2 Case Study: Internet of Things (IoT) Monitoring

An IoT sensor network combines location and event matching for efficient operation:

- ✓ Location Systems: Sensors and actuators register their location and function with a central Directory Service.
- ✓ Distributed Event Matching: Sensors publish data events (e.g., {temperature: 55, location: A4}). Actuators or dashboards subscribe using Content-Based Routing filters (e.g., temperature > 50 AND location IN [A4, B4]). The event matching system efficiently routes the alert only to the relevant subscribing component (e.g., a fan in the specified zone), avoiding unnecessary network broadcast and conserving resources.

9. Conclusion

Coordination is the bedrock of reliable distributed systems. The techniques reviewed—from fundamental clock synchronization algorithms that define causality, to mutual exclusion protocols that guarantee resource integrity, and election methods that establish temporary authority—provide the tools necessary to manage the inherent complexity of a networked environment. Modern approaches like gossip protocols provide scale and resilience, while event matching decouples components for flexibility. The continuing focus on advanced Consensus Algorithms (like Paxos and Raft) and the integration of physical and logical time will drive the next generation of highly consistent and scalable geo-distributed systems.

References

[1] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7), 558–565.

[2] Ricart, G., & Agrawala, A. K. (1981). An optimal algorithm for mutual exclusion in computer networks. Communications of the ACM, 24(1), 9–17.

[3] Garcia-Molina, H. (1982). Elections in a distributed computing system. IEEE Transactions on Computers, C-31(1), 48–59.

[4] Van Renesse, R., & Birman, K. P. (2020). Gossip-based dissemination and aggregation in large-scale networks. ACM Transactions on Computer Systems (TOCS), 37(1), 1-35.

[5] Tanenbaum, A. S., & Van Steen, M. (2017). Distributed Systems: Principles and Paradigms (3rd Edition). Pearson.

[6] Cugola, G., & Margara, L. (2012). The internet of publications/subscriptions. ACM Transactions on Internet Technology (TOIT), 12(1), 1-46.