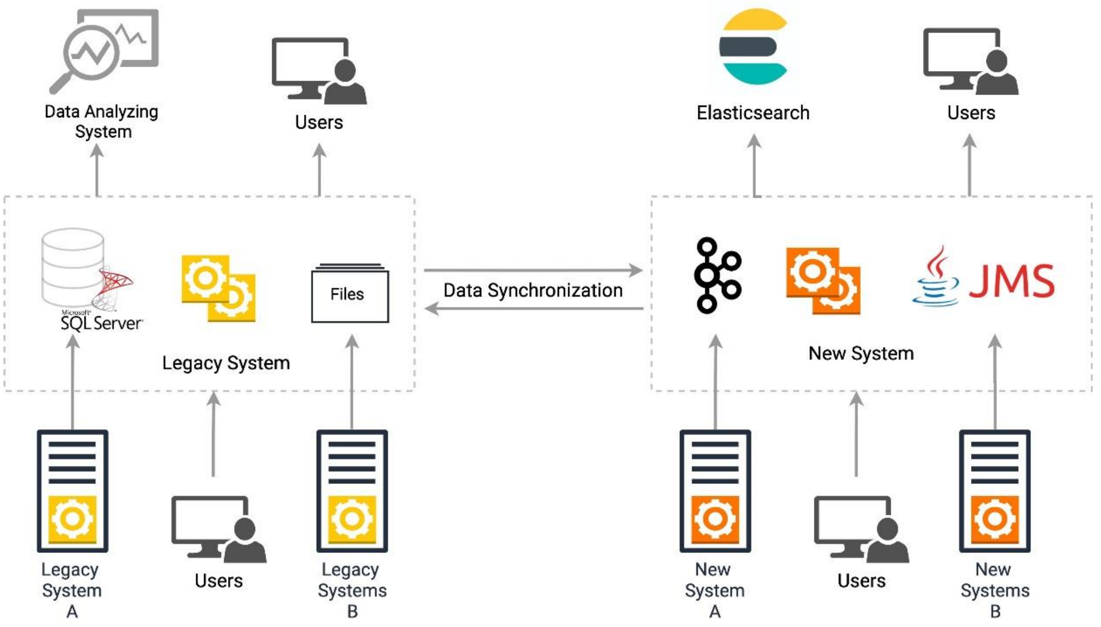
UNIT–III:

Synchronization in Distributed System, Clock Synchronization, Mutual Exclusion, Election algorithms, Atomic transactions, Deadlocks in Distributed Systems.

Synchronization in Distributed Systems

Introduction:

Synchronization in distributed systems refers to the coordination of processes and data across multiple, independent machines (nodes) in a network. These systems often need to ensure consistency, order of execution, or coordination among processes to function correctly. Due to the absence of a global clock and the possibility of network delays, synchronization becomes more complex in a distributed environment compared to centralized systems.



In distributed systems, synchronization is essential for:

- **Maintaining data consistency.**
- **Managing process coordination.**
- **Ensuring reliable communication between nodes.**

- Controlling access to shared resources to prevent conflicts.

Challenges in Synchronization

Distributed systems face several key challenges when it comes to synchronization:

Lack of Global Clock: Unlike single-node systems where a global clock can be used to synchronize events, distributed systems have no shared clock. Each node maintains its local clock, and clocks may differ across nodes due to clock drift.

Network Latency: Messages between nodes may take different amounts of time to travel, making it difficult to determine the order of events based solely on message arrival times.

Node Failures: Nodes in a distributed system may fail or become unreachable, making synchronization challenging if certain nodes are unavailable or slow.

Concurrency: Multiple processes may run simultaneously on different nodes, leading to issues like race conditions, deadlocks, or inconsistent states if synchronization is not handled properly.

Types of Synchronization in Distributed Systems

Synchronization can be broadly divided into two categories:

Clock Synchronization: Ensures that all nodes in the system agree on the current time or event order.

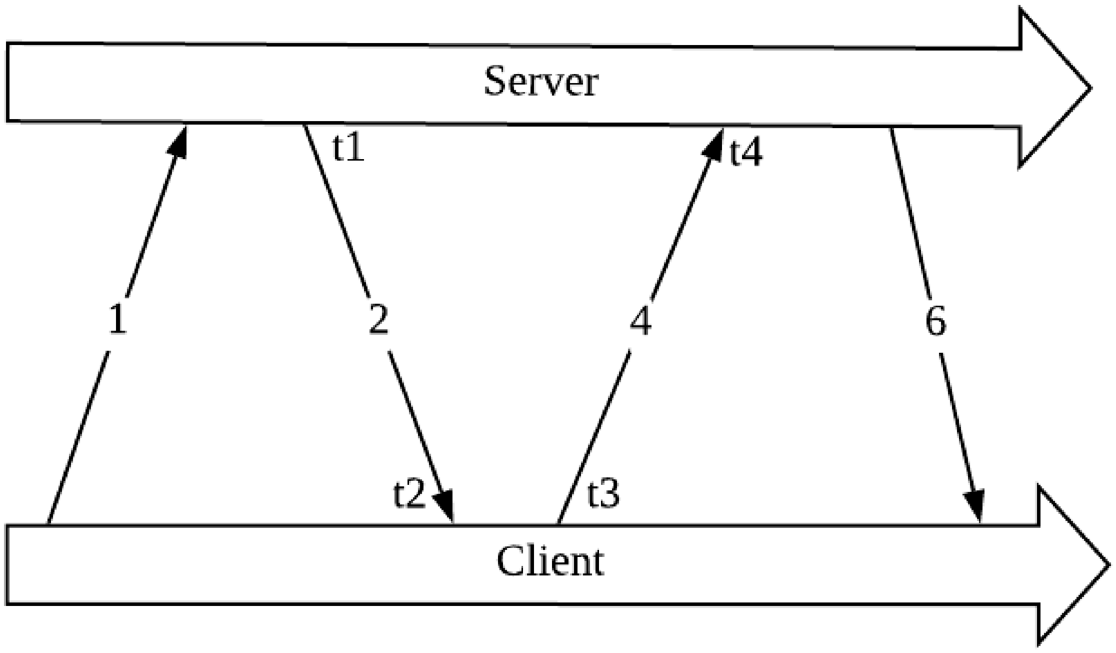
Data/Process Synchronization: Ensures that data or processes running on different nodes remain consistent and coordinated.

Practical Use Cases of Synchronization

- Distributed Databases: Synchronization ensures that data remains consistent across multiple replicas. Consensus protocols like Paxos or Raft are used to ensure that all nodes agree on updates to the database.
- Cloud Services: Cloud platforms rely on synchronization techniques to coordinate distributed services, such as load balancers and distributed file systems.
- File Systems: Distributed file systems like the Google File System (GFS) and Hadoop Distributed File System (HDFS) use synchronization mechanisms to manage file access and ensure data consistency across multiple machines.
- Distributed Locking: In distributed applications, distributed locks are used to control access to shared resources, ensuring only one instance of an application modifies the resource at a time

Clock Synchronization

Clock synchronization in distributed systems is the process of ensuring that multiple computers, or nodes, agree on the time. This is crucial for tasks such as ordering events, coordinating actions, or maintaining consistency in distributed systems where each node has its own local clock. Without a common time reference, it becomes difficult to determine the order of events and ensure that distributed systems function correctly.

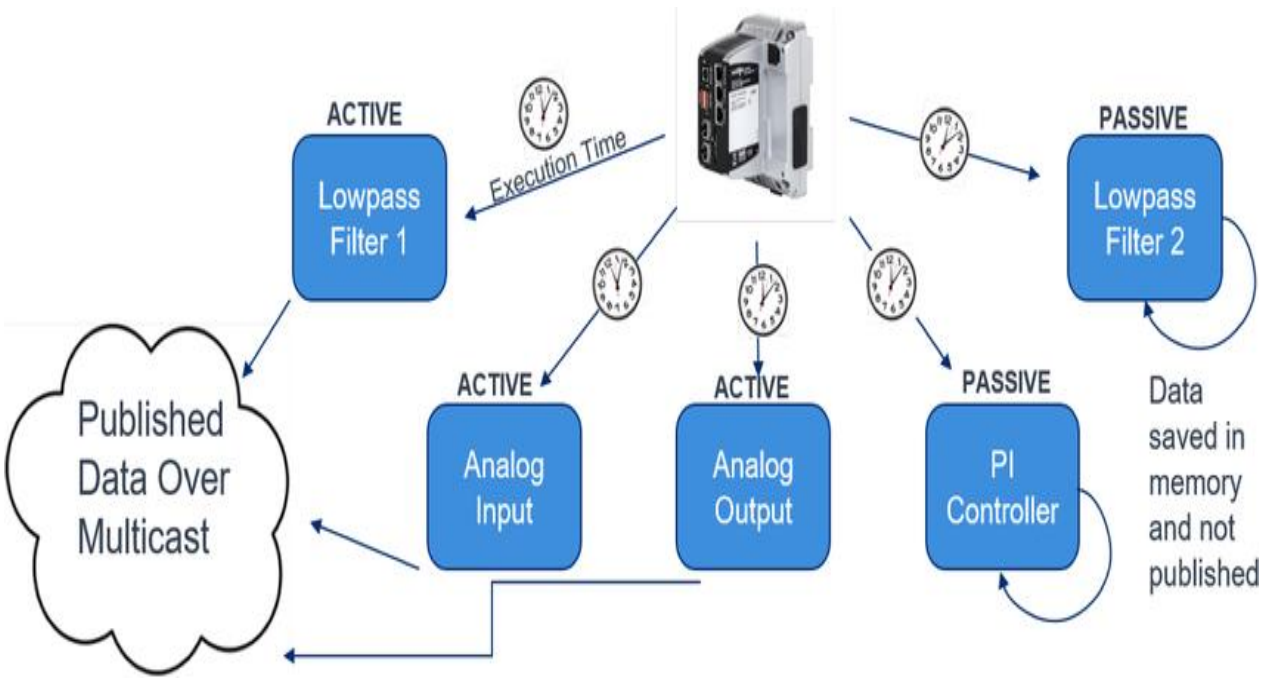


---

2. Challenges in Clock Synchronization

- **Absence of a Global Clock: Each node in a distributed system has its own local clock, which may drift over time due to differences in hardware. There is no global clock that provides a universally agreed-upon time.**
- **Clock Drift: Physical clocks are not perfectly accurate. They may drift apart (i.e., tick at different rates), leading to inconsistencies across nodes.**
- **Variable Network Delays: Communication between nodes can experience unpredictable network delays, making it challenging to synchronize clocks precisely.**

- **Fault Tolerance: Some nodes may fail, or there may be malicious behavior in the network. Synchronization protocols must handle failures gracefully and maintain clock consistency.**



3. Types of Clocks in Distributed Systems

- **Physical Clocks: These clocks attempt to represent actual wall-clock time (real-world time). Synchronizing physical clocks involves ensuring that all nodes agree on a common reference time.**
- **Logical Clocks: Instead of using real-world time, logical clocks track the order of events. They provide event sequencing but do not correspond to actual physical time.**

---

4. Physical Clock Synchronization

Physical clock synchronization ensures that all nodes in a distributed system have clocks that are as close as possible to a standard reference time, such as UTC (Coordinated Universal Time). The two main categories of physical clock synchronization are:

a. External Clock Synchronization

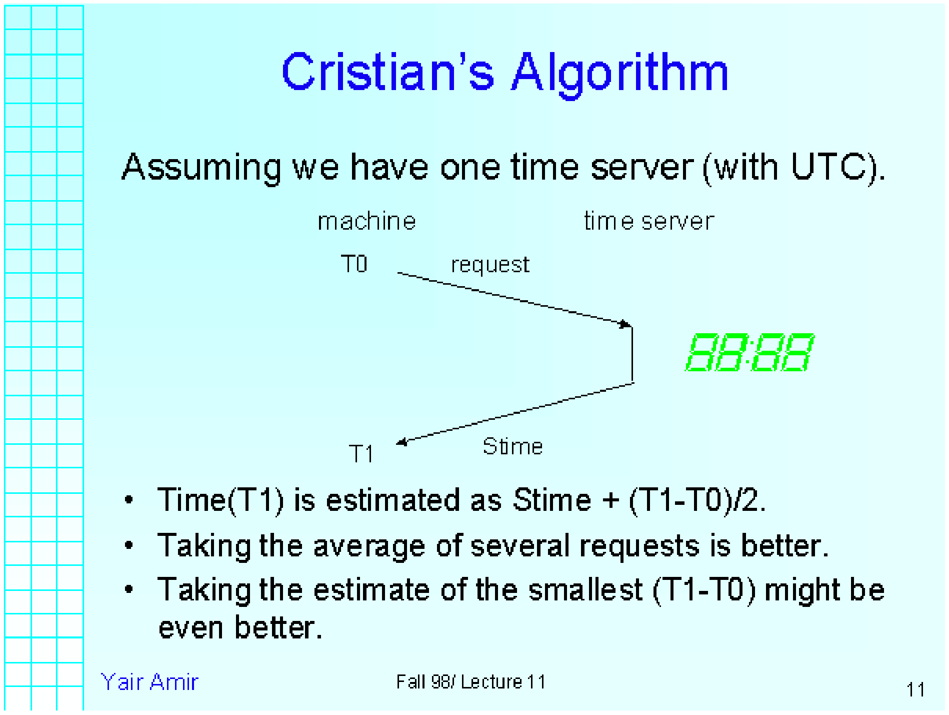
This involves synchronizing the clocks of nodes with an external, reliable time source like a time server.

Network Time Protocol (NTP):

- **Overview: NTP is one of the most commonly used protocols for clock synchronization. It synchronizes clocks over a network by adjusting for round-trip network delays.**
- **Structure: NTP uses a hierarchical system with stratum levels. Stratum 1 servers are directly connected to a high-precision time source (e.g., an atomic clock or GPS), while other servers (stratum 2, 3, etc.) synchronize their time from higher-stratum servers.**
- **Operation: NTP clients periodically send requests to NTP servers, and the servers respond with the current time. The client adjusts its clock based on the server's time, while also accounting for network delays.**
- **Accuracy: NTP can typically achieve synchronization accuracy within milliseconds on a local network and tens of milliseconds on a wide-area network.**
- **Offset and Delay Calculation:**
  - **Offset: The time difference between the client's clock and the server's clock.**
  - **Round-trip delay: The total time taken for a request to travel to the server and the server's response to return to the client. NTP uses round-trip delay to estimate how much time to adjust the client's clock by.**

Cristian's Algorithm:

- **Overview: This algorithm is used by a client to synchronize its clock with a time server.**

- 

Cristian's Algorithm

Assuming we have one time server (with UTC).

machine
time server
T0
request
88:88
T1
Stime

- Time(T1) is estimated as Stime + (T1-T0)/2.
- Taking the average of several requests is better.
- Taking the estimate of the smallest (T1-T0) might be even better.

Yair Amir Fall 98/ Lecture 11 11

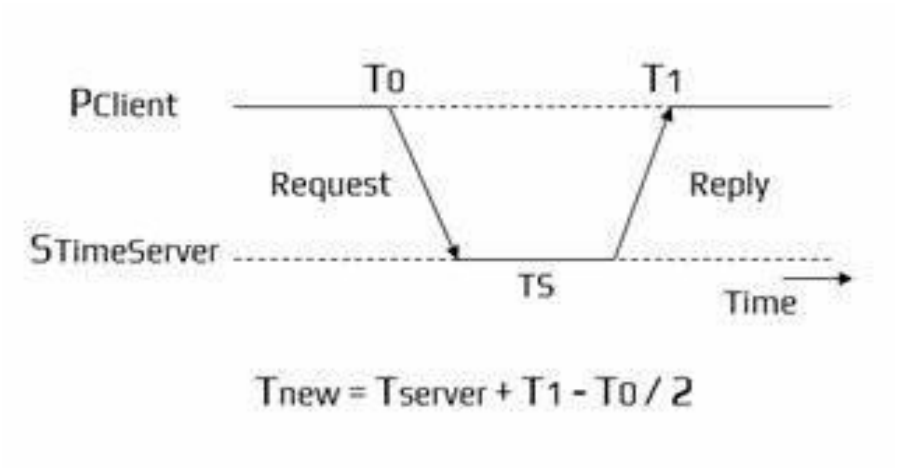
-

Process:

The client sends a time request to the server.

The server responds with its current time.

The client adjusts its clock to the server's time, considering the round-trip delay.

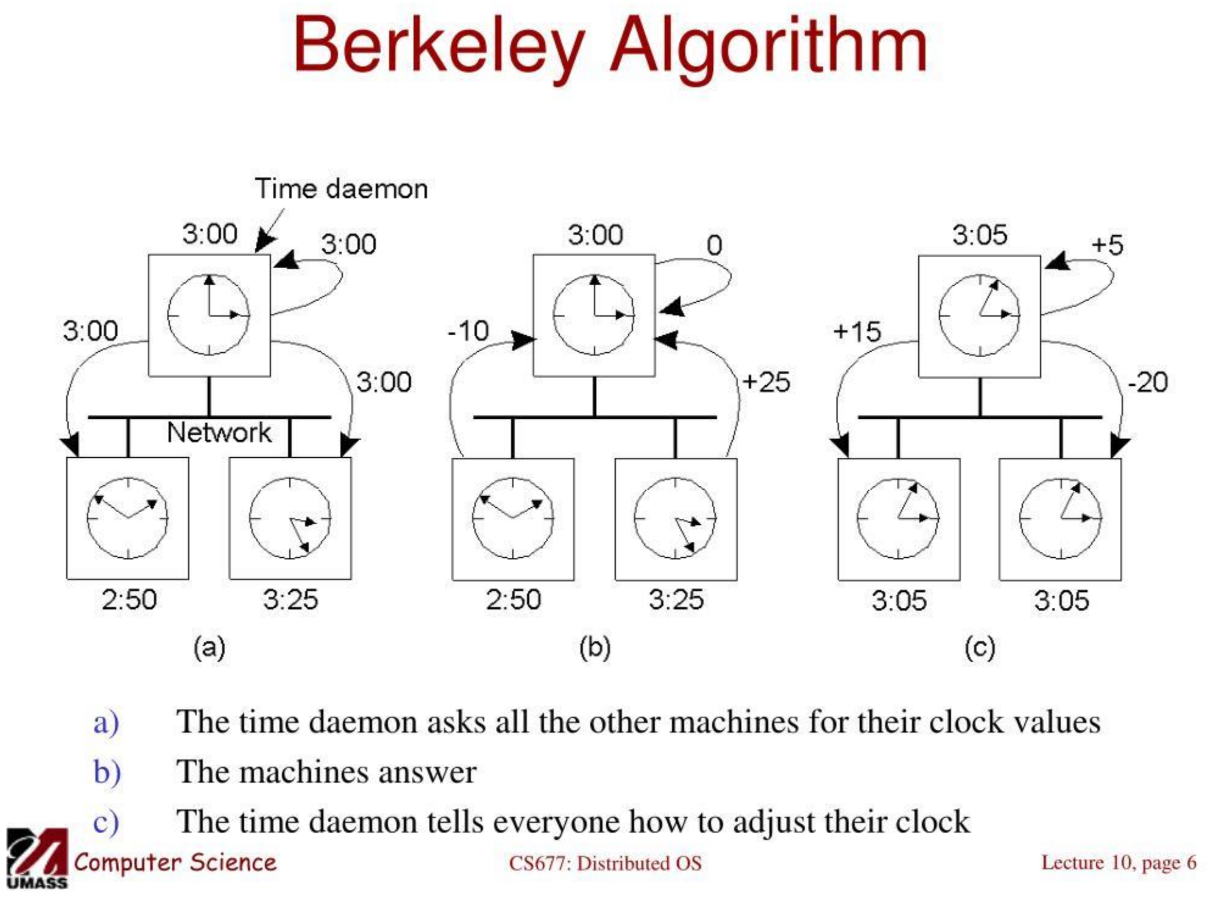


- **Assumption:** **It assumes that the round-trip delay is symmetric, meaning the time taken for a message to travel from the client to the server is the same as the time taken for the response to return.**

- **Limitation:** **If the network delay is unpredictable or asymmetric, the algorithm's accuracy can suffer.**

Berkeley Algorithm:

- **Overview: Unlike NTP and Cristian's Algorithm, which synchronize clocks with an external time source, the Berkeley Algorithm synchronizes clocks internally across nodes within a distributed system.**
- **Process:**



Berkeley Algorithm

a) The time daemon asks all the other machines for their clock values

b) The machines answer

c) The time daemon tells everyone how to adjust their clock

UMASS Computer Science CS677: Distributed OS Lecture 10, page 6

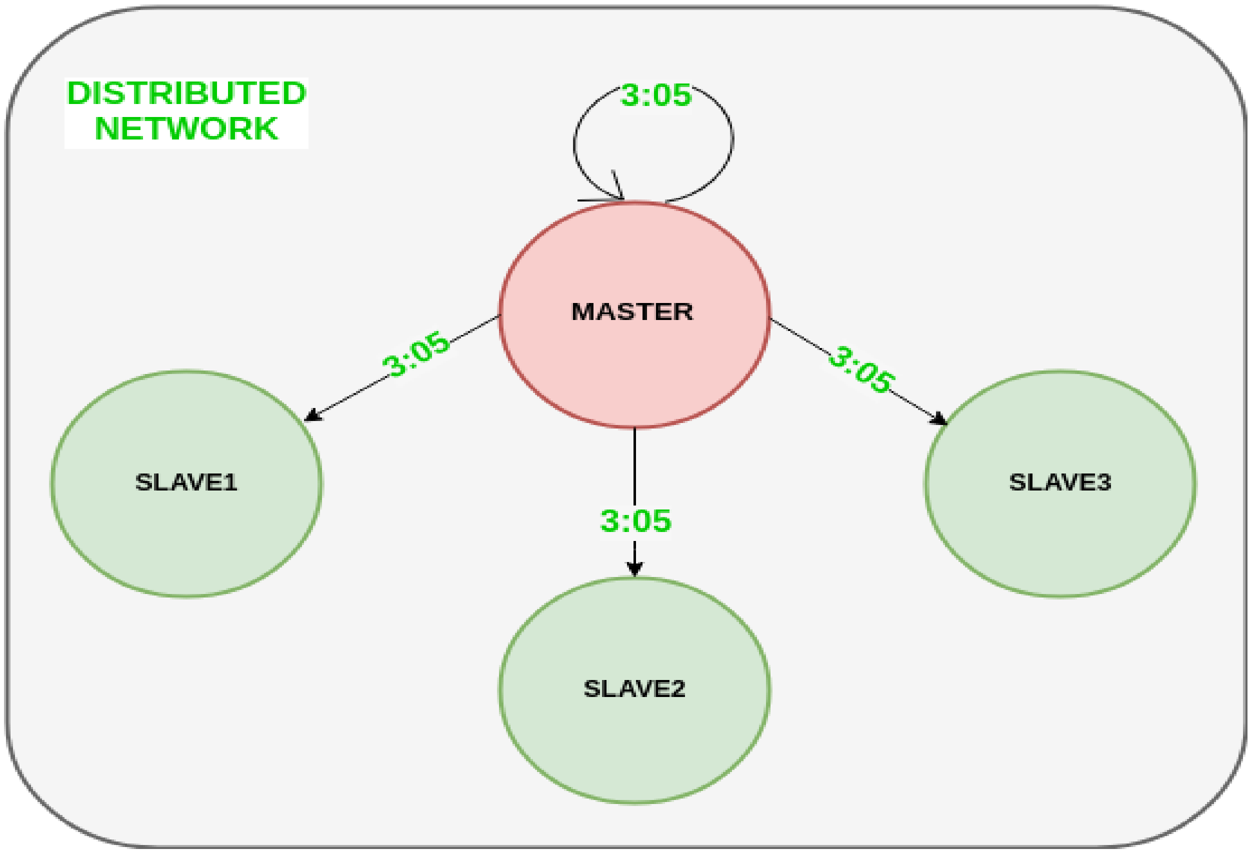
•

1. A coordinator node polls all nodes in the system to get their local times.

2. The coordinator computes the average of all times and calculates the offset for each node.

3. The coordinator instructs each node to adjust its clock by a certain amount (either forward or backward) to bring all clocks closer to the average.

- **Use Case: This algorithm is used in systems where no external reference clock is available or necessary.**



b. Internal Clock Synchronization

In this approach, nodes attempt to synchronize their clocks with each other rather than with an external time source. The goal is to ensure that the maximum difference between any two clocks in the system remains below a certain bound (called precision).

- **Bounded Drift Rate: Nodes attempt to maintain a bounded drift rate, ensuring that their clocks do not drift too far apart from each other. This can be achieved by periodically adjusting clocks to reduce discrepancies.**

Skew and Precision:

  - **Skew: The difference in time between two clocks at any given moment.**
  - **Precision: The maximum allowed skew between clocks in the system. Synchronization aims to keep the skew below this bound.**

---

5. Logical Clock Synchronization

Logical clocks provide a mechanism for ordering events in distributed systems without relying on physical time. These clocks help maintain the causal relationships between events, ensuring that distributed processes execute in a consistent order.

a. Lamport Timestamps

- **Overview: Introduced by Leslie Lamport, Lamport timestamps allow events in distributed systems to be ordered in a way that respects causality.**

Rules:

1. **Incrementing the clock: Each process increments its local logical clock for every event it processes.**
2. **Sending a message: When a process sends a message, it attaches its current logical clock value to the message.**
3. **Receiving a message: When a process receives a message, it updates its local clock to be greater than the maximum of its own clock and the received clock. This ensures that the receiving process's logical clock is always ahead of the sender's clock, maintaining causal order.**

- **Causal Order: If event A happens before event B (denoted as A → B), then the logical clock of A will be smaller than the logical clock of B. However, Lamport timestamps cannot detect concurrent events (events that are not causally related).**

b. Vector Clocks

- **Overview: Vector clocks improve upon Lamport timestamps by providing a mechanism to detect concurrency in addition to causal relationships.**
- **Structure: Each process maintains a vector of logical clocks, where each entry corresponds to the logical clock of another process in the system.**
- **Rules:**
    1. **Each process increments its own clock for each event it processes.**
    2. **When sending a message, a process attaches its vector clock to the message.**
    3. **When receiving a message, the receiver updates its vector by taking the element-wise maximum of its own vector and the received vector.**
    4. **Two events are causally related if the vector clock of one event is less than or equal to the vector clock of the other. If neither event's vector clock is less than the other, the events are concurrent.**

- **Concurrency Detection: Vector clocks can determine when two events are concurrent (i.e., they are not causally related). This is not possible with Lamport timestamps.**

---

6. Hybrid Approaches to Clock Synchronization

Hybrid approaches combine elements of both physical and logical clocks to achieve efficient and accurate synchronization.

- **Hybrid Logical Clocks (HLC):**
  - **Overview: HLC is a hybrid approach that incorporates both physical clocks and logical clocks to provide causality guarantees while leveraging real-time information.**
  - **Concept: Each event has a physical timestamp and a logical component. This hybrid approach ensures causal ordering while maintaining accuracy with real-world time.**

---

7. Skew, Drift, and Convergence in Clock Synchronization

- **Skew: The difference between two clocks at a specific point in time. Minimizing skew is crucial for maintaining synchronization between nodes.**
- **Drift: The rate at which clocks diverge over time. Drift occurs because no two clocks run at exactly the same speed. Clock synchronization algorithms periodically resynchronize clocks to mitigate drift.**
- **Convergence: The process by which multiple clocks in a distributed system become synchronized, reducing the skew between them. Convergence is essential for ensuring consistency in distributed systems.**

---

8. Clock Synchronization Algorithms in Practical Systems

- **Google Spanner: Spanner, Google's distributed database, uses TrueTime, a clock synchronization protocol that combines NTP with atomic clocks and GPS for highly accurate synchronization. TrueTime provides bounded uncertainty on clock synchronization, allowing Spanner to ensure strict consistency across data centers.**
- **Amazon Dynamo: Dynamo, a distributed storage system, uses vector clocks to track the causality of updates. This enables it to reconcile conflicts that arise due to concurrent updates by different nodes.**

9. Clock Synchronization in Fault-Tolerant Systems

Fault-tolerant clock synchronization is essential for maintaining consistency in distributed systems even in the presence of failures or network issues. Algorithms like Paxos and Raft rely on accurate clock synchronization to ensure consensus in distributed systems.

- **Fault Tolerance: Clock synchronization algorithms must handle network failures, message loss, and node crashes gracefully. Redundant communication, failure detection mechanisms, and quorum-based**