Chapter 5
TIME AND STATE IN DISTRIBUTED SYSTEM

1) Time in Distributed Systems

- Time is a fundamental concept that describes the sequence, duration and progression of events in the universe.
- In distributed systems, time refers to the concept of ordering and coordinating events across multiple nodes or processes that operate independently, often without a shared clock.
- Since nodes in a distributed system may be geographically dispersed and have their own local clocks, achieving a consistent notion of time is challenging due to clock skew, drift, and network latency.
- Time in this context is critical for ensuring consistency, coordination, and correct event ordering in distributed applications.
- The distributed system do not have global physical time. Each machine in a distributed system has its own clock providing the physical time.
- Time synchronization is essential to know at what time of day a particular event occurred at a particular computer within a system.

- Clock Synchronization deals with the understanding of temporal ordering of events produced by concurrent processes. It is useful for synchronizing senders and receivers.
- Multiple unrelated processes running on different machines should be able in agreement with and be able to make consistent decisions about the ordering of events in the system.
- For ordering of events to prevent duplicate updates and maintain consistency of the distributed data, we use timestamps.

2) Physical Clocks

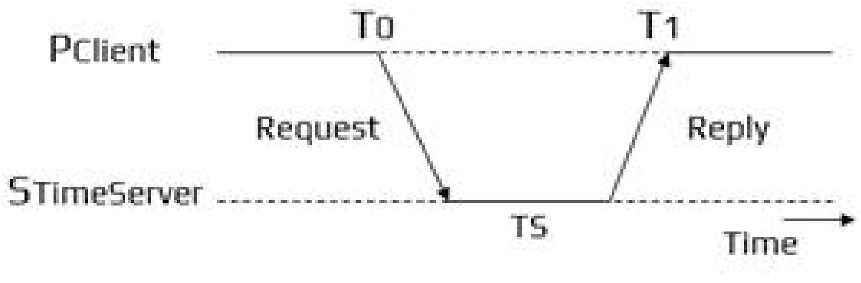
- Each node in distributed systems contains an electronic device that counts oscillations in a crystal at a definite frequency and store division of count to frequency in a register to provide time. Such device is called physical clock and the time shown is physical time.
- Since, different computers in a distributed system have different crystals that run at different rates, the physical clock gradually get out of synchronization and provide different time values. Due to this, it is very difficult to handle and maintain time critical real time systems. Consistency of distributed data during any modification is based on time factor.
- The n crystals on the n computers will run at slightly different rates, causing the clocks gradually to get out of synchronization and give different values.
- Thus it is difficult to handle deterministic systems.

- The algorithms for synchronization of physical clocks are as follows:

  - <u>Christian's method</u>
    - ✓ Cristian's algorithm relies on the existence of a time server as a physical clock synchronization algorithm.
    - ✓ The time server maintains its clock by using a radio clock or other accurate time source (UTC), then all other computers in the system stay synchronized with it.
    - ✓ A time client will maintain its clock by making a remote procedure call to the time server.
    - ✓ This method achieve synchronization only if the round-trip times between client and time server are sufficiently short compared to the required accuracy.
    - ✓ <u>Algorithm</u>:

      Cristian's Algorithm works between a process P, and a time server S connected to a source of UTC (Coordinated Universal Time).

      - P requests the time from S
      - After receiving the request from P, S prepares a response and appends the time T from its own clock.
      - P then sets its time to be T + RTT/2 ; RTT (Round Trip Time)



$$T_{new} = T_{server} + T_1 - T_0 / 2$$

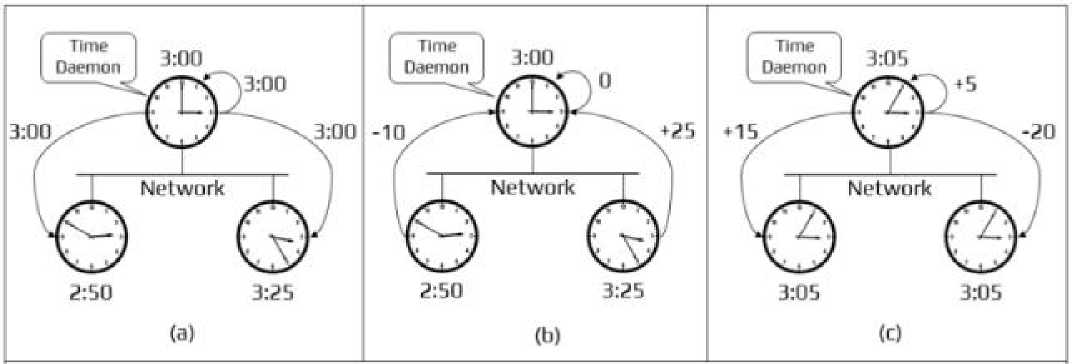
      - Further accuracy can be gained by making multiple requests to S and using the response with the shortest RTT.
      - If a time server fails, the synchronization is impossible.

✧ <u>Berkeley's method</u>

- ✓ Berkeley algorithm is also a physical clock synchronization algorithm based on centralized system. This algorithm is more suitable for systems where a radio clock is not present.
- ✓ This system has no way of making sure of the actual time other than by maintaining a global average time as the global time.
- ✓ A time server will periodically fetch the time from all the time clients, average the results, and then report back to the clients the adjustment that needs be made to their local clocks to achieve the average.

- ✓ It is an algorithm for internal synchronization. The steps of algorithm are as follows:

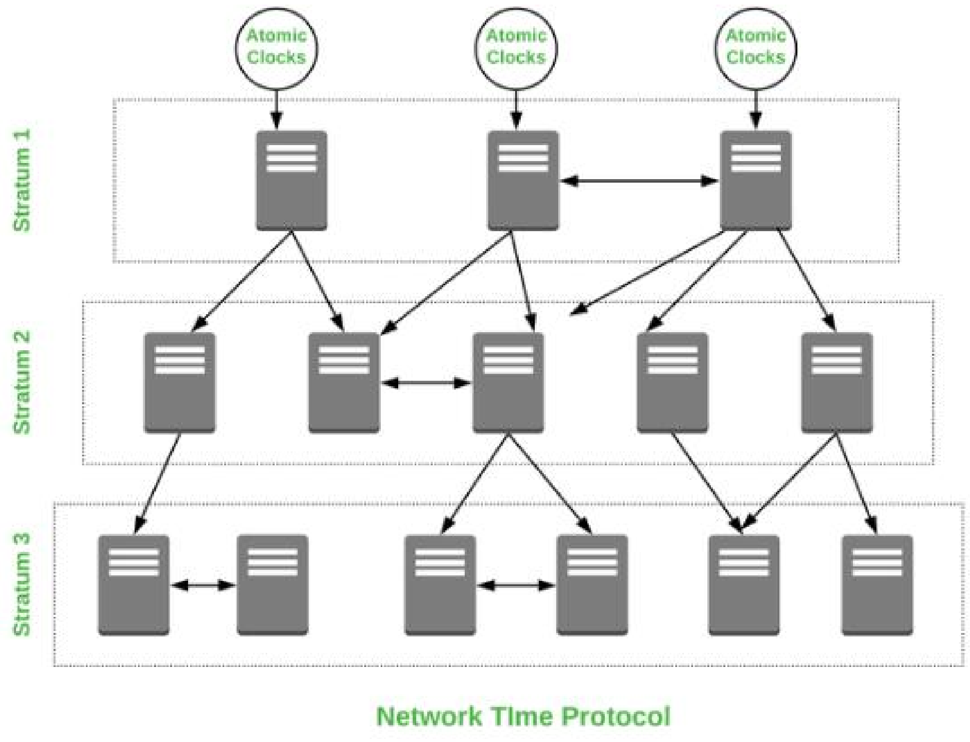
  i. A computer is chosen as a master.
  ii. All other computers are slaves.
  iii. Master periodically polls for the time of slaves and the slaves send back their clock values to master.
  iv. The master estimates local time of each slave by observing the round-trip times.
  v. Master calculates average of obtained time including its own time.
  vi. While calculating average, it eliminates faulty clocks by choosing a subset of clocks that do not differ from one another by more than a specified amount.
  vii. The master then sends the amount by which each slave should adjust their clock which may be positive or negative i.e. the master tells all the slaves to advance their clocks to new time or wait until some specific reduction has been achieved.
  viii. If the master fails, one of the slaves can be elected to take the place of master.



(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.

✧ <u>Network time protocol</u>

- ✓ An architecture that enable clients, across the Internet to be synchronized accurately to UTC.
- ✓ It synchronizes against many time servers.
- ✓ A machine will usually try to synchronize with many servers, using the best of all the results to set its time.



- ✓ Primary servers are connected directly to a time server. Secondary servers are synchronized with primary servers.
- ✓ The logical hierarchy of server connection is called synchronization subnet. Each level of synchronization subnet is called stratum.
- ✓ Lowest level executes in user's workstation.
- ✓ Server with high stratum numbers are liable to have less accurate clocks.

3) Logical Clocks

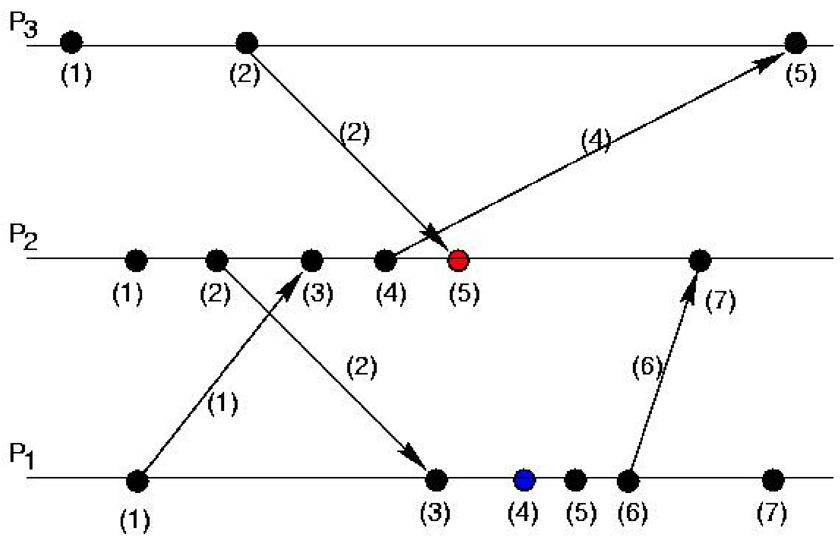
- A logical clock is defined as the clock that does not care about the time of the day at which an event occurred but that all the processes can agree on the order in which the related events occurred.
- Logical clock is a virtual clock that records the relative ordering of events in a process.
- It is a monotonically increasing software counter. It is realized whenever relative ordering of events is more important than the physical time.

- Physical clocks are not needed to be synchronized. The value of logical clock is used to assign time stamps to the events.

- <u>Lamport's Logical Clock</u>

- Lamport invented a simple mechanism by which the "happened before" ordering can be captured numerically.
- "->" is called as "happened before" relationship.

- <u>Algorithm:</u>
  i. If for some process P_i: a -> b, then a -> b.
  ii. For any message m, send(m) -> receive(m)
  iii. If a, b and c are events such that a->b and b-->c, then a -> c.
  iv. If a -> b, a casually affects event b.
  v. If a -> e and e -> a are false, then a and e are concurrent events, which can be written as a || e.



<u>For sending,</u>
time = timestamp + 1
send (message, time)

<u>For receiving,</u>
receive (message, time)
t_s = max(sender time, own time) + 1

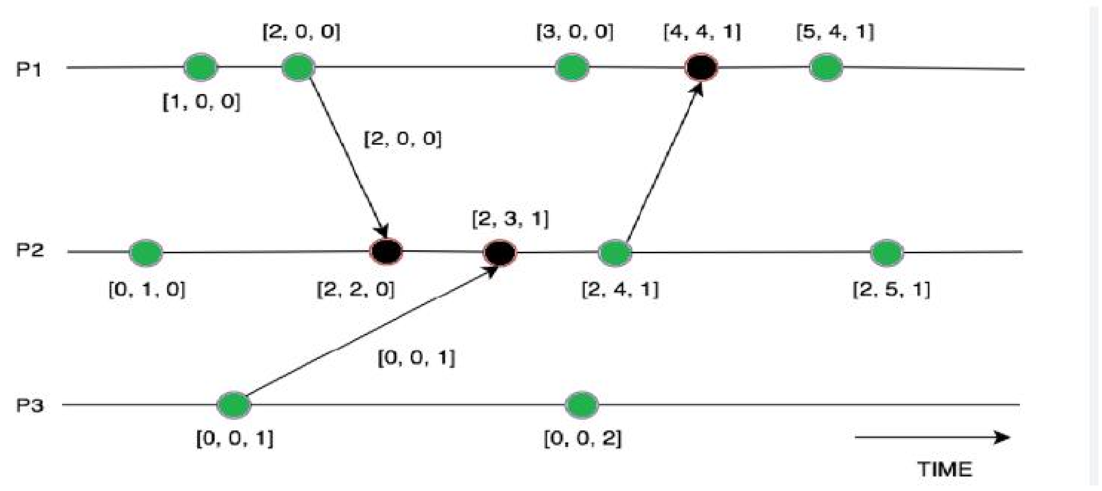
Goal: consistent ordering of events
Limitation: a → b implies L(a) < L(b) BUT ,L(a) < L(b) doesn't imply a → b.

4) Vector Clocks

- Mattern and Fidge developed a clock to overcome the shortcomings of Lamport's Logical Clock.
- Vector clock is a clock that gives ability to decide whether two events are causally related or not by looking at their time stamps.
- A vector clock for a system of N processes is an array of N integers. Each process keeps its own vector clock Vi used to time stamp local events.
- The disadvantage is that it takes more amount of storage and message payload proportional to the number of processes.

- <u>Algorithm</u>:
  i. Initially all clocks are zero.
  ii. Each time a process experiences an internal event, it increments its own logical clock in the vector by one.
  iii. Each time a process prepares to send a message, it increments its own logical clock in the vector by one and then sends its entire vector along with the message.
  iv. Each time a process receives a message, it increments its own logical clock in the vector by one and updates each element in its vector by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

- With vector clock, we have additional knowledge that if V(e) < V(e') then e->e'.



Assignment

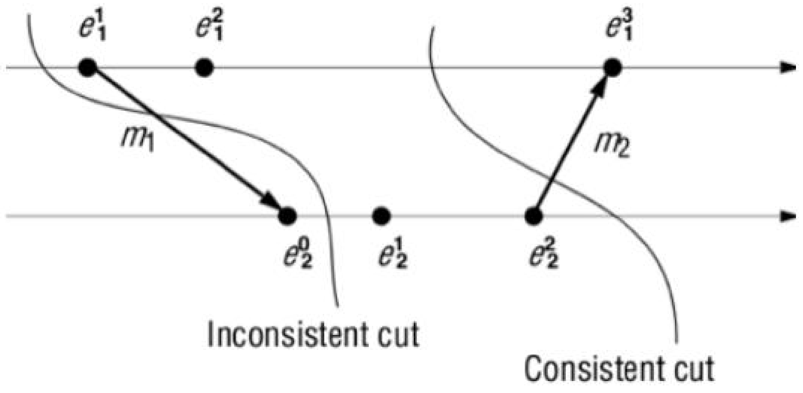
5) Clock Synchronization
6) Distributed debugging
7) Causal Ordering of Messages

- For any two casually related messages m1 and m2, if send(m1) -> send(m2) then for every recipient of m1 and m2, m1 must deliver before m2.
- But messages which are not casually related can be delivered by different nodes in different order.

8) Global State and State Recording

- In a distributed system, global state consists of local state of each process (message sent and message received) and state of each channel (message sent but not received).
- The challenge is to record the global state. The problem is due to lack of global clock because of which the local states are recorded at different time for each process.
- Uses:
  i. To track the usage of resources
  ii. To watch for interrupts
  iii. To mobilize the systems as per demands, etc.

- Consistent Global State
- A global state is consistent if for any received message in the state the corresponding send is also in the state.
- A cut C is said to be consistent, if for each event it contains, it also contains all the events that happened before that event.



- Importance:
  i. Distributed Deadlock Detection
  ii. Termination Detection
  iii. Checkpoints