SYNCHRONIZATION

• Clock Synchronization

A Distributed System is a collection of computers connected via a high-speed communication network. In the distributed system, the hardware and software components communicate and coordinate their actions by message passing. Each node in distributed systems can share its resources with other nodes. So, there is a need for proper allocation of resources to preserve the state of resources and help coordinate between the several processes. To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.

Clock synchronization is the mechanism to synchronize the time of all the computers in the distributed environments or system.

Assume that there are three systems present in a distributed environment. To maintain the data i.e. to send, receive and manage the data between the systems at the same time in a synchronized manner you need a clock that has to be synchronized. This process to synchronize data is known as Clock Synchronization.

Synchronization in distributed systems is more complicated than in centralized systems because of the use of distributed algorithms.

Properties of Distributed algorithms to maintain Clock synchronization:

- Relevant and correct information will be scattered among multiple machines.
- The processes make the decision only on local information.
- Failure of the single point in the system must be avoided.
- No common clock or other precise global time exists.
- In the distributed systems, the time is ambiguous.

As the distributed system has its own clocks. The time among the clocks may also vary. So, it is possible to synchronize all the clocks in a distributed environment.

Types of Clock Synchronization

- Physical clock synchronization
- Logical clock synchronization
- Mutual exclusion synchronization

● Physical Clock

A computer clock typically has a quartz crystal, a counter register and a constant register.

A constant register stores a constant value dependent on the quartz crystal's oscillation frequency.

The counter register decrements by 1 for each quartz crystal oscillation.

When the counter register reaches zero, an interrupt is issued and its value is reinitialized to the constant register. Each interrupt is termed a clock tick.

To make the computer clock work like a regular clock, the following steps are taken:

1. The constant register value is set to 60 clock ticks per second.
2. Computer clocks are synchronized with real time.
3. In UNIX, time starts at 0000 on January 1, 1970.
4. The system prompts the user to enter the current date and time during bootup.
5. The system calculates the ticks after the fixed starting date and time from the entered value.
6. To keep the clock running, the interrupt service routine advances the tick count at each clock tick.

Drifting of clocks:

- A clock always runs at a constant rate because its quartz crystal oscillates at a fixed frequency. However, due to crystal variances the rates at which two clocks run are generally different.
- The difference in oscillation period between two clocks may be quite minor but the difference collected over many oscillations leads to an observable difference

in timings of two clocks regardless of how precisely they were initialized to the same value.

- As a result, as time passes a computer clock drifts away from the real time clock that was utilized for its initial set up.

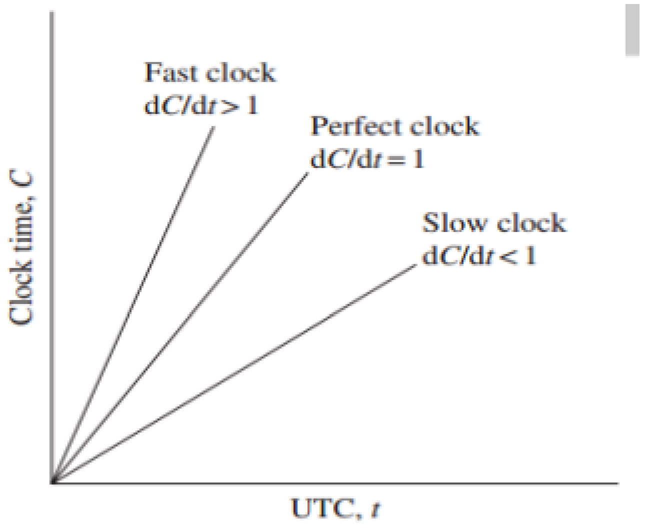


Fig : Behavior of clocks

- After synchronization with a perfect clock, slow and fast clocks drift in opposing directions from the perfect clock as shown in above figure.
- This is because dC/dt<1 for slow clocks and dC/dt>1 for fast clocks.
- A DS requires the following types of clock synchronization:
    a. <u>Synchronization of computer clocks with real time clocks</u>

    This type of synchronization is essential for real time applications. It enables the system to exchange information about the timing of events with other systems and users. The Coordinated Universal Time (UTC) is a popular external time source for synchronizing computer clocks with real time.

    b. <u>Mutual synchronization of the clocks of different nodes of the system</u>

    This type of synchronization is necessary for applications that demand a constant view of time across all nodes of DS,

Basic Terminologies

- Time: time of a clock
- Frequency: rate at which clock progress
- Offset: difference between the time reported by a clock and the real time.
- Skew: difference in the frequencies of the clock and the perfect clock

Physical Clock synchronization algorithms:

Every computer contains a clock which is an electronic device that counts the oscillations in a crystal at a particular frequency. Synchronization of these physical clocks to some known high degree of accuracy is needed. This helps to measure the time relative to each local clock to determine order between events.

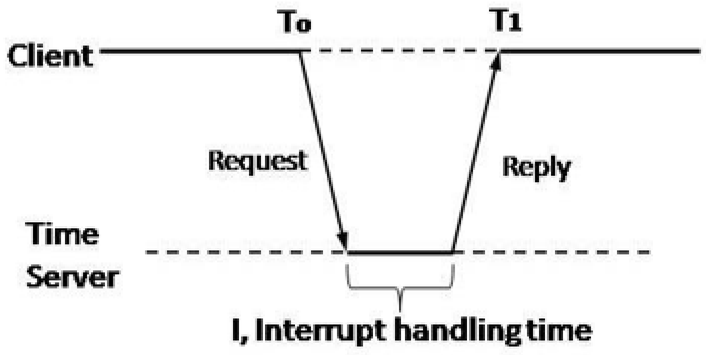
Physical clock synchronization algorithms can be classified as centralized and distributed.

1.Centralized clock synchronization algorithms

These have one node with a real-time receiver and are called time server nodes. The clock time of this node is regarded as correct and used as reference time.

The goal of this algorithm is to keep the clocks of all other nodes synchronized with the time server node.

i. Cristian's Algorithm



In this method each node periodically sends a message to the server. When the time server receives the message it responds with a message T, where T is the current time of the server node.

Assume the clock time of client be To when it sends the message and T1 when it receives the message from server. To and T1 are measured using the same clock so the best estimate of time for propagation is (T1-To)/2.

When the reply is received at the client's node, its clock is readjusted to T+(T1-T0)/2. There can be unpredictable variation in the message propagation time between the nodes hence (T1-T0)/2 is not good to be added to T for calculating current time.

For this several measurements of T1-To are made and if these measurements exceed some threshold value then they are unreliable and discarded. The average of the remaining measurements is calculated and the minimum value is considered accurate and half of the calculated value is added to T.

Advantage-It assumes that no additional information is available.

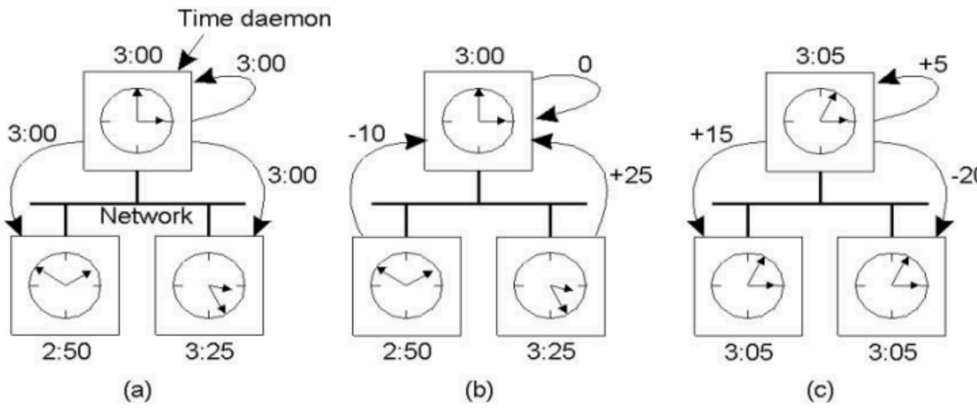
Disadvantage- It restricts the number of measurements for estimating the value.

ii.The Berkeley Algorithm

This is an active time server approach where the time server periodically broadcasts its clock time and the other nodes receive the message to correct their own clocks.

In this algorithm the time server periodically sends a message to all the computers in the group of computers. When this message is received each computer sends back its own clock value to the time server. The time server has a prior knowledge of the approximate time required for propagation of a message which is used to readjust the clock values. It then takes a fault tolerant average of clock values of all the computers. The calculated average is the current time to which all clocks should be readjusted.

The time server readjusts its own clock to this value and instead of sending the current time to other computers it sends the amount of time each computer needs for readjustment. This can be a positive or negative value and is calculated based on the knowledge the time server has about the propagation of messages.



2.Distributed algorithms

Distributed algorithms overcome the problems of centralized by internally synchronizing for better accuracy. One of the two approaches can be used:

i.Global Averaging Distributed Algorithms

In this approach the clock process at each node broadcasts its local clock time in the form of a "resync" message at the beginning of every fixed-length resynchronization interval. This is done when its local time equals To+iR for some integer i, where To is a fixed time agreed by all nodes and R is a system parameter that depends on total nodes in a system.

After broadcasting the clock value, the clock process of a node waits for time T which is determined by the algorithm.

During this waiting the clock process collects the resync messages and the clock process records the time when the message is received which estimates the skew after the waiting is done. It then computes a fault-tolerant average of the estimated skew and uses it to correct the clocks.

ii.Localized Averaging Distributed Algorithms

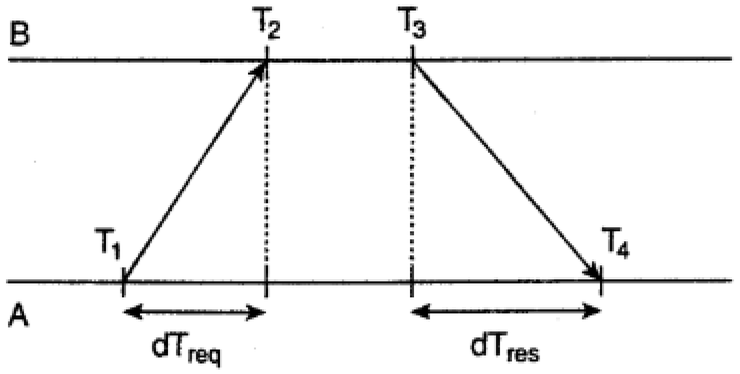
The global averaging algorithms do not scale as they need a network to support broadcast facility and a lot of message traffic is generated.

Localized averaging algorithms overcome these drawbacks as the nodes in distributed systems are logically arranged in a pattern or ring.

Each node exchanges its clock time with its neighbors and then sets its clock time to the average of its own clock time and of its neighbors.

3. Simple Network Time Protocol

It is recommended in a network environment where server is root and client is leaf node.



Time offset t=((t2-t1)+(t3-t4))/2

Current time=t4+t

- **Logical Clocks**

Logical Clocks refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual timespan. A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system. Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.

Example :

If we go outside then we have made a full plan at which place we have to go first, second and so on. We don't go second place first and then first place. We always maintain the procedure or an organization that is planned before. In a similar way, we should do the operations on our PCs one by one in an organized way.

Suppose, we have more than 10 PCs in a distributed system and every PC is doing its own work but then how do we make them work together. There comes a solution to this i.e. LOGICAL CLOCK.

Method-1:

To order events across processes, try to sync clocks in one approach.

This means that if one PC has a time of 2:00 pm then every PC should have the same time which is quite not possible. Not every clock can sync at one time. Then we can't follow this method.

Method-2:

Another approach is to assign Timestamps to events.

Taking the example into consideration, this means if we assign the first place as 1, second place as 2, third place as 3 and so on. Then we always know that the first place will always come first and then so on. Similarly, If we give each PC their individual number then it will be organized in a way that the 1st PC will complete its process first and then second and so on.

BUT, Timestamps will only work as long as they obey causality.

What is causality ?

Causality is fully based on HAPPEN BEFORE RELATIONSHIP.

Taking a single PC only if 2 events A and B are occurring one by one then TS(A) < TS(B). If A has a timestamp of 1, then B should have a timestamp more than 1, then only happen before the relationship occurs.

Taking 2 PCs and event A in P1 (PC.1) and event B in P2 (PC.2) then also the condition will be TS(A) < TS(B). Take example- suppose you are sending a message to someone at 2:00:00 pm, and the other person is receiving it at 2:00:02 pm.Then it's obvious that TS(sender) < TS(receiver).

Properties Derived from Happen Before Relationship –

Transitive Relation –

If, TS(A) <TS(B) and TS(B) <TS(C), then TS(A) < TS(C)

Causally Ordered Relation –

a->b, this means that a is occurring before b and if there are any changes in a it will surely reflect on b.

Concurrent Event –

This means that not every process occurs one by one, some processes are made to happen simultaneously i.e., A || B.

Lamport's Logical Clocks:

The algorithm of Lamport timestamps is a simple algorithm used to determine the order of events in a distributed computer system. As different nodes or processes will typically not be perfectly synchronized, this algorithm is used to provide a partial ordering of events with minimal overhead, and conceptually provide a starting point for the more advanced vector clock method.

For synchronization of logical clocks, Lamport established a relation termed as "happens-before" relation.

Happens- before relation is a transitive relation, therefore if p→q and q→r, then p→r.

If two events, a and b, occur in different processes which do not at all exchange messages amongst them, then a→b is not true, but neither is b→a which is antisymmetry.

The algorithm follows some simple rules:

A process increments its counter before each event in that process.

When a process sends a message, it includes its counter value with the message.

On receiving a message, the counter of the recipient is updated, if necessary, to the greater of its current counter and the timestamp in the received message. The counter is then incremented by 1 before the message is considered received.

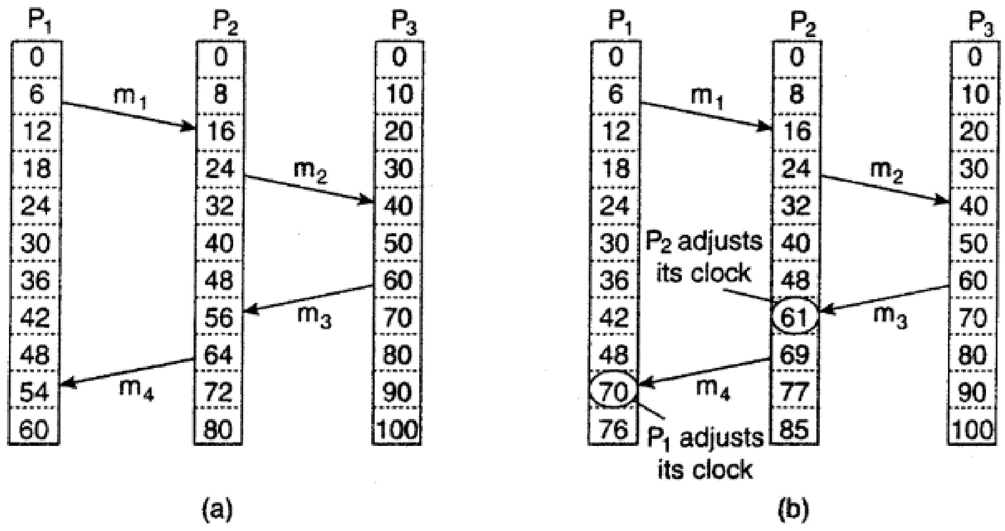


Figure 6-9. (a) Three processes, each with its own clock. The clocks run at different rates. (b) Lamport's algorithm corrects the clocks.

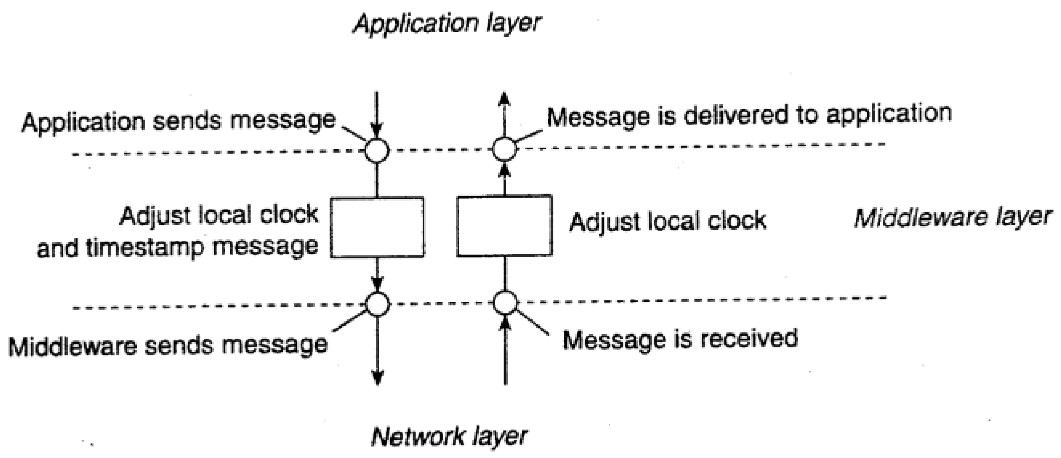


Figure 6-10. The positioning of Lamport's logical clocks in distributed systems.

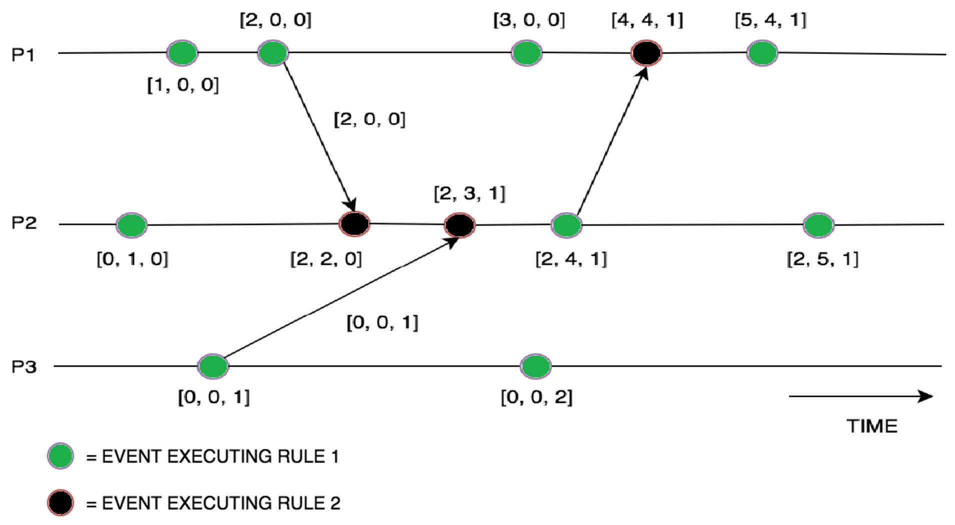
Vector Timestamp Ordering:

Vector clock is an algorithm that generates partial ordering of events and detects casual violations in DS. This algorithm helps us label every process with a vector with an integer for each local clock of every process within the system.

For N given processes there will be a vector/array of size N.

Working of vector clock algorithm:

1. Initially all clocks are set to zero
2. Every time an internal event occurs in a process the value of the process's logical clock in the vector is incremented by 1.
3. Also, every time a process sends a message the value of the process’s logical clock in the vector is incremented by 1.


P1
P2
P3
[1, 0, 0]
[2, 0, 0]
[3, 0, 0]
[4, 4, 1]
[5, 4, 1]
[2, 0, 0]
[2, 3, 1]
[0, 1, 0]
[2, 2, 0]
[2, 4, 1]
[2, 5, 1]
[0, 0, 1]
[0, 0, 1]
[0, 0, 2]
TIME
= EVENT EXECUTING RULE 1
= EVENT EXECUTING RULE 2