**Clocks Synchronization**

Clock synchronization involves aligning the clocks of computers or nodes, enabling efficient data transfer, smooth communication, and coordinated task execution.

Clock synchronization in distributed systems refers to the process of ensuring that all clocks across various nodes or computers in the system are set to the same time or at least have their times closely aligned.

* In a distributed system, where multiple computers communicate and collaborate over a network, each computer typically has its own local clock.
* However, due to factors such as hardware differences, network delays, and clock drift (inaccuracies in timekeeping), these local clocks can drift apart over time.

**Importance**:

Event ordering, data integrity and conflict resolution, Fault detection and recovery, security and authentication, consistency and coherence.

In synchronization, there are three types of clocks.

1. **Physical Clock**: - Time isn't a big issue in traditional centralized systems, where one or more CPUs share a common bus. The entire system shares the same understanding of time, right or wrong, it is consistent. - In distributed systems, this is not the case. Every system, though, **has its own timer** that keeps the clock running. These clocks are based on the oscillation of a piezoelectric crystal or a similar integrated circuit. They are not flawless, but they are relatively precise, reliable, and accurate. This implies that the clocks will **differ** **from the correct time**. Every timer is different in terms of characteristics — characteristics that might change with time, temperature. Thus, each system's time will drift away from the true time at a different rate and perhaps in a different direction (slow or fast) hence the need to adjust the clocks accordingly.
2. **Logical Clock:** - Logical clocks mean creating a protocol on all computers in a distributed system so that the computers can keep **a uniform ordering** of happenings inside some virtual time range. In a distributed system, **a logical clock is a technique for recording temporal and causative links.** Because distributed systems may lack a physically synchronized global clock, a logical clock provides for **the global ordering** of occurrences from various processes in certain systems.
3. **Vector Clocks**

In Lamport’s clock, if x -> y, then T(x)<T(y). But this does not tell about the relationship between events x and that’s because Lamport’s clock do not capture causality. The causal relationship between messages is captured through vector clocks.

Vector Clock is an algorithm that creates a partial ordering of occurrences and identifies causality breaches in a distributed system. Such clocks extend on vector time to provide for a logically coherent picture of the distributed system; they identify if a contributed activity has triggered another activity. It essentially captures all the causal relationships. This approach assists in labelling each process within the system with a vector (a list of

numbers) including an integer for each local clock. As a result, for every N process, there will be a vector of size N.

Algorithm:

- All of the clocks are initialized to zero.

- When an internal event happens in a process, the number of the process's logical clock in the vector is increased by one.

- Also, every time a process sends a message, the value of the process's logical clock in the vector is incremented by one.

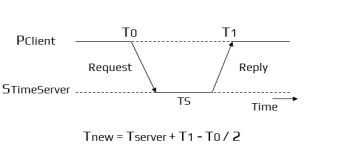
- Every time a process receives a message, the value of the process's logical clock in the vector is incremented by one, and moreover, each element is adjusted by calculating the maximum value of the vector clock and the vector value in the incoming message.

**Clock synchronization Algorithms:**

1. **Cristian Algorithm**

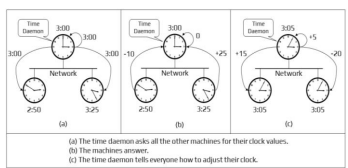
Cristian’s Algorithm is a centralized clock synchronization algorithm used to synchronize time with a time server by client processes. This algorithm works well with a low latency network where the round-trip time — time duration between the start of request and end of corresponding response — is short as compared to the accuracy. It is an approach in which the client approaches the server.

A client sends a request to a time server for its current value of the UTC time. The client records the time its request was submitted and the time it got the response. Then, the client changes its current time with the value received from the server **plus** its **estimate of the delay** in obtaining this value, resulting in the total time required to submit the query and get the response, which is (T1−T0)/2. The new time value is thus Tserver+(T1−T0)/2.

****

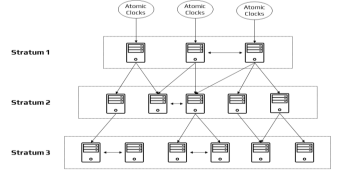
**Berkeley’s Algorithm**

The Berkeley Algorithm is a centralized clock synchronization mechanism in a distributed system that implies no computer has a precise timing source. The algorithm was developed by Gusella and Zatti at the University of California, Berkeley in 1989. This algorithm is an example of an active time server approach: the time server periodically sends a message to all the computers in the group. When the message is received, each computer then sends back its own clock value to the time server. It is an approach in which the server approaches the client. The time server has prior knowledge of the approximate time required for the propagation of the message which is used to readjust the message. The time server then takes the average of all clock values of all the computers. All clocks should be readjusted to the current time which is the calculated average. The time server readjusts its own clock value to this value and instead of sending the current time to other computers, it sends the amount of time each computer needs for readjustment. The value may be positive or negative.

How the algorithm works:

Network Protocol

Network Time Protocol is a standard followed by synchronization clocks **on the internet**. It is a **decentralized algorithm**. The Network Time Protocol (NTP) is a commonly employed Internet Engineering Task Force (IETF) standard (RFC 1305). (RFC 1305). The main servers are directly linked to a precise and dependable **UTC time source**. They are the foundations of hierarchical time service, with additional servers becoming operational as we go away from the roots. The common configuration includes UTC time servers at big government institutions at stratum 1, institutional time servers or Internet service providers' time servers at stratum 2, and most users linking to academic time servers at stratum 3. NTP may synchronize computers in three modes: first is the client-server mode, the second is the multicast mode, and the last is the symmetrical (peer) mode. In the client/server mode, the client makes queries to the server upon startup and on a regular basis thereafter. In a way similar to Cristian's technique, it tracks the time at which the request and response are delivered and received in order to factor out network latency as much as feasible. Because the server multicasts its time value on a regular basis, the multicast mode is frequently more efficient. On a local area network with multicast capabilities, time resynchronization can be accomplished in a single message rather than two messages per client (e.g., Ethernet). However, in order to assess the network latency and adjust for it, the clients must first conduct a few client/server queries. However, if the network parameters change over time, the multicast mode's accuracy will be inferior to that of the client-server mode. The Simple Network Time Protocol (RFC 2030) is a Network Time Protocol adaptation that supports operation in a stateless remote procedure call mode or multicast mode. It is designed for use in contexts where a complete NTP implementation is neither required nor warranted. SNTP is intended to be utilized at the endpoints of the synchronization subnet (high stratum) rather than for time server synchronization.



**Lamport’s Algorithm**

In a distributed system, it is not necessary for the clocks to be absolutely synchronized. If two processes do not interact with each other, it is not necessary that their clocks need to be synchronized because the lack of synchronization would not matter. It is not important for all processes to agree on what the current time is but they should agree on the order in which events occur. In a distributed system, Lamport Clocks are a basic mechanism for **identifying the sequence of events**. It gives a **"Happened-Before"** sequencing of occurrences. If there is no “happened-before” relationship, then the events are considered **concurrent.** Algorithm: The “Happened before” relation between a and b is a -> b, which means ‘a’ happened before ‘b’. The criteria for logical clocks are: - Clock 1: Ci (a) < Ci(b), Ci -> Logical Clock, **if ‘a’ happened before ‘b’, then the time of ‘a’ will be less than ‘b’ in a particular process**.

Clock 2: Ci(a) < Cj(b), Clock value of Ci(a) is less than Cj(b).

To be continued………