LOGICAL TIME

Logical clocks are based on capturing chronological and causal relationships of processes and ordering events based on these relationships.

Precise physical clocking is not possible in distributed systems. The asynchronous distributed systems spans logical clock for coordinating the events.Three types of logical clock are maintained in distributed systems:

- Scalar clock
- Vector clock
- Matrix clock

In a system of logical clocks, every process has a logical clock that is advanced using a set of rules. Every event is assigned a timestamp and the causality relation between events can be generally inferred from their timestamps.

The timestamps assigned to events obey the fundamental monotonicity property; that is, if an event a causally affects an event b, then the timestamp of *a* is smaller than the timestamp of *b*.

Differences between physical and logical clock

Physical Clock	Logical Clock
A physical clock is a physical procedure combined with a strategy for measuring that procedure to record the progression of time.	A logical clock is a component for catching sequential and causal connections in a dispersed framework.
The physical clocks are based on cyclic processes such as a celestial rotation.	A logical clock allows global ordering on events from different processes.

A Framework for a system of logical clocks

A system of logical clocks consists of a time domain T and a logical clock C. Elements of T form a partially ordered set over a relation <. This relation is usually called the happened before or causal precedence.

The logical clock C is a function that maps an event e in a distributed system to an element in the time domain T denoted as C(e).

$$C : H \mapsto T \text{ such that}$$

for any two events e_i and e_j, $e_i \rightarrow e_j \Rightarrow C(e_i) < C(e_j)$.

This monotonicity property is called the **clock consistency condition.**When T and C satisfy the following condition,

$$e_i \rightarrow e_j \Leftrightarrow C(e_i) < C(e_j)$$

Then the system of clocks is **strongly consistent.**

Implementing logical clocks

The two major issues in implanting logical clocks are:

Data structures: representation of each process

Protocols: rules for updating the data structures to ensure consistent conditions.

Data structures:

Each process $p_i$maintains data structures with the given capabilities:

• A local logical clock (lc_i), that helps process p_i measure itsown progress.

• A logical global clock (gc_i), that is a representation of process p_i's local view of the logical global time. It allows this process to assignconsistent timestamps to its local events.

Protocol:

The protocol ensures that a process's logical clock, and thus its view of theglobal time, is managed consistently with the following rules:

Rule 1: Decides the updates of the logical clock by a process. It controls send, receive and other operations.

Rule 2: Decides how a process updates its global logical clock to update its view of the global time and global progress. It dictates what information about the logical time is piggybacked in a message and how this information is used by the receiving process to update its view of the global time.

SCALAR TIME

Scalar time is designed by Lamport to synchronize all the events in distributed systems. A Lamport logical clock is an incrementing counter maintained in each process. This logical clock has meaning only in relation to messages moving between processes. When a process receives a message, it resynchronizes its logical clock with that sender maintainingcausal relationship.

The Lamport's algorithm is governed using the following rules:

- The algorithm of Lamport Timestamps can be captured in a few rules:
- All the process counters start with value 0.
- A process increments its counter for each event (internal event, message sending, message receiving) in that process.

- When a process sends a message, it includes its (incremented) counter value with the message.
- On receiving a message, the counter of the recipient is updated to the greater of its current counter and the timestamp in the received message, and then incremented by one.

- If C_i is the local clock for process P_i then,
- if a and b are two successive events in P_i, then $C_i(b) = C_i(a) + d1$, where $d1 > 0$
- if a is the sending of message m by P_i, then m is assigned timestamp $t_m = C_i(a)$
- if b is the receipt of m by P_j, then $C_j(b) = \max\{C_j(b), t_m + d2\}$, where $d2 > 0$

Rules of Lamport's clock

Rule 1: $C_i(b) = C_i(a) + d1$, where $d1 > 0$

Rule 2: The following actions are implemented when p_i receives a message m with timestamp C_m:

a) $C_i = max(C_i, C_m)$

b) Execute Rule 1

c) deliver the message

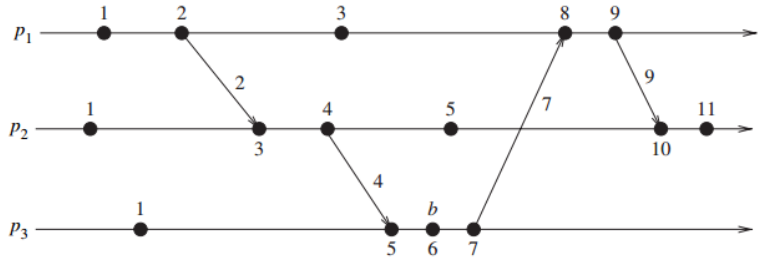

Fig : Evolution of scalar time

Basic properties of scalar time:

1. Consistency property: Scalar clock always satisfies monotonicity. A monotonic clock only increments its timestamp and never jump.Hence it is consistent.

$$C(e_i) < C(e_j)$$

2. Total Reordering: Scalar clocks order the events in distributed systems.But all the events do not follow a common identical timestamp. Hence a tie breaking mechanism is essential to order the events. The tie breaking is done through:

- Linearly order process identifiers.
- Process with low identifier value will be given higher priority.

The term (t, i) indicates timestamp of an event, where t is its time of occurrence and i is the identity of the process where it occurred.

The total order relation ($\prec$) over two events x and y with timestamp (h, i) and (k, j) is given by:

$$x \prec y \Leftrightarrow (h < k \textbf{ or } (h = k \textbf{ and } i < j))$$

A total order is generally used to ensure liveness properties in distributed algorithms.

3. Event Counting

If event e has a timestamp h, then h−1 represents the minimum logical duration, counted in units of events, required before producing the event e. This is called **height** of the event e. h-1 events have been produced sequentially before the event e regardless of the processes that produced these events.

4. No strong consistency

The scalar clocks are not strongly consistent is that the logical local clock and logical global clock of a process are squashed into one, resulting in the loss causal dependency information among events at different processes.

VECTOR TIME

The ordering from Lamport's clocks is not enough to guarantee that if two events precede one another in the ordering relation they are also causally related. Vector Clocks use a vector counter instead of an integer counter. The vector clock of a system with N processes is a vector of N counters, one counter per process. Vector counters have to follow the following update rules:

- Initially, all counters are zero.
- Each time a process experiences an event, it increments its own counter in the vector by one.
- Each time a process sends a message, it includes a copy of its own (incremented) vector in the message.
- Each time a process receives a message, it increments its own counter in the vector by one and updates each element in its vector by taking the maximum of the value in its own vector counter and the value in the vector in the received message.

The time domain is represented by a set of n-dimensional non-negative integer vectors in vector time.

Rules of Vector Time

Rule 1: Before executing an event, process pi updates its local logical time as follows:

$$vt_i[i] := vt_i[i] + d \qquad (d > 0)$$

Rule 2: Each message m is piggybacked with the vector clock vt of the sender process at sending time. On the receipt of such a message (m,vt), process p_i executes the following sequence of actions:

1. update its global logical time

$$1 \le k \le n \ : \ vt_i[k] := max(vt_i[k], vt[k])$$

2. execute R1

3. deliver the message m

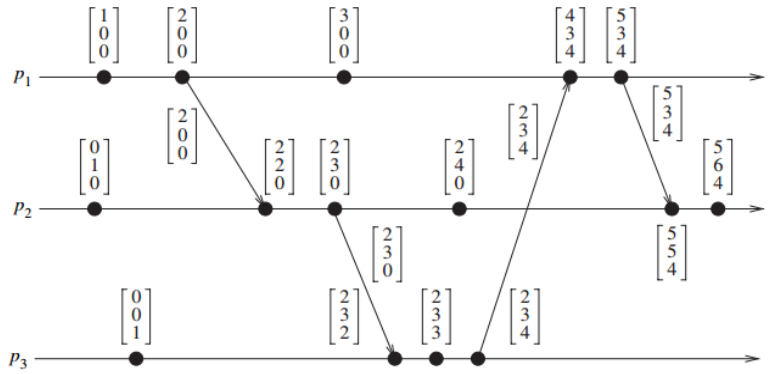

Fig : Evolution of vector scale

Basic properties of vector time

1. Isomorphism:

- "→" induces a partial order on the set of events that are produced by a distributed execution.
- If events x and y are timestamped as vh and vk then,

$$x \rightarrow y \Leftrightarrow vh < vk$$

- $$x \parallel y \Leftrightarrow vh \parallel vk.$$
- There is an isomorphism between the set of partially ordered events produced by a distributed computation and their vector timestamps.
- If the process at which an event occurred is known, the test to compare two timestamps can be simplified as:

$$x \rightarrow y \Leftrightarrow vh[i] \le vk[i]$$

$$x \parallel y \Leftrightarrow vh[i] > vk[i] \wedge vh[j] < vk[j].$$

2. Strong consistency

The system of vector clocks is strongly consistent; thus, by examining the vector timestamp of two events, we can determine if the events are causally related.

3. Event counting

If an event e has timestamp vh, vh[j] denotes the number of events executed by process p_j that causally precede e.

Vector clock ordering relation

$t = t' \Leftrightarrow \forall i\ t[i] = t'[i]$
$t \neq t' \Leftrightarrow \exists i\ t[i] \neq t'[i]$
$t \leq t' \Leftrightarrow \forall i\ t[i] \leq t'[i]$
$t < t' \Leftrightarrow (t \leq t' \text{ and } t \neq t')$
$t \,||\, t' \Leftrightarrow \text{not } (t < t' \text{ or } t' < t)$

t[i]- timestamp of process i.

1.18 PHYSICAL CLOCK SYNCHRONIZATION: NEWTWORK TIME PROTOCOL (NTP)

Centralized systems do not need clock synchronization, as they work under a common clock. But the distributed systems do not follow common clock: each system functions based on its own internal clock and its own notion of time.The time in distributed systems is measured in the following contexts:

- The time of the day at which an event happened on a specific machine in the network.
- The time interval between two events that happened on different machines in the network.
- The relative ordering of events that happened on different machines in the network.

Clock synchronization is the process of ensuring that physically distributed processors have a common notion of time.

Due to different clocks rates, the clocks at various sites may diverge with time, and periodically a clock synchrinization must be performed to correct this clock skew in distributed systems. Clocks are synchronized to an accurate real-time standard like UTC (Universal Coordinated Time). Clocks that must not only be synchronized with each other but also have to adhere to physical time are termed **physical clocks**. This degree of synchronization additionally enables to coordinate and schedule actions between multiple computers connected to a common network.

1.18.1 Basic terminologies:

If C_a and C_b are two different clocks, then:

- **Time:** The time of a clock in a machine p is given by the function $C_p(t)$,where $C_p(t)= t$ for a perfect clock.
- **Frequency:** Frequency is the rate at which a clock progresses. The frequency at time t of clock $C_a$is$C_a^{'}(t)$.
- **Offset:** Clock offset is the difference between the time reported by a clock and the real time. The offset of the clock C_a is given by $C_a(t)- t$. The offset of clock C a relative to C_b at time $t \geq 0$ is given by $C_a(t)- C_b(t)$
- **Skew:** The skew of a clock is the difference in the frequencies of the clock and the perfect clock. The skew of a clock C_a relative to clock C_b at timet is $C_a^{'}(t)- C_b^{'}(t)$.
- **Drift (rate):** The drift of clock C_a the second derivative of the clock value with respect to time. The drift is calculated as:

$$C_a''(t) - C_b''(t).$$

1.18.2 Clocking Inaccuracies

Physical clocks are synchronized to an accurate real-time standard like UTC (Universal Coordinated Time). Due to the clock inaccuracy discussed above, a timer (clock) is said to be working within its specification if:

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho.$$

ρ- maximum skew rate.

1. Offset delay estimation

A time service for the Internet - synchronizes clients to UTC Reliability from redundant paths, scalable, authenticates time sources Architecture. The design of NTP involves a hierarchical tree of time servers with primary serverat the root synchronizes with the UTC. The next level contains secondary servers, which act as a backup to the primary server. At the lowest level is the synchronization subnet which has the clients.

2. Clock offset and delay estimation

A source node cannot accurately estimate the local time on the target node due to varying message or network delays between the nodes.This protocol employs a very common practice of performing several trials and chooses the trial with the minimum delay.

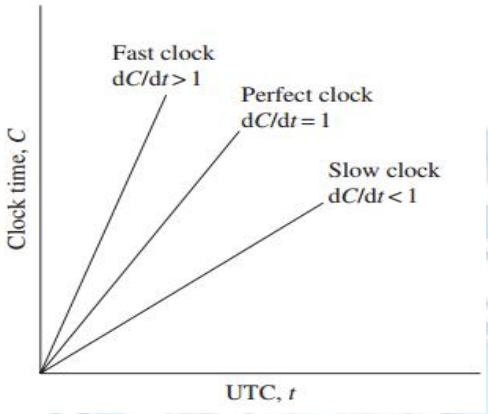

Fig 1.24: Behavior of clocks

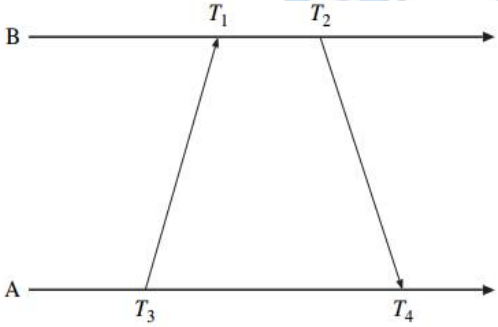

Fig 1.30 a) Offset and delay estimation between processes from same server

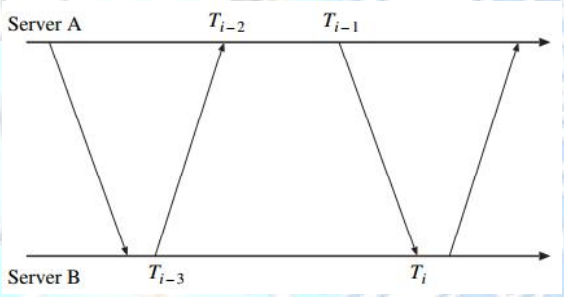

Fig 1.30 b) Offset and delay estimation between processes from different servers

Let T1, T2, T3, T4 be the values of the four most recent timestamps. The clocks A and B are stable and running at the same speed. Let a = T1 − T3 and b = T2 − T4. If the network delay difference from A to B and from B to A, called **differential delay**, is

small, the clock offset θand roundtrip delayδ of B relative to A at time T4are approximately given by the following:

$$\theta = \frac{a+b}{2}, \quad \delta = a - b$$

Each NTP message includes the latest three timestamps T1, T2, andT3, while T4 is determined upon arrival.