SSN COLLEGE OF ENGINEERING
RECORD SHEET

16/08/2021 Monday	UCS 1701 - Distributed Systems Continuous Assessment Test - I	Sukeerthi 185001175 CSE - C

Part - B

13. (i) The reasons due to which physical clocks are not suitable for Distributed Systems :-

Since the concept of causality between events is fundamental to the design and analyses of parallel and distributed computing. Usually, causality is tracked using physical time. But in distributed systems, it is not possible to have a global physical time and only possible to realize an approximation of it. Consequently, if the physical clocks are not synchronized precisely, the causality relation between events may not be precisely captured. This makes it incompatible with distributed systems in which the rate of occurrence of events is several magnitudes higher and the event execution time is several magnitudes smaller.

The problems with Physical clocks can be attributed to

* Clock Drift - Clock ticks at different rates thus creating an ever-widening gap in perceived time.

* Clock Skew - Difference between two clocks at one point in time.

For quartz crystal clocks, typical drift rate is about 1 second every 106 seconds = 11.6 days.

Best atomic clocks have drift rate of one second in 1013 seconds = 300,000 years.

→ Quartz clock runs at the rate of 1.5 microseconds slower for every 35 days.

→ Within a single process, or between processes on the same computer the order in which two events occur can be determined using the physical clock, But

→ Between two different computers in a distributed system The order in which two events occur cannot be determined using physical clocks, since those clocks cannot be synchronized perfectly.

→ In centralized systems, when one or more processors share a common bus, time is not much of a concern since the entire system shares the same understanding of time.

→ In distributed systems, each system has its own timer that drives the clock. These timers are based on the oscillation of a quartz crystal or equivalent IC. Although they are reasonably precise, stable and accurate they are not perfect. The clocks will drift away from true time. Each timer has different characteristics that might change with time, temperature etc. This implies that each systems will drift away from true time at a different rate and in a different direction.

(ii) Algorithms / Methods of Physical Clock Synchronization

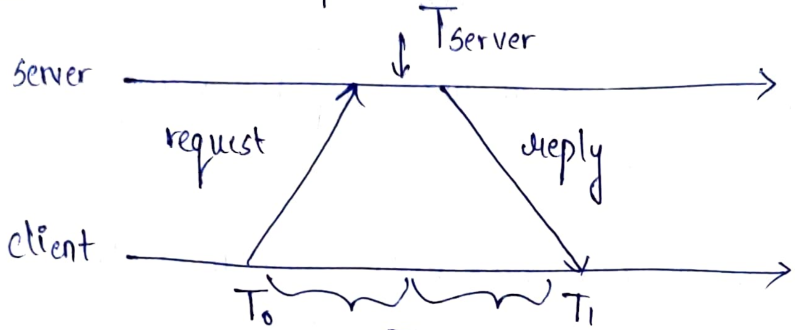
1) → Cristian's algorithm

2) → Berkely algorithm

3) → Network time protocol (Internet)

1)

Cristian's Method



$\frac{T_1 - T_0}{2}$ = estimated overhead in each direction

$\frac{T_1 - T_0}{2}$ → round-trip time

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

→ Round trip times between processes are often reasonably short in practice, yet theoretically unbounded.

→ Practical estimate possible if round-trip times are sufficiently short in compared to required accuracy.

2) Berkely Algorithm

→ Machines run time daemon process that implements protocol

→ One machine is elected as the server and others are slaves

→ Master polls each machine periodically and ask each machine for time.

→ Can use Cristian's algorithm to compensate for network latency.

→ When results are received, master computes average of times- including master's time.

→ Send offset by which each clock needs adjustment to each slave

→ It has provisions for ignoring readings from clocks whose skews is too large.

→ Computes a fault - tolerant average

→ If master fails, any slave can take over

3) Network Time protocol

→ Enables clients across Internet to be accurately synchronized to UTC despite message delays.

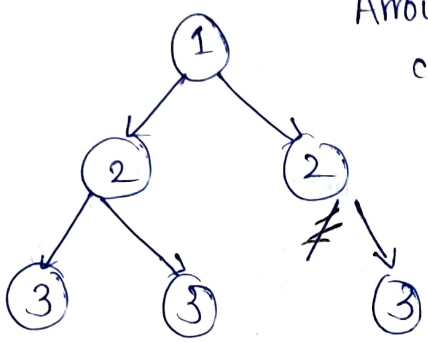
→ Primary servers are connected directly to a time source such as a radio clock receiving UTC, secondary servers are synchronized with primary servers.

→ The servers are connected in a logical hierarchy called a synchronization subnet whose levels are called strata.

→ Primary servers ~~as~~ occupy stratum 1 : they are the root

→ Stratum 2 servers are secondary servers that are synchronized directly with the primary servers.

→ Stratum 3 servers are synchronized with stratum 2 servers and so on.



Arrows denote synchronization control

numbers denote strata.

The round trip delay is estimated as

$$D_i = (T_i - T_{i-3}) - [(T_{i-1}) - (T_{i-2})]$$

For 15 computers, local drift rate $< 2 \times 10^{-5}$ max round-trip 10 ms and the clocks can be synchronized to with 20-25 ms.

In practice, a source node cannot accurately estimate the local time on the target node due to varying message/network delays between the nodes.

This protocol employs a common practice of performing several trials and chooses the one with minimum delay.

11. Need for Logical Clock

→ For many purposes it is sufficient to know the order in which events occurred.

→ Lamport introduced logical time, synchronize logical clocks.

→ An event may be an instruction execution, may be a function execution etc.

→ Events include message send/ receive.

Within a single process or between two processes on the same computer, the order in which two events occur can be determined using the physical clock.

Between two different computers in a distributed system, the order in which two events occur cannot be determined using a physical clock, since those clocks cannot be synchronized perfectly.

Working of Lamport's Logical Clock

Lamport defined the happened before relation (→) which describes a causal ordering of events:

1. If a and b are events in the same process and a occured before b, then a → b

2. If a is the event of sending a message m in one process, b is the event of receiving that message m in another process then a → b.

if $a \to b$, and $b \to c$ then $a \to c$ (the relation "$\to$" is transitive)

Each process P_i has a logical clock C_i

→ Clock C_i can assign a value $C_i(a)$ to any event a in process P_i

→ The value $C_i(a)$ is called the timestamp of event a in process P_i

→ The value $C(a)$ is called the timestamp of event a in whatever process it occurred.

* if $a \to b$, then $C(a) < C(b)$

* For any two events a and b in the same process if a happens before b, then $C_i(a) < C_i(b)$

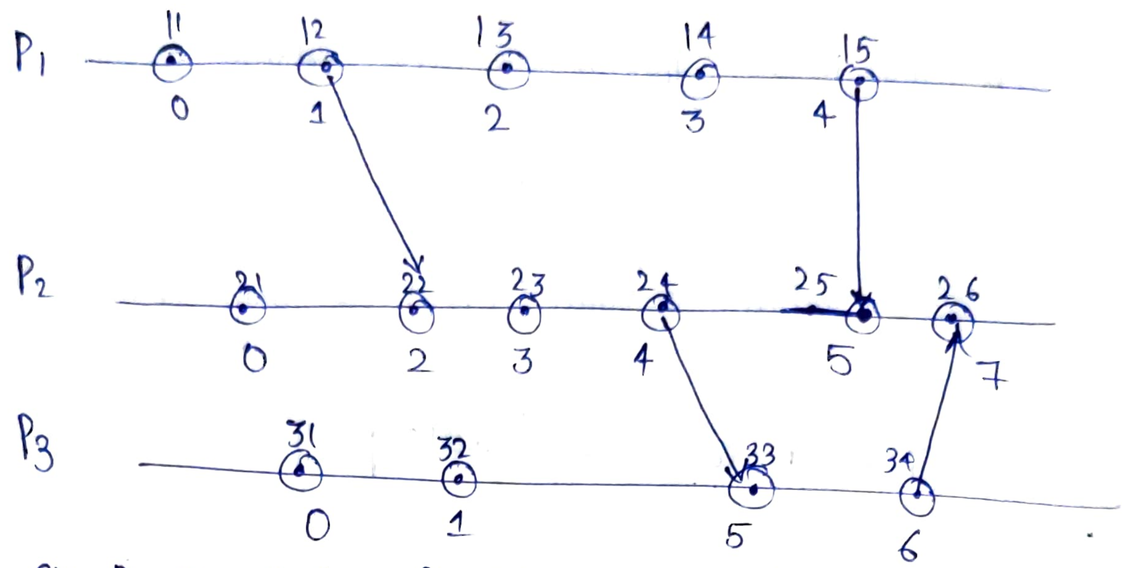
* If event a is the event of sending a message m in process P_i and event b → receiving that same message m in a ~~dr~~ P_k process then $C_i(a) < C_k(b)$

[IR1] Clock C_i must be incremented between any two successive events in process P_i $C_i := C_i + d$, $(d > 0)$ usually $d = 1$

[IR2] If event a is the event of sending a message m in process P_i, then it is assigned a timestamp $C_{msg} = C_i(a)$. When that is received by P_k, C_k is set to a value greater than or equal to present value and greater than C_{msg}.

$C_k := \max(C_k, C_{msg} + d)$, $(d > 0)$ usually $d = 1$

(ii)



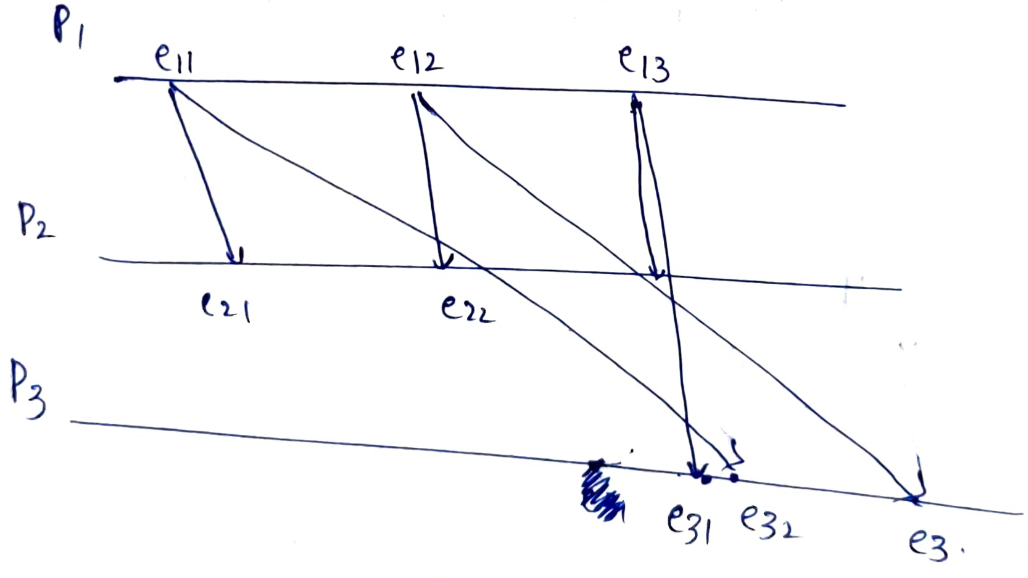
$e_{11} = 0$ $e_{12} = 1$ $e_{13} = 2$ $e_{14} = 3$ $e_{15} = 4$

$e_{21} = 0$ $e_{22} = 2$ $e_{23} = 3$ $e_{24} = 4$ $e_{25} = 5$ $e_{26} = 7$

$e_{31} = 0$ $e_{32} = 1$ $e_{33} = 5$ $e_{34} = 6$

Part - C

14)



all clocks are initialy [0,0,0]

e_{11} => send event

i = sender
j = receiver

$P_1 : C_1 [1,0,0]$

e_{12} send event

P_1 : $C_1[2,0,0]$

e_{13} send event

P_1 : $C_1[3,0,0]$

e_{21} $C_2[0,0,0,0]$ $tm[1,0,0]$ $[i=1, j=2]$

(i) $C_2[1] == tm[1] - 1$? true

(ii) $C_2[2,3] >= tm[2,3]$? true.

Accept message and update clock

P_2 clock : $[1,0,0]$ $(C_2$

e_{22} $C_2[1,0,0]$ $tm[2,0,0]$ $[i=1, j=2]$

(i) $C_2[1] == tm[1] - 1$? true

(ii) $C_2[2,3] >= tm[2,3]$? true

Accept message and update clock

P_2 clock : $[2,0,0]$ (C_2)

e_{23} $C_2[2,0,0]$ $tm[3,0,0]$ $[i=1, j=2]$

(i) $C_2[1] == tm[1] -$? true

(ii) $C_2[2,3] >= tm[2,3]$? true

Accept message and update clock.

P_2 clock = $[3,0,0]$ (C_2)

e_{31} $C_3[0,0,0]$ tm $[3,0,0]$ $[i=1, j=3]$

(i) $C_3[i] == tm[i]-1$? false

~~Buffer the message and update clock~~

~~P_2 clock : $C_2[3,0,0]$~~

Buffer the message in P_3 and try next message.

Donot update C_3 clock

e_{32} $C_3[0,0,0]$ tm $[1,0,0]$ $[i=1, j=3]$

(i) $C_3[i] == tm[i]-1$? true

(ii) $C_3[2,3] >= tm[2,3]$? true

Accept message and update clock

P_3 clock : $C_3[1,0,0]$

Clock value update. Attempt redelivery of buffer quantities

e_{31} : $C_3[1,0,0]$ tm $[3,0,0]$ $[i=1, j=3]$

(i) $C_3[1] == tm[1]-1$? false

Buffer the message in P_3 and attempt ~~&~~ redelivery later.

Do not update clock C_3. P_3 buffer : $e_{31}(3,0,0)$

e_{33} : $C3\ [1,0,0]$ tm $[2,0;0]$

$[i=1,\ j=3]$

(i) $C3[1] == tm[1] - 1$? true.

(ii) $C3[2,3] >= tm[2,3]$? true.

Accept the message and update the clock

C_3 clock : $[2,0,0]$

Clock updated. Attempt redelivery of buffer messages

e_{31}'' : $C3\ [2,0,0]$ tm $[3,0,0]$

$[i=1,\ j=3]$

(i) $C3[1] == tm[1] - 1$? true

(ii) $C3[2,3] >= tm[2,3]$? true

Accept the message and update the clock

C_3 clock : $[3,0,0]$