CS6650 - Scalable Distributed Systems

Name - Mahvash Maghrabi

Homework Set #3

Late days left : 4

Answer the following questions using explanations and diagrams as needed. No implementation needed.

Q1. A clock is reading 10:27:54.0 (hr:min:sec) when it is discovered to be 4 seconds fast. Explain why it is undesirable to set it back to the right time at that point and show (numerically) how it should be adjusted so as to be correct after 8 seconds have elapsed.

When a clock is 4 seconds ahead of time it clearly depicts that the clock has been running fast since a long time. If the clock is simply adjusted back by 4 seconds it may become unsynchronized with other clocks in the system as the other clocks may be running at the correct speed. So it is important to adjust the clock in a way that it maintains synchronization with other clocks in the system.

The clock can be adjusted so as to be correct after 8 seconds have elapsed :

E refers to an errant clock that reads 10:27:54.0 while the real time is 10:27:50.

We adjust our software clock S to tick at rate chosen so that it will be correct after 8 seconds

S = c(E - Tskew) + Tskew,

where, Tskew = 10:27:54 and c is to be found.

S = Tskew+4 (the correct time) when E = Tskew+8,
so: Tskew+ 4 = c(Tskew + 8 - Tskew) + Tskew, and c is 0.5.

Hence, S = 0.5(E - Tskew) + Tskew
where , Tskew ≤ E ≤ Tskew+8).

Q2. A client attempts to synchronize with a time server. It records the round-trip times and timestamps returned by the server in the table below…..

As the client tries to synchronize with a time server the client needs to know timestamp of the time when the request was sent and the timestamp of the time when the response was received and the round-trip time. The client needs to select the time from the table which has the lowest round trip which in this table is 20 and the time corresponding to it is 10:54:28:342.
To get the accuracy the maximum and minimum round trip difference should be divided by 2. Therefore the accuracy is +/- 2.5 ms. If the time difference between sending and receiving a message in the system is at least 8ms then we need to subtract this value from each round-trip time before dividing by 2. Which gives us (25 - 7.5 - 22 + 7.5) / 2 = 2.5 ms which remains the same.

Q3. A system of four processes, (P1, P2, P3, P4) performs the following events…

Lamport Timestamps : Lamport's logical clock was created by Leslie Lamport hence the name. It determines the order of events occurring.

Algorithm:

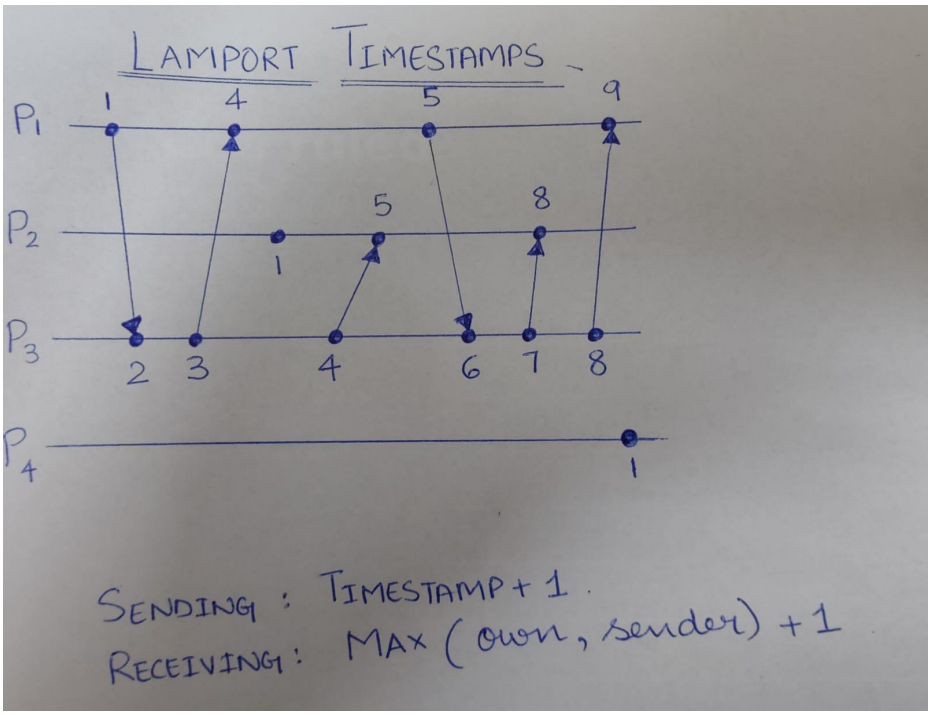
[Reference : https://www.geeksforgeeks.org/lamports-logical-clock/]

- **Happened before relation(->):** a -> b, means 'a' happened before 'b'.
- **Logical Clock:** The criteria for the logical clocks are:
  - [C1]: C_i (a) < C_i(b), [C_i -> Logical Clock, If 'a' happened before 'b', then time of 'a' will be less than 'b' in a particular process.]
  - [C2]: C_i(a) < C_j(b), [Clock value of C_i(a) is less than C_j(b)]

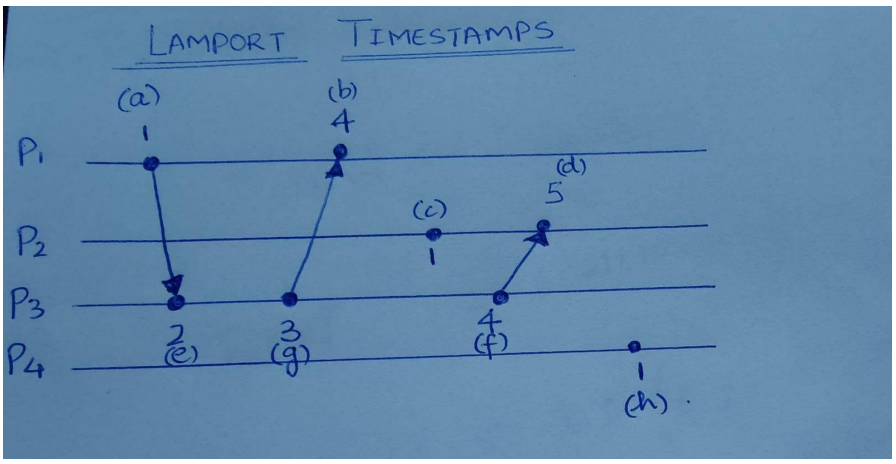
Lamport timestamps:
a. P1 sends a message to P3 (to event e).
Lamport Timestamp: 2
b. P1 receives a message from P3 (from event g).
Lamport Timestamp: 4
c. P2 executes a local event.
Lamport Timestamp: 1
d. P2 receives a message from P3 (from event f).
Lamport Timestamp: 5

e. P3 receives a message from P1 (from event a).
Lamport Timestamp: 6
f. P3 sends a message to P2 (to event d).
Lamport Timestamp: 8
g. P3 sends a message to P1 (to event b).
Lamport Timestamp: 9
h. P4 executes a local event.
Lamport Timestamp: 1

Working of Lamport Timestamps



The events are in conjunction with each other so even if the part on the right is removed all the events will still be considered and it will look something like this :



The lamport timestamps are as follows for each event :
a - 1
b - 4
c - 1
d - 5
e - 2
f - 4
g - 3
h - 1

Vector Timestamps : These are a variation on Lamport timestamps that use a vector of counters to represent the state of the entire system at each event. Vector Clock generates a partial ordering of events and detects causality violations in a distributed system.

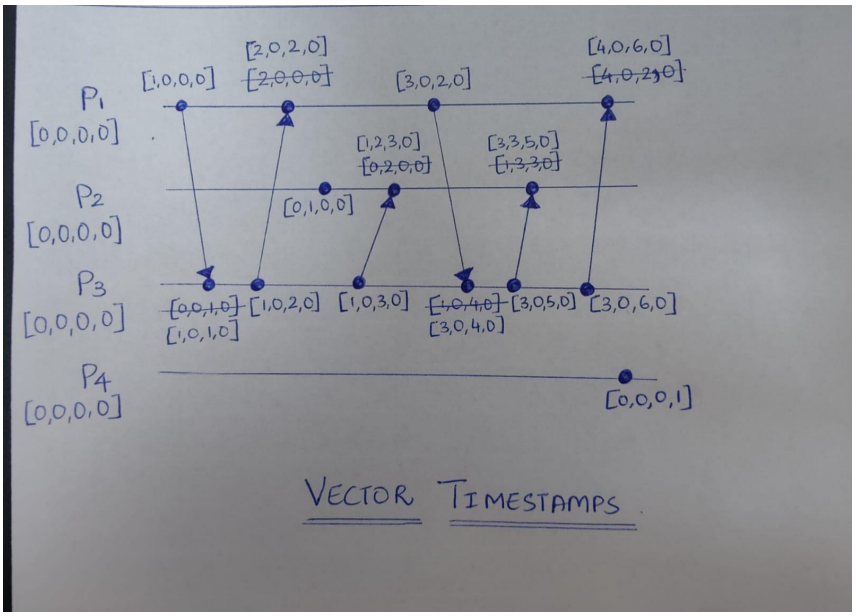
Algorithm : [Reference : https://www.geeksforgeeks.org/vector-clocks-in-distributed-systems/]

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

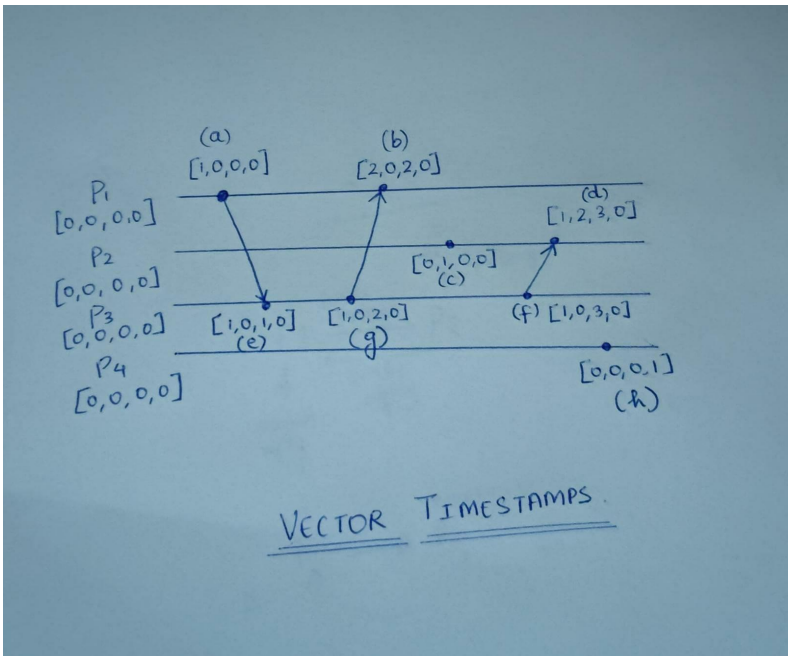
Vector timestamps:

a. P1 sends a message to P3 (to event e).
Timestamp: (1,0,1,0)
b. P1 receives a message from P3 (from event g).
Timestamp: (2,0,2,0)
c. P2 executes a local event.
Timestamp: (0,1,0,0)
d. P2 receives a message from P3 (from event f).
Timestamp: (1,2,3,0)
e. P3 receives a message from P1 (from event a).
Timestamp: (3,0,4,0)
f. P3 sends a message to P2 (to event d).
Timestamp: (3,3,5,0)
g. P3 sends a message to P1 (to event b).
Timestamp: (4,0,6,0)
h. P4 executes a local event.
Timestamp: (0,0,0,1)

Working of Vector Timestamps



The events are in conjunction with each other so even if the part on the right is removed all the events will still be considered and it will look something like this :



The vector timestamps are as follows for each event :
a - [1,0,0,0]
b - [2,0,2,0]
c - [0,1,0,0]
d - [1,2,3,0]
e - [1,0,1,0]
f - [1,0,3,0]
g - [1,0,2,0]
h - [0,0,0,1]

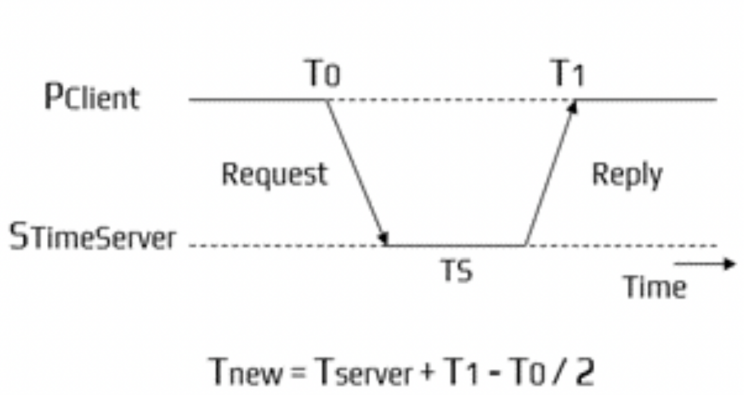
3b) Two events are said to be concurrent if one vector timestamp is neither greater than nor less than the other element while doing an element by element comparison. A concurrent vector is a mix of greater than, less than, or equal elements. Hence event b & h are concurrent with event d.

Q4. You are synchronizing your clock from a time server using Cristian's algorithm and observe the following times:
timestamp at client when the message leaves the client: 6:22:15.100
timestamp generated by the server: 6:21:10.700
timestamp at client when the message is received at client: 6:22:15.250

To what value do you set the client's clock?

If the best-case round-trip message transit time is 124 msec (0.124 sec), what is the error of the clock on the client?

Cristian's Algorithm is a clock synchronization algorithm is used to synchronize time with a time server by client processes. The algorithm works well when the networks have Low-latency and the round trip time is short as compared to accuracy. The round trip time refers to the time between the start of the request and the end of the corresponding response.



The algorithm is based on the formula :

Tclient = Tserver +(T1-T0) / 2

Where,
Tclient refers to the synchronized clock time,
Tserver refers to the clock time returned by the server,
T0 refers to the time at which request was sent by the client process,
T1 refers to the time at which response was received by the client process

According to the formula :

Tclient = 6:21:10.700 (6:22:15.250 - 6:22:15.100)/2
= 6:21:10.700 (150 ms)/2
= 6:21:10.775

The accuracy here is : +/-75 ms

If the best-case round-trip message transit time is 124 msec
Then the error of the clock is :
6:21:10.700 + 124ms/2 = 6:21:10.762
Error = 75 - 62 = 13ms

Q5. Is it possible to implement either a reliable or an unreliable (process) failure detector using an unreliable communication channel?

A5. It is possible to implement an unreliable failure detector using an unreliable communication channel as the unreliable communication channel is a channel that can drop, delay, or corrupt messages during transmission. The reliable failure detector requires a synchronous system and it cannot be built on an unreliable channel because a dropped message and a failed process cannot be distinguished unless the unreliability of the channel can be masked while it provides a guaranteed upper bound on message delivery times. Also if it is built up on an unreliable channel the dropped message and the failed process cannot be distinguished. If a channel that drops messages but also guarantees that at least one messages was not dropped can be used to create a reliable failure detector.

Q6. Give a formula for the maximum throughput of a mutual exclusion system in terms of the synchronization delay.

A6. The formula for maximum throughput of a mutual exclusion is

Let S = Synchronization delay, and

M = Minimum time spent in a critical section by any process.

Maximum throughput is 1/(S + M) critical section entries per second

Q7. Adapt the central server algorithm for mutual exclusion to handle the crash failure of any client (in any state),assuming that the server is correct and given a reliable failure detector. Comment on whether the resultant system is fault-tolerant. What would happen if a client that possesses the token is wrongly suspected to have failed?

A7. In the central server algorithm mutual exclusion can be achieved by employing a server that grants permission to enter the critical section. To enter this section the process sends a request message to the the server andwaits for a reply. The reply consists a token giving permission to enter the critical section. If no other process have the token when the request was made then the server replies immediately and gives the token. If the token is is being used by another process then the server does not reply, but queues the request. When the prior process leaves the critical section it sends a message to the server, where it gives back the token.

Here the server uses the reliable failure detector which detects if any client has crashed. If client

Crashed after getting the token then the server behaves like the client has returned the token. If it still keeps receiving the token prior to the client crashing it just ignores it. Therefore the resultant system is not fault-tolerant. If the client that crashed was a client holding the token then the data of the application that was being protected by the critical section will be in an unknown state when client starts to access it.

If a client that possesses the token is wrongly suspected to have failed there is a danger that the two processes might be given the permission to be executed concurrently in the critical section. This will eventually lead to a race condition which will result in data corruption or inconsistencies.

Q8.Construct a solution to reliable, totally ordered multicast in a synchronous system, using a reliable multicast and a solution to the consensus problem.

A8.

- To achieve reliable, totally-ordered multicast (RTO-multicast) we can use a combination of a reliable multicast (R-multicast) algorithm and also the consensus algorithm.
- The process is supposed to attach a totally ordered and a unique identifier to a message before R-multicasting it to all the receivers. Each of the processes maintains two sets of messages:
  - One of the set of messages it has R-delivered
  - Another is the set of messages it has RTO-delivered

- The process also keeps note of the messages that have not yet been RTO-delivered.
- The process then proposes a message that has not been RTO-delivered as the next message to deliver.
- Then the consensus algorithm is performed, in here all processes participate to agree on the next message to be RTO-delivered.
- Each process after RTO-delivering the message removes it from the set of R-delivered messages.
- Hence, every process delivers messages in the order of the concatenation of the sequence of consensus results.
- As the consensus results given to different correct processes are identical we get a RTO multicast
- In RTO multicast all processes deliver the same set of messages in the same order and without duplicates or omitting.
- This solution is fault-tolerant as it can handle failures of individual processes.
- It assumes that the system is synchronous which means that the timing of the events is well-defined and all the processes have access to a global clock.