Chapter-5 Time & State in DS

Physical Clock

Every computer or device have their own clock time, which is known as physical clock. These timers are based either on the oscillation of a quartz crytal, or equivalent IC.
Every computer contains a clock which is an electronic device that counts the oscillations in a crystal at a particular frequency which is known as physical clock.

- Cristian's Algorithm
- Berkeley Algorithm

Unfortunately, period of crystal oscillation varies slightly. If it oscillates faster, more ticks per real second, so clock runs faster; similar for slower clocks. So we need clock synchronization.

- **Internal syncronization**: Requires the clocks of the nodes to be synchronized to within a pre-specified bound. May not be syncronized to external reference.
- **External Syncronization:** Requires the clocks to be synchronized to within a pre-specified bound of an external reference clock

Logical clock

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.
Logical clock is a mechanism for capturing causal and chronological relationships in a distributed system. A physically synchronous global clock may not be present in a distributed system. In such systems a logical clock allows global ordering on events from different processes.

- Lamport's Algorithm
- Vectors Clock

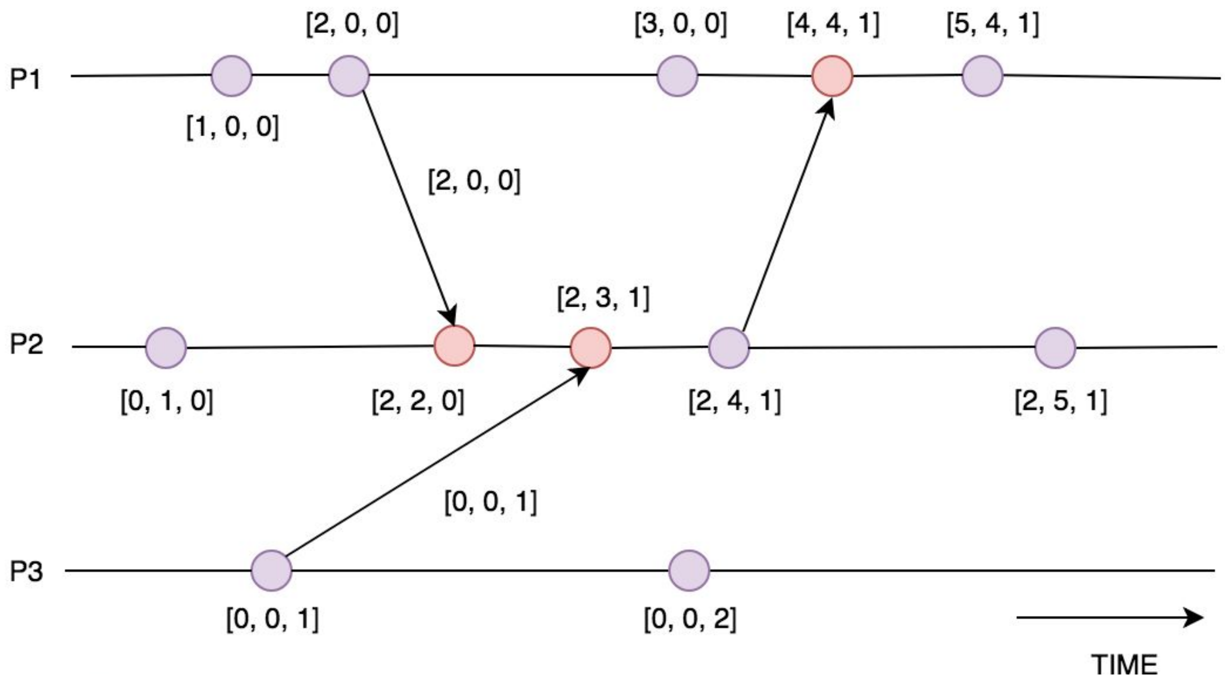
Need

1. For causally ordering events in a distributed system
2. To determine when one thing happened before another

Vector clock

It is an algorithm for generating a partial ordering of events in a distributed system. It detects causality violations. Like the Lamport timestamps interprocess messages contain the state of the sending process's logical clock.

[source: https://cseweb.ucsd.edu/classes/sp16/cse291-e/applications/ln/lecture4.html]



Each process P Each process Pi has a clock has a clock Ci, which is a vector of size n , which is a vector of size n

The clock The clock Ci assigns a vector assigns a vector Ci(a)to any event a at Pi

Update rules Update rules:

- Ci[i]++ for every event at process i]++ for every event at process i
- If a is send of message m from i to j with vector timestamp t If a is send of message m from i to j with vector timestamp tm,then on receipt of m

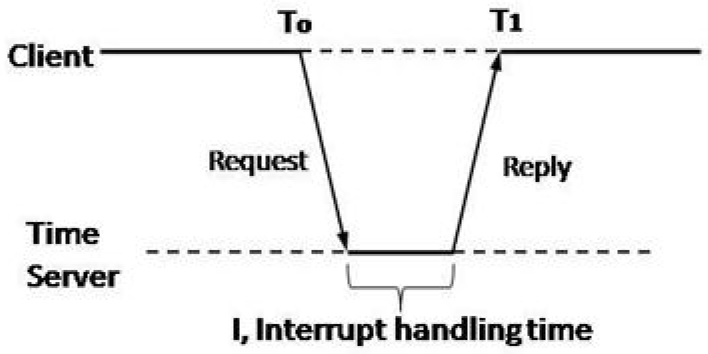
Cj[k] = max(Cj[k], tm[k]) for all k [k]) for all k

What are the drawbacks?

1. Entire vector is sent with message
2. All vector elements (n) have to be checked on every message

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 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 same clock so best estimate of time for propagation is (T1-To)/2.
- When the reply is received at clients 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 Berkley 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 positive or negative value and is calculated based on the knowledge the time server has about the propagation of message.

[source ; http://profile.iiita.ac.in/bibhas.ghoshal/lecture_slides/distributed/Clock.pdf]
[source : https://homes.cs.washington.edu/~arvind/cs425/lectureNotes/clocks-2.pdf]

Some Points to Note

1. **Cristian's algorithm**

- Can also give external synchronization if the time server is sync'ed with external clock reference
- Requires a special node with a time source
- Prone to failure of the central server

2. **Berkeley's algorithm**

- Can be used for internal synchronization only
- No separate time source needed, one of the nodes can be elected as leader and then act as the time server
- Note that the actual time of the central server does not matter, enough for it to tick at around the same rate as other clocks to compute average correctly (why?
- Failures are handled by electing new leader

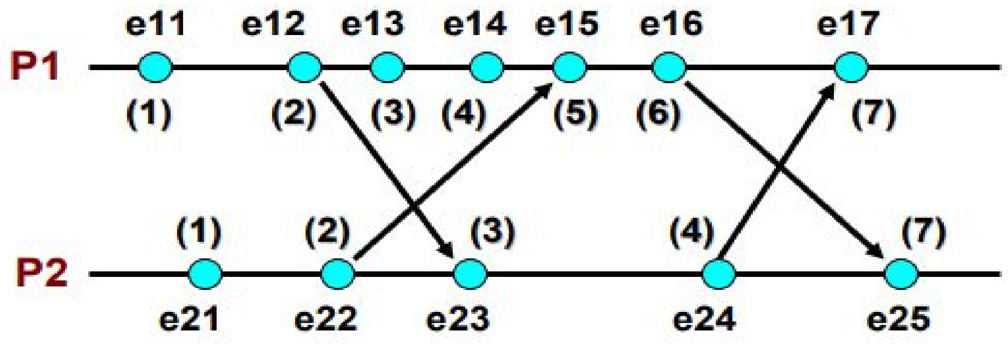
Lamport's Logical Clocks:

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 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.



Limitation

a → b implies C(a) < C(b) b implies C(a) < C(b)

BUT

C(a) < C(b) doesn C(a) < C(b) doesn't imply a t imply a → b !! **(Solution is vector clock)**

Event ordering

One way to define an order of events in a distributed system would be to have a physical clock. So "happened before" event can be described using t1 < t2.

Let's establish a more precise definition of "happens before" (→) in such a system.

- In the same process, if event a comes before event b, then a → b
- When process i sends some message at a to process j and process j acknowledges this message at b, then a → b
- Happens before is transitive. a → b and b → c, then a → c.

Two distinct events a and b are concurrent if a 6→ b and b 6→ a. Assume a 6→ a for any event a. So → is an irreflexive partial ordering on the set of all events in the system. The ordering is only partial because events can be concurrent in which case it is not known which event happened first.

Causal Ordering

Causal message ordering is a partial ordering of messages in a distributed computing environment. It places a restriction on communication between processes by requiring that if the transmission of message mi to process pk necessarily preceded the transmission of message mj to the same process, then the delivery of these messages to that process must be ordered such that mi is delivered before mj .

Global ordering

To totally order the events in a system, the events are ordered according to their times of occurrence. In case two or more events occur at the same time, an arbitrary total ordering ≺ of processes is used. To do this, the relation ⇒ is defined as follows:

If a is an event in process Pi and b is an event in process Pj , then a ⇒ b if and only if either:

i. Ci(a) < Cj (a) or

ii. Ci(a)= Cj (a) and Pi ≺ Pj

There is total ordering because for any two events in the system, it is clear which happened first.