Synchronization

Physical Clocks, Logical Clocks

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

1. Synchronous Systems
   - Physical Clocks
     - Quartz Clocks
     - Atomic Clocks
     - GPS
   - Network Time Protocol
   - Totally Ordered Multicast
2. Asynchronous Systems
   - Happens Before Relationship
   - Totally Ordered Mutual Exclusion

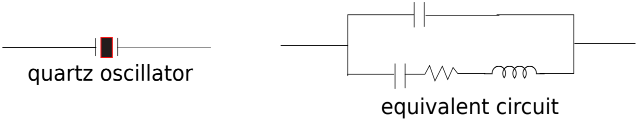
Outline

1. Synchronous Systems
   - Physical Clocks
     - Quartz Clocks
     - Atomic Clocks
     - GPS
   - Network Time Protocol
   - Totally Ordered Multicast
2. Asynchronous Systems
   - Happens Before Relationship
   - Totally Ordered Mutual Exclusion

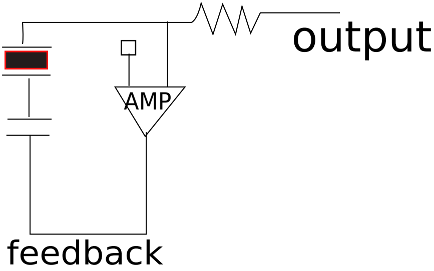
Quartz Based Clock

Quartz Oscillator

- Computers clock use a quartz crystal to generate a clock signal.
- Quartz is a piezoelectric material – generates a voltage, when subjected to mechanical stress.
- Resistant to temperature fluctuations.



Quartz Clock II



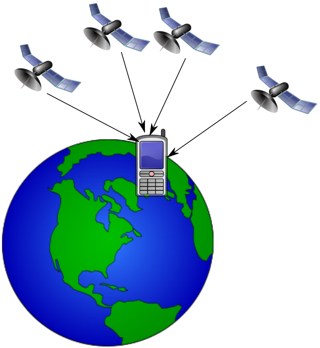
- The quartz oscillator is a part of a self feedback loop.
- It typically oscillates at 32 KHz.
- Processors generate a higher frequency by dividing this clock.
- The clock drift is ± 15 seconds per month (6 ppm).
- A regular quartz clock is not suitable for large distributed

Atomic Clock

Atomic Clock

- Uses a Caesium-133 atom as an oscillator.
- Uses a similar feedback based circuit as the quartz clock.
- Accuracy : 10^{-8} ppm

Use of Atomic Clock: GPS



- Each satellite broadcasts its position (x_i, y_i, z_i) and time t_i
- The time is obtained through an atomic clock.

Finding the Position through GPS

- Current position: (x, y, z)
- The drift between the receiver clock and the atomic clocks is d.
- The time at which the received receives the message is t_r.
- Setup equation:

$$\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} = (t_r - t_i + d) \times c$$

- c is the speed of light
- For four unknowns x, y, z, d, we need at least four equations

Finding the Position through GPS

- Current position: (x, y, z)
- The drift between the receiver clock and the atomic clocks is d.
- The time at which the received receives the message is t_r.
- Setup equation:

$$\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} = (t_r - t_i + d) \times c$$

- c is the speed of light
- For four unknowns x, y, z, d, we need at least four equations

Hence, we need at least four satellites.

Outline

1. Synchronous Systems
   - Physical Clocks
     - Quartz Clocks
     - Atomic Clocks
     - GPS
   - Network Time Protocol
   - Totally Ordered Multicast
2. Asynchronous Systems
   - Happens Before Relationship
   - Totally Ordered Mutual Exclusion

Network Time Protocol

- There are a set of network time servers that have accurate clocks (stratum 1).
- These servers might in turn synchronize with servers that have even more accurate clocks (stratum 0).
- A client machine needs to contact a NTP time server and find the drift between the clocks.
- There are different clock synchronization algorithms.

Cristian's Algorithm

1. Client sends a request to the server at its local time t_1.
2. Server receives it at its local time t_2.
3. Server sends a reply at its local time t_3.
4. Client receives the replay at t_4.

Calculating the Drift - Δ

If we assume that the jitter in the network is 0, then the request and response take the same amount of time. We have

$$\begin{aligned} t_2 - (t_1 + \Delta) &= t_4 + \Delta - t_3 \\ \Rightarrow \Delta &= \frac{(t_2 - t_1) + (t_3 - t_4)}{2} \end{aligned} \quad (1)$$

Cristian's Algorithm

1. Client sends a request to the server at its local time t_1.
2. Server receives it at its local time t_2.
3. Server sends a reply at its local time t_3.
4. Client receives the replay at t_4.

Calculating the Drift - Δ

If we assume that the jitter in the network is 0, then the request and response take the same amount of time. We have

$$\begin{aligned} t_2 - (t_1 + \Delta) &= t_4 + \Delta - t_3 \\ \Rightarrow \Delta &= \frac{(t_2 - t_1) + (t_3 - t_4)}{2} \end{aligned} \tag{1}$$

Shift the clock of the client by Δ

Berkeley Algorithm

- A master is chosen by some method among a group of nodes.
- The master uses Cristian's algorithm to find the clock drift with each slave.
- The master computes the mean value of the drift.
- The master sends an update to each slave regarding the amount that the slave needs to shift its clock.
- This ensures that the clocks of most slaves are relatively synchronized with each other.
- The algorithm also aims to minimize the amount by which each slave needs to adjust its clock.

Outline

Totally Order Multicast with Synchronized Clocks

Problem

Nodes randomly send messages to a subset of other nodes. The network has a non-deterministic delay. It is bounded by Δ. Ensure that all the messages are delivered in the same order at all nodes.

Solution

Sender: Timestamp every message with local time.
Receiver:

1. For a message with timestamp t, transfer it to the receive queue at time $t + \Delta$.
2. Deliver the messages in the receive queue in the order of their timestamps.

Outline

Definitions

- Our distributed system does not have a notion of global time .
- It contains a set of processes.
- Each process issues its own set of events .
- A process can send a message to another process.

Happens-before relationship($\rightarrow$)

1. If a process issues event a before b, then $a \rightarrow b$.
2. If event a is the sending of a message by one process and b is its receipt by another process. Then $a \rightarrow b$.
3. If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Definitions - II

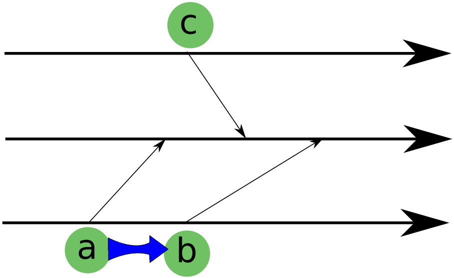
- If $a \nrightarrow b$ and $b \nrightarrow a$, then $a \bowtie b$(concurrent)
- If a happens before b, then we say that a causally affects b
- Let us assign a number to each event: $\tau(a)$
- We want it to satisfy some conditions
  - Clock Condition : $(a \rightarrow b) \Rightarrow \tau(a) < \tau(b)$
  - C1: If $a \rightarrow b$ and they belong to the same process, then $\tau(a) < \tau(b)$
  - C2: If a represents a send, and b is its receipt, then $\tau(a) < \tau(b)$

Enforcing the Clock Condition

- Every process keeps a clock that is initialized to 0. Process i's clock is τ_i.
- Each process increments τ_i between two successive events.
- If event a is the sending of an event by process i, then this process embeds $\tau_i(a)$ in the message.
  - $\tau(a) = \tau_i(a)$
- Let b be the receive event at process j.
  - $\tau_j = \tau_j(b) = max(\tau_j, \tau_i(a)) + 1$
  - $\tau(b) = \tau_j(b)$
- This method provides a partial ordering .

Vector Clocks: Motivation

- Clock Condition: $a \rightarrow b$ implies $\tau(a) < \tau(b)$
- Is it true that: $\tau(a) < \tau(b)$ implies $a \rightarrow b$
  - This would mean that $a \bowtie b$ implies $\tau(a) = \tau(b)$
  - Not True



Vector Clocks: Design

Vector Clock

- If there are n processes, every process maintains a n element array $\mathcal{V}_i$
- Process i increments $\mathcal{V}_i(i)$ before sending or receiving a message, and on every internal event.
- Every message is timestamped with the vector clock of the sender
- The receiver merges the clocks:
  - Assume: i sends a message to j
  - $\forall k, \mathcal{V}_j(k) = max(\mathcal{V}_i(k), \mathcal{V}_j(k))$
- $\mathcal{V}_i < \mathcal{V}_j \Rightarrow (\forall k, \mathcal{V}_i(k) \leq \mathcal{V}_j(k)) \wedge (\exists k, \mathcal{V}_i(k) < \mathcal{V}_j(k))$

Additional Properties

1. $\mathcal{V}_a < \mathcal{V}_b \Leftrightarrow a \rightarrow b$
2. $(\mathcal{V}_a \not\leq \mathcal{V}_b) \wedge (\mathcal{V}_a \not\geq \mathcal{V}_b) \Leftrightarrow a \bowtie b$

Outline

Total Ordering $\Rightarrow$

- Let us consider two events a and b belonging to processes i and j
  - $a \Rightarrow b$, if $\tau_i(a) < \tau_j(b)$
  - $a \Rightarrow b$, if $\tau_i(a) = \tau_j(b)$, and $i \prec j$

Ordered Mutual Exclusion Problem

- A certain resource can be owned by only one process. It must be explicitly granted and released.
- Different requests must be granted in the order in which they were made.
- If no process hangs forever after taking the resource, every request is ultimately granted.

Algorithm for Solving the Mutual Exclusion Problem

Resource Request

1. To request a resource, P_i sends a message: (T_M, i) to all nodes, and also puts the message in its request queue . $T_M = \tau_i$
2. When P_j receives (T_M, i), it places it in its request queue, and sends a timestamped acknowledgement .

Resource Release

1. P_i removes any (T_M, i) messages in its queue, and sends a timestamped P_i releases message to all other processes.
2. When process P_j receives a release message from process i, it removes any request message from process i in its request queue.

Algorithm - II

Resource Access

Access the resource when both these conditions are met:

1. (T_M, i) is the earliest message in the queue.
2. The process has received a message with timestamp greater than T_M from every other process.

Proof – Main Idea

Objectives

1. If the resource is free, then some process will get it.
2. No two processes can get the resource at the same time.
3. Processes get the resource in the order of the requests.

Discussion

- If a process is getting a resource, then there are two possibilities
  1. It has seen requests by all other processes.
  2. It has not seen the request of some set of processes, but it has seen messages that precede them.

Time, clocks, and the ordering of events in a distributed system by Leslie Lamport, Communications of the ACM, 1978