Clock Synchronization

- Physical clocks
- Logical clocks
- Vector clocks

Physical clocks

Problem

Sometimes we simply need the exact time, not just an ordering.

Solution

Universal Coordinated Time (UTC):

- Based on the number of transitions per second of the cesium 133 atom (pretty accurate).
- At present, the real time is taken as the average of some 50 cesium-clocks around the world.
- Introduces a leap second from time to time to compensate that days are getting longer.

Note

UTC is broadcast through short wave radio and satellite. Satellites can give an accuracy of about ± 0.5 ms.

Physical clocks

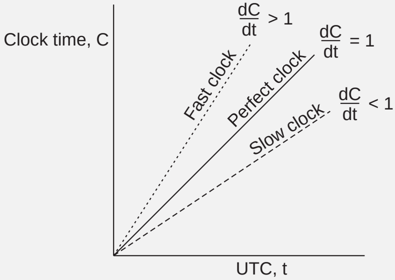
Problem

Suppose we have a distributed system with a UTC-receiver somewhere in it $\Rightarrow$ we still have to distribute its time to each machine.

Basic principle

- Every machine has a timer that generates an interrupt H times per second.
- There is a clock in machine p that ticks on each timer interrupt. Denote the value of that clock by $C_p(t)$, where t is UTC time.
- Ideally, we have that for each machine p, $C_p(t) = t$, or, in other words, $dC/dt = 1$.

Physical clocks



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

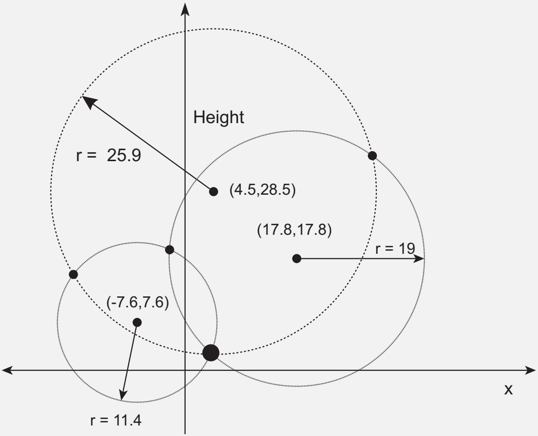
Goal

Never let two clocks in any system differ by more than δ time units $\Rightarrow$ synchronize at least every $\delta/(2\rho)$ seconds.

Global positioning system

Basic idea

You can get an accurate account of time as a side-effect of GPS.



Global positioning system

Problem

Assuming that the clocks of the satellites are accurate and synchronized:

- It takes a while before a signal reaches the receiver
- The receiver's clock is definitely out of synch with the satellite

Global positioning system

Principal operation

- Δ_r: unknown deviation of the receiver's clock.
- x_r, y_r, z_r: unknown coordinates of the receiver.
- T_i: timestamp on a message from satellite i
- $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i.
- Measured distance to satellite i: $c \times \Delta_i$
  (c is speed of light)
- Real distance is

$$d_i = c\Delta_i - c\Delta_r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

Observation

4 satellites $\Rightarrow$ 4 equations in 4 unknowns (with Δ_r as one of them)

Global positioning system

Principal operation

- Δ_r: unknown deviation of the receiver's clock.
- x_r, y_r, z_r: unknown coordinates of the receiver.
- T_i: timestamp on a message from satellite i
- $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i.
- Measured distance to satellite i: $c \times \Delta_i$
  (c is speed of light)
- Real distance is

$$d_i = c\Delta_i - c\Delta_r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

Observation

4 satellites $\Rightarrow$ 4 equations in 4 unknowns (with Δ_r as one of them)

Global positioning system

Principal operation

- Δ_r: unknown deviation of the receiver's clock.
- x_r, y_r, z_r: unknown coordinates of the receiver.
- T_i: timestamp on a message from satellite i
- $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i.
- Measured distance to satellite i: $c \times \Delta_i$
  (c is speed of light)
- Real distance is

$$d_i = c\Delta_i - c\Delta_r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

Observation

4 satellites $\Rightarrow$ 4 equations in 4 unknowns (with Δ_r as one of them)

Global positioning system

Principal operation

- Δ_r: unknown deviation of the receiver's clock.
- x_r, y_r, z_r: unknown coordinates of the receiver.
- T_i: timestamp on a message from satellite i
- $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i.
- Measured distance to satellite i: $c \times \Delta_i$
  (c is speed of light)
- Real distance is

$$d_i = c\Delta_i - c\Delta_r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

Observation

4 satellites $\Rightarrow$ 4 equations in 4 unknowns (with Δ_r as one of them)

Global positioning system

Principal operation

- Δ_r: unknown deviation of the receiver's clock.
- x_r, y_r, z_r: unknown coordinates of the receiver.
- T_i: timestamp on a message from satellite i
- $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i.
- Measured distance to satellite i: $c \times \Delta_i$
  (c is speed of light)
- Real distance is

$$d_i = c\Delta_i - c\Delta_r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

Observation

4 satellites $\Rightarrow$ 4 equations in 4 unknowns (with Δ_r as one of them)

Global positioning system

Principal operation

- Δ_r: unknown deviation of the receiver's clock.
- x_r, y_r, z_r: unknown coordinates of the receiver.
- T_i: timestamp on a message from satellite i
- $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i.
- Measured distance to satellite i: $c \times \Delta_i$
  (c is speed of light)
- Real distance is

$$d_i = c\Delta_i - c\Delta_r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

Observation

4 satellites $\Rightarrow$ 4 equations in 4 unknowns (with Δ_r as one of them)

Global positioning system

Principal operation

- Δ_r: unknown deviation of the receiver's clock.
- x_r, y_r, z_r: unknown coordinates of the receiver.
- T_i: timestamp on a message from satellite i
- $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i.
- Measured distance to satellite i: $c \times \Delta_i$
  (c is speed of light)
- Real distance is

$$d_i = c\Delta_i - c\Delta_r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

Observation

4 satellites $\Rightarrow$ 4 equations in 4 unknowns (with Δ_r as one of them)

Global positioning system

Principal operation

- Δ_r: unknown deviation of the receiver's clock.
- x_r, y_r, z_r: unknown coordinates of the receiver.
- T_i: timestamp on a message from satellite i
- $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i.
- Measured distance to satellite i: $c \times \Delta_i$
  (c is speed of light)
- Real distance is

$$d_i = c\Delta_i - c\Delta_r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

Observation

4 satellites $\Rightarrow$ 4 equations in 4 unknowns (with Δ_r as one of them)

Clock synchronization principles

Principle I

Every machine asks a time server for the accurate time at least once every $\delta/(2\rho)$ seconds (Network Time Protocol).

Note

Okay, but you need an accurate measure of round trip delay, including interrupt handling and processing incoming messages.

Clock synchronization principles

Principle II

Let the time server scan all machines periodically, calculate an average, and inform each machine how it should adjust its time relative to its present time.

Note

Okay, you'll probably get every machine in sync. You don't even need to propagate UTC time.

Fundamental

You'll have to take into account that setting the time back is never allowed $\Rightarrow$ smooth adjustments.

The Happened-before relationship

Problem

We first need to introduce a notion of ordering before we can order anything.

The happened-before relation

- If a and b are two events in the same process, and a comes before b, then $a \to b$.
- If a is the sending of a message, and b is the receipt of that message, then $a \to b$
- If $a \to b$ and $b \to c$, then $a \to c$

Note

This introduces a partial ordering of events in a system with concurrently operating processes.

The Happened-before relationship

Problem

We first need to introduce a notion of ordering before we can order anything.

The happened-before relation

- If a and b are two events in the same process, and a comes before b, then $a \rightarrow b$.
- If a is the sending of a message, and b is the receipt of that message, then $a \rightarrow b$
- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Note

This introduces a partial ordering of events in a system with concurrently operating processes.

The Happened-before relationship

Problem

We first need to introduce a notion of ordering before we can order anything.

The happened-before relation

- If a and b are two events in the same process, and a comes before b, then $a \rightarrow b$.
- If a is the sending of a message, and b is the receipt of that message, then $a \rightarrow b$
- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Note

This introduces a partial ordering of events in a system with concurrently operating processes.

Logical clocks

Problem

How do we maintain a global view on the system's behavior that is consistent with the happened-before relation?

Solution

Attach a timestamp $C(e)$ to each event e, satisfying the following properties:

- P1 If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$.
- P2 If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$.

Problem

How to attach a timestamp to an event when there's no global clock $\Rightarrow$ maintain a consistent set of logical clocks, one per process.

Logical clocks

Problem

How do we maintain a global view on the system's behavior that is consistent with the happened-before relation?

Solution

Attach a timestamp $C(e)$ to each event e, satisfying the following properties:

- P1 If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$.
- P2 If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$.

Problem

How to attach a timestamp to an event when there's no global clock $\Rightarrow$ maintain a consistent set of logical clocks, one per process.

Logical clocks

Problem

How do we maintain a global view on the system's behavior that is consistent with the happened-before relation?

Solution

Attach a timestamp $C(e)$ to each event e, satisfying the following properties:

- P1 If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$.
- P2 If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$.

Problem

How to attach a timestamp to an event when there's no global clock $\Rightarrow$ maintain a consistent set of logical clocks, one per process.

Logical clocks

Solution

Each process P_i maintains a local counter C_i and adjusts this counter according to the following rules:

1: For any two successive events that take place within P_i, C_i is incremented by 1.

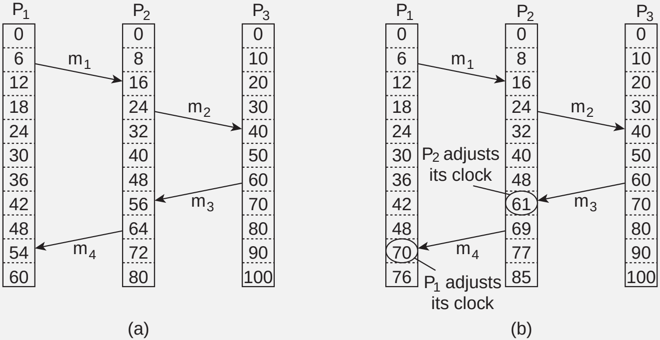
2: Each time a message m is sent by process P_i, the message receives a timestamp $ts(m) = C_i$.

3: Whenever a message m is received by a process P_j, P_j adjusts its local counter C_j to $\max\{C_j, ts(m)\}$; then executes step 1 before passing m to the application.

Notes

- Property P1 is satisfied by (1); Property P2 by (2) and (3).
- It can still occur that two events happen at the same time. Avoid this by breaking ties through process IDs.

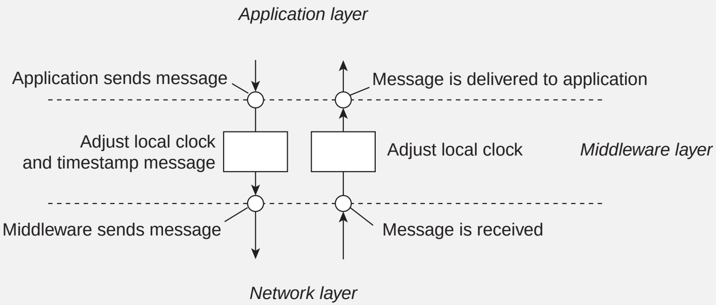
Logical clocks – example



Logical clocks – example

Note

Adjustments take place in the middleware layer

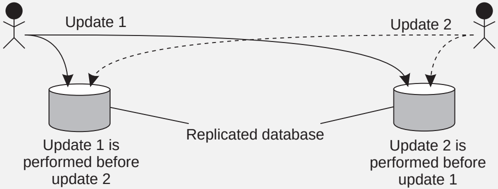


Example: Totally ordered multicast

Problem

We sometimes need to guarantee that concurrent updates on a replicated database are seen in the same order everywhere:

- P_1 adds \$100 to an account (initial value: \$1000)
- P_2 increments account by 1%
- There are two replicas



Result

In absence of proper synchronization:
replica #1 $\leftarrow$ \$1111, while replica #2 $\leftarrow$ \$1110.

Example: Totally ordered multicast

Solution

- Process P_i sends timestamped message msg_i to all others. The message itself is put in a local queue $queue_i$.
- Any incoming message at P_j is queued in $queue_j$, according to its timestamp, and acknowledged to every other process.

P_j passes a message msg_i to its application if:

(1) msg_i is at the head of $queue_j$
(2) for each process P_k, there is a message msg_k in $queue_j$ with a larger timestamp.

Note

We are assuming that communication is reliable and FIFO ordered.

Example: Totally ordered multicast

Solution

- Process P_i sends timestamped message msg_i to all others. The message itself is put in a local queue $queue_i$.
- Any incoming message at P_j is queued in $queue_j$, according to its timestamp, and acknowledged to every other process.

P_j passes a message msg_i to its application if:

(1) msg_i is at the head of $queue_j$
(2) for each process P_k, there is a message msg_k in $queue_j$ with a larger timestamp.

Note

We are assuming that communication is reliable and FIFO ordered.

Example: Totally ordered multicast

Solution

- Process P_i sends timestamped message msg_i to all others. The message itself is put in a local queue $queue_i$.
- Any incoming message at P_j is queued in $queue_j$, according to its timestamp, and acknowledged to every other process.

P_j passes a message msg_i to its application if:

(1) msg_i is at the head of $queue_j$
(2) for each process P_k, there is a message msg_k in $queue_j$ with a larger timestamp.

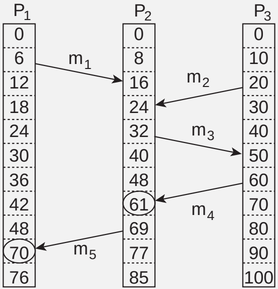
Note

We are assuming that communication is reliable and FIFO ordered.

Vector clocks

Observation

Lamport's clocks do not guarantee that if $C(a) < C(b)$ that a causally preceded b



Observation

Event a: m_1 is received at $T = 16$;
Event b: m_2 is sent at $T = 20$.

Note

We cannot conclude that a causally precedes b.

Vector clocks

Solution

- Each process P_i has an array $VC_i[1..n]$, where $VC_i[j]$ denotes the number of events that process P_i knows have taken place at process P_j.
- When P_i sends a message m, it adds 1 to $VC_i[i]$, and sends VC_i along with m as vector timestamp $vt(m)$. Result: upon arrival, recipient knows P_i's timestamp.
- When a process P_j delivers a message m that it received from P_i with vector timestamp $ts(m)$, it

  (1) updates each $VC_j[k]$ to $\max\{VC_j[k], ts(m)[k]\}$
  (2) increments $VC_j[j]$ by 1.

Question

What does $VC_i[j] = k$ mean in terms of messages sent and received?

Vector clocks

Solution

- Each process P_i has an array $VC_i[1..n]$, where $VC_i[j]$ denotes the number of events that process P_i knows have taken place at process P_j.
- When P_i sends a message m, it adds 1 to $VC_i[i]$, and sends VC_i along with m as vector timestamp $vt(m)$. Result: upon arrival, recipient knows P_i's timestamp.
- When a process P_j delivers a message m that it received from P_i with vector timestamp $ts(m)$, it

  (1) updates each $VC_j[k]$ to $\max\{VC_j[k], ts(m)[k]\}$
  (2) increments $VC_j[j]$ by 1.

Question

What does $VC_i[j] = k$ mean in terms of messages sent and received?

Causally ordered multicasting

Observation

We can now ensure that a message is delivered only if all causally preceding messages have already been delivered.

Adjustment

P_i increments $VC_i[i]$ only when sending a message, and P_j "adjusts" VC_j when receiving a message (i.e., effectively does not change $VC_j[j]$).

P_j postpones delivery of m until:

- $ts(m)[i] = VC_j[i] + 1$.
- $ts(m)[k] \leq VC_j[k]$ for $k \neq i$.

Causally ordered multicasting

Observation

We can now ensure that a message is delivered only if all causally preceding messages have already been delivered.

Adjustment

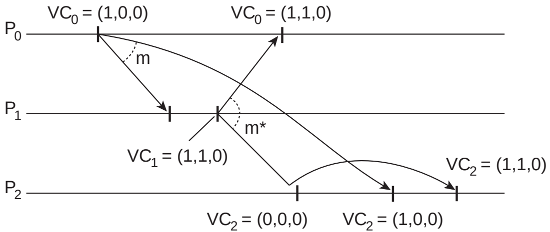
P_i increments $VC_i[i]$ only when sending a message, and P_j "adjusts" VC_j when receiving a message (i.e., effectively does not change $VC_j[j]$).

P_j postpones delivery of m until:

- $ts(m)[i] = VC_j[i] + 1$.
- $ts(m)[k] \leq VC_j[k]$ for $k \neq i$.

Causally ordered multicasting

Example

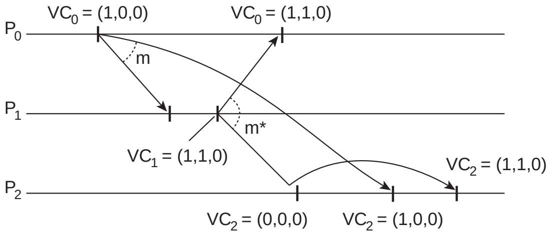


Example

Take $VC_2 = [0,2,2]$, $ts(m) = [1,3,0]$ from P_0. What information does P_2 have, and what will it do when receiving m (from P_0)?

Causally ordered multicasting

Example



Example

Take $VC_2 = [0,2,2]$, $ts(m) = [1,3,0]$ from P_0. What information does P_2 have, and what will it do when receiving m (from P_0)?