Time and Clock

Real primary standard is based on the rotation of earth.

De facto primary standard uses atomic clocks.

1 sec = 9,192,631,770 orbital transitions of Cs$^{133}$ atom.

86400 atomic seconds = 1 solar day - 3 ms. Atomic clocks are fast becoming cost effective.

If the physical clocks are not perfectly synchronized, how do we decide if one event happened before another?

The central concept here is that of causality.
Sequential and Concurrent events

*Sequential* implies “Totally ordered in time.” In a single process, there is one clock. So total ordering is feasible.

This is not true in a distributed system. It leads to the following two issues:

- Physical clock synchronization
- Can we define sequential and concurrent events without using physical clocks?
Causality

Joke ↖ Re: joke (↑ implies causally ordered before)

Message sent ↖ message received

Local ordering a ↖ b ↖ c (based on the local clock)

Rule 1. If a, b are two events in a single process P, and the time of a is less than the time of b then a ↖ b.

Rule 2. If a = sending a message, and b = receipt of that message, then a ↖ b.

Rule 3. a ↖ b ↖ b ↖ c □ a ↖ c
Example

\[\begin{align*}
a &\prec d & \text{since} & (a \prec b \land b \prec c \land c \prec d) \\
e &\prec d & \text{since} & (e \prec f \land f \prec d)
\end{align*}\]

\(\prec\) defines a PARTIAL order.

Is \(g \prec f\) or \(f \prec g\)? NO. They are concurrent.

Concurrency = absence of causal order.
**Logical Clocks**

**LC** is a counter. Its value respects causal ordering.

With each process there is an **LC**. Initially **LC** = 0. **LC** is incremented in such a way that it satisfies the following condition:

\[ a \prec b \implies LC(a) < LC(b) \]

**LC1.** Each time a local event takes place, increment **LC**.

**LC2.** Append the value of **LC** to outgoing messages.

**LC3.** When receiving a message, set **LC** to

\[ 1 + \max \text{(local LC, message LC)} \]

Note that **LC(a) < LC(b)** does **NOT** imply \( a \prec b \).
What about a total order?

Yes, it is important for some applications. Examples?

Causal order is strengthened to define a total order (\(<\)) among events.

Let \(a, b\) be events in processes \(i\) and \(j\) respectively. Then

\[
a \prec b \quad \text{iff} \quad \text{LC}(a) < \text{LC}(b),
\]

or

\[
\text{LC}(a) = \text{LC}(b) \quad \text{and} \quad i < j
\]

The (id, LC) value sent out with each message is called its *timestamp*.

\[
a \prec b \nsubsetneq a \prec b, \text{ but the converse is not true.}
\]
Vector Clocks

Causality detection can be an important issue in applications like group communication.

Timestamps cannot detect causality. However, causality can be detected using vector clocks. The goal is to satisfy the following condition:

\[ a < b \quad \text{if} \quad VC(a) < VC(b) \]
Vector Clock of an event in a system of 8 processes

$VC(a)[j]$ denotes the $j$th component of $VC(a)$

**Definition.** $VC(a) < VC(b)$ iff

\[
\begin{align*}
\forall i : 0 \leq i \leq N-1 : VC(a)[i] &\leq VC(b)[i], \text{ and} \\
\forall j : 0 \leq j \leq N-1 : VC(a)[j] &< VC(b)[j],
\end{align*}
\]

So, $[3, 3, 4, 5, 3, 2, 1, 4] < [3, 3, 4, 5, 3, 2, 2, 5]$

But, $[3, 3, 4, 5, 3, 2, 1, 4]$ and $[3, 3, 4, 5, 3, 2, 2, 3]$ are not comparable to each other
**Implementing VC**

![Vector Clock Diagram]

**VC1.** Increment VC[i]) for each local event in process i.

**VC2.** Append VC to every outgoing message.

**VC3.** When process j receives a message with a vector timestamp T from another process, it first increments the jth component VC[j] of its own vector clock, (i.e. VC[j] := VC[j] + 1) and then updates its vector clock as follows:

∀k: 0 ≤ k ≤ N-1:: VC[k] := max(T[k], VC[k]).
Physical Clock Synchronization

Question 1. Why is physical clock synchronization so important?

Question 2. With atomic clocks becoming affordable, should we care about physical clock synchronization?

Types of Synchronization

- External Synchronization
- Internal Synchronization
- Phase Synchronization

Types of clocks

- Unbounded 0, 1, 2, 3, ... 
- Bounded 0, 1, 2, ... M-1, 0, 1, ...

Unbounded clocks are not realistic, but sometimes they are easier to deal with in algorithms.
What are these?

Drift rate $\square$
Clock skew $\square$
Resynchronization interval $R$

Challenges

Drift is unavoidable
Accounting for propagation delay
Processing delay
Faulty clocks
Internal Synchronization

A simple averaging algorithm

Step 1. Read every clock in the system.
Step 2. Discard "bad" clock values and substitute them by the value of the local clock.
Step 3. Update the clock using the average of these values.

Synchronization is maintained if $n > 3t+1$ where $t$ is the number of faulty (may be 2-faced) clocks. Why?

$R = \frac{\Box}{\Box(3t+1)}$. Why?