Logical Clocks

CS 451 Lecture

Prof. R.K. Joshi
Dept of CSE
IIT Bombay
Synchronizing between machines

- In a distributed system, machines are spatially separated.
- They communicate by exchanging messages.
- Delays are not negligible.
- It’s sometimes impossible to say which of the 2 events occurred first? (happened before relation: only partial ordering)
Lamport’s Contribution

- Partial ordering defined by happened before relation
- Extend it to provide a consistent total ordering of all events in a distributed system
happened before Relation

When do you say event a happened before event b?

When a’s real time stamp is earlier than that of b

A problem with this scheme is that all events are not observable from a given system and all real clocks are not synchronized.
If $a$, $b$ are events in the same process, and $a$ comes before $b$, then $a \rightarrow b$

If $a$ is the sending of a message by one process and $b$ is receiving of message by another, $a \rightarrow b$

If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

If $a \nsim \rightarrow b$ and $b \nsim \rightarrow$ then $a$ and $b$ are said to be concurrent
Example Space-Time Diagram

Process P  Process Q  Process R
Example

Process P
pl

Process Q
ql

Process R
Example
Example

Process P

Process Q

Process R
Example
Example

Process P

Process Q

Process R
Example
Example
Example
Example

Process P

Process Q

Process R
Example
What can be said about the events in the above example?

- If $a \rightarrow b$, then you will be able to move along the time lines and message links from $a$ to $b$ in the space time diagram.
- $a \rightarrow b$ means that it is possible for $a$ to causally affect $b$.
- Two events are concurrent if neither can causally affect the other.
Which events are concurrent?

And which of them can be ordered?

Not all can be totally ordered

How to come up with a relation which will allow us to totally order all events in a distributed system

\[\rightarrow\text{In next class!}\]
Towards Defining a total order: Logical Clocks

Clock $C_i$ for process $P_i$ assigns a number $C_i(a)$ to any event $a$ in $P_i$.

This number is called Time Stamp for event $a$.

No assumptions about relation to physical time: hence logical clock.
Conditions satisfied by the system of clocks

- If \( a \rightarrow b \) then \( C(a) < C(b) \)

if \( a, b \) are in same process \( P_i \)
\[ C_i(a) < C_i(b) \]

if \( a \) is send event in \( P_i \) and \( b \) is the receive event in \( P_j \)
\[ C_i(a) < C_j(b) \]
Example

With $\rightarrow$ relation, there is no total order as there is always a possibility of concurrent events. If you apply clocks to order, equal time stamps are still a problem.
How to implement the above conditions for clock values for the happened before relation?

If $a$ and $b$ are two successive events in same process, $a \rightarrow b$
implement $C_i(b) = C_i(a) + k \ (k>0)$

If $a$ is send event in $P_i$ and its corresponding receive event is $b$ in $P_j$,
implement $C_j(b) = \max (C_i(a), C_j(b-1)) + k \ (k>0)$
Obtaining Total Order with relation

If a is an event in Pi and b is an event in Pj, then a \( \rightarrow \) b iff

Either \( C_i(a) < C_j(b) \) or

\( C_i(a) = C_j(b) \) and Pi \( \prec \) Pj such that relation \( \prec \) totally orders processes
Example

With \( \Rightarrow \) relation, there is total order as there is always a total order:

\[ e_{11}, e_{12}, e_{13}, e_{14}, e_{21}, e_{22}, e_{23}, e_{15}, e_{24}, e_{16} \]
Reference Reading Material for the course