Introduction

- How do you assign order to events happening in a distributed system?
  - No central clock.
  - Theoretically impossible to fully synchronize clocks.
  - Decentralized
Outline

- Introduction
- The Partial Ordering
- Logical Clocks
- Total Ordering of Events
- Physical Clocks
- Applications
- Conclusion

The Partial Ordering

- Definition: “happened before”, denoted as →,
  1) If a and b are events in the same process, and a comes before b, then a → b.
  2) If a is the sending of a message by one process and b is the receipt of the same message by another process, then a → b.
  3) If a → b and b → c then a → c.
     Two distinct events a and b are said to be concurrent if a! → b and b! → a.
  4) a! → a for any event a.

• p1→p3, q1→q7
• p1→q2, q1→r4
• p1→q4, p1→r4, p3 concurrent q3
Logical Clocks

- Define clock $C_i(a)$ on processor $P_i$ as a function that assigns a number to event “$a$”.
  - $C_i: a \rightarrow N_0$
- Define $C(a) = C_i(a)$ if $a$ is an event on $P_i$.
- Clock Condition:
  - For all $a,b$: if $a \rightarrow b$, then $C_i(a) < C_i(b)$

Logical Clocks: Satisfying clock condition

- IR1. Each process $P_i$ increments $C_i$ between any two successive events.
- IR2.
  - (a) If event $a$ is the sending of a message $m$ by process $P_i$, then the message $m$ contains a timestamp $T_m = C_i(a)$.
  - (b) Upon receiving a message $m$, process $P_i$ sets $C_i$ greater than or equal to its present value and greater than $T_m$. 
Total Ordering of Events

- Using logical clocks it is simple to produce a total ordering of events ($\Rightarrow$)
  - $a \Rightarrow b$ if and only if either
    1. $C_i(a) < C_j(b)$
    2. $C_i(a) = C_j(b)$ and $P_i < P_j$.
  - $a \rightarrow b$ implies $a \Rightarrow b$

Outline

- Background
- The Partial Ordering
- Logical Clocks
- Total Ordering of Events
  - Physical Clocks
- Applications
- Conclusion
Strong Clock Condition

- What happens when some communication is out of band?
  - Remember IR2 (b) condition
    - Upon receiving a message \( m \), process \( P_i \) sets \( C_i \) greater than or equal to its present value and greater than \( T_m \).
  - Because message was send out of band, it is possible that \( a \rightarrow b \), but \( C^{<a>} > C^{<b>} \).

- Strong Clock Condition
  - Let \( a \rightarrow b \), as a happening before \( b \).
  - For any events \( a, b \) in \( L \), if \( a \rightarrow b \) then \( C^{<a>} < C^{<b>} \).

Physical Clock

- Introduce a physical clock
  - Accurate
    - There exists a constant \( \kappa \ll 1 \), such that for all \( i \):
      \[
      \text{sgn} \left( \frac{dC_i(t)}{dt} - 1 \right) < \kappa
      \]
  - Synchronized
    - For all \( i, j \):
      \[
      \left| C_i(t) - C_j(t) \right| < \varepsilon
      \]

- Avoid anomalous behavior
  - For any \( i, j, t \):
    \[
    C_i(t + \mu) - C_j(t) > 0
    \]
    Where \( \mu \) is the transmission speed.
  - \( \varepsilon/(1- \kappa) \leq \mu \) must hold
Physical Clock: Synchronization

- When $P_i$ sends a message $m$ at physical time $t$ to $P_j$, $m$ contains a timestamp $T_m = C(t)$.
- Upon receiving a message $m$ at time $t'$, process $P_j$ sets $C_j(t') = \max(C_j(t' - 0), T_m + \text{min delay})$.
- Theorem states that given the bounds on maximum number of hops and if messages are sent frequently enough, synchronization condition holds:
  - For all $i, j$: $|C_i(t) - C_j(t)| < \varepsilon$

Applications

- Granting exclusive right to a resource
  - Use logical clocks to assign ordering to requests (done individually at each process)
  - Move on to next task as soon as got confirmation of a release.
- Updates in Peer-to-Peer network
Conclusion

- Questions?