

Time, Clocks, and the Ordering of Events in a Distributed System

By: Leslie Lamport

Presented by: Konstantin Berlin

<http://www.cs.umd.edu/~kberlin/cmcs714/cmcs714pres.ppt>

1

Introduction

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

2

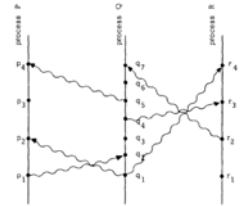
Outline

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

3

The Partial Ordering

- Definition: "happened before", denoted as \rightarrow ,
 - 1) If a and b are events in the same process, and a comes before b, then $a \rightarrow b$.
 - 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 $a \rightarrow b$.
 - 3) If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.
 - 4) If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$. Two distinct events a and b are said to be *concurrent* if $a \not\rightarrow b$ and $b \not\rightarrow a$.

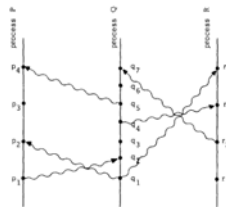


- $p1 \rightarrow p3, q1 \rightarrow q7$
- $p1 \rightarrow q2, q1 \rightarrow r4$
- $p1 \rightarrow q4, p1 \rightarrow r4, p3$ concurrent $q3$

4

Logical Clocks

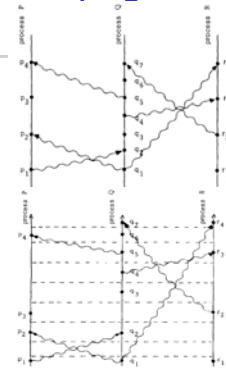
- Define clock $C_i \langle a \rangle$ on processor P_i as function that assigns number to event "a".
 - $C_i: a \rightarrow N_0$
- Define $C \langle a \rangle = C_i \langle a \rangle$ if a is event on P_i .
- Clock Condition:
 - For all a,b: if $a \rightarrow b$, then $C \langle a \rangle < C \langle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



5

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 $T_m = C_i \langle a \rangle$.
 - (b) Upon receiving a message m, process P_j sets C_j greater than or equal to its present value and greater than T_m .



5

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
 - $C_i(a) < C_j(b)$
 - $C_i(a) = C_j(b)$ and $P_i < P_j$.
 - $a \rightarrow b$ implies $a \Rightarrow b$

7

Outline

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

8

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 sent 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 $C(a) < C(b)$.

9

Physical Clock

- Introduce a physical clock
 - Accurate
 - There exists a constant $\kappa \ll 1$, such that for all i : $|dC_i(t)/dt - 1| < \kappa$
 - Synchronized
 - For all i, j : $|C_i(t) - C_j(t)| < \epsilon$
- Avoid anomalous behavior
 - For any i, j, t : $C_i(t + \mu) - C_j(t) > 0$. Where μ is the transmission speed.
 - $\epsilon/(1 - \kappa) \leq \mu$ must hold

10

Physical Clock: Synchronization

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

11

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

12



Conclusion

- Questions?