CSE 486/586 Distributed Systems

Logical Time

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time

- Clock skews do happen
- Cristian’s algorithm
  - One server
  - Server-side timestamp and one-way delay estimation
- NTP (Network Time Protocol)
  - Hierarchy of time servers
  - Estimates the actual offset between two clocks
  - Designed for the Internet

Then Came a Breakthrough…

- We cannot sync multiple clocks perfectly.
- But why did we want to synchronize clocks in the first place?

Then Came a Breakthrough…

- If we just want to order events happened at different processes, we don’t need to synchronize physical clocks.
- We just need to be able to determine the ordering.
- So the concept of logical time:
  - First proposed by Leslie Lamport in the 70’s
  - Based on causality of events
  - Defined relative time, not absolute time
- Critical observation: time (ordering) only matters if two or more processes interact, i.e., send/receive messages.

Abstract View

- Background: we’ll think of a program as a collection of actions: instruction, send, and receive events.
- Above is what we will deal with most of the time.
  - This is the execution view of a distributed system.
- Ordering question: what do we ultimately want?
  - Taking two events and determine the ordering of the two.

What Ordering?

- What kind of orderings can we determine right away?
  - Events in the same process
  - Send/receive events
Lamport Timestamps
- Goal: take any two events, and determine the ordering of the two.
- It uses a single number to do so.
- Basic idea
  - But each process needs to know a time value

Logical Clocks
- (Lamport algorithm assigns logical timestamps.)
- Each process uses a counter with initial value of zero
  - A process increments its counter when a send or an instruction happens at it. The counter is assigned to the event as its timestamp.
  - A send (message) event carries its timestamp
  - For a receive (message) event the counter is updated by max(local clock, message timestamp) + 1

Walk-Thru
- Algorithm
  - All processes use a counter (clock) with initial value of zero
  - A process increments its counter when a send or an instruction happens at it. The counter is assigned to the event as its timestamp
  - A send (message) event carries its timestamp
  - For a receive (message) event the counter is updated by max(local clock, message timestamp) + 1

Happened Before
- Define a logical relation happened-before (→) among events:
  - On the same process: a → b, if time(a) < time(b)
  - If p1 sends m to p2: send(m) → receive(m)
  - (Transitivity) If a → b and b → c then a → c
  - Shows causality of events (a chain of events that are causally related)

CSE 486/586 Administrivia
- PA2A is out.
- PA1 grading is going on. Will post grades as soon as it’s done.
- TA info
  - Tom Sherwood: TBD
  - Chang Min Park: Tuesdays 1pm - 4pm
  - Sixu Piao: Wednesdays 2pm - 5pm
  - Chen Yuan: Thursdays 9am - 12 pm
  - Sampreeth Boddi Reddy: Thursdays 2 pm - 4 pm
  - Bekir Oguzhan Turkkan: Fridays 9am - 12pm
  - Sahil Gupta: TBD

Find the Mistake: Lamport Logical Time
Corrected Example: Lamport Logical Time

Vector Timestamps
- With Lamport clock
  - e "happened-before" f \(\iff\) timestamp(e) < timestamp(f), but
  - timestamp(e) < timestamp(f) \(\nRightarrow\) e "happened-before" f
- Idea?
  - Each process keeps a separate clock & pass them around.
  - Each process learns about what happened in all others.

Vector Logical Clocks
- Vector Logical time addresses the issue:
  - All processes use a vector of counters (logical clocks), \(i^{th}\) element is the clock value for process \(i\), initially all zero.
  - Each process \(i\) increments the \(i^{th}\) element of its vector upon an instruction or send event. Vector value is timestamp of the event.
  - A send(message) event carries its vector timestamp (counter vector).
  - For a receive(message) event, \(V_{receiver}[j] = \max(V_{receiver}[j], V_{message}[j])\), if \(j\) is not self.
  - \(V_{receiver}[j] + 1\), otherwise
- Key point
  - You update your own clock. For all other clocks, rely on what other processes tell you and get the most up-to-date values.

Comparing Vector Timestamps
- \(VT_1 = VT_2\)
  - \# \(VT_1[i] = VT_2[i]\), for all \(i = 1, \ldots, n\)
- \(VT_1 \ll VT_2\)
  - \# \(VT_1[i] < VT_2[i]\), for all \(i = 1, \ldots, n\)
- \(VT_1 < VT_2\)
  - \# \(VT_1 < VT_2\) & \(\exists j (1 \leq j \leq n \land VT_1[j] < VT_2[j])\)
- \(VT_1\) is concurrent with \(VT_2\)
  - \# (not \(VT_1 \ll VT_2\) AND not \(VT_2 \ll VT_1\))
The Use of Logical Clocks

- Is a design decision
- NTP error bound
  - Local: a few ms
  - Wide-area: 10's of ms
- If your system doesn’t care about this inaccuracy, then NTP should be fine.
- Logical clocks impose an arbitrary order over concurrent events anyway
  - Breaking ties: process IDs, etc.

Summary

- Relative order of events enough for practical purposes
  - Lamport’s logical clocks
  - Vector clocks
- Next: How to take a global snapshot

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta at UIUC.