CSE 486/586 Distributed Systems
Logical Time

Steve Ko
Computer Sciences and Engineering
University at Buffalo

CSE 486/586, Spring 2014

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
• Logical time
  – For ordering events, relative time should suffice.
  – Will continue today

Basics: State Machine
• State: a collection of values of variables
• Event: an occurrence of an action that changes the state, (i.e., instruction, send, and receive)
• As a program,
  – We can think of all possible execution paths.
• At runtime,
  – There’s only one path that the program takes.
• Equally applicable to
  – A single process
  – A distributed set of processes

Ordering Basics
• Why did we want to synchronize physical clocks?
• What we need: Ordering of events.
• Arises in many different contexts...

Abstract View
• Above is what we will deal with most of the time.
• Ordering question: what do we ultimately want?
  – Taking two events and determine which one happened before the other one.

What Ordering?
• Ideal?
  – Perfect physical clock synchronization
• Reliably?
  – Events in the same process
  – Send/receive events
Lamport Timestamps

• Lamport algorithm assigns logical timestamps:
  • 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
  • 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

Logical Clocks

CSE 486/586 Administrivia

• PA2 will be out very soon.
  – Sorry for my delay; waiting on new TAs first.
• Please pay attention to your coding style.

Corrected Example: Lamport Logical Time

Find the Mistake: Lamport Logical Time

One Issue

3 and 7 are logically concurrent events
Vector Timestamps

- With Lamport clock
  - `e "happened-before" f` if `timestamp(e) < timestamp (f)`, but
  - `timestamp(e) < timestamp (f)` `⇒` `e "happened-before" f`

- Idea?

![Diagram of Vector Timestamps]

Vector Logical Clocks

- Vector logical time addresses the issue:
  - All processes use a vector of counters (logical clocks), `P`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

![Diagram of Vector Logical Clocks]

Comparing Vector Timestamps

- `VT_1 = VT_2`, `iff` `VT_1[i] = VT_2[i]`, for all `i = 1, ..., n`
- `VT_1 <= VT_2`, `iff` `VT_1[i] <= VT_2[i]`, for all `i = 1, ..., n`
- `VT_1 < VT_2`, `iff` `VT_1 < VT_2 & \exists j (1 <= j <= n & VT_1[j] < VT_2[j])`
- `VT_1` is concurrent with `VT_2`, `iff` `(not VT_1 <= VT_2 AND not VT_2 <= VT_1)`

Find a Mistake: Vector Logical Time

![Diagram of Find a Mistake: Vector Logical Time]

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.