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?
- 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.
- 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

```
<table>
<thead>
<tr>
<th>Event</th>
<th>a</th>
<th>b</th>
<th>m1</th>
<th>c</th>
<th>d</th>
<th>m2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>1</td>
<td>2</td>
<td></td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
```

- But each process needs to know a time value

A Catch

- 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

CSE 486/586 Administrivia

- PA2A is out. Two points:
  - Multicast: Need to send each message to every instance including the one that sends the message. Just create 5 connections (5 sockets) and send a message 5 times through different connections.
  - ContentProvider: Don’t call it directly. Don’t share anything with the main activity. Consider it an almost separate app only accessible via ContentResolver.

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

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

Find the Mistake: Lamport Logical Time

```
<table>
<thead>
<tr>
<th>Event</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
```

Clock Value

Message
Vector Timestamps

- With Lamport clock
  - e “happened-before” f ⇔ timestamp(e) < timestamp(f), but
    timestamp(e) ≤ timestamp(f) ∨ 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\textsuperscript{th} element is the clock value for process i, initially all zero.
  - Each process increments the i\textsuperscript{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\textsuperscript{receiver}[j] = Max(V\textsuperscript{receiver}[j], V\textsuperscript{message}[j]), if j is not self.
    V\textsuperscript{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\textsubscript{i} = VT\textsubscript{j},
  - if VT\textsubscript{i}[i] = VT\textsubscript{j}[i], for all i = 1, …, n
- VT\textsubscript{i} <= VT\textsubscript{j},
  - if VT\textsubscript{i}[j] <= VT\textsubscript{j}[j], for all i = 1, …, n
- VT\textsubscript{i} < VT\textsubscript{j},
  - if VT\textsubscript{i}[j] <= VT\textsubscript{j}[j] ∧ (1 <= j <= n ∧ VT\textsubscript{i}[j] < VT\textsubscript{j}[j])
- VT\textsubscript{i} is concurrent with VT\textsubscript{j},
  - if (not VT\textsubscript{i} <= VT\textsubscript{j} AND not VT\textsubscript{j} <= VT\textsubscript{i})
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.