Last Time

- Clock skew does happen
- Cristian’s algorithm
  - One server
  - Server-side timestamp with one-way delay estimate
- NTP (Network Time Protocol)
  - Hierarchy of time servers
  - Estimates the offset between two clocks
  - Designed for the Internet
- Logical clocks stamp events, not times
Logical clocks

• We cannot sync multiple clocks perfectly.
• Thus, if we want to order events happening at different processes (remember the ticket reservation example?), we cannot rely on physical clocks.
• Then came logical time.
  – First proposed by Leslie Lamport in the 70s [1]
  – 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.

*Time, Clocks, and the Ordering of Events in a Distributed System* [1] is required reading.
Ordering Basics

• Why did we want to synchronize physical clocks?
• We actually need ordering of events.
• Arises in many different contexts…
Abstract View

• Think of a program as a collection of actions: instructions, send, and receive events.

• This 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?
  – Determine the ordering of (any?) two events.
What Ordering?

- What would be ideal?
  - Perfect physical clock synchronization

- Without this, what is reliable?
  - Events in the same process
  - Send/receive events
Lamport Timestamps

- **Goal:** take any two events, and determine their ordering.
- **It uses a single number to do so.**
Logical Clocks

• Lamport’s algorithm assigns logical timestamps:
  • All processes use a counter (clock) with initial value of zero
  • A process increments its counter when executing a send or an instruction.
  • The counter is assigned to the event as its timestamp.
  • A send event carries its timestamp in the message
  • On a receive event, the process counter is updated by max(local clock, message timestamp) + 1

• Defines a logical relation happened-before (→) over events:
  • On the same process: \( a \rightarrow b \), if \( time(a) < time(b) \)
  • If \( p_1 \) sends \( m \) to \( p_2 \): \( send(m) \rightarrow receive(m) \)
  • Transitive: If \( a \rightarrow b \) and \( b \rightarrow c \) then \( a \rightarrow c \)
  • Shows causality of events
Corrected: Lamport Logical Time

- **Clock Value**
- **timestamp**
- **Message**

Diagram showing the relationship between physical time and the events in Lamport Logical Time.
Lamport Logical Time Ambiguities

3 and 7 are logically concurrent events.
Vector Timestamps

- With Lamport clock
  - \( e \rightarrow f \Rightarrow \text{timestamp}(e) < \text{timestamp}(f) \), but
  - \( \text{timestamp}(e) < \text{timestamp}(f) \Rightarrow e \rightarrow f \)

- Idea?
  - Each process keeps a separate clock & passes all around.
  - Each process learns about what happened in all others.

![](diagram.png)
Vector Logical Clocks

• Vector Logical time:
  • 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 event carries its vector timestamp
  • For a receive event, $V_{receiver}[j] =$
    • $\text{Max}(V_{receiver}[j], V_{message}[j])$, 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.
Find a Mistake:
Vector Logical Time

Physical Time

p 1
0,0,0,0
1,0,0,0
(1,0,0,0)
2,0,0,0
1,1,0,0
(2,0,0,0)
3,0,2,2
2,0,2,0
(2,0,2,0)
4,0,2,2
2,0,2,2
2,0,2,1
(2,0,2,1)
2,0,2,3
(2,0,2,3)

p 2
0,0,0,0
1,1,0,0
(2,0,0,0)
1,2,0,0
(1,2,0,0)
3,0,2,2
2,0,2,0
(2,0,2,0)
4,0,2,2
2,0,2,2
2,0,2,1
(2,0,2,1)
2,0,2,3
(2,0,2,3)

p 3
0,0,0,0
2,0,2,0
2,2,3,0
4,2,4,2
4,2,5,3

p 4
0,0,0,0
2,0,2,2
(2,0,2,2)

n,m,p,q Vector logical clock
(vector timestamp) Message
Comparing Vector Timestamps

- $\text{VT}_1 = \text{VT}_2$, 
  - iff $\text{VT}_1[i] = \text{VT}_2[i]$, for all $i = 1, \ldots, n$

- $\text{VT}_1 \leq \text{VT}_2$, 
  - iff $\text{VT}_1[i] \leq \text{VT}_2[i]$, for all $i = 1, \ldots, n$

- $\text{VT}_1 < \text{VT}_2$, 
  - iff $\text{VT}_1 \leq \text{VT}_2 \& \exists j (1 \leq j \leq n \& \text{VT}_1[j] < \text{VT}_2[j])$

- $\text{VT}_1$ is concurrent with $\text{VT}_2$ 
  - iff (not $\text{VT}_1 \leq \text{VT}_2$ AND not $\text{VT}_2 \leq \text{VT}_1$)
The Use of Logical Clocks

• Is a design decision
• NTP error bound
  – Local: a few ms
  – Wide-area: 10s 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
References


Acknowledgements

- Slides by Steve Ko (used with permission) and lightly modified by Ethan Blanton.
- These slides contain material developed and copyrighted by Indranil Gupta at UIUC.