BRINGING ORDER TO THE GALAXY

THE EMPIRE

BRINGING ORDER TO THE GALAXY
SINCE: 1977
WAITLIST

Departmental policy: priority to CSE students

I have cleared some people. I will clear a few more.
Course homepage

http://web.eecs.umich.edu/~manosk/eecs591-w17.html
RESEARCH PROJECT

- Have a crazy idea? Talk to me about it.
- Work on my crazy idea. Ask me about it.
- Feeling adventurous? Volunteer for project X.
Synchrony vs Asynchrony

Synchronous systems
- Known bound on message delivery
- Known bound on processing speed
- Considered a strong assumption

Asynchronous systems
- **No bound** on message delivery
- **No bound** on processing speed
- Weak assumption = less vulnerable
- Asynchronous ≠ slow

This lecture: asynchronous + no process failures
Ordering events in a distributed system

What does it mean for an event to “happen before” another event?
What is a distributed system?

A collection of distinct processes that:

• are spatially separated
• communicate with one another by exchanging messages
• have non-negligible communication delay
• do not share fate
• have separate physical clocks

(Imperfect, unsynchronized)
Non-distributed system

- A single clock
- Each event has a timestamp
- Compare timestamps to order events

Distributed system

- Each process has its own clock
- Each clock runs at a different speed
- Cannot directly compare clocks

Leslie Lamport said:

"an absolute temporal ordering is not what you want in a distributed system anyway."
an absolute temporal ordering is not what you want in a distributed system anyway

Leslie Lamport

Because temporal ordering is not observable. You cannot read two separate clocks simultaneously!

Why not?

Very important point:
if a system is to meet a specification correctly, then that specification must be given in terms of events observable within the system
ORDERING EVENTS WITHOUT PHYSICAL CLOCKS

Modeling a process:

- A set of instantaneous events with an a priori total ordering
- Events can be local, sends, or receives.
Happened-before” relation, denoted: \[ \rightarrow \]

- If \( a \) and \( b \) are events on the same process and \( a \) comes before \( b \), then \( a \rightarrow b \)
Ordering events without physical clocks

“Happened-before” relation, denoted: $\rightarrow$

- If $a$ is the sending of a message by one process and $b$ is the receipt of the same message by another process, then $a \rightarrow b$
ORDERING EVENTS WITHOUT PHYSICAL CLOCKS

“Happened-before” relation, denoted: $\rightarrow$

- If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$
Ordering events without physical clocks

Putting it all together
Can arrows go backwards?
Can cycles be formed?

No, because the same event would happen at two different times
ORDERING EVENTS WITHOUT PHYSICAL CLOCKS

Are all events related by $\rightarrow$?
A partial order

The set of events \( q \) such that \( q \rightarrow p \) are the events that could have influenced \( p \) in some way.

\[ \{a, b, e, f, h\} \]
If two events could not have influenced each other, it doesn’t matter when they happened relatively to each other.

$h$ and $d$ are concurrent: $h \leftrightarrow d$, $d \leftrightarrow h$
Goal: generate a \textbf{total} order that is consistent with the happened-before partial order
Lamport clocks

Define a function $\text{LC}$ such that:

$$ p \rightarrow q \Rightarrow \text{LC}'(p) < \text{LC}'(q) $$

(the Clock condition)

Implement $\text{LC}$ by keeping a local $\text{LC}_i$ at each process $i$
Lamport clocks

Single process

1 6 37 1145
Lamport clocks

Across processes

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

<table>
<thead>
<tr>
<th></th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
```

\[ b \rightarrow h \Rightarrow LC(b) < LC(h) \]

\[ g \rightarrow h \Rightarrow LC(g) < LC'(h) \]
Putting in all together
Is this correct?
Generating a total order

- Order messages by LC
- Ties are broken by unique process ID
Lamport clocks implement the Clock condition

\[ p \rightarrow q \Rightarrow LC(p) < LC(q) \]

But is that all we need?
FIFO delivery

\[ send_i(m) \to send_i(m') \Rightarrow deliver_j(m) \to deliver_j(m') \]
When more processes are involved, causal delivery is needed:

$$send_i(m) \rightarrow send_j(m') \Rightarrow deliver_k(m) \rightarrow deliver_k(m')$$
Gap detection: Given two events $e$ and $e'$, where $LC(e) < LC(e')$, determine whether some other event $e''$ exists such that

$$LC(e) < LC(e'') < LC(e')$$
**Gap detection**: Given two events \( e \) and \( e' \), where \( LC(e) < LC(e') \), determine whether some other event \( e'' \) exists such that

\[
LC(e) < LC(e'') < LC(e')
\]

Lamport clocks don't provide gap detection!
**How to implement causal delivery?**

(in other words, when is it safe to deliver $m'$?)

a) Wait to receive a message with higher LC from each channel

b) Implement better clocks!
FROM CLOCKS TO STRONG CLOCKS

\[ p \rightarrow q \Rightarrow LC(p) < LC(q) \]

Clock condition

\[ p \rightarrow q \Leftrightarrow LC(p) < LC(q) \]

Strong clock condition
Causal histories

The set of events $q$ such that $q \rightarrow p$ are the events that could have influenced $p$ in some way.

$$\theta(g) = \{a, b, e, f, h\}g$$
IMPLEMENTING STRONG CLOCKS
(the hard way)

• Initialize $\theta := \emptyset$

• For send and local events $e$, $\theta(e) := \theta \cup \{e\}$

• For receive events $e = \text{recv}(m)$, $\theta(e) := \theta \cup \{e\} \cup \theta(m)$
IMPLEMENTING STRONG CLOCKS
(the hard way)

Strong clock condition: $p \rightarrow q \iff \theta(p) \subseteq \theta(q)$
IMPLEMENTING STRONG CLOCKS
(the hard way)

Strong clock condition: \( p \rightarrow q \iff \theta(p) \subset \theta(q) \)
Vector clocks

Each process keeps a vector of natural numbers $VC$, one for each process.

**Update rules**

If $e_i$ is a local or send event at process $i$:
$$VC(e_i)[i] := VC[i] + 1$$

If $e_i$ is a receive event of message $m$:
$$VC(e_i) := max\{VC, VC(m)\}$$
$$VC(e_i)[i] := VC[i] + 1$$
**Vector clocks**

\[ VC(e_i)[i] = \text{number of events executed by process } i \text{ (including } e_i) \]

\[ VC(e_i)[j] = \text{number of events executed by process } j \text{ that causally precede } e_i \]
Comparing vector clocks

Equality

\[ V = V' \equiv \forall k : 1 \leq k \leq n : V[k] = V'[k] \]

(i.e. all elements are the same)

Inequality

\[ V < V' \equiv (V \neq V') \land (\forall k : 1 \leq k \leq n : V[k] \leq V'[k]) \]

Examples: \([2,0,0] < [2,0,1] < [3,0,1] < [4,1,1]\)

Strong clock condition: \( p \rightarrow q \iff VC(p) < VC(q) \)
Comparing vector clocks

Strong clock condition: \( p \rightarrow q \iff VC(p) < VC(q) \)
CAUSAL DELIVERY

A “monitor” process wants to record all messages (e.g. deadlock detection, system snapshot, etc)

- Processes send copies of their messages to the monitor
- Only increment the local component of VC for send events
CAUSAL DELIVERY RULES

Monitor keeps an array $D$, where $D[i]$ is the number of messages delivered from process $i$

Monitor delivers message $m$ from process $j$ when:

$$D[j] = VC(m)[j] - 1$$

$$D[k] \geq VC(m)[k], \forall k \neq j$$
Causal Delivery

\[
D[j] = VC(m)[j] - 1
\]

<table>
<thead>
<tr>
<th></th>
<th>(0, 0)</th>
<th>(1, 0)</th>
<th>(1, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

\[
D[k] \geq VC(m)[k], \forall k \neq j
\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>✔️</th>
<th>✔️</th>
<th>✔️</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>❌</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>