Distributed Systems

06. Logical Clocks

Paul Krzyzanowski
Rutgers University
Fall 2015
Logical clocks

Assign sequence numbers to messages
– All cooperating processes can agree on order of events
– vs. *physical clocks*: report time of day

Assume no central time source
– Each system maintains its own local clock
– No total ordering of events
  • No concept of *happened-when*
Happened-before

Lamport’s “happened-before” notation

\[ a \rightarrow b \] event \( a \) happened before event \( b \)

e.g.: \( a \): message being sent, \( b \): message receipt

Transitive:

if \( a \rightarrow b \) and \( b \rightarrow c \) then \( a \rightarrow c \)
Assign a “clock” value to each event
  – if $a \rightarrow b$ then $\text{clock}(a) < \text{clock}(b)$
  – since time cannot run backwards

If $a$ and $b$ occur on different processes that do not exchange messages, then neither $a \rightarrow b$ nor $b \rightarrow a$ are true
  – These events are **concurrent**
  – Otherwise, they are **causal**
Event counting example

- Three systems: $P_0$, $P_1$, $P_2$
- Events $a$, $b$, $c$, …
- Local event counter on each system
- Systems occasionally communicate
Event counting example

P_1: a, b, c, d, e, f
   1, 2, 3, 4, 5, 6

P_2: j, g
    1, 2, 3

P_3: k
    1, 2
Event counting example

Bad ordering:

- $e \rightarrow h$ but $5 \geq 2$
- $f \rightarrow k$ but $6 \geq 2$
Lamport’s algorithm

• Each message carries a timestamp of the sender’s clock

• When a message arrives:
  – if receiver’s clock < message_timestamp
    set system clock to (message_timestamp + 1)
  – else do nothing

• Clock must be advanced between any two events in the same process
Lamport’s algorithm

Algorithm allows us to maintain time ordering among related events

– Partial ordering
Applying Lamport’s algorithm

We have good ordering where we used to have bad ordering:

- e → h and 5 < 6
- f → k and 6 < 7
• Algorithm needs monotonically increasing software counter

• Incremented at least when events that need to be timestamped occur

• Each event has a Lamport timestamp attached to it

• For any two events, where \( a \rightarrow b \):
  \[
  L(a) < L(b)
  \]
Problem: Identical timestamps

Local events sequenced:

\[ a \rightarrow b, \ b \rightarrow c, \ldots \]

Lamport imposes a send→receive relationship:

\[ i \rightarrow c, \ f \rightarrow d, \ d \rightarrow g, \ldots \]

Concurrent events (e.g., \( b \& g; \ i \& k \)) may have the same timestamp … or not
Unique timestamps (total ordering)

We can force each timestamp to be unique

– Define global logical timestamp \((T_i, i)\)
  
  • \(T_i\) represents local Lamport timestamp
  
  • \(i\) represents process number (globally unique)
    
    – e.g., (host address, process ID)

– Compare timestamps:

\[(T_i, i) < (T_j, j)\]

if and only if

\[T_i < T_j \text{ or }\]

\[T_i = T_j \text{ and } i < j\]

Does not necessarily relate to actual event ordering
Unique (totally ordered) timestamps
Problem: Detecting causal relations

If $L(e) < L(e')$

– We cannot conclude that $e \rightarrow e'$

By looking at Lamport timestamps

– We cannot conclude which events are causally related

Solution: use a vector clock
Vector clocks

Rules:

1. Vector initialized to 0 at each process
   \[ V_i[j] = 0 \] for \( i, j = 1, \ldots, N \)

2. Process increments its element of the vector in local vector before timestamping event:
   \[ V_i[i] = V_i[i] + 1 \]

3. Message is sent from process \( P_i \) with \( V_i \) attached to it

4. When \( P_j \) receives message, compares vectors element by element and sets local vector to higher of two values
   \[ V_j[i] = \max(V_i[i], V_j[i]) \] for \( i = 1, \ldots, N \)

For example,
received: \([0, 5, 12, 1]\)
have: \([2, 8, 10, 1]\)
new timestamp: \([2, 8, 12, 1]\)
Comparing vector timestamps

Define

\[ V = V' \iff V[i] = V'[i] \text{ for } i = 1 \ldots N \]
\[ V \leq V' \iff V[i] \leq V'[i] \text{ for } i = 1 \ldots N \]

For any two events \( e, e' \)

- if \( e \rightarrow e' \) then \( V(e) < V(e') \)
- \( \ldots \) just like Lamport's algorithm
- if \( V(e) < V(e') \) then \( e \rightarrow e' \)

Two events are **concurrent** if neither

\[ V(e) \leq V(e') \text{ nor } V(e') \leq V(e) \]
Vector timestamps

(0,0,0)

P_1

(0,0,0)

P_2

(0,0,0)

P_3

a b
c d
e f
Vector timestamps

Event timestamp

a (1,0,0)
Vector timestamps

<table>
<thead>
<tr>
<th>Event</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>(1,0,0)</td>
</tr>
<tr>
<td>b</td>
<td>(2,0,0)</td>
</tr>
</tbody>
</table>

(0,0,0) P1

(1,0,0) a, b

(2,0,0) c, d

(0,0,0) e, f

P2

P3
Vector timestamps

Event | timestamp
--- | ---
a | (1,0,0)
b | (2,0,0)
c | (2,1,0)
Vector timestamps

<table>
<thead>
<tr>
<th>Event</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>(1,0,0)</td>
</tr>
<tr>
<td>b</td>
<td>(2,0,0)</td>
</tr>
<tr>
<td>c</td>
<td>(2,1,0)</td>
</tr>
<tr>
<td>d</td>
<td>(2,2,0)</td>
</tr>
<tr>
<td>e</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
</tr>
</tbody>
</table>
Vector timestamps

Event | timestamp
---|---
a | (1,0,0)
b | (2,0,0)
c | (2,1,0)
d | (2,2,0)
e | (0,0,1)
Vector timestamps

Event | timestamp
--- | ---
a | (1,0,0)
b | (2,0,0)
c | (2,1,0)
d | (2,2,0)
e | (0,0,1)
f | (2,2,2)
Vector timestamps

Event | timestamp
---|---
a | (1,0,0)
b | (2,0,0)
c | (2,1,0)
d | (2,2,0)
e | (0,0,1)
f | (2,2,2)

concurrent events
Vector timestamps

- Event timestamp
  - a: (1,0,0)
  - b: (2,0,0)
  - c: (2,1,0)
  - d: (2,2,0)
  - e: (0,0,1)
  - f: (2,2,2)

Concurrent events:
Vector timestamps

<table>
<thead>
<tr>
<th>Event</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>(1,0,0)</td>
</tr>
<tr>
<td>b</td>
<td>(2,0,0)</td>
</tr>
<tr>
<td>c</td>
<td>(2,1,0)</td>
</tr>
<tr>
<td>d</td>
<td>(2,2,0)</td>
</tr>
<tr>
<td>e</td>
<td>(0,0,1)</td>
</tr>
<tr>
<td>f</td>
<td>(2,2,2)</td>
</tr>
</tbody>
</table>

concurrent events
Vector timestamps

<table>
<thead>
<tr>
<th>Event</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>(1,0,0)</td>
</tr>
<tr>
<td>b</td>
<td>(2,0,0)</td>
</tr>
<tr>
<td>c</td>
<td>(2,1,0)</td>
</tr>
<tr>
<td>d</td>
<td>(2,2,0)</td>
</tr>
<tr>
<td>e</td>
<td>(0,0,1)</td>
</tr>
<tr>
<td>f</td>
<td>(2,2,2)</td>
</tr>
</tbody>
</table>

concurrent events
Generalizing Vector Timestamps

- A “vector” can be an list of tuples:
  - For processes $P_1, P_2, P_3, \ldots$
  - Each process has a globally unique Process ID, $P_i$ (e.g., $\text{MAC\_address}:\text{PID}$)
  - Each process maintains its own timestamp: $T_{P1}, T_{P2}, \ldots$
  - Vector: $\{<P_1, T_{P1}>, <P_2, T_{P2}>, <P_3, T_{P3}>, \ldots\}$

- Any one process may have only partial knowledge of others
  - New timestamp for a received message:
    - Compare all matching sets of process IDs: set to highest of values
    - Any non-matched $<P, T>$ sets get added to the timestamp
  - For a happened-before relation:
    - At least one set of process IDs must be common to both timestamps
    - Match all corresponding $<P, T>$ sets: $A:<P_i, T_a>$, $B:<P_i, T_b>$
    - If $T_a \leq T_b$ for all common processes $P$, then $A \rightarrow B$
Summary: Logical Clocks & Partial Ordering

• Causality
  – If $a \rightarrow b$ then event $a$ can affect event $b$

• Concurrency
  – If neither $a \rightarrow b$ nor $b \rightarrow a$ then one event cannot affect the other

• Partial Ordering
  – Causal events are sequenced

• Total Ordering
  – All events are sequenced
The End