Lecture 8: Logical Time

Why is time important?
- In real life,
  - Just because 😊
- In distributed systems,
  - Fairness in mutual exclusion
  - Avoiding deadlocks
  - Tracking dependence in case of failure
  - Consistency of replicated data
  - Knowledge about progress for garbage collection
  - Concurrency measurements

Example
- Consider a bank system running on a distributed system.
  - The data is replicated on multiple servers.
- Some account initially has $200.
- A deposit of $1000 is made to the account, and server A handles the transaction.
  - The deposit is time-stamped with A's local time of 9:22pm.
  - All data is synchronized at all servers.
- A withdrawal of $500 is then made, and server B handles the transaction.
  - The withdrawal is time-stamped with B's local time of 9:15pm.
- Server C queries the transaction logs of both servers.
  - What will happen?

We need to synchronize time
Here are some clock synchronization algorithm(s)
Christian’s algorithm

- Christian [1989] suggested using a time server that helps different machines (processes) adjust their clocks.

Problems?

- Delays in processing the messages and buffer delays and other network stuff.
- P can’t set the clock to t blindly

Problems?

- Delays in the actual message transfer from the network
- P can’t set the clock to t blindly

Problems?

- P sets clock to \( t + \frac{(RTT + \text{min}2 - \text{min}1)}{2} \)
All problems fixed?

- No!
  - We still have a single point of failure.
  - We still might have errors in computing the clock value \( t \).

- What can we do?
  - Distribute the responsibility of the time server.
  - Berkeley algorithm
  - Design an architecture for the time protocol, which would statistically minimize clock errors.
  - Network Time Protocol (NTP)

But do we really need to synchronize clocks?

We still have a chance of error, when working with time

Network Time Protocol (NTP)

Back to our Example

- Consider a bank system running on a distributed system.
  - The data is replicated on multiple servers.
- Some account initially has $200.
- A deposit of $1000 is made to the account.
  - The deposit is time-stamped with A's local time of 9:22pm.
  - All data is synchronized at all servers.
- A withdrawal of $500 is then made, and server B handles the transaction.
  - The withdrawal is time-stamped with B's local time of 9:15pm.
- Server C queries the transaction logs of both servers.
  - What will happen?

Logical Time

It's all about the sequence of events!
Ordering of events

- Definitions:
  - Multiple processes running on independent machines.
  - Events occurring at a process could be a step, send, or receive.

- Ordering assumptions:
  - Events occurring at the same process have a natural ordering.
    - The process can keep track of that using local clock.
  - When a message is sent,
    - The send event then the receive event, then the message is delivered to the other process.

Potential causal ordering

- Or what we call the happened-before relationship (→).

  - HB1: within a single process if e happened before e',
    - e → e'
  - HB2: for any message m,
    - send(m) → receive(m)
  - HB3: if e → e' and e' → e'',
    - e → e''
  - HB4: if e (→) e' and e' (→) e,
    - e || e'
    - Which means that they are concurrent

Example

Logical clocks

- To design a logical clock algorithm, you need to define:
  - Data structures used locally by a process to define the logical time
  - Protocol used among processes to update these data structures

- The protocol must specify:
  - R1: How a process adjusts its logical time when a local event occurs
    - Times are piggybacked with messages sent.
  - R2: How a process adjusts its logical time when a message is received
Lamport's algorithm (Scalar time)

- Data structures:
  - Each process (i) keeps track of a single non-negative integer $C_i$

- Protocol:
  - R1: before executing an event locally $C_i = C_i + d$
  - R2: when message is received with a time value of $C_{msg}$
    - $C_i = \max(C_i, C_{msg})$
    - Execute R1
    - Deliver the message

Example

- Group exercise time!

Lamport Timestamps

<table>
<thead>
<tr>
<th>Instruction or step</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>0</td>
</tr>
</tbody>
</table>

Initial counters (clocks)

Lamport Timestamps

<table>
<thead>
<tr>
<th>Instruction or step</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>ts = 1</td>
</tr>
<tr>
<td>P2</td>
<td>ts = 1</td>
</tr>
<tr>
<td>P3</td>
<td>ts = 1</td>
</tr>
</tbody>
</table>

Message send
Lamport Timestamps

Message carries

\[ ts = \max(0, 1) + 1 \]
\[ = 2 \]

Lamport Timestamps

Message carries

\[ ts = \max(2, 2) + 1 \]
\[ = 3 \]

Lamport Timestamps

\[ ts = \max(3, 4) + 1 \]
\[ = 5 \]

Lamport Timestamps

\[ ts = \max(2, 2) + 1 \]
\[ = 3 \]
Obeying Causality

Properties of scalar time

- Consistency
- Total ordering
- Event counting
- But...
  - No strong consistency

Vector time

- Data structures:
  - Each process \( i \) keeps track of a vector of \( n \) non-negative integers \( vti[1 \ldots n] \)

- Protocol:
  - R1: before executing an event locally \( vti[i] = vti[i] + d \)
  - R2: when message is received with a time value of \( vt \)
    - For all values in the vector \( vti[k] = \max(vti[k], vt[k]) \)
    - Execute R1
    - Deliver the message
Example

- Group exercise!

Vector Timestamps

Initial counters (clocks)

Message(0,0,1)