CS 555: DISTRIBUTED SYSTEMS
[LOGICAL CLOCKS]

Shrideep Pallickara
Computer Science
Colorado State University

September 3, 2019

Frequently asked questions from the previous class survey

- NTP
  - Why use the closest server to synchronize?
Topics covered in this lecture

- Clock synchronization
  - Berkley Algorithm
  - Cristian’s Algorithm
  - Synchronization in wireless settings
- Logical Clocks
  - Lamport’s clocks

THE BERKLEY ALGORITHM
The Berkley Algorithm for time synchronization

- In NTP, the time server is passive
  - Server merely responds to queries from the server
- The time server is **active** in the Berkley Algorithm
  - Polls every machine periodically for their time
- Collects responses and computes the **average**
  - Tell machines to **slowdown** or **speedup** their clocks

Berkley Algorithm: An example

![Network diagram showing time synchronization between Machine 1, Machine 2, and Machine 3 with a network daemon and their time readings.](image)
Berkley Algorithm: An example

September 3, 2019
Professor: SHRIDEEP PALLICKARA

CS555: Distributed Systems [Fall 2019]
Dept. Of Computer Science, Colorado State University

L3.7

Berkley Algorithm: An example

September 3, 2019
Professor: SHRIDEEP PALLICKARA

CS555: Distributed Systems [Fall 2019]
Dept. Of Computer Science, Colorado State University

L3.8
Time in the Berkley Algorithm

- Not necessary for the computed time to agree with real time
- It is sufficient that all machines agree on some time
- No harm done if there is no contact with external machines

Cristian’s Algorithm
Cristian’s algorithm

- Uses a time server to synchronize services
  - The server receives signals from a UTC source
- When a request is received, the server process supplies the time

Clock synchronization using a time server

![Diagram showing clock synchronization](image)
Some notations

- A process \( p \) requests time in a message \( m_r \).
- Process \( p \) receives time value in a message \( m_t \).
- Time \( t \) is inserted in \( m_t \) at the last possible moment by Server \( S \).
- Process \( p \) records total round-trip time \( T_{\text{round}} \).
  - To send request \( m_r \) and to receive the reply \( m_t \).

Measurement of the round trip

- This is reasonably accurate if clock drift rate is small.
- Round-trip time in a LAN is 1-10 ms.
  - If the clock drift rate is \( 10^{-6} \) seconds/second?
    - That clock would vary at most by \( 10^{-8} \) seconds \((10^{-6} \times 10^{-2})\) or \( 10^{-5} \) milliseconds.
Estimating what the time at $p$ should be [1/2]

- A simple estimate of this is $t + T_{round}/2$
  - Elapsed time is split equally before and after $S$ placed $t$ in $m_t$

- If the value of the minimum transmission time $min$ is known (or conservatively estimated)?
  - We can determine accuracy of the result

Estimating what the time at $p$ should be [2/2]

- The earliest point at which $S$ could have placed the time in $m_t$?
  - $min$ after $p$ dispatched $m_r$

- The latest point at which $S$ could have placed the time in $m_t$?
  - $min$ before $m_t$ arrived at $p$

- Time by $S$'s clock when $m_t$ the arrives at $p$
  - Range: $[t + min, t + T_{round} - min]$
  - Width of the range is $T_{round} - 2min$
  - Accuracy is: $\pm(T_{round}/2 - min)$
An interesting example

- An interesting case involves Google's TrueTime API
  - Explicitly reports the confidence interval on the local clock
  - When you ask it for the current time, you get back two values:
    - [earliest, latest]
    - The earliest possible and the latest possible timestamp.

Theoretical limit for coping with multiple faulty clocks?

- Dolev et al in 1986
- If $f$ is the number of faulty clocks out of $N$
  - Then we must have $N > 3f$ if other, correct, clocks are still to be able to achieve agreement
CLOCK SYNCHRONIZATION IN WIRELESS NETWORKS

Sensor network settings

- Nodes are resource constrained
- Multihop routing is expensive
- Optimize algorithms for energy consumption
Reference broadcast synchronization (RBS)

- Objective is not to provide all nodes with UTC time
- Goal is to internally synchronize all clocks
- RBS is not a two-way protocol
  - Only the receivers synchronize
- In RBS, a sender broadcasts a reference message that will allow its receivers to adjust their clocks

Sensor network communications

- Time to propagate a signal to other nodes
  - Roughly constant
  - Assume that there is no multi-hop routing
- Wireless networks based on contention protocol
- Propagation time is measured from the moment message leaves NIC of sender
  - Two elements no longer play role in estimating delay
    - Time to construct the message
    - Time to access the network
Critical paths in determining network delays:
Regular Settings

Critical paths in determining network delays: RBS
When a node broadcasts a reference message $m$

- Each node $p$, records $T_{p,m}$
  - The time it received $m$
  - Based on $p$’s local clock

- Two nodes, $p$ and $q$ can compare delivery times to compute mutual, relative offset

$$\text{Offset}[p,q] = \frac{\sum_{k=1}^{M} T_{p,k} - T_{q,k}}{M}$$

$M$ is the total number of reference messages

---

**Logical Clocks**

September 3, 2019

CS555: Distributed Systems (Fall 2019)
Dept. Of Computer Science, Colorado State University
Physical time in a distributed system is problematic

- This is not because of the effects of special relativity, which are negligible or non-existent for normal computers
  - Unless you count computers travelling in spaceships

- It is because of the **inability to accurately timestamp** events at different nodes
  - We need this to order any pairs of events

If two processes do not interact with each other?

- Their clocks **need not** be synchronized

- Lack of synchronization is not observable
  - Does not cause problems
Logical clocks

- Within a single process, events are ordered uniquely by times shown on local clock
- But we cannot synchronize clocks perfectly across a distributed system [Lamport 1978]
  - We cannot use physical time to find out the order of an arbitrary pair of events in a distributed system

We can use a scheme that is similar to physical causality to order events

1. If two events occurred at the same process $p_i$ ($i=1, 2, ..., N$)?
   - Then they occurred in the order in which $p_i$ observes them
     - This is the order $\rightarrow_i$

2. When a message is sent between processes?
   - The event of sending the message occurred before the event of receiving the message
The $\rightarrow$ relation

- Lamport called the **partial ordering** obtained by generalizing the previous 2 relationships
  - The *happened-before* or *happens-before* relation
- Sometimes also known as the relation of *causal ordering* or *potential causal ordering*

Lamport’s logical clocks

- The **happens-before** relation $\Rightarrow$
  - $a$ and $b$ are events in the process; and $a$ occurs before $b$
    - Then $a \Rightarrow b$ is true
  - $a$ is event of message sent by one process;
    $b$ is event of message being received in another process
    - Then $a \Rightarrow b$ is true
Some more things about the happens-before relation

- If \( a \rightarrow b \) and \( b \rightarrow c \); then \( a \rightarrow c \)
  - Transitive

- If events \( x \) and \( y \) occur in processes that do not exchange messages, then ...
  - \( x \rightarrow y \) is not true
  - But, neither is \( y \rightarrow x \)
  - These events are said to be concurrent

Events occurring at three processes

- \( a \rightarrow b \) and \( c \rightarrow d \)
  - These occur within the same process
- \( b \rightarrow c \) and \( d \rightarrow f \)
  - Events that correspond to sending and receiving messages
- We can use transitivity to say \( a \rightarrow f \)
- No relationship between \( a \) and \( e \); these are concurrent \( a \parallel e \)
If the $\rightarrow$ relation holds between two processes

- The first event might or might-not have caused the second
  - The $\rightarrow$ relation only captures potential causality
    - i.e. two events can be related by $\rightarrow$ without a real connection between them

- EXAMPLE 1: If the server receives a request and sends a response?
  - Then reply is caused by the request

- EXAMPLE 2: A process might receive a request and subsequently issue another message
  - But this could be one that it issues every 5 minutes anyway

A simple example of Lamport timestamps

[Diagram showing timelines for events X, Y, Z with timestamps 1, 2, 3, 4, 5]
An example of Lamport's algorithm:

Each message carries the sending time according to the sender's clock.

Each clock runs at a constant (but different rate)
Implementing Lamport’s clocks

1. Before executing an event; $P_i$ executes
   \[ C_i = C_i + 1 \]

2. When $P_i$ sends a message $m$ to $P_j$; it sets $m$’s timestamp $ts(m)$ to $C_i$ in previous step

3. Upon receipt of message $m$, $P_j$ adjusts its own local counter
   \[ C_j = \max \{C_j, ts(m)\} \]
   do step (1) and deliver message

The positioning of Lamport’s clocks in distributed systems

Application layer
- Application sends message
- Adjust local clock and timestamp message
- Message is delivered to application

Middleware layer
- Adjust local clock
- Message is received

Network layer
- Middleware sends message
- Message is received
An application of Lamport’s clock:
User has $1000 in bank account initially

- Add $100 to account
- Update with 1% interest

San Francisco  
New York  

Add $100 … Total: $1100  
Give 1% interest on total = $11  
**Balance:** $1111

Give 1% interest … Total: $1010  
Add $100  
**Balance:** $1110

There is a difference when the orders are reversed

- Our objective for now is consistency
- Both copies must be exactly the same
Use Lamport’s clock to order messages

- Process puts received messages into local queue
  - Ordered according to the message’s timestamp
- Message can be delivered only if it is **acknowledged** by all the other processes
- If a message is at the head of the queue, and acknowledged by all processes
  - It is delivered and processed

Lamport’s Clocks order events based on the happened-before relationship

- If \( a \) happened before \( b \), then \( C(a) < C(b) \)
- But nothing can be said about two events \( a \) and \( b \) by merely comparing their values
- \( C(a) < C(b) \)?
  - Does not mean \( a \) happened before \( b \)

September 3, 2019
Professor: SHRIDEEP PALICKARA

CS555: Distributed Systems [Fall 2019]
Dept. Of Computer Science, Colorado State University
Let’s look a little closer

- $T_{\text{snd}}(m_i)$: Time $m_i$ was sent
- $T_{\text{rcv}}(m_i)$: Time $m_i$ was received
- $T_{\text{snd}}(m_i) < T_{\text{rcv}}(m_i)$
- BUT
  - $T_{\text{snd}}(m_i) < T_{\text{rcv}}(m_j)$?
  - NO

Concurrent message transmissions

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>6</th>
<th>12</th>
<th>18</th>
<th>24</th>
<th>30</th>
<th>36</th>
<th>42</th>
<th>48</th>
<th>70</th>
<th>76</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>61</td>
<td>69</td>
<td>77</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>m2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sending $m_3$ MAY HAVE depended on $m_1$

$T_{\text{rcv}}(m_1) < T_{\text{snd}}(m_2)$

But sending of $m_2$ has nothing to do with receipt of $m_1$

Lamport clocks do not capture causality
The contents of this slide-set are based on the following references

- [Matrix Clocks](http://en.wikipedia.org/wiki/Matrix_clocks)