

TCCS 558: APPLIED DISTRIBUTED COMPUTING

Coordination

Wes J. Lloyd
 Institute of Technology
 University of Washington - Tacoma

OBJECTIVES

- Assignment #2 Questions
- Feedback from 11/16
- Ch. 6 - Coordination
 - Clock synchronization
 - Logical clocks, Lamport clocks
 - Vector clocks
 - Distributed mutual exclusion

November 21, 2017
TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma
L15.2

FEEDBACK FROM 11/16

- When using "Gossip" message flooding, is there a possibility that a node is sent a message multiple times?
 - YES
- Is this redundant and possibly inefficient
 - YES
- How does NTP work in an **ad-hoc system**? (unstructured peer-to-peer?)
 We might have some nodes to be synced with atomic clocks (or lower levels), but how can we make sure ALL nodes in the system have access to the "synchronized" nodes
- NTP is UDP (time changes too quick to resend failed msgs)
- NTP typically operates in client/server mode (msgs sent to specific IPs)
- Other modes include broadcast and multicast

November 21, 2017
TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma
L15.3

FEEDBACK - 2

- What is the purpose of random graphs?
 - Seem circular?
 - Higher edge probability means more edges per node
 - More edges per node means higher probability
 - Does the number of rounds required for anti-entropy depend on edge probability?
 - YES, the probability graph depicts differences between push, pull, push/pull
- How is gossiping different from flooding?
 - Very similar, gossiping is a way of "thinking about" message interaction and developing the algorithm
- What if you want to reintroduce an item that was previously removed that has a death certificate?
 - The death certificate would need to be deleted...

November 21, 2017
TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma
L15.4

FEEDBACK - 3

- Why are the clocks not synchronized on campus?

November 21, 2017
TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma
L15.5

CHAPTER 6 - COORDINATION

- 6.1 Clock Synchronization
 - Physical clocks
 - Clock synchronization algorithms
- 6.2 Logical clocks
 - Lamport clocks
 - Vector clocks
- 6.3 Mutual exclusion
- 6.4 Election algorithms
- 6.6 Distributed event matching (*light*)
- 6.7 Gossip-based coordination (*light*)

November 21, 2017
TCCS558: Applied Distributed Computing [Fall 2017]
 Institute of Technology, University of Washington - Tacoma
L15.6



CH. 6.1: CLOCK SYNCHRONIZATION

L15.7

COMPUTING: CLOCK CHALLENGES

- How do we synchronize computer clocks with real-world clocks?
- How do we synchronize computer clocks with each other?

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.8

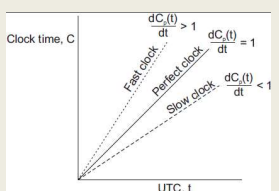
CLOCK SYNCHRONIZATION

- **UTC services:** use radio and satellite signals to provide time accuracy to 50ns
- **Time servers:** Server computers with UTC receivers that provide accurate time
- **Precision (π):** how close together a set of clocks may be
- **Accuracy:** how correct to actual time clocks may be
- **Internal synchronization:** Sync local computer clocks
- **External synchronization:** Sync to UTC clocks
- **Clock drift:** clocks on different machines gradually become out of sync due to crystal imperfections, temperature differences, etc.
- **Clock drift rate:** typical is 31.5s per year
- **Maximum clock drift rate (ρ):** clock specifications include one

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.9

CLOCK SYNCHRONIZATION - 2

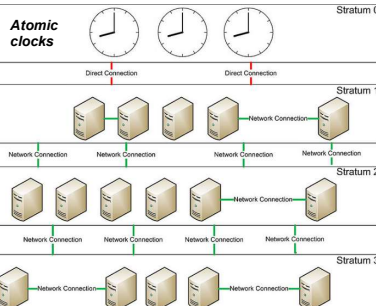
- If two clocks drift from UTC in opposite directions, after time Δt after synchronization, they may be 2ρ apart.
- Clocks must be resynchronized every $\pi/2\rho$ seconds
- **Network time protocol**
- Provide coordination of time for servers
- Leverage distributed network of time servers



November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.10

NETWORK TIME PROTOCOL

- Servers organized into stratum
- Stratum-1 servers have UTC receivers and are sync'd with atomic clocks
- Servers connect with closest NTP server for time synchronization
- Servers assume role as NTP server at stratum+1

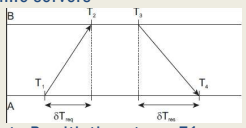


November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.11

NTP - 2

- Must estimate network delays when synchronizing with remote UTC receiver clocks / time servers

Time server B



Client A

1. A sends message to B, with timestamp T_1
2. B records time of receipt T_2 (from local clock)
3. B returns response with send time T_3 , and receipt time T_2
4. A records arrival of T_4

- Assuming propagation delay of $A \rightarrow B \rightarrow A$ is the same
- Estimate propagation delay: $\theta = T_3 + \frac{(T_2 - T_1) + (T_4 - T_3)}{2} - T_4 = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$
- Add delay to time

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.12

NTP - 3

- Cannot set clocks backwards (recall "make" file example)
- Instead, temporarily slow the progress of time to allow fast clock to align with actual time
- Change rate of clock interrupt routine
- Slow progress of time until synchronized
- NTP accuracy is within 1-50ms
- In Ubuntu Linux, to quickly synchronize time:


```
$apt install ntp ntpdate
```
- Specify local timeservers in /etc/ntp.conf


```
server time.u.washington.edu iburst
server bigben.cac.washington.edu iburst
```
- Shutdown service (sudo service ntp stop)
- Run ntpdate: (sudo ntpdate time.u.washington.edu)
- Startup service (sudo service ntp start)

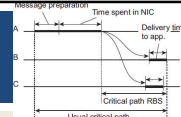
November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.13

BERKELEY ALGORITHM

- Berkeley time daemon server actively polls network to determine average time across servers
- Suitable when no machine has a UTC receiver
- Time daemon instructs servers how much to adjust clocks to achieve precision
- Accuracy can not be guaranteed
- Berkeley is an internal clock synchronization algorithm

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.14

CLOCK SYNCHRONIZATION IN WIRELESS NETWORKS



- Sensor networks bring unique challenges for clock synchronization
 - Address resource constraints:** limited power, multihop routing slow
- Reference broadcast synchronization (RBS)**
 - Provides precision of time, not accuracy as in Berkeley
 - No UTC clock available
 - RBS sender broadcasts a reference message to allow receivers to adjust clocks
 - No multi-hop routing
 - Time to propagate a signal to nodes is roughly constant
 - Message propagation time does not consider time spent waiting in NIC for message to send
 - Wireless network resource contention may force wait before message even **can** be sent

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.15

REFERENCE BROADCAST SYNCHRONIZATION (RBS)

- Node broadcasts reference message m
- Each node p records time $T_{p,m}$ when m is received
- $T_{p,m}$ is read from node p's clock
- Two nodes p and q can exchange delivery times to estimate mutual relative offset
- Then calculate relative average offset for the network:

$$Offset[p, q] = \frac{\sum_{k=1}^M (T_{p,k} - T_{q,k})}{M}$$
- Where M is the total number of reference messages sent
- Nodes can simply store offsets instead of frequently synchronizing clocks to save energy

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.16

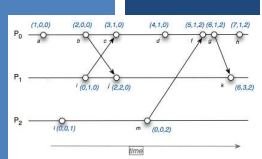
REFERENCE BROADCAST SYNCHRONIZATION (RBS) - 2

- Cloud skew: over time clocks drift apart
- Averages become less precise
- Elson et al. propose using standard linear regression to predict offsets, rather than calculating them
- IDEA: Use node's history of message times in a simple linear regression to continuously refine a formula with coefficients to predict time offsets:

$$Offset[p, q](t) = at + \beta$$

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.17

CH. 6.2: LOGICAL CLOCKS



November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.18

LOGICAL CLOCKS

- In distributed systems, synchronizing to actual time may not be required...
- It may be sufficient for every node to simply agree on a current time (e.g. logical)
- Logical clocks** provide a mechanism for capturing chronological and **causal** relationships in a distributed system
- Think **counters** . . .
- Leslie Lamport [1978] seminal paper showed that absolute clock synchronization often is not required
- Processes simply need to agree on the order in which events occur

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.19

LOGICAL CLOCKS - 2

- Happens-before relation**
- $A \rightarrow B$: **Event A**, happens before **event B**...
- All processes must agree that **event A** occurs first
- Then afterward, **event B**
- Actual time not important. . .
- If **event A** is the event of proc P1 sending a msg to a proc P2, and **event B** is the event of proc P2 receiving the msg, then $A \rightarrow B$ is also true. . .
- The assumption here is that message delivery takes time
- Happens before is a transitive relation:
- $A \rightarrow B, B \rightarrow C$, therefore $A \rightarrow C$

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.20

LOGICAL CLOCKS - 3

- If two events, say event X and event Y do not exchange messages, not even via third parties, then $X \rightarrow Y$ and $Y \rightarrow X$ **can not be determined**
- Within the system, these events appear **concurrent**
- Concurrent**: nothing can be said about when the events happened, or which event occurred first
- Clock time, C, must always go forward (increasing), never backward (decreasing)
- Corrections to time can be made by adding a positive value, but never by subtracting one

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.21

LOGICAL CLOCKS - 4

- Three processes each with local clocks
- Lamport's algorithm** corrects their values

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.22

LOGICAL CLOCKS

- Events:**

P ₁	P ₂	P ₃	P ₁	P ₂	P ₃
0	0	0	0	0	0
6	8	10	6	8	10
12	16	20	12	16	20
18	24	30	18	24	30
24	32	40	24	32	40
30	40	50	30	40	50
36	48	60	36	48	60
42	56	70	42	56	70
48	64	80	48	64	80
54	72	90	54	72	90
60	80	100	60	80	100

- 6: P1 send m1 to P2
- 16: P2 receives m1
- 24: P2 sends m2 to P3
- 40: P3 receives m2
- 60: P3 sends m3 to P2
- 56: P2 receives m3
- 56: P2 clock reset=61**
- 64: P2 sends m4 to P1
- 54: P1 receives m4
- 70: P1 clock reset=70**

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.23

LAMPOR LOGICAL CLOCKS - IMPLEMENTATION

- Negative values not possible
- When a message is received, and the local clock is before the timestamp when then message was sent, the local clock is updated to message_sent_time + 1

- Clock is incremented before an event: sending a message, receiving a message, some other internal event
 P_i increments C_i : $C_i \leftarrow C_i + 1$
- When P_i send msg m to P_j , m 's timestamp is set to C_i
- When P_j receives msg m , P_j adjusts its local clock
 $C_j \leftarrow \max\{C_j, ts(m)\}$
- Ties broken by considering Proc ID: $i < j$; $\langle 40, i \rangle < \langle 40, j \rangle$

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.24

TOTAL-ORDERED MULTICASTING

- Consider concurrent updates to a replicated database
- Communication latency between DB1 and DB2 is 250ms

- Initial Account balance: \$1,000**
- Update #1: Deposit \$100**
- Update #2: Add 1% Interest**
- Total Ordered Multicasting needed**

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.25

TOTAL-ORDERED MULTICASTING EXAMPLE

Total Ordered Multicasting
 Logical clocks with Acknowledgements

$m_1 (10, P_1, \text{ADD } \$100)$
 $m_2 (12, P_2, \text{1% interest})$
 ↑
 Timestamps

TWO PROCESSES WITH RELATED DB-REPLICAS
 TWO PROCESSES AT DIFFERENT LOCATIONS AT ~250ms TIME
 RECEIVING DELAY
 Applying messages placed in queues ordered by timestamp

	P1 QUEUE	P2 QUEUE	ACK FIRST	ACKS →	P1 ACK'D	P2 ACK'D
ASST FIRST	P1 ACK'D	P2 ACK'D	ASST FIRST	ASST →	✓	✓
ASST	✓	✓	ASST →	ASST →	✓	✓
ASST SECOND	✓	✓				

Each message must be Acknowledged by every process in the system before operations in queue can be applied to the local DB...

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.26

TOTAL-ORDERED MULTICASTING - 2

- Each message timestamped with local logical clock of sender
- Multicast message is conceptually sent to the sender
- Assumptions:
 - Messages from same sender received in order they were sent
 - No messages are lost
- When messages arrive they are placed in local queue ordered by timestamp
- Receiver multicasts acknowledgement of message receipt to other processes
 - Time stamp of message receipt is lower the acknowledgement
- This process **replicates** queues across sites
- Process delivers messages to application only when message at the head of the queue has been acknowledged by every process in the system

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.27

TOTAL-ORDERED MULTICASTING - 3

- Can be used to provide replicated state machines (RSMs)
- Concept is to replicate event queues at each node
- (1) **Using logical clocks** and (2) **exchanging acknowledgement messages**, allows for events to be **"totally"** ordered in replicated event queues
- Events can be applied **"In order"** to each (distributed) replicated state machine (RSM)

Pass to other machines

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.28

VECTOR CLOCKS

- Lamport clocks don't help to determine causal ordering of messages
- Vector clocks capture causal histories and can be used as an alternative
- What is causality?

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.29

WHAT IS CAUSALITY?

- Consider the messages:

- P2 receives m1, and subsequently sends m3
- Causality:** Sending m3 **may** depend on what's contained in m1
- P2 receives m2, receiving m2 is **not** related to receiving m1
- Is sending m3 causally dependent on receiving m2?**

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.30

VECTOR CLOCKS

- Vector clocks keep track of **causal history**
- If two local events happened at process P, then the causal history $H(p2)$ of event p2 is $\{p1, p2\}$
- P sends messages to Q (event p3)
- Q previously performed event q1
- Q records arrival of message as q2
- Causal histories merged at Q $H(q2) = \{p1, p2, p3, q1, q2\}$
- Fortunately, can simply store history of last event, as a vector clock $\rightarrow H(q2) = (3, 2)$
- Each entry corresponds to the last event at the process

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.31

VECTOR CLOCKS - 2

- Each process maintains a vector clock which
 - Captures number of events at the local process (e.g. logical clock)
 - Captures number of events at all other processes
- Causality is captured by:
 - For each event at P_i , the vector clock (VC_i) is incremented
 - The msg is timestamped with VC_i; and sending the msg is recorded as a new event at P_i
 - P_j adjusts its VC_j choosing the **max** of: the message timestamp - or- the local vector clock (VC_j)

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.32

VECTOR CLOCKS - 3

- P_j knows the # of events at P_i based on the timestamps of the received message
- P_j learns how many events have occurred at other processes based on timestamps in the vector
- These events **"may be causally dependent"**
- In other words:** they may have been necessary for the message(s) to be sent...

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.33

VECTOR CLOCKS EXAMPLE

Local clock is underlined

CAUSALITY

$ts(m_2)$	$ts(m_4)$	$ts(m_2) < ts(m_4)$	$ts(m_2) > ts(m_4)$	Conclusion
(2,1,0)	(4,3,0)	Yes	No	m2 may causally precede m4

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.34

VECTOR CLOCKS EXAMPLE - 2

$ts(m_2)$	$ts(m_4)$	$ts(m_2) < ts(m_4)$	$ts(m_2) > ts(m_4)$	Conclusion
(4,1,0)	(2,3,0)	No	No	m2 and m4 may conflict

- P3 can't determine if m4 may be causally dependent on m2
- Is m4 causally dependent on m3?**

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.35

QUESTIONS

November 21, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L15.36

