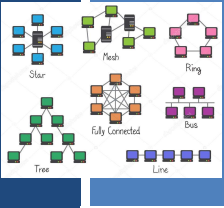


TCSS 558:  
APPLIED DISTRIBUTED COMPUTING

Chapter 4 – Communication  
Chapter 6 – Coordination

Wes J. Lloyd  
School of Engineering  
and Technology  
University of Washington - Tacoma



1

OBJECTIVES

- Assignment 2 - questions
- Feedback from 2/25
- Chapter 4.4: Multicast Communication
- Chapter 6: Coordination
- Chapter 6.1: Clock Synchronization
- Chapter 6.2: Logical Clocks

February 27, 2020

TCSS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.2

2

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (8 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 7.875**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 6.5**

February 27, 2020

TCSS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.3

3

FEEDBACK FROM 2/25

- **What Is client polling?**
- Types of asynchronous RPC:
- **Client polling**
  - Client (*using separate thread*) continually polls server for result
- When making an asynchronous RPC call, create a separate thread that is dedicated to querying the server if it has the result.
- Polling will repeatedly query the server at repeating intervals (e.g. every 10 seconds) to see if the result is available

February 27, 2020

TCSS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.4

4

FEEDBACK - 2

- **Assignment #2**
- **A server takes a port number and a membership server port?**  
**Can you describe the difference of these 2 port numbers?**
- For the TCP membership server option:
  - # New TCP server startup CLI:
  - `java -jar GenericNode.jar ts <listen-port> <membership-server-IP>`
- You may use a HARD CODED port for the membership server  
For example: **port 4410**.
- To deploy, first launch the membership server (single-node TCP key-value store) to listen on a special port (e.g. 4410).
- If using Docker check membership server's IP address
- When launching all nodes include the membership-server IP address
- If not using docker, just specify "localhost" - nodes assume port 4410

February 27, 2020

TCSS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.5

5

FEEDBACK - 3

- In-progress transactions should be written to a **data structure that is synchronized to prevent concurrent writes** to the same key.
  - Only **one** transaction, can change a given key, at a time
- **Will all server nodes need access to this data structure?**
- Every server node maintains its own data structure that stores key-value pairs
- The two-phase commit protocol is used to synchronize key-pairs to every node's data local store

February 27, 2020

TCSS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.6

6

SHORT-HAND-CODES FOR MEMBERSHIP TRACKING APPROACHES

- Include readme.txt or doc file with instructions in submission
- Must document membership tracking method
- **S-1:** Static file membership tracking only = 0 pts
- **T-1:** TCP membership tracking only = +5 pts (*should be dynamic once servers point to membership server*)
- **U-1:** UDP membership tracking only = +10 pts (*automatically discovers nodes with no configuration*)
- **S+T-2:** Static file + TCP membership tracking = +15 pts (*Static file is not reread to refresh membership during operation*)
- **S+U-2:** Static file + UDP membership tracking = +15 pts (*Static file is not reread to refresh membership during operation*)
- **SD+U-2:** Static file + UDP membership tracking = +20 pts (*Static file is periodically reread to refresh membership during operation*)
- **T+U-2:** TCP + UDP membership tracking = 20 pts (*both dynamic*)

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.7

7

CH. 4 COMMUNICATION

L15.8

8

CHAPTER 4

- 4.1 Foundations
  - Protocols
  - Types of communication
- 4.2 Remote procedure call
- 4.3 Message-oriented communication
  - Socket communication
  - Messaging libraries
  - Message-Passing Interface (MPI)
  - Message-queueing systems
  - Examples
- 4.4 Multicast communication
  - Flooding-based multicasting
  - Gossip-based data dissemination

These sections feature many details, Our focus is on the "big picture"

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.9

9

CH. 4.4: MULTICAST COMMUNICATION

Multicast

one to many  
X = subscriber

Apache ActiveMQ

L15.10

10

MESSAGE FLOODING

- For deterministic topologies (such as hypercube), design of efficient flooding scheme is much simpler
- If the overlay network is structured, this gives us a deterministic topology
- Schlosser et al [2002] – offer simple and efficient broadcasting scheme that relies on keeping track of neighbors per dimension

0000 0001 1000 1001  
0010 0011 1010 1011  
0100 0101 1100 1101  
0110 0111 1110 1111

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.11

11

MESSAGE FLOODING - 2

0000 0001 1000 1001  
0010 0011 1010 1011  
0100 0101 1100 1101  
0110 0111 1110 1111

- **Hypercube Broadcast**
- N(1001) starts the network broadcast
- N(1001) neighbors {0001,1000,1011,1101}
- N(1001) Sends message to all neighbors
- **>>Edge Labels (which bit is changed?, 1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup>...)**
- Edge to 0001 – labeled 1 – change the 1<sup>st</sup> bit
- Edge to 1000 – labeled 4 – change the 4<sup>th</sup> bit
- Edge to 1011 – labeled 3 – change the 3<sup>rd</sup> bit
- Edge to 1101 – labeled 2 – change the 2<sup>nd</sup> bit
- **RULE: nodes only forward along edges with a higher dimension**
- Node 1101 receives message on edge labeled 2
- Broadcast msg is only forwarded on **higher** valued edges (>2)

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.12

12

### MESSAGE FLOODING - 3

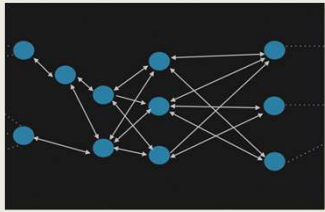
- **Hypercube:** forward msg along edges with higher dimension
- Node(1101)-neighbors {0101,1100,1001,1111}
- Node (1101) - incoming broadcast edge = 2
- **Label Edges:**
- Edge to 0101 - labeled 1 - change the 1<sup>st</sup> bit
- Edge to 1100 - labeled 4 - change the 4<sup>th</sup> bit \***<FORWARD>**\*
- Edge to 1001 - labeled 2 - change the 2<sup>nd</sup> bit
- Edge to 1111 - labeled 3 - change the 3<sup>rd</sup> bit \***<FORWARD>**\*
- N(1101) broadcast - forward only to N(1100) and N(1111)
- (1100) and (1111) are the **higher dimension edges**
- Broadcast requires just:  $N-1$  messages, where nodes  $N=2^n$ ,  $n$ =dimensions of hypercube

February 27, 2020 TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma L15.13

13

### GOSSIP BASED DATA DISSEMINATION

- When structured peer-to-peer topologies are not available
- Gossip based approaches support multicast communication over unstructured peer-to-peer networks
- General approach is to leverage how gossip spreads across a group
- This is also called "epidemic behavior"...
- Data updates for a specific item begin at a specific node



February 27, 2020 TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma L15.14

14

### INFORMATION DISSEMINATION

- **Epidemic algorithms:** algorithms for large-scale distributed systems that spread information
- Goal: "infect" all nodes with new information as fast as possible
- **Infected:** node with data that can spread to other nodes
- **Susceptible:** node without data
- **Removed:** node with data that is unable to spread data

February 27, 2020 TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma L15.15

15

### EPIDEMIC PROTOCOLS

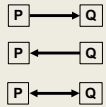
- **Gossiping**
- Nodes are randomly selected
- One node, randomly selects any other node in the network to propagate the network
- Complete set of nodes is known to each member

February 27, 2020 TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma L15.16

16

### ANTI ENTROPY DISSEMINATION MODEL FOR GOSSIPING

- **Anti-entropy:** Propagation model where node P picks node Q at random and exchanges message updates
- Akin to random walk
- **PUSH:** P only **pushes** its own updates to Q
- **PULL:** P only **pulls** in new updates from Q
- **TWO-WAY:** P and Q send updates to each other (i.e. a push-pull approach)
- Push only: hard to propagate updates to last few hidden susceptible nodes
- Pull: better because susceptible nodes can pull updates from infected nodes
- Push-pull is better still

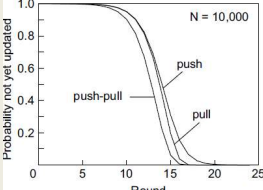


February 27, 2020 TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma L15.17

17

### ANTI ENTROPY EFFECTIVENESS

- **Round:** span of time during which every node takes initiative to exchange updates with a randomly chosen node
- The number of rounds to propagate a single update to all nodes requires  $O(\log(N))$ , where  $N$ =number of nodes
- Let  $p_i$  denote probability that node P has not received msg m after the  $i$ th round.
- For pull, push, and push-pull based approaches:



10,000 nodes →

February 27, 2020 TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma L15.18

18

## RUMOR SPREADING

- Variant of epidemic protocols
- Provides an approach to **"stop"** message spreading
- Mimics "gossiping" in real life
- Rumor spreading:**
  - Node P** receives new data **Item X**
  - Contacts an arbitrary **node Q** to push update
  - Node Q** reports already receiving **Item X** from another node
  - Node P** may loose interest in spreading the rumor with probability =  $p_{\text{stop}}$ , let's say 20% . . . (or 0.20)

February 27, 2020
TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma
L15.19

19

## RUMOR SPREADING - 2

- Does not guarantee all nodes will be updated
- The fraction of nodes  $s$ , that remain susceptible grows relative to the probability that node **P** stops propagating when finding a node already having the message
- Fraction of nodes not updated remains < 0.20 with high  $p_{\text{stop}}$
- Susceptible nodes ( $s$ ) vs. probability of stopping  $p_{\text{stop}}$  →

February 27, 2020
TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma
L15.20

20

## REMOVING DATA

- Gossiping is good for spreading data
- But how can data be removed from the system?**
- Idea is to issue **"death certificates"**
  - Act like data records, which are spread like data
  - When death certificate is received, data is deleted
  - Certificate is held to prevent data element from reinitializing from gossip from other nodes
  - Death certificates time-out after expected time required for data element to clear out of entire system
  - A few nodes maintain death certificates forever

February 27, 2020
TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma
L15.21

21

## DEATH CERTIFICATE EXAMPLE

- For example:**
  - Node P** keeps death certificates forever
  - Item X** is removed from the system
  - Node P** receives an update request for **Item X**, but **also** holds the death certificate for **Item X**
  - Node P** will recirculate the death certificate across the network for **Item X**

February 27, 2020
TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma
L15.22

22

## 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*)

February 27, 2020
TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma
L15.23

23

## CHAPTER 6 - COORDINATION

- How can processes synchronize and coordinate data?
- Process synchronization
  - Coordinate cooperation to grant individual processes temporary access to shared resources (e.g. a file)
- Data synchronization
  - Ensure two sets of data are the same (data replication)
- Coordination
  - Goal is to manage interactions and dependencies between activities in the distributed system
  - Encapsulates synchronization

February 27, 2020
TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma
L15.24

24

## COORDINATION - 2

- Synchronization challenges begin with **time**:
  - How can we synchronize computers, so they all agree on the time?
  - How do we measure and coordinate when things happen?
- Fortunately, for synchronization in distributed systems, it is often sufficient to only agree on a relative ordering of events
  - E.g. not actual time

February 27, 2020

TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.25

25

## COORDINATION - 3

- Groups of processes often appoint a **coordinator**
- **Election algorithms** can help elect a leader
- Synchronizing access to a shared resource is achieved with **distributed mutual exclusion** algorithms
- Also in chapter 6:
  - Matching subscriptions to publications in publish-subscribe systems
  - Gossip-based coordination problems:
    - Aggregation
    - Peer sampling
    - Overlay construction

February 27, 2020

TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.26

26

## CH. 6.1: CLOCK SYNCHRONIZATION



L15.27

27

## CLOCK SYNCHRONIZATION

- Example:
  - "make" is used to compile source files into binary object and executable files
  - As an optimization, make only compiles files when the "last modified time" of source files is more recent than object and executables
- Consider if files are on a shared disk of a distributed system where there is no agreement on time
- Consider if the program has 1,000 source files

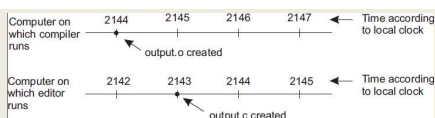
February 27, 2020

TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.28

28

## TIME SYNCHRONIZATION PROBLEM FOR DISTRIBUTED SYSTEMS



- Updates from different machines, may have clocks set to different times
- Make becomes confused with which files to recompile

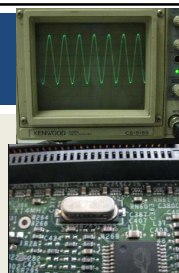
February 27, 2020

TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.29

29

## PHYSICAL CLOCKS



- **Computer timers**: precisely machined quartz crystals
- When under tension, they oscillate at a well defined frequency
- In analog electronics/communications crystals once used to set the frequency of two-way radio transceivers for
- Today, crystals are associated with a counter and holding register on a digital computer.
- Each oscillation decrements a counter by one
- When counter gets to zero, an interrupt fires
- Can program timer to generate interrupt, let's say 60 times a second, or another frequency to track time

1960s ERA radio crystal →



February 27, 2020



TCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.30

30

COMPUTER CLOCKS

- Digital clock on computer sets base time
- Crystal clock tracks forward progress of time
  - Translation of wave "ticks" to clock pulses
- CMOS battery on motherboard maintains clock on power loss
- Clock skew**: physical clock crystals are not exactly the same
  - Some run at slightly different rates
- Time differences accumulate as clocks drift forward or backward slightly
- In an automobile, where there is no clock synchronization, clock skew may become noticeable over months, years



February 27, 2020

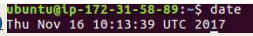
TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.31

31

UNIVERSAL COORDINATED TIME

- Universal Coordinated Time (UTC)**
  - Worldwide standard for time keeping
  - Equivalent to Greenwich Mean Time (United Kingdom)
  - 40 shortwave radio stations around the world broadcast a short pulse at the start of each second (WWV)
  - World wide "atomic" clocks powered by constant transitions of the non-radioactive caesium-133 atom
    - 9,162,631,770 transitions per second
- Computers track time using UTC as a base
  - Avoid thinking in local time, which can lead to coordination issues
  - Operating systems may translate to show local time



February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.32

32

COMPUTING: CLOCK CHALLENGES

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

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.33

33

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 ( $\pi$ )**: 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 ( $\rho$ )**: clock specifications include one

February 27, 2020

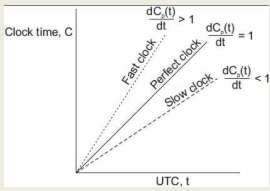
TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.34

34

CLOCK SYNCHRONIZATION - 2

- If two clocks drift from UTC in opposite directions, after time  $\Delta t$  after synchronization, they may be  $2\rho$  apart.
  - $\rho$  - clock drift rate,  $\pi$  - clock precision (max 50ns)
- Clocks must be resynchronized every  $\pi/2\rho$  seconds
- Network time protocol**
- Provide coordination of time for servers
- Leverage distributed network of time servers



February 27, 2020

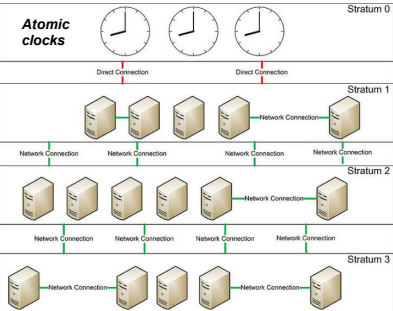
TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.35

35

NETWORK TIME PROTOCOL

- Servers organized into strataums
- 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



February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

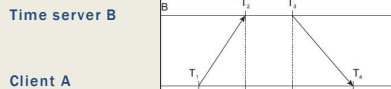
L15.36

36



## NTP - 2

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



- A sends message to B, with timestamp  $T_1$
  - B records time of receipt  $T_2$  (from local clock)
  - B returns response with send time  $T_3$ , and receipt time  $T_4$
  - 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

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.37

37

## 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`)

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.38

38

## 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

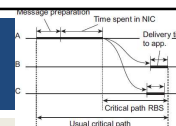
February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.39

39

## 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

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.40

40

## 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

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.41

41

## 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:

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

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.42

42

CH. 6.2: LOGICAL CLOCKS

time

February 27, 2020TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.43

43

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

February 27, 2020TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.44

44

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  $A \rightarrow C$

February 27, 2020TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.45

45

LOGICAL CLOCKS - 3

- If two events, say event X and event Y do not exchange messages, not even via third parties, then the sequence of  $X \rightarrow Y$  vs.  $Y \rightarrow X$  **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

February 27, 2020TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.46

46

LOGICAL CLOCKS - 4

- Three processes each with local clocks
- Lamport's algorithm** corrects process clock values
- Always propagate the most recent known value of logical time

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

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

February 27, 2020TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.47

47

LOGICAL CLOCKS

- Events:**

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

February 27, 2020TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.48

48



LAMPORT 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  
 $C_j \leftarrow \max\{C_j, \text{timestamp}(m)\}$
- Ties broken by considering Proc ID:  $i < j$ ;  $\langle 40, i \rangle < \langle 40, j \rangle$

February 27, 2020

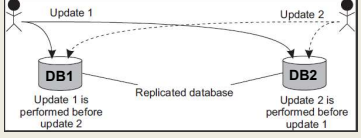
TCSS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.49

49

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

February 27, 2020

TCSS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.50

50

TOTAL-ORDERED MULTICASTING  
EXAMPLE

- Two messages ( $m_1, m_2$ ) must be distributed, to two processes ( $p_1, p_2$ )
- We assume messages have correct lamport clock timestamps
  - $m_1(10, p_1, \text{add } \$100)$
  - $m_2(12, p_2, \text{add } 1\% \text{ interest})$
- Each process maintains a queue of messages
- Arriving messages are placed into queues ordered by the lamport clock timestamp
- In each queue, each message must acknowledged by every process in the system before operations can be applied to the local database

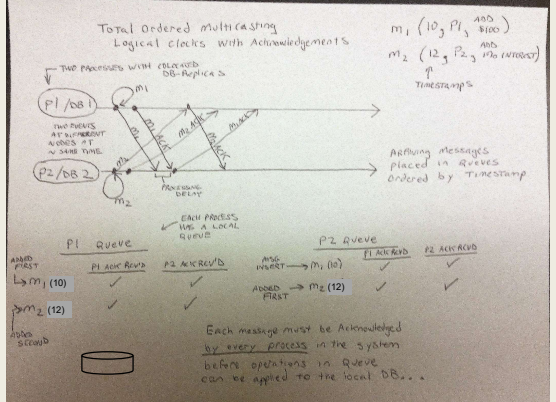
February 27, 2020

TCSS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.51

51

TOTAL-ORDERED MULTICASTING EXAMPLE



52

TOTAL-ORDERED MULTICASTING - 2

- Each message timestamped with local logical clock of sender
- Multicast message is conceptually "sent" to the sender (itself)
- 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
- Messages delivered to application (database) only when message at the head of the queue has been acknowledged by every process in the system

February 27, 2020

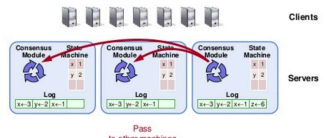
TCSS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.53

53

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)



February 27, 2020

TCSS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.54

54

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?

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.55

55

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?*

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.56

56

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 → H(q2) = (3,2)
- Each entry corresponds to the last event at the process

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.57

57

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 Pi, the vector clock (VCi) is incremented
  - The msg is timestamped with VCi and sending the msg is recorded as a new event at Pi
  - Pj adjusts its VCj choosing the **max** of: the message timestamp –or– the local vector clock (VCj)

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.58

58

VECTOR CLOCKS - 3

- Pj knows the # of events at Pi based on the timestamps of the received message
- Pj 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...

February 27, 2020

TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.59

59

VECTOR CLOCKS EXAMPLE

- Local clock is underlined

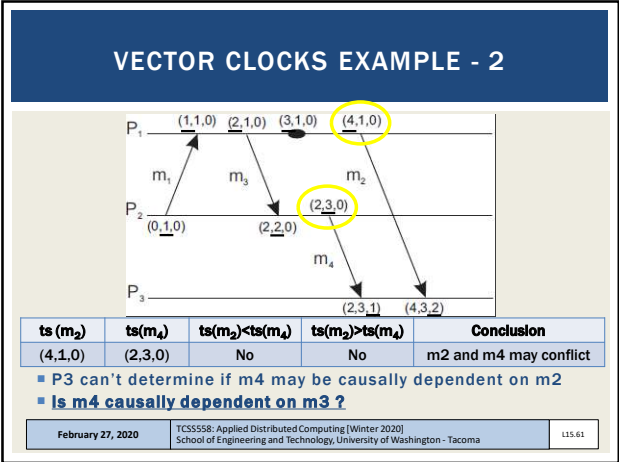
ts (m2)	ts (m4)	ts(m2)<ts(m4)	ts(m2)>ts(m4)	Conclusion
(2,1,0)	(4,3,0)	Yes	No	m2 may causally precede m4

February 27, 2020

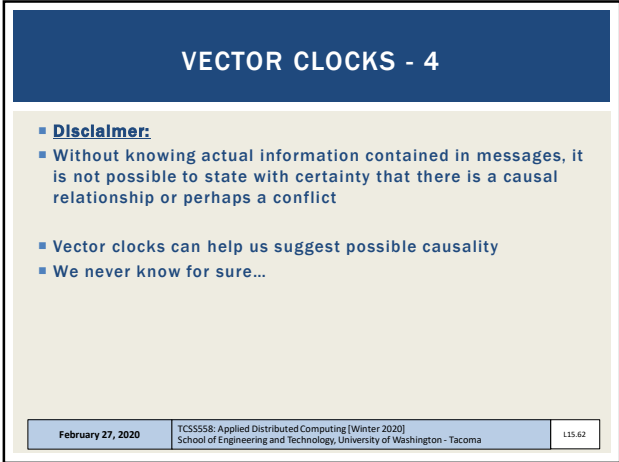
TCCS558: Applied Distributed Computing [Winter 2020]  
School of Engineering and Technology, University of Washington - Tacoma

L15.60

60



61



62



63