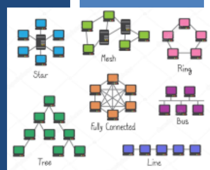
TCSS 558: APPLIED DISTRIBUTED COMPUTING

Chapter 6 – Coordination - III

Wes J. Lloyd
School of Engineering
& Technology (SET)
University of Washington - Tacoma



1

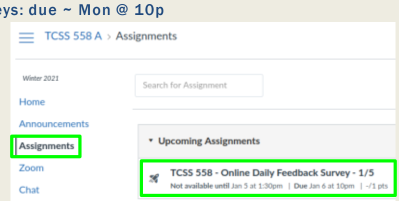
OBJECTIVES – 3/2

- Questions from 2/28
- Assignment 2: Replicated Key Value Store
- Chapter 6: Coordination
  - Chapter 6.2: Logical Clocks
    Vector Clocks
- Class Activity 4 – Total Ordered Multicasting
- Chapter 6: Coordination
  - Chapter 6.3: Distributed Mutual Exclusion

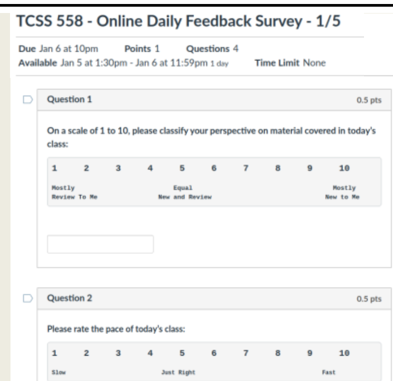
2

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Available After Each Class
- Extra credit available for completing surveys ***ON TIME***
- Tuesday surveys: due by ~ Wed @ 10p
- Thursday surveys: due ~ Mon @ 10p



3



4

MATERIAL / PACE

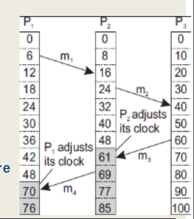
- Please classify your perspective on material covered in today's class (24 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.25** (↓ - *previous 6.57*)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.65** (↓ - *previous 5.81*)

5

FEEDBACK FROM 2/28

- ***Will adjusting the time using Lamport's algorithm affect the order of other messages in the process?***
- Node P_2 receives a message m_3 from node P_3 at time=56
- The timestamp of m_3 from P_3 is time=60
- We assume that P_3 sends the message at time=60
- P_2 must receive m_3 at P_3's time+1
- P_2's clock is adjusted to time=61
- Because of the happens-before relation, we still say m_1 and m_2 happen before m_3
- What if m_5 arrives at P_3 from P_4 at time=50
- M5 is time stamped at time=24
- Does the sending of m_2 from P_2 happen before the sending of m_5 from P_4?
- This question asks about causality

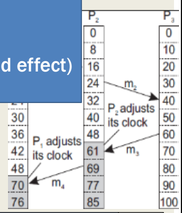


6

FEEDBACK FROM 2/28

- ***Will adjusting the time using Lamport's algorithm affect the order of other messages in the process?***
- Node P_2 receives a message m_1 from node P_1 at time=56
- The tim
- We assu time=60
- P_2 must
- P_2's clo
- Because we still say m_1 and m_2 happen before m_3
- What if m_5 arrives at P_3 from P_4 at time=50
- M5 is time stamped at time=24
- Does the sending of m_2 from P_2 happen before the sending of m_5 from P_4?
- This question asks about causality

KEY CONCEPT:
Lamport clock's are insufficient to determine causality (cause and effect)



7

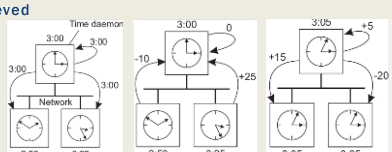
FEEDBACK - 2

- ***What is the difference between clock time accuracy vs. precision?***
- **Accuracy** is how close a given set of time measurements are to their true value
- Assume the atomic clock (UTC receiver) is the authority
- For a given node A, the time **accuracy** will be the error relative to the authority,
- **Precision** is how close two time measurements are to each other
  - Here neither is an authority
- **Example:**
- Consider if the UTC receiver indicates the time is 12:00:00 AM
- Node A and Node B both indicate the time is 12:00:25 AM
- What is the accuracy of Node A and B's sense of time?
- What is the precision of time between Node A and Node B?

8

FEEDBACK - 3

- ***Please explain the Berkeley algorithm***
- In the absence of a time authority, a **time daemon** polls the system to assess the consensus of time
- Nodes report their time offset relative to the time daemon
- Time daemon calculates average time, tells other nodes to advance or slow their clocks to the new time has been achieved



9

FEEDBACK - 3

- ***Please explain the Berkeley algorithm***
- In the absence of a time authority, a **time daemon** polls the system to assess the consensus of time
- Nod… on
- Time… to adva… achi…

Notice how the average time difference is 5:

average time = (-10 + 25) / 3 (nodes) = +5

Time daemon adjusts clock to new consensus of time

10

FEEDBACK - 4

- ***Please provide an example of RBS***
- RBS used for time synchronization in wireless sensor networks
- RBS: assumes no node has accurate time
- ***GOAL***: internally synchronize clocks – same as Berkeley
- RBS: to save wireless network bandwidth, 2-way time sync not performed. Instead receivers note the time offset when msgs are received
- RBS: assumes message delivery time is roughly constant (*no multi-hop message delivery*)
- RBS: ignores time spent to construct message, and time spent to access network (*only records receipt time stamps*)
- RBS: receiver notes time when message arrives ($T_{p,m}$)
  assumes constant message delivery time (single-hop)

11

FEEDBACK - 5

- RBS: two nodes p and q communicate their relative offset
- P tells q when it received msg, Q tells p when it received msg
- Assume message delivery time is same
- P and Q exchange message receipt times
- Calculate average offset between P and Q (how much time varies)
- Difference between p and q time represents avg clock offset

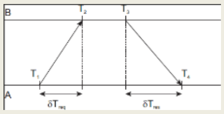
$$Offset[p,q] = \frac{\sum_{k=1}^{M}(T_{p,k} - T_{q,k})}{M}$$

- Offset is calculated for a series of M messages
- Nodes don't adjust clocks to save battery – instead node offset
- UNFORUNATELY: Average clock offset is not constant
- Avg clock offset is precise at first, but then clocks drift apart
- Elson et al. propose to use linear regression to estimate how clocks will drift apart over time using historical p and q offsets

12

FEEDBACK - 6

- ***NTP example:***
  Time server B



θ = clock offset
δ = propagation delay

Client A

- Assume: $\delta T_{req} = \delta T_{res} = 50$
- $T_1=50$, $T_2(@A)=100$, $T_2=200$, $T_3=300$, $T_3(@A)=200$, $T_4=250$
- Calculate clock offset (θ) between A and B
- $\theta = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$ $\delta = \frac{(T_4 - T_1) - (T_3 - T_2)}{2}$
- **What is the clock offset between A and B?**

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.13

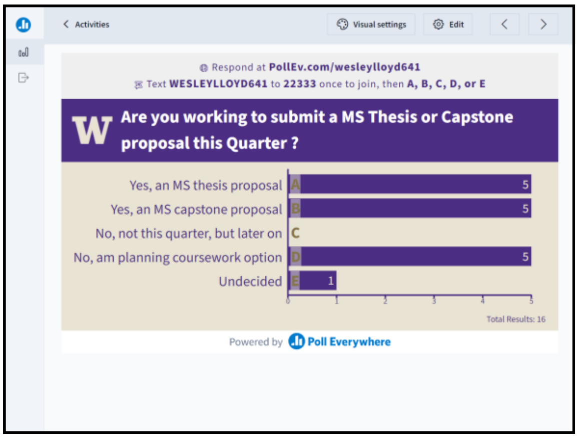
13

FEEDBACK - 7

- ***How do the formulas shown in class work; can we get some practice problems with them so we can better know how to use them in a class activity or an exam?***
- Class Activity 3
- Class Activity 4
- Practice Final Exam – next Thursday 3/9

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.14

14



15

OBJECTIVES – 3/2

- Questions from 2/28
- **Assignment 2: Replicated Key Value Store**
- Chapter 6: Coordination
  - Chapter 6.2: Logical Clocks
    Vector Clocks
- Class Activity 4 – Total Ordered Multicasting
- Chapter 6: Coordination
  - Chapter 6.3: Distributed Mutual Exclusion

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.16

16

SHORT-HAND-CODES FOR MEMBERSHIP TRACKING APPROACHES

- Include readme.txt or doc file with instructions in submission
- Must document membership tracking method

>> please indicate which types to test <<

ID	Description
F	Static file membership tracking – file is not reread
FD	Static file membership tracking DYNAMIC - file is periodically reread to refresh membership list
T	TCP membership tracking – servers are configured to refer to central membership server
U	UDP membership tracking - automatically discovers nodes with no configuration

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.17

17

ASSIGNMENT 2

- **Sunday March 12th**
- Goal: Replicated Key Value Store
- Team signup posted on Canvas under 'People'
- Builds off of Assignment 1 GenericNode
- Focus on TCP client/server w/ replication
- How to track membership for data replication?
  - Can implement multiple types of membership tracking for extra credit

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.18

18

OBJECTIVES – 3/2

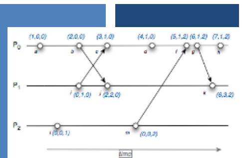
- Questions from 2/28
- Assignment 2: Replicated Key Value Store
- Chapter 6: Coordination
  - Chapter 6.2: Logical Clocks
    Vector Clocks
- Class Activity 4 – Total Ordered Multicasting
- Chapter 6: Coordination
  - Chapter 6.3: Distributed Mutual Exclusion

19

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)*

20

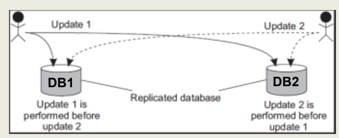


CH. 6.2: LOGICAL CLOCKS

21

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

22

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, add $100)
- m_2(12, p_2, add 1% 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 be acknowledged by every process in the system before operations can be applied to the local database

23

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, add $100)

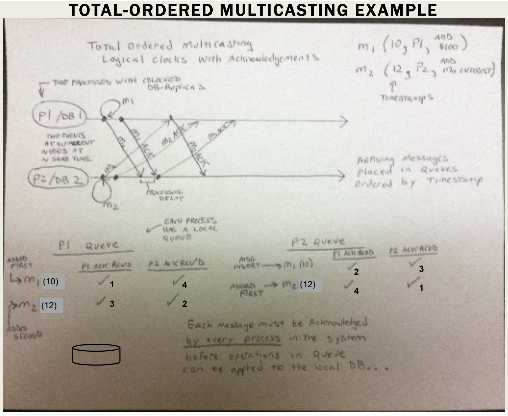
Key point:
Multicast messages are also received by the sender (*itself*)

- Arriving messages are placed into queues ordered by the Lamport clock timestamp

- In each queue, each message must be acknowledged by every process in the system before operations can be applied to the local database

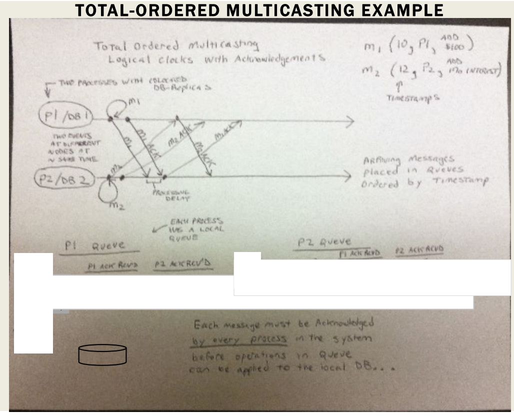
24

TOTAL-ORDERED MULTICASTING EXAMPLE



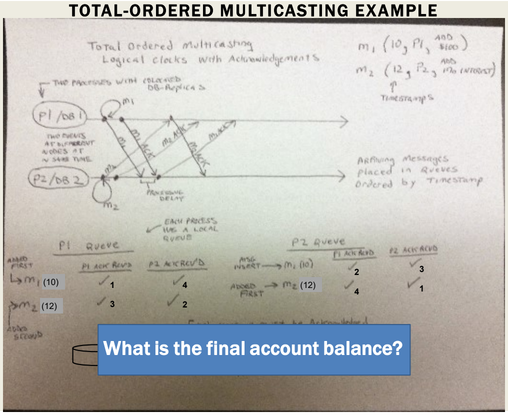
25

TOTAL-ORDERED MULTICASTING EXAMPLE



26

TOTAL-ORDERED MULTICASTING EXAMPLE



27

TOTAL-ORDERED MULTICASTING - 2

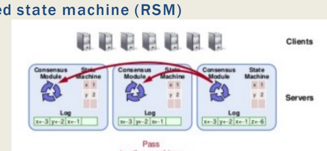
- Each message timestamped with local logical clock of sender
- **Multicast messages are also received by the sender (*itself*)**
- Assumptions:
  - Messages from same sender received in order they were sent
  - No messages are lost
- When messages arrive they are placed in local queue ordered by timestamp
- 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

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.28

28

TOTAL-ORDERED MULTICASTING - 3

- Can be used to implement 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)



March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.29

29

OBJECTIVES – 3/2

- Questions from 2/28
- Assignment 2: Replicated Key Value Store
- Chapter 6: Coordination
  - Chapter 6.2: Logical Clocks
  - **Vector Clocks**
- Class Activity 4 – Total Ordered Multicasting
- Chapter 6: Coordination
  - Chapter 6.3: Distributed Mutual Exclusion

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.30

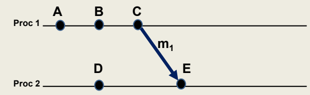
30

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
- But what is causality? …

31

WHAT IS CAUSALITY?



- Having a causal relationship between two events (A and E) indicates that event E results from the occurrence of event A.
- When one event results from another, there is a causal relationship between the two events.
- This is also referred to as cause and effect.

32

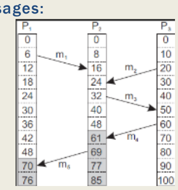
CAUSALITY - 2

- **Disclaimer:**
- Without knowing actual information contained in messages, it is not possible to state with certainty that there is a causal relationship or perhaps a conflict
- Lamport/Vector clocks can help us suggest possible causality
- But we never know for sure…

33

CAUSALITY - 3

- 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?***

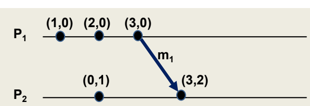
34

VECTOR CLOCKS

- Vector clocks help 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

35

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 (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)

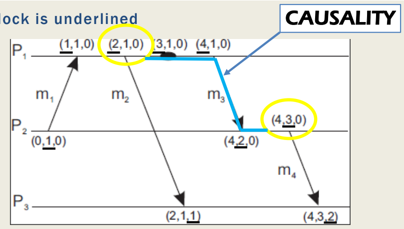
36

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…

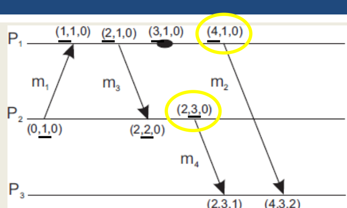
VECTOR CLOCKS EXAMPLE

- Local clock is underlined



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

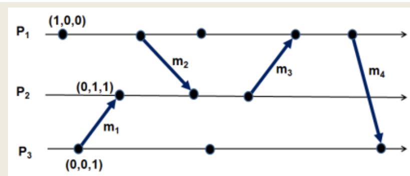
VECTOR CLOCKS EXAMPLE - 2



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

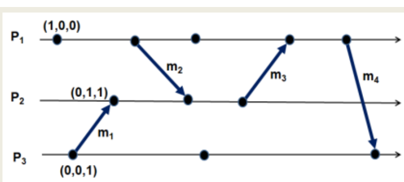
- P3 can't determine if m4 may be causally dependent on m2
- **Is m4 causally dependent on m3 ?**

VECTOR CLOCKS EXAMPLE - 3



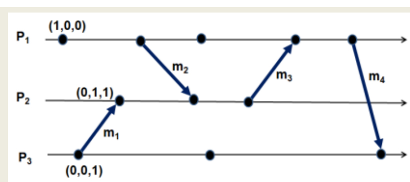
- Provide a vector clock label for unlabeled events

VECTOR CLOCKS EXAMPLE - 4



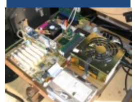
- TRUE/FALSE:
- The sending of message m_3 is causally dependent on the sending of message m_1.
- The sending of message m_2 is causally dependent on the sending of message m_1.

VECTOR CLOCKS EXAMPLE - 5



- TRUE/FALSE:
- P_1 (1,0,0) and P_3 (0,0,1) may be concurrent events.
- P_2 (0,1,1) and P_3 (0,0,1) may be concurrent events.
- P_1 (1,0,0) and P_2 (0,1,1) may be concurrent events.

WE WILL RETURN AT 2:40 PM



43

OBJECTIVES – 3/2

- Questions from 2/28
- Assignment 2: Replicated Key Value Store
- Chapter 6: Coordination
  - Chapter 6.2: Logical Clocks
    Vector Clocks
- **Class Activity 4 – Total Ordered Multicasting**
- Chapter 6: Coordination
  - Chapter 6.3: Distributed Mutual Exclusion

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.44

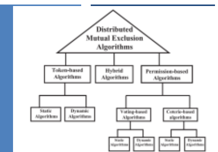
44

OBJECTIVES – 3/2

- Questions from 2/28
- Assignment 2: Replicated Key Value Store
- Chapter 6: Coordination
  - Chapter 6.2: Logical Clocks
    Vector Clocks
- Class Activity 4 – Total Ordered Multicasting
- Chapter 6: Coordination
  - **Chapter 6.3: Distributed Mutual Exclusion**

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.45

45



CH. 6.3: DISTRIBUTED MUTUAL EXCLUSION

L17.46

46

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS

- Coordinating access among distributed processes to a shared resource requires **Distributed Mutual Exclusion**
- **Algorithms in 6.3**
- Token-ring algorithm
- ***Permission-based algorithms:***
- Centralized algorithm
- Distributed algorithm (Ricart and Agrawala)
- Decentralized voting algorithm (Lin et al.)

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.47

47

TOKEN-BASED ALGORITHMS

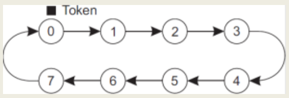
- Mutual exclusion by passing a “token” between nodes
- Nodes often organized in ring
- Only one token, holder has access to shared resource
- Avoids starvation: ***everyone gets a chance to obtain lock***
- Avoids deadlock: easy to avoid

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.48

48

TOKEN-RING ALGORITHM

- Construct overlay network
- Establish logical ring among nodes



- Single token circulated around the nodes of the network
- Node having token can access shared resource
- If no node accesses resource, token is constantly circulated around ring

49

TOKEN-RING CHALLENGES

1. If token is lost, token must be regenerated
   - **Problem**: may accidentally circulate multiple tokens
2. Hard to determine if token is lost
   - What is the difference between token being lost and a node holding the token (***lock***) for a long time?
3. When node crashes, circular network route is broken
   - Dead nodes can be detected by adding a receipt message for when the token passes from node-to-node
   - When no receipt is received, node assumed dead
   - Dead process can be "jumped" in the ring

50

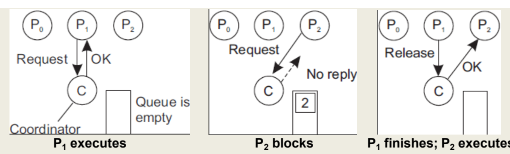
DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS - 3

- **Permission-based algorithms**
- Processes must require permission from other processes before first acquiring access to the resource
  - CONTRAST: Token-ring did not ask nodes for permission
- **Centralized algorithm**
- Elect a single leader node to coordinate access to shared resource(s)
- Manage mutual exclusion on a distributed system similar to how mutual exclusion is managed for a single system
- Nodes must all interact with leader to obtain ***"the lock"***

51

CENTRALIZED MUTUAL EXCLUSION

Permission granted from coordinator V No response from coordinator



- When resource not available, coordinator can block the requesting process, or respond with a reject message
- P2 must ***poll*** the coordinator if it responds with reject otherwise can wait if simply blocked
- Requests are granted permission fairly using FIFO queue
- Just three messages: (request, grant (OK), release)

52

CENTRALIZED MUTUAL EXCLUSION - 2

- **Issues**
- Coordinator is a single point of failure
- Processes can't distinguish dead coordinator from ***"blocking"*** when resource is unavailable
  - No difference between CRASH and BLOCK (*for a long time*)
- Large systems, coordinator becomes performance bottleneck
  - Scalability: Performance does not scale
- **Benefits**
- Simplicity:
  Easy to implement compared to distributed alternatives

53

DISTRIBUTED ALGORITHM

- Ricart and Agrawala [1981], use total ordering of all events
  - Leverages Lamport logical clocks
- Package up resource request message (AKA Lock Request)
- Send to all nodes
- Include:
  - Name of resource
  - Process number
  - Current (logical) time
- Assume messages are sent reliably
  - No messages are lost

54

DISTRIBUTED ALGORITHM - 2

- When each node receives a request message they will:

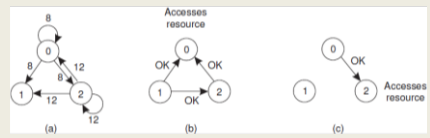
1. Say OK (***if the node doesn't need the resource***)
2. Make **no reply**, queue request (***node is using the resource)***
3. ***If node is also waiting to access the resource:*** perform a timestamp comparison -
   1. Send OK if requester has lower logical clock value
   2. Make **no reply** if requester has higher logical clock value

- Nodes sit back and wait for all nodes to grant permission
- Requirement: every node must know the entire membership list of the distributed system

55

DISTRIBUTED ALGORITHM - 3

- Node 0 and Node 2 simultaneously request access to **resource**
- Node 0's time stamp is lower (8) than Node 2 (12)
- Node 1 and Node 2 grant Node 0 access
- Node 1 is not interested in the resource, it OKs both requests



- **In case of conflict, lowest timestamp wins!**
  - Node 2 rejects its own request (12) in favor of node 0 (8)

56

CHALLENGES WITH DISTRIBUTED ALGORITHM

- **Problem:** Algorithm has N points of failure !
- Where N = Number of Nodes in the system
- **No Reply Problem:** When node is accessing the resource, it does not respond
  - Lack of response can be confused with **failure**
  - ***Possible Solution:*** When node receives request for resource it is accessing, always send a reply either granting or denying permission (ACK)
  - Enables requester to determine when nodes have died

57

CHALLENGES WITH DISTRIBUTED ALGORITHM - 2

- **Problem**: Multicast communication required –or- each node must maintain full group membership
  - Track nodes entering, leaving, crashing…
- **Problem**: Every process is involved in reaching an agreement to grant access to a shared resource
  - This approach ***may not scale*** on resource-constrained systems
- **Solution**: Can relax total agreement requirement and proceed when a **simple majority** of nodes grant permission (>50%)
  - *Presumably any one node locking the resource prevents agreement*
  - *If one node gets majority of acknowledges no other can*
  - *Requires every node to know size of system (# of nodes)*
- **Problem:** 2 concurrent transactions get 50% permission → deadlock?
- Distributed algorithm for mutual exclusion works best for:
  - Small groups of processes
  - When memberships rarely change

58

DECENTRALIZED ALGORITHM

- Lin et al. [2004], decentralized voting algorithm
- Resource is replicated N times
- Each replica has its own coordinator …(N coordinators)
- Accessing resource requires majority vote: total votes (m) > N/2 coordinators
- **Assumption #1:** When coordinator does not give permission to access a resource (because it is busy) it will inform the requester

59

DECENTRALIZED ALGORITHM - 2

- **Assumption #2:** When a coordinator crashes, it recovers quickly, but will have forgotten votes before the crash.
- Approach assumes coordinators reset **arbitrarily** at any time
- **Risk**: on crash, coordinator forgets it previously granted permission to the shared resource, and on recovery it errantly grants permission again
- **The Hope**: if coordinator crashes, *upon recovery*, ***the node granted access to the resource has already finished before the restored coordinator grants access again . . .***

60

DECENTRALIZED ALGORITHM - 3

- With 99.167% coordinator availability (30 sec downtime/hour) chance of violating correctness **is so low** it can be neglected in comparison to other types of failure
- Leverages fact that a new node must obtain a majority vote to access resource, ***which requires time***

N	m	p	Violation
8	5	3 sec/hour	$< 10^{-15}$
8	6	3 sec/hour	$< 10^{-18}$
16	9	3 sec/hour	$< 10^{-27}$
16	12	3 sec/hour	$< 10^{-36}$
32	17	3 sec/hour	$< 10^{-52}$
32	24	3 sec/hour	$< 10^{-73}$

N	m	p	Violation
8	5	30 sec/hour	$< 10^{-10}$
8	6	30 sec/hour	$< 10^{-11}$
16	9	30 sec/hour	$< 10^{-18}$
16	12	30 sec/hour	$< 10^{-24}$
32	17	30 sec/hour	$< 10^{-35}$
32	24	30 sec/hour	$< 10^{-49}$

N = number of resource replicas, m = required "majority" vote
p=seconds per hour coordinator is offline

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.61

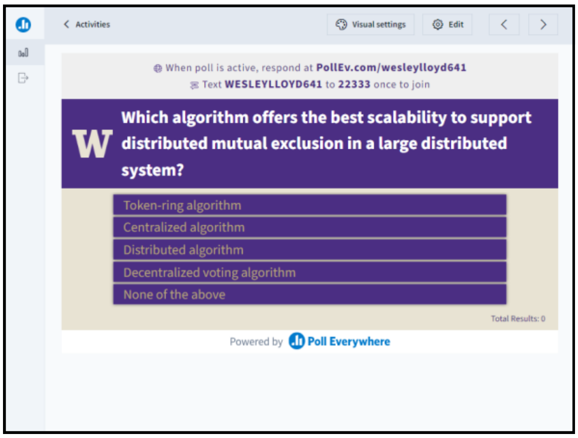
61

DECENTRALIZED ALGORITHM - 4

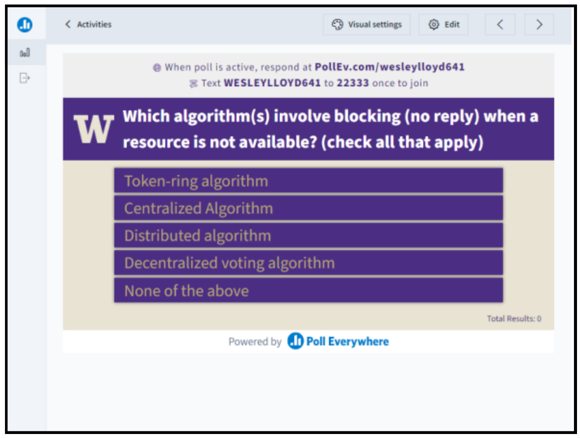
- **Back-off Polling Approach for *permission-denied*:**
- If permission to access a resource is denied via majority vote, process can poll to gain access again with a ***random*** delay (***known as back-off***)
- Node waits for a random amount, retries…
- If too many nodes compete to gain access to a resource, majority vote can lead to low resource utilization
  - ***No one can achieve majority vote to obtain access to the shared resource***
  - *Mimics elections where with too many candidates, where no one candidate can get >50% of the total vote*
- Problem Solution detailed in [Lin et al. 2014]

March 2, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L17.62

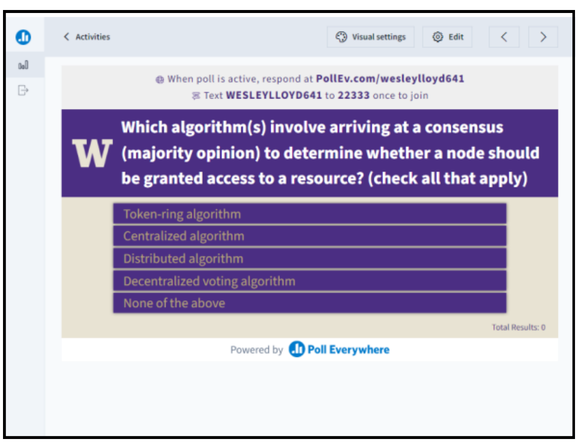
62



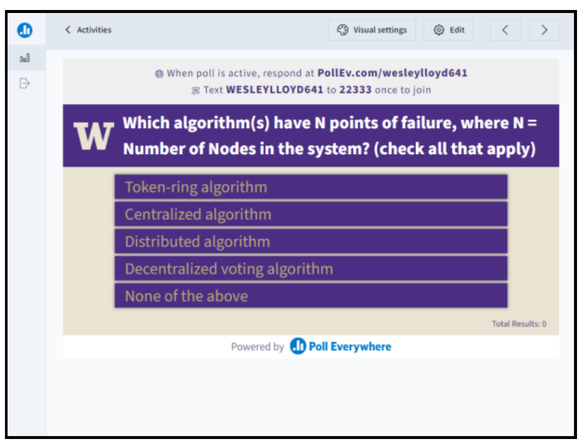
63



64



65



66

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS REVIEW

- Which algorithm offers the best scalability to support distributed mutual exclusion in a large distributed system?
- (A) Token-ring algorithm
- (B) Centralized algorithm
- (C) Distributed algorithm
- (D) Decentralized voting algorithm

67

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS REVIEW - 2

- Which algorithm(s) involve blocking (no reply) when a resource is not available?
- (A) Token-ring algorithm
- (B) Centralized algorithm
- (C) Distributed algorithm
- (D) Decentralized voting algorithm

68

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS REVIEW - 3

- Which algorithm(s) involve arriving at a consensus (majority opinion) to determine whether a node should be granted access to a resource?
- (A) Token-ring algorithm
- (B) Centralized algorithm
- (C) Distributed algorithm
- (D) Decentralized voting algorithm

69

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS REVIEW - 4

- Which algorithm(s) have N points of failure, where N = Number of Nodes in the system?
- (A) Token-ring algorithm
- (B) Centralized algorithm
- (C) Distributed algorithm
- (D) Decentralized voting algorithm

70

QUESTIONS

71