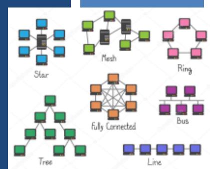
TCSS 558: APPLIED DISTRIBUTED COMPUTING

Chapter 6 – Coordination - II

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



1

OBJECTIVES – 2/29

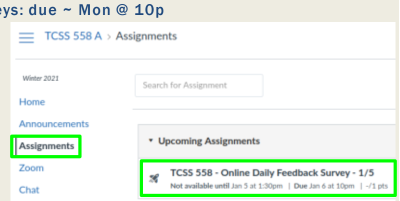
- Questions from 2/27
- Assignment 3: Replicated Key Value Store
- Chapter 6: Coordination
  - Chapter 6.2: Logical Clocks
    Vector Clocks
- Class Activity 4 – Total Ordered Multicasting
- Chapter 6: Coordination
  - Chapter 6.3: Distributed Mutual Exclusion

February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.2

2

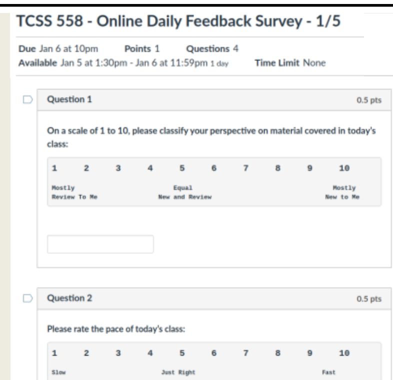
ONLINE DAILY FEEDBACK SURVEY

- 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



February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.3

3



February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.4

4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (21 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.67** (↑ - *previous 6.09*)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.52** (↑ - *previous 5.36*)

February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.5

5

FEEDBACK FROM 2/27

- ***Can you please explain again the graph mentioned in Rumor Spreading which is plotted between P_stop and s ?***

February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.6

6

RUMOR SPREADING

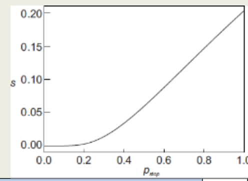
- Variant of epidemic protocols
- Provides an approach to **"stop"** message spreading
- Mimics "gossiping" in real life

- **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_{stop}, let's say 20% . . . (or 0.20)

7

RUMOR SPREADING - 2

- p_{stop}, is the probability node will stop spreading once contacting a node that already has the message
- Rumor spreading does not guarantee all nodes will be updated
- 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 < 20% with high p_{stop}
- Susceptible nodes (s) vs. probability of stopping →

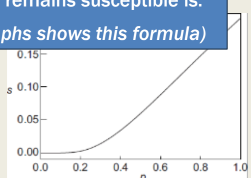


8

- Once P finds a node Q that already has the message X, P begins evaluating whether it should stop spreading message X
- P decides randomly when to stop spreading the message X
- With p_{stop},=.20, the odds are 1 in 5 that P will stop
- *On average*, after 5 attempts, P will stop trying to spread the message X
- The number of nodes that remains susceptible is: $s=e^{1(\frac{1}{pstop}+1)(1-s)}$ *(this graphs shows this formula)*

message

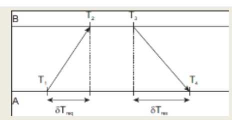
- Fraction of nodes not updated remains < 20% with high p_{stop}
- Susceptible nodes (s) vs. probability of stopping →



9

REVIEW: NTP EXAMPLE

- Time server B

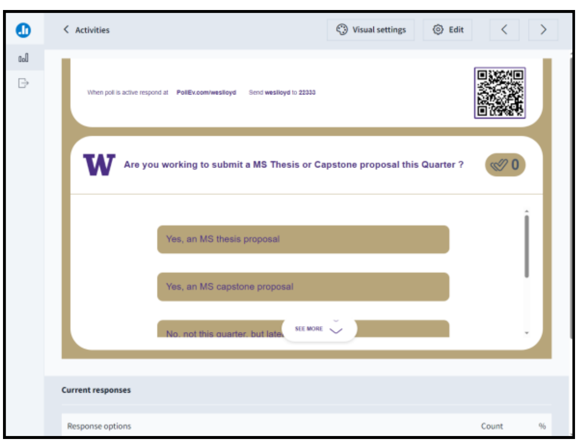


θ = clock offset
δ = propagation delay

Client A

- Assume: δT_{req}= δT_{res} *(request latency equals response latency)*
- 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 propagation delay between A and B?**
- **What is the clock offset between A and B?**

10



11

OBJECTIVES – 2/29

- Questions from 2/27
- **Assignment 3: 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

12

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

13

ASSIGNMENT 3

- **Sunday March 10th**
- Goal: Replicated Key Value Store
- Team signup to be posted on Canvas under 'People'
- Build off of Assignment 2 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
- REQUIREMENT: 'store' command needs to output 1 key-value pair per line using ASCII text (no binary)

14

OBJECTIVES – 2/29

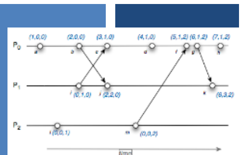
- Questions from 2/27
- Assignment 3: Replicated Key Value Store
- Chapter 6: Coordination
  - **Chapter 6.2: Logical Clocks**
    Vector Clocks
- Class Activity 4 – Total Ordered Multicasting
- Chapter 6: Coordination
  - Chapter 6.3: Distributed Mutual Exclusion

15

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)*

16

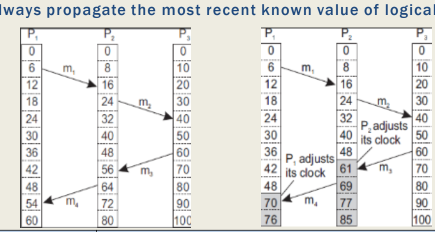


CH. 6.2: LOGICAL CLOCKS

17

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

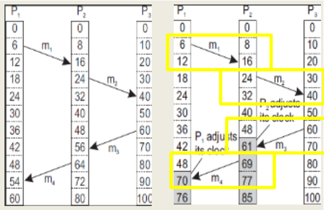


18

LOGICAL CLOCKS

- **Events**:

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
69: P2 sends m4 to P1
54: P1 receives m4
70: P1 clock reset=70



19

LAMPORT LOGICAL CLOCKS - IMPLEMENTATION

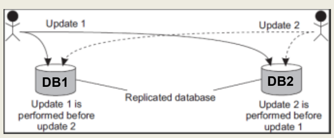
- Negative values not possible
- When a message is received, and the local clock is before the timestamp when then message was sent, the local clock is updated to message_sent_time + 1

1. Clock is incremented before an event: (*sending-a-message, receiving-a-message, some-other-internal-event*)
   Pi increments Ci: Ci ← Ci + 1
2. When Pi send msg m to Pj, m's timestamp is set to Ci
3. When Pj receives msg m, Pj adjusts its local clock
   Cj ← max{Cj, timestamp(m)}
4. Ties broken by considering Proc ID: i<j; <40,i> < <40,j>
   Both Lamport clocks are = 40
   The winner has a higher alphanumeric Process ID
   J (winner) is greater than i, alphabetically

20

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

21

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

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)

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

23

TOTAL-ORDERED MULTICASTING EXAMPLE

Total Ordered Multicasting
Logical Clocks with Acknowledgements

messages:
m_1 (10, P_1, add $100)
m_2 (12, P_2, add 1% interest)
↑
timestamps

Two processes with collocated DB replicas:

P_1/DB_1 →

P_2/DB_2 →

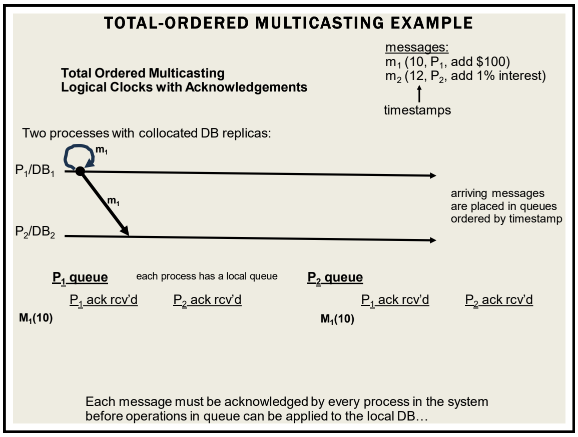
arriving messages are placed in queues ordered by timestamp

P_1 queue — each process has a local queue — **P_2 queue**

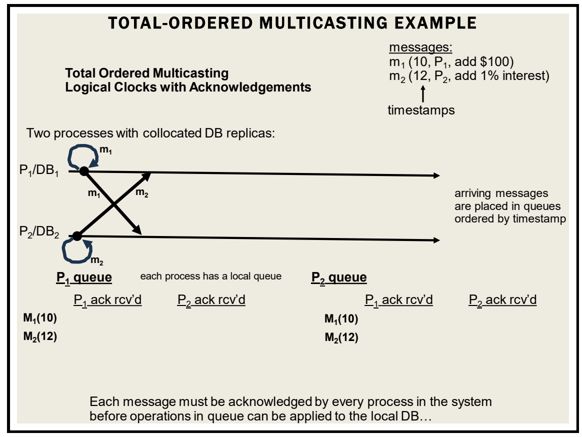
P_1 ack rcv'd | P_2 ack rcv'd | P_1 ack rcv'd | P_2 ack rcv'd

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

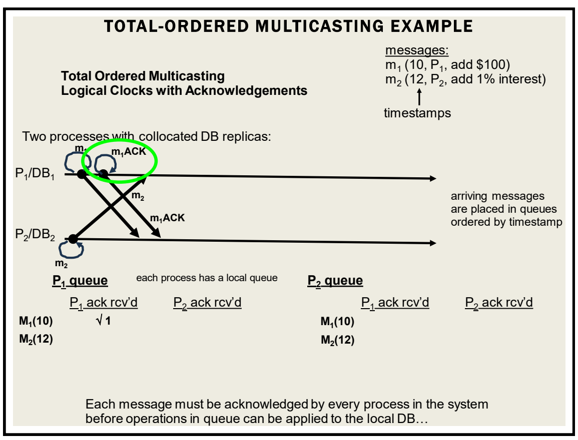
24



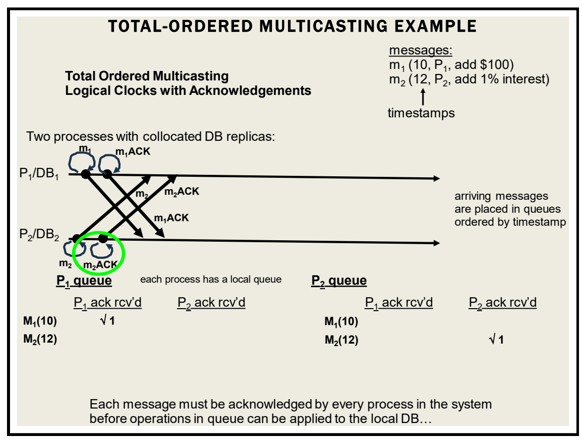
25



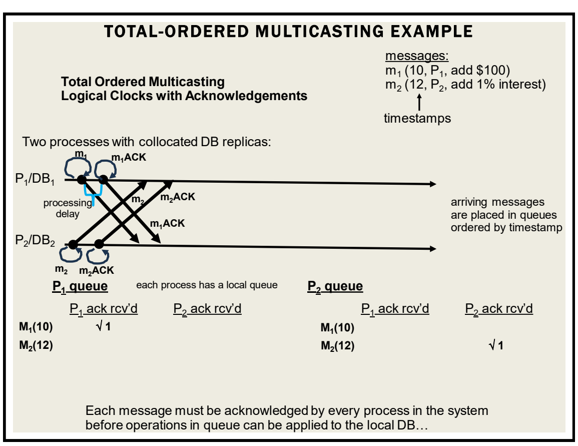
26



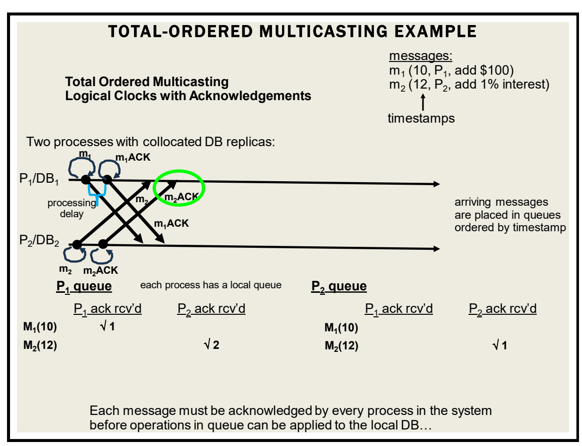
27



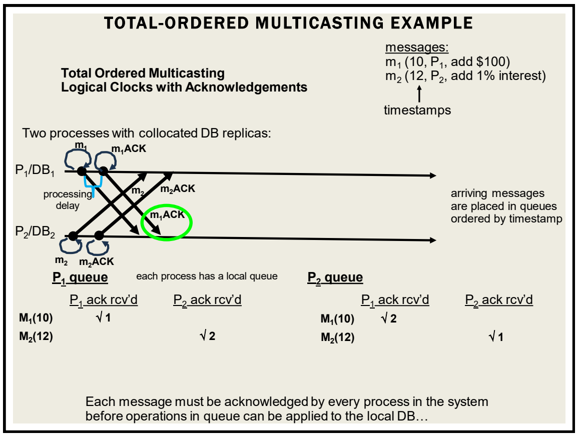
28



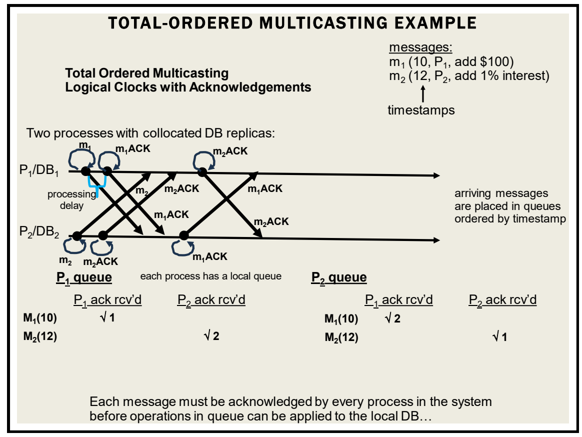
29



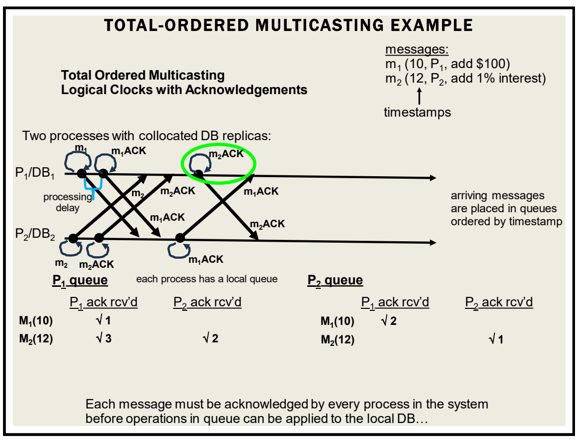
30



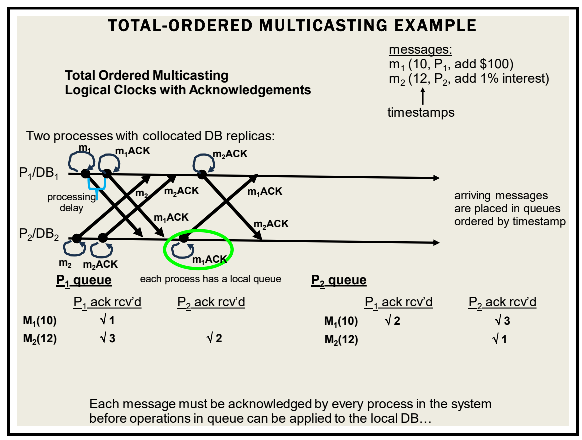
31



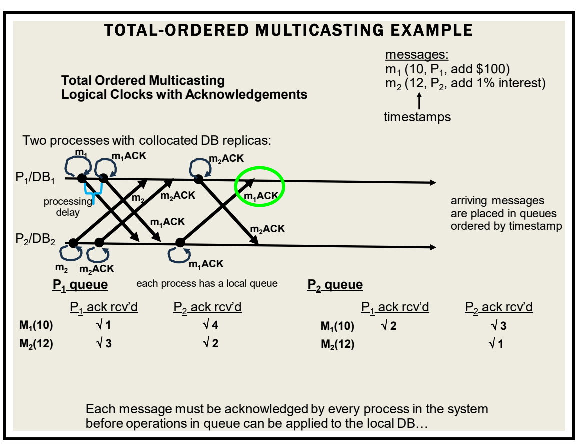
32



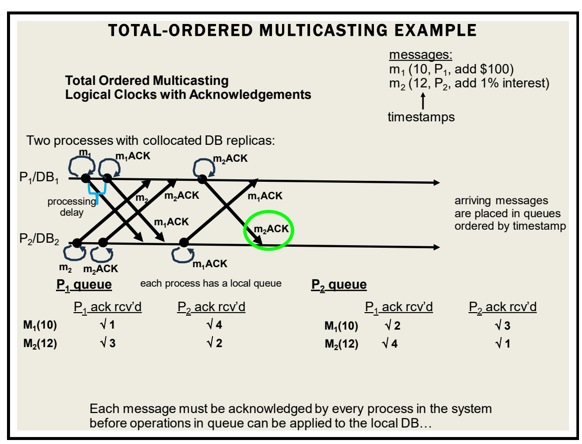
33



34



35



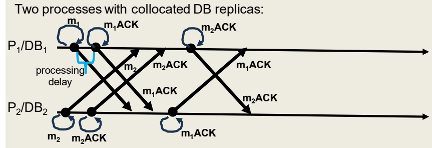
36

TOTAL-ORDERED MULTICASTING EXAMPLE

Total Ordered Multicasting
Logical Clocks with Acknowledgements

messages:
m_1 (10, P_1, add $100)
m_2 (12, P_2, add 1% interest)
↑
timestamps

Two processes with collocated DB replicas:



arriving messages are placed in queues ordered by timestamp

P_1 queue — each process has a local queue — **P_2 queue**

	P_1 ack rcv'd	P_2 ack rcv'd
M_1(10)	√1	√4
M_2(12)	√3	√2

	P_1 ack rcv'd	P_2 ack rcv'd
M_1(10)	√2	√3
M_2(12)	√4	√1

What is the final account balance?

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

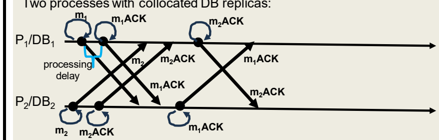
37

TOTAL-ORDERED MULTICASTING EXAMPLE

Total Ordered Multicasting
Logical Clocks with Acknowledgements

messages:
m_1 (10, P_1, add $100)
m_2 (12, P_2, add 1% interest)
↑
timestamps

Two processes with collocated DB replicas:



arriving messages are placed in queues ordered by timestamp

P_1 queue — each process has a local queue — **P_2 queue**

	P_1 ack rcv'd	P_2 ack rcv'd
M_1(10)	√1	√4
M_2(12)	√3	√2

$1,000 + $100 = $1,100
$1,100 * 1% = $1,111

	P_1 ack rcv'd	P_2 ack rcv'd
M_1(10)	√2	√3
M_2(12)	√4	√1

$1,000 + $100 = $1,100
$1,100 * 1% = $1,111

Messages are processed in timestamp order
Messages aren't processed until all acknowledgements are received

38

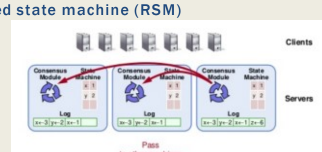
TOTAL-ORDERED MULTICASTING - 2

- 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

39

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)



40

OBJECTIVES – 2/29

- Questions from 2/27
- Assignment 3: 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

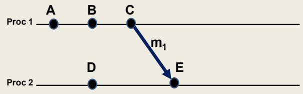
41

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? …

42

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.

43

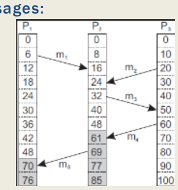
CAUSALITY - 2

- **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…

44

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?***

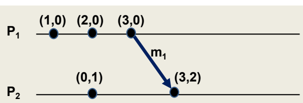
45

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

46

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)

47

VECTOR CLOCKS - 3

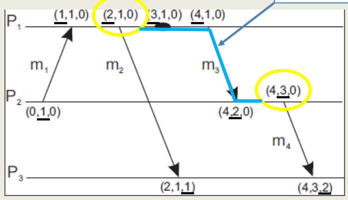
- Pj knows the # of events at Pi based on the timestamps of the received message
- Pj learns how many events have occurred at other processes based on timestamps in the vector
- These events ***"may be causally dependent"***
- **In other words:** they may have been necessary for the message(s) to be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

48

VECTOR CLOCKS EXAMPLE

- Local clock is underlined

CAUSALITY

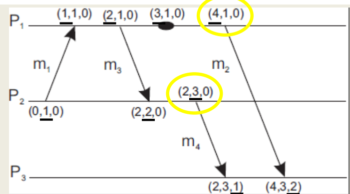


m_2	m_4	$m_2 < m_4$	$m_2 > m_4$	Conclusion
(2,1,0)	(4,3,0)	Yes	No	m2 may causally precede m4

February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.49

49

VECTOR CLOCKS EXAMPLE - 2



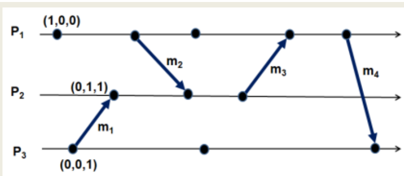
m_2	m_4	$m_2 < m_4$	$m_2 > m_4$	Conclusion
(4,1,0)	(2,3,0)	No	No	m2 and m4 may conflict

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

February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.50

50

VECTOR CLOCKS EXAMPLE - 3

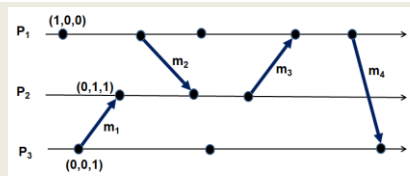


- Provide a vector clock label for unlabeled events

February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.51

51

VECTOR CLOCKS EXAMPLE - 4

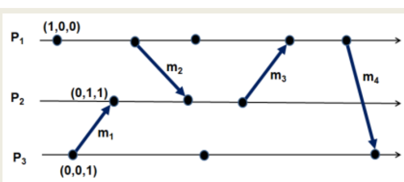


- TRUE/FALSE:
- The sending of message m_3 is causally dependent on the sending of message m_1.
- The sending of message m_2 is causally dependent on the sending of message m_1.

February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.52

52

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.

February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.53

53

WE WILL RETURN AT 2:40 PM

54

OBJECTIVES – 2/29

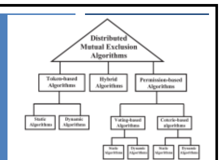
- Questions from 2/27
- Assignment 3: Replicated Key Value Store
- Chapter 6: Coordination
  - Chapter 6.2: Logical Clocks
    Vector Clocks
- **Class Activity 4 – Total Ordered Multicasting**
- Chapter 6: Coordination
  - Chapter 6.3: Distributed Mutual Exclusion

55

OBJECTIVES – 2/29

- Questions from 2/27
- Assignment 3: 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**

56



CH. 6.3: DISTRIBUTED MUTUAL EXCLUSION

57

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.)

58

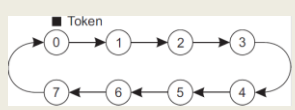
TOKEN-BASED ALGORITHMS

- 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

59

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

60

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

61

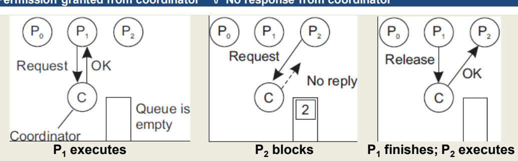
DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS - 3

- **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"***

62

CENTRALIZED MUTUAL EXCLUSION

Permission granted from coordinator ∨ 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)

63

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

64

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

65

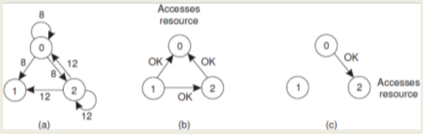
DISTRIBUTED ALGORITHM - 2

- 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

66

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)

67

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

68

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

69

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

70

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 . . .***

71

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

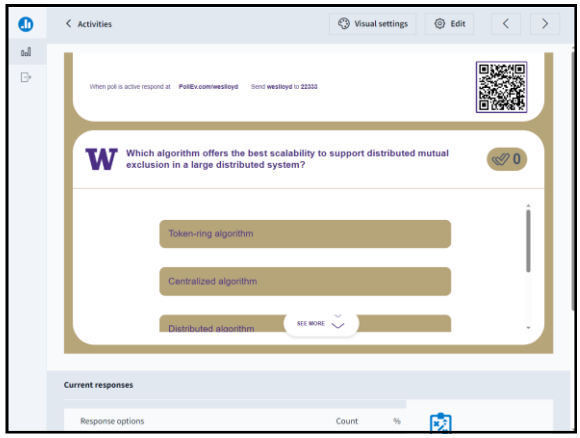
72

DECENTRALIZED ALGORITHM - 4

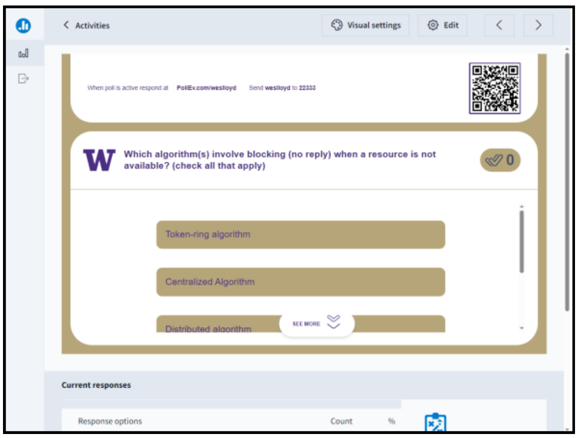
- **Back-off Polling Approach for *permission-denied*:**
- If permission to access a resource is denied via majority vote, process can poll to gain access again with a ***random*** delay (***known as back-off***)
- Node waits for a random amount, retries...
- If too many nodes compete to gain access 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]

February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.73

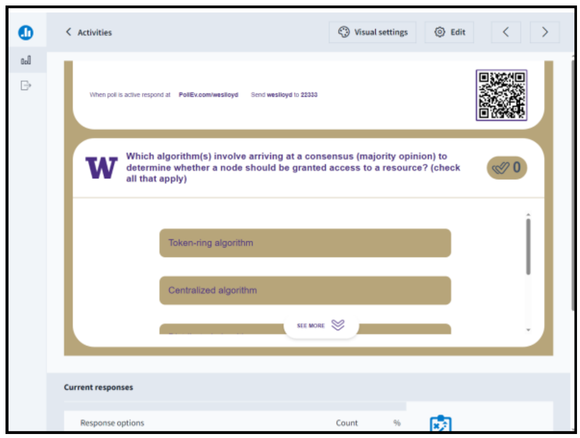
73



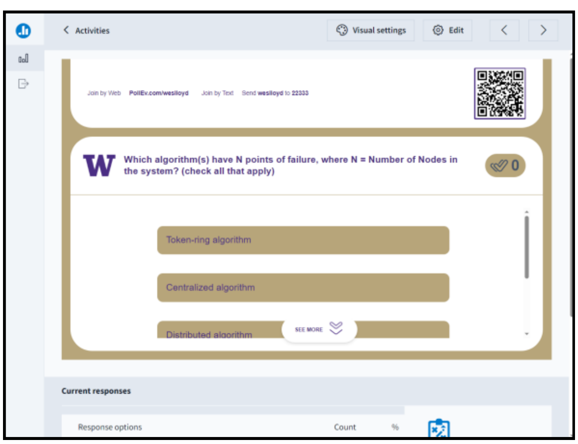
74



75



76



77

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

February 29, 2024 | TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma | L16.78

78

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

79

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

80

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

81

QUESTIONS

82