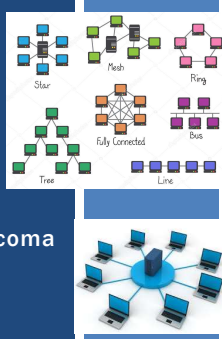


## TCSS 558: APPLIED DISTRIBUTED COMPUTING

### Coordination

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



## OBJECTIVES

- Assignment #2 Questions
- Feedback from 11/21
- Assignment #3 / Final Exam
  
- Ch. 6 - Coordination
  - Vector clocks
  - Distributed mutual exclusion
- Raft Consensus Algorithm

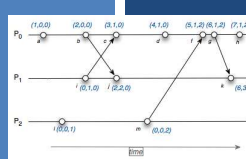
November 28, 2017    TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma    L16.2

## CHAPTER 6 - COORDINATION

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

November 28, 2017    TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma    L16.3

## CH. 6.2: LOGICAL CLOCKS



November 28, 2017    TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma    L16.4

## LOGICAL CLOCKS - 4

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

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

P<sub>1</sub> adjusts its clock to 61  
 P<sub>2</sub> adjusts its clock to 69  
 P<sub>3</sub> adjusts its clock to 77

November 28, 2017    TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma    L16.5

## LOGICAL CLOCKS

- **Events:**

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

P<sub>1</sub> adjusts its clock to 61  
 P<sub>2</sub> adjusts its clock to 69  
 P<sub>3</sub> adjusts its clock to 77

- 6: P<sub>1</sub> send m<sub>1</sub> to P<sub>2</sub>
- 16: P<sub>2</sub> receives m<sub>1</sub>
- 24: P<sub>2</sub> sends m<sub>2</sub> to P<sub>3</sub>
- 40: P<sub>3</sub> receives m<sub>2</sub>
- 60: P<sub>3</sub> sends m<sub>3</sub> to P<sub>2</sub>
- 56: P<sub>2</sub> receives m<sub>3</sub>
- 56: P<sub>2</sub> clock reset=61
- 64: P<sub>2</sub> sends m<sub>4</sub> to P<sub>1</sub>
- 54: P<sub>1</sub> receives m<sub>4</sub>
- 70: P<sub>1</sub> clock reset=70

November 28, 2017    TCSS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma    L16.6

### TOTAL-ORDERED MULTICASTING

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

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

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L16.7

### TOTAL-ORDERED MULTICASTING EXAMPLE

Total Ordered Multicasting  
 Logical clocks with Acknowledgements

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L16.8

### 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)

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L16.9

### VECTOR CLOCKS

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

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L16.10

### WHAT IS CAUSALITY?

- Consider the messages:

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	61	70
48	69	80
70	77	90
76	85	100

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

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L16.11

### 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

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017]  
 Institute of Technology, University of Washington - Tacoma L16.12

### VECTOR CLOCKS - 2

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

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.13

### VECTOR CLOCKS - 3

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

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.14

### VECTOR CLOCKS EXAMPLE

Local clock is underlined

**CAUSALITY**

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

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.15

### VECTOR CLOCKS EXAMPLE - 2

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

- $P_3$  can't determine if  $m_4$  may be causally dependent on  $m_2$
- Is  $m_4$  causally dependent on  $m_3$  ?**

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.16

### VECTOR CLOCKS - 4

- 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
- Vector clocks can help us suggest possible causality
- We never know for sure...

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.17

## CH. 6.3: DISTRIBUTED MUTUAL EXCLUSION

L16.18

## DISTRIBUTED MUTUAL EXCLUSION

- Coordinating access among distributed processes to a shared resource requires **Distributed Mutual Exclusion**
- **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**

November 28, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
Institute of Technology, University of Washington - Tacoma
L16.19

## 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

November 28, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
Institute of Technology, University of Washington - Tacoma
L16.20

## 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 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

November 28, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
Institute of Technology, University of Washington - Tacoma
L16.21

## DISTRIBUTED MUTUAL EXCLUSION - 2

- **Permission-based algorithms**
- Processes must require permission from other processes before first acquiring access to the resource
- **Centralized algorithm**
- Elect a single leader node to coordinate access to shared resource(s)
- Manage mutual exclusion on a distributed system similar to how it mutual exclusion is managed for a single system
- Nodes must all interact with leader to obtain “**the lock**”

November 28, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
Institute of Technology, University of Washington - Tacoma
L16.22

## CENTRALIZED MUTUAL EXCLUSION

Permission granted from coordinator    No response from coordinator

**P<sub>1</sub> executes**

**P<sub>2</sub> blocks**

**P<sub>1</sub> finishes; P<sub>2</sub> executes**

- 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 granted permission fairly using FIFO queue
- Just three messages: (request, grant, release)

November 28, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
Institute of Technology, University of Washington - Tacoma
L16.23

## CENTRALIZED MUTUAL EXCLUSION - 2

- **Issues**
- Coordinator is a single point of failure
- Processes can't distinguish dead coordinator from “permission denied”
  - 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

November 28, 2017
TCCS558: Applied Distributed Computing [Fall 2017]  
Institute of Technology, University of Washington - Tacoma
L16.24

### 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

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.25

### 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. Perform a timestamp comparison (*If node is waiting to access the resource*), then:
    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

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.26

### DISTRIBUTED ALGORITHM - 3

- If Node 0 and Node 2 simultaneously request access
- Node 0's time stamp is lower (8) than Node 2 (12)
- Node 1 and Node 2 grant Node 0 access
- Notice that Node 1 also grants Node 2 permission

(a) (b) (c)

- **In case of conflict, lowest timestamp wins!**

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.27

### CHALLENGES WITH DISTRIBUTED ALGORITHM

- **Problem:** Algorithm has N points of failure !
- Where N = Number of Nodes in the system
- **Problem:** When node is accessing the resource, it does not respond
  - Lack of response can be confused with **failure**
  - **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

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.28

### 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
  - *Presumably any one node locking the resource prevents agreement*
- Distributed algorithm for mutual exclusion works best for:
  - Small groups of processes
  - When memberships rarely change

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.29

# QUESTIONS

November 28, 2017 TCCS558: Applied Distributed Computing [Fall 2017] Institute of Technology, University of Washington - Tacoma L16.30

