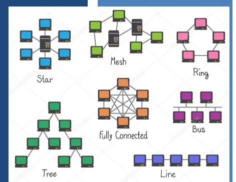
TCSS 558: APPLIED DISTRIBUTED COMPUTING

Chapter 6 - Coordination

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



1

OBJECTIVES

- Feedback from 3/5
- Assignment 2 - Questions
- Thursday: Chapter 6.4: Election Algorithms
- Thursday: Final Exam Review

2

MATERIAL / PACE

- Please classify your perspective on material covered in today's class:
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.9 (↓)**
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.9 (↓)**

3

FEEDBACK FROM 3/5

- Assignment #2:
  For inter-server communication for the two-phase commit protocol, can we use RMI or UDP?
- We are thinking about using multicast to communicate and get a response back for inter-server communication. We aren't sure how to receive a response back from multicast.
- Or we were thinking about using RMI to remotely invoke a method that checks current operations done on the other servers.
- Any suggestions?

4

FEEDBACK - 2

- Assignment #2:
  For inter-server communication for the two-phase commit protocol, can we use RMI or UDP?
- The idea is that the leader has a for-loop that iterates through the nodes in the membership list and establishes a TCP connection with each one at a time.
- An optimization is to use multiple threads to perform these TCP connections in parallel.
- *There is no extra credit for this.*
- Don't use UDP. It's not reliable, and we need the two-phase commit messages to be reliably exchanged.
- RMI could work, but the idea in assignment #2 is to use TCP for the two-phase commit protocol.

5

FEEDBACK - 3

- Assignment #2:
- ***We have a multi-threaded implementation, but if multiple nodes simultaneously receive requests then multiple nodes are acting as a leader. Each leader would try to open a socket with the other nodes in the network, and port in use error will be thrown.***
- ***How do we coordinate this?***
- If there is an error trying to establish the TCP connection, then the leader should retry.

6

FEEDBACK - 4

- Assignment #2:
- ***I saw you said there are no extra points for multi-threading TCP connections between the servers. There are 10 points in the assignment description for "server threads". Is this just meaning that the 10 points are if each server processes their client requests with threads then?***
- That's correct!
- Multi-threading the leader conversations with the nodes is a nice-to-have from an efficiency standpoint,
- but it doesn't prevent the two-phase commit from working correctly.

7

FEEDBACK - 5

- How will dynamic membership tracking approaches be tested for Assignment 2?
- A multi-node system will be set up, for example, with 3 nodes.
- A few transactions will be performed
  - put a 123
  - put b 456
- A new node will be added or deleted
- A few transactions will be performed
  - put c 789
  - put d aaa
- The new node should have keys c,d but not a,b
- Deletes nodes should no longer be accessible

8

FEEDBACK - 6

- Troubleshooting UDP Broadcast on Docker
  - Try using local host
  - Broadcast to the local subnet
  - Use the first three numbers of the IP, replace the last with "0"
  - Example: Docker container IP 172.17.0.2, broadcast IP "**172.17.0.0**"
- Does a new node joining the system need to learn the set of keys from existing nodes?
  - No, this is out of the scope of assignment 2
- Is there extra credit for implementing Docker swarm to have an overlay network for assignment 2?
  - Docker swarm instructions are provided for those interested in exploring configuration of a Docker swarm cluster to test their key-value store with nodes spanning multiple cloud VMs.
  - This involves essentially configuration. There is no extra credit.

9

SHORT-HAND-CODES FOR MEMBERSHIP TRACKING APPROACHES

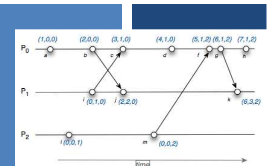
- Include readme.txt or doc file with instructions in submission
- Must document membership tracking method
- **S-1:** Static file membership tracking only = 0 pts
- **T-1:** TCP membership tracking only = +5 pts (*should be dynamic once servers point to membership server*)
- **U-1:** UDP membership 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*)

10

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)*

11



CH. 6.2: LOGICAL CLOCKS

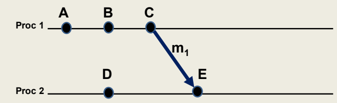
12

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? …

13

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.

14

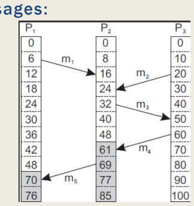
CAUSALITY - 2

- **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…

15

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?***

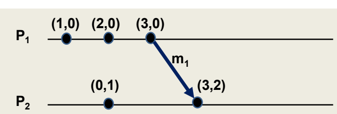
16

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

17

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)

18

VECTOR CLOCKS - 3

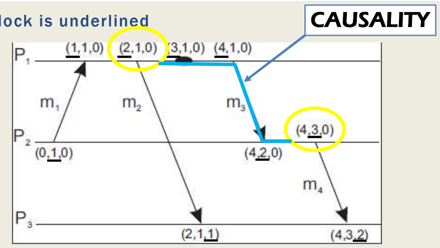
- Pj knows the # of events at Pi based on the timestamps of the received message
- Pj learns how many events have occurred at other processes based on timestamps in the vector
- These events ***"may be causally dependent"***
- **In other words:** they may have been necessary for the message(s) to be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

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.19

19

VECTOR CLOCKS EXAMPLE

- Local clock is underlined

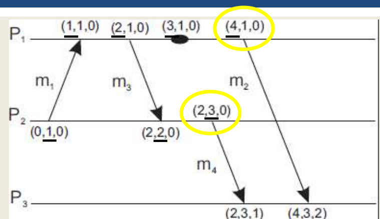


m_2	m_4	$m_2 < m_4$	$m_2 > m_4$	Conclusion
(2,1,0)	(4,3,0)	Yes	No	m2 may causally precede m4

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.20

20

VECTOR CLOCKS EXAMPLE - 2



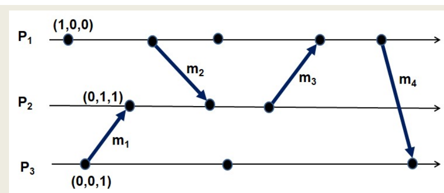
m_2	m_4	$m_2 < m_4$	$m_2 > m_4$	Conclusion
(4,1,0)	(2,3,0)	No	No	m2 and m4 may conflict

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

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.21

21

VECTOR CLOCKS EXAMPLE - 3

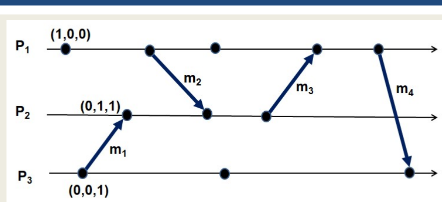


- Provide a vector clock label for unlabeled events

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.22

22

VECTOR CLOCKS EXAMPLE - 4

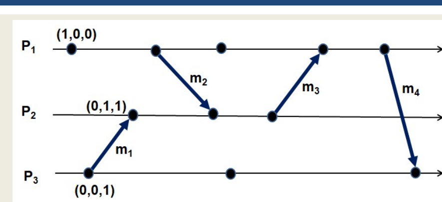


- TRUE/FALSE:
- The sending of message m_3 is causally dependent on the sending of message m_1.
- The sending of message m_2 is causally dependent on the sending of message m_1.

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.23

23

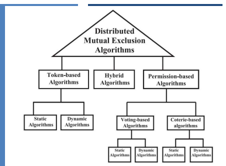
VECTOR CLOCKS EXAMPLE - 5



- 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.

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.24

24



CH. 6.3: DISTRIBUTED MUTUAL EXCLUSION

25

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.)

26

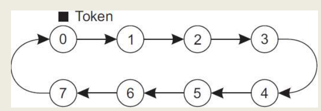
TOKEN-BASED ALGORITHMS

- 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

27

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

28

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

29

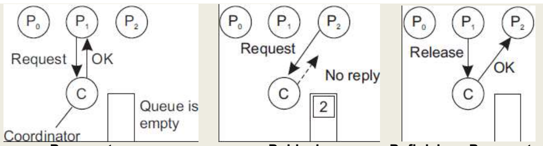
DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS - 3

- **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 it mutual exclusion is managed for a single system
- Nodes must all interact with leader to obtain ***"the lock"***

30

CENTRALIZED MUTUAL EXCLUSION

Permission granted from coordinator ∨ No response from coordinator



P_1 executes P_2 blocks P_1 finishes; P_2 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 (OK), release)

31

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

32

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

33

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

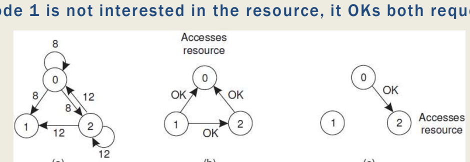
- Nodes sit back and wait for all nodes to grant permission

- Requirement: every node must know the entire membership list of the distributed system

34

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 (1@) in favor of node 0 (8)

35

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

36

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*
  - *If one node gets majority of acknowledges no other can*
  - *Requires every node to know size of system (# of nodes)*
- Distributed algorithm for mutual exclusion works best for:
  - Small groups of processes
  - When memberships rarely change

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.37

37

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

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.38

38

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 . . .***

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.39

39

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 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.40

40

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 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.41

41

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

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.42

42

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS REVIEW - 2

- Which algorithm(s) involve blocking when a resource is not available?
- (A) Token-ring algorithm
- (B) Centralized algorithm
- (C) Distributed algorithm
- (D) Decentralized voting algorithm

43

DISTRIBUTED MUTUAL EXCLUSION ALGORITHMS REVIEW - 3

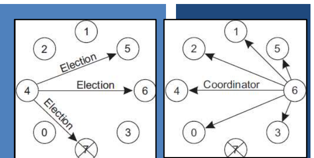
- Which algorithm(s) involve arriving at a consensus to determine whether a node should be granted access to a resource?
- (A) Token-ring algorithm
- (B) Centralized algorithm
- (C) Distributed algorithm
- (D) Decentralized voting algorithm

44

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

45



CH. 6.4: ELECTION ALGORITHMS

46

ELECTION ALGORITHMS

- Many distributed systems require one process to act as a coordinator, initiator, or provide some special role
- Generally any node (or process) can take on the role
  - In some situations there are special requirements
  - <u>Resource requirements</u>: compute power, network capacity
  - <u>Data</u>: access to certain data/information
- Assumption:
  - Every node has access to a “node directory”
  - Process/node ID, IP address, port, etc.
  - Node directory may not know “current” node availability
- Goal of election: at conclusion all nodes agree on a coordinator

47

ELECTION ALGORITHMS

- Consider a distributed system with N processes (*or nodes*)
- Every process has an identifier id(P)
- Election algorithms attempt to locate the highest numbered process to designate as coordinator
- **<u>Algorithms:</u>**
- Bully algorithm
- Ring algorithm
- Elections in wireless environments
- Elections in large-scale systems

48

BULLY ALGORITHM

- When ***any*** process notices the coordinator is no longer responding to requests, it initiates an election
- Process P_k initiates an election as follows:
  1. P_k sends an ELECTION message to all processes with higher process IDs (P_{k+1}, P_{k+2}, … P_{N-1})
  2. If no one responds, P_k wins the election and becomes coordinator
  3. If a "higher-up" process answers (P_{k+n}), it will take over and run the election. P_k will quit sending ELECTION messages.
- When the higher numbered process receives an ELECTION message from a lower-numbered colleague, it responds with "OK", indicating it's alive, and it takes over the election.

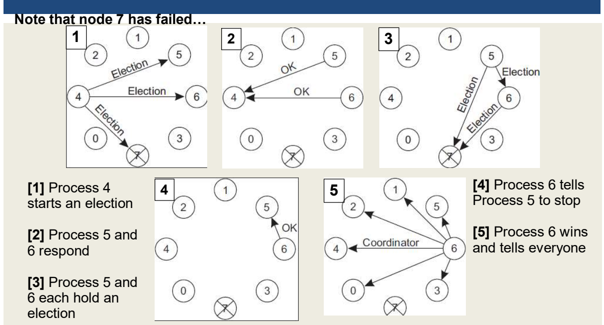
49

BULLY ALGORITHM - 2

- The higher numbered process then holds an election with ***only*** higher numbered processes (nodes).
- Eventually ***all*** processes give up except one, and the remaining process becomes the new coordinator.
- The coordinator announces victory by sending all processes a message stating it is starting as the coordinator.
- If a higher numbered node that was previously down comes back up, it holds an election, and ultimately takes over the coordinator role.
- The process with the *"biggest"* ID in town always wins.
- Hence the name, **bully algorithm**

50

BULLY ALGORITHM - 3



51

BULLY ALGORITHM - 4

- Every node knows who is participating in the distributed system
  - Each node has a group membership directory
- First process to notice the leader is offline launches a new election
- GOAL: Find the highest number node that is running
  - Loop over the nodes until the highest numbered node is found
  - May require multiple election rounds
- Highest numbered node is always the ***"BULLY"***

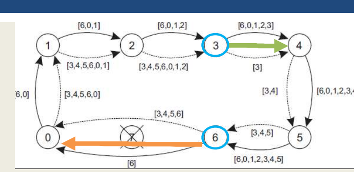
52

RING ALGORITHM

- Election algorithm based on a network of nodes in logical ring
- *Does not use a token*
- Any process (P_k) starts the election by noticing the coordinator is not functioning
  1. P_k builds an **election message**, and sends to its successor in the ring
     - If successor is down, successor is skipped
     - Skips continue until a running process is found
  2. When the **election message** is passed around, each node adds its ID to a *separate* **active node list**
  3. When **election message** returns to P_k, P_k recognizes its own identifier in the **active node list**. Message is changed to COORDINATOR and "**elected(P_k)**" message is circulated.
     - Second message announces P_k is the NEW coordinator

53

RING: MULTIPLE ELECTION EXAMPLE



- **PROBLEM:** Two nodes start election at the same time: P_3 and P_6
- P_3 sends **ELECT(P_3)** message, P_6 sends **ELECT(P_6)** message
  - P_3 and P_6 both circulate ELECTION messages at the same time
- Also circulated with ELECT message is an **active node list**
- Each node adds itself to the **active node list**
- Each node votes for the highest numbered candidate
- P_6 wins the election because it's the candidate with the **highest ID**

54

ELECTIONS WITH WIRELESS NETWORKS

- Assumptions made by traditional election algorithms not realistic for wireless environments:
  - >>> Message passing is reliable
  - >>> Topology of the network does not change
- A few protocols have been developed for elections in ad hoc wireless networks
- Vasudevan et al. [2004] solution handles failing nodes and partitioning networks.
  - Best leader can be elected, rather than just a random one

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.55

55

VASUDEVAN ET AL. WIRELESS ELECTION

1. Any node (***source***) (P) starts the **election** by sending an ELECTION message to immediate neighbors (any nodes in range)
2. Receiving node (Q) designates sender (P) as parent
3. (Q) Spreads election message to neighbors, ***but not to parent***
4. Node (R), receives message, designates (Q) as parent, and spreads ELECTION message to neighbors, ***but not to parent***
5. Neighbors that have already selected a parent immediately respond to R.
   - If ***all*** neighbors already have a parent, R is a leaf-node and will report back to Q quickly.
   - When reporting back to Q, R includes metadata regarding battery life and resource capacity
6. Q eventually acknowledges the ELECTION message sent by P, and also indicates the most eligible node (based on battery & resource capacity)

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.56

56

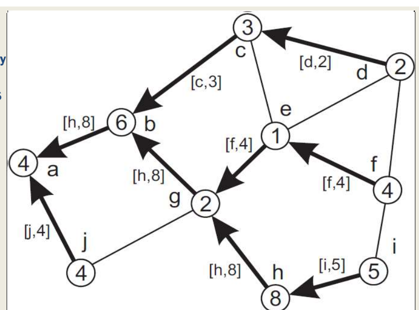
WIRELESS ELECTION - 2 SOURCE NODE: [A]

Node [A] initiates election: find the highest capacity

Election messages propagated to all nodes

Each node reports to its parent node with best capacity

Node A then facilitates Node H becoming leader



March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.57

57

WIRELESS ELECTION - 3

- When multiple elections are initiated, nodes only join one
- Source node tags its ELECTION message with unique identifier, to uniquely identify the election.
- With minor adjustments protocol can operate when the network partitions, and when nodes join and leave

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.58

58

ELECTIONS FOR LARGE-SCALE SYSTEMS

- Large systems often require several nodes to serve as coordinators/leaders
- These nodes are considered ***"super peers"***
- ***Super peers*** must meet operational requirements:

1. Network latency from normal nodes to ***super peers*** must be low
2. ***Super peers*** should be evenly distributed across the overlay network (ensures proper load balancing, availability)
3. Must maintain set ratio of ***super peers*** to normal nodes
4. ***Super peers*** must not serve ***too many*** normal nodes

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.59

59

ELECTIONS FOR DHT BASED SYSTEMS

- DHT-based systems use a bit-string to identify nodes
- **Basic Idea**: Reserve fraction of ID space for super peers
- Reserve first $\log_2(N)$ bits for super-peer IDs
- m=number of bits of the identifier
- k=# of nodes each node is responsible for (Chord system)
- **Example:**
- For a system with m=8 bit identifier, and k=3 keys per node
- Required number of super peers is $2^{(k-m)} \cdot N$, where N is the number of nodes
  - In this case N=32
  - **Only 1 super peer is required for every 32 nodes**

March 10, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L18.60

60

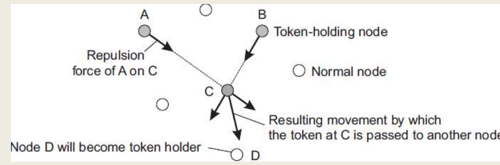
SUPER PEERS IN AN M-DIMENSIONAL SPACE

- Given an overlay network, the idea is to position superpeers throughout the network so they are evenly disbursed
- **Use tokens:**
- Give N tokens to N randomly chosen nodes
- No node can hold more than (1) token
- Tokens are "repelling force". Other tokens move away
- All tokens exert the same repelling force
- This automates token distribution across an overlay network

61

OVERLAY TOKEN DISTRIBUTION

- Gossping protocol is used to disseminate token location and force information across the network
- If forces acting on a node with a token exceed a **threshold**, token is moved away
- Once nodes hold token for awhile they become superpeers



62

QUESTIONS

63

RESEARCH DIRECTIONS

64

CLOUD AND DISTRIBUTED SYSTEMS RESEARCH GROUP

- Meetings on Wednesdays from 12 (12:30) to 1:30pm
- MDS 202
- *MDS is just south of Cherry Parkes*

The CDS group collaborates on research projects spanning Serverless computing (FaaS), Containerization, Infrastructure-as-a-Service (IaaS) cloud, virtualization, infrastructure management, and performance and cost modeling of application deployments. Our research aims to demystify the myriad of options to guide software developers, engineers, scientists, and practitioners to intelligently harness cloud computing to improve performance and scalability of their applications, while reducing hosting costs.

65

EXTRA SLIDES

66

DIRECTIONAL GOSSIPING

- Taking network topology into account can help
- When gossiping, nodes connected to only a few other nodes are more likely to be contacted

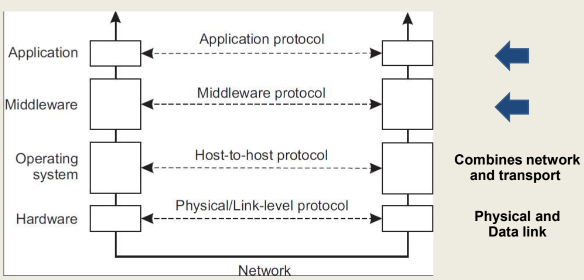
67

LANGUAGE BASED SUPPORT

- Leads to simpler application development
- Helps with providing access transparency
  - Differences in data representation, and how object is accessed
  - Inter-language parameter passing issues resolved: → *just 1 language*
- Well known example: ***Java Remote Method Invocation*** RPC equivalent embedded in Java

68

ADAPTED REFERENCE MODEL



69

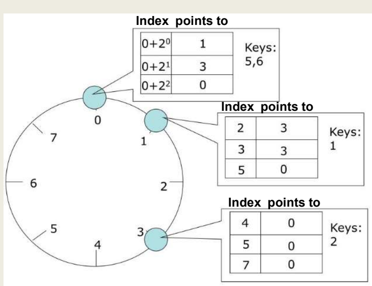
CHORD SYSTEM – FINGER TABLE

- Each node keeps maintains a finger table with m entries
  - m is the number of bits in the hash key
  - Distance of the entries increases exponentially
- Contents of each node's finger table:
  for i=0 to m-1
  finger table entry for node n:
  index: $n+2^i$ → points to: $n+2^i \bmod 2^m$
- The first entry of finger table is the node's immediate successor (an extra successor field is not needed).
- Each time a node looks up a key k, it passes the query to the closest node to k in the finger table that is not greater than k
- With finger tables, the number of nodes contacted to find a successor in an N-node network is O(log N).

70

CHORD SYSTEM – 2

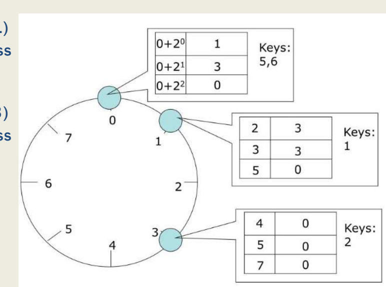
- Keys have m-bits
- m=3
- Always pass query for key k to index in the finger table that is not greater than k
- Example: key (k=7)
- Query arrives at (0)
  - 0: → (index=4, pass to 0), key 7 is adjacent



71

CHORD SYSTEM – 2

- Example (k=7)
- Query arrives at (1)
  - 1: → (index=5, pass to 0), key 7 is adjacent
- Query arrives at (3)
  - 1: → (index=7, pass to 0), key 7 is adjacent
- Example (k=6)



72