Logistics

• HW2 is due tomorrow.

• HW3 has been released.
  • Due on March 7th, Monday.
  • You can only use up to 24 hours of the total 48 hours grace period provided for late submissions.
Today’s agenda

• Wrap up Mutual Exclusion
  • Extending Maekawa’s algorithm to break deadlocks.

• Leader Election
  • Chapter 15.3
Recap: Mutual exclusion in distributed systems

- Classical algorithms for mutual exclusion in distributed systems.
  - Central server algorithm
  - Ring-based algorithm
  - Ricart-Agrawala Algorithm
  - Maekawa Algorithm
Maekawa Algorithm: Actions

- state = \textit{Released}, voted = false
- enter() at process \( P_i \):
  - state = \textit{Wanted}
  - Multicast \textit{Request} message to all processes in \( V_i \)
  - Wait for \textit{Reply (vote)} messages from all processes in \( V_i \) (including vote from self)
  - state = \textit{Held}
- exit() at process \( P_i \):
  - state = \textit{Released}
  - Multicast \textit{Release} to all processes in \( V_i \)
Maekawa Algorithm: Actions (contd.)

• When Pi receives a Request from Pj:
  if (state == Held OR voted = true)
    queue Request
  else
    send Reply to Pj and set voted = true

• When Pi receives a Release from Pj:
  if (queue empty)
    voted = false
  else
    dequeue head of queue, say Pk
    Send Reply only to Pk
    voted = true
Analysis: Maekawa Algorithm

- **Safety:**
  - When a process $P_i$ receives replies from all its voting set $V_i$ members, no other process $P_j$ could have received replies from all its voting set members $V_j$.

- **Liveness**
  - Not satisfied. Can have deadlock!

- **Ordering:**
  - Not satisfied.
Breaking deadlocks

• Maekawa algorithm can be extended to break deadlocks.
• Compare Lamport timestamps before replying (like Ricart-Agrawala).
• But is that enough?
  • System of 6 processes \{0, 1, 2, 3, 4, 5\}. 0, 1, 2 want to enter critical section:
    • \(V_0 = \{0, 1, 2\}\): 0, 2 send reply to 0, but 1 sends reply to 1;
    • \(V_1 = \{1, 3, 5\}\): 1, 3 send reply to 1, but 5 sends reply to 2;
    • \(V_2 = \{2, 4, 5\}\): 4, 5 send reply to 2, but 2 sends reply to 0;
  • Suppose \((L_1, P_1) < (L_0, P_0) < (L_2, P_2)\).
  • Deadlock can still happen based on when messages are received.
    • \(P_5\) receives \(P_2\)’s request before \(P_1\)’s, and replies back to \(P_2\) first.
  • We need a way to take back the reply.
Breaking deadlocks

• Say Pi’s request has a smaller timestamp than Pj.
• If Pk receives Pj’s request after replying to Pi, send fail to Pj.
• If Pk receives Pi’s request after replying to Pj, send inquire to Pj.
• If Pj receives an inquire and at least one fail, it sends a relinquish to release locks, and deadlock breaks.
Breaking deadlocks

- System of 6 processes \{0, 1, 2, 3, 4, 5\}. 0, 1, 2 want to enter critical section:
  - \(V_0 = \{0, 1, 2\}\): 0, 2 send reply to 0, but 1 sends reply to 1;
  - \(V_1 = \{1, 3, 5\}\): 1, 3 send reply to 1, but 5 sends reply to 2;
  - \(V_2 = \{2, 4, 5\}\): 4, 5 send reply to 2, but 2 sends reply to 0;

- Suppose \((L_1, P_1) < (L_0, P_0) < (L_2, P_2)\).
- P2 will send fail to itself when it receives its own request after P0.
- P5 will send inquire to P2 when it receives P1’s request.
- P2 will send relinquish to \(V_2\). P5 and P4 will set “voted = false”. P5 will reply to P1.
- P1 can now enter CS, followed by P0, and then P2.
Mutual exclusion in distributed systems

- Classical algorithms for mutual exclusion in distributed systems.
  - Central server algorithm
    - Satisfies safety, liveness, but not ordering.
    - $O(1)$ bandwidth, and $O(1)$ client and synchronization delay.
    - Central server is scalability bottleneck.
  - Ring-based algorithm
    - Satisfies safety, liveness, but not ordering.
    - Always uses bandwidth, $O(N)$ client and synchronization delay
  - Ricart-Agrawala algorithm
    - Satisfies safety, liveness, and ordering.
    - $O(N)$ bandwidth, $O(1)$ client and synchronization delay.
  - Maekawa algorithm
    - Satisfies safety, but not liveness and ordering.
    - $O(\sqrt{N})$ bandwidth, $O(1)$ client and synchronization delay.
Today’s agenda

• Wrap up Mutual Exclusion
  • Extending Maekawa’s algorithm to break deadlocks.

• Leader Election
  • Chapter 15.3

• Goal:
  • What is leader election in distributed systems?
  • How do we elect a leader?
  • To what extent can we handle failures when electing a leader?
Why Election?

• Example: Your Bank account details are replicated at a few servers, but one of these servers is responsible for receiving all reads and writes, i.e., it is the leader among the replicas

  • What if servers disagree about who the leader is?
  • What if there are two leaders per customer?
  • What if the leader crashes?

  *Each of the above scenarios leads to inconsistency*
More motivating examples

• The root server in a group of NTP servers.
• The master in Berkeley algorithm for clock synchronization.
• In the sequencer-based algorithm for total ordering of multicasts, the “sequencer” = leader.
• The central server in the “central server algorithm” for mutual exclusion.
• Other systems that need leader election: Apache Zookeeper, Google’s Chubby.
Leader Election Problem

• In a group of processes, elect a Leader to undertake special tasks
  • And let everyone know in the group about this Leader

• What happens when a leader fails (crashes)
  • Some process detects this (using a Failure Detector!)
  • Then what?

• Focus of this lecture: Election algorithm. Its goal:
  1. Elect one leader only among the non-faulty processes
  2. All non-faulty processes agree on who is the leader
Calling for an Election

• Any process can call for an election.

• A process can call for at most one election at a time.

• Multiple processes are allowed to call an election simultaneously.
  • All of them together must yield only a single leader

• The result of an election should not depend on which process calls for it.
Election Problem, Formally

- A run of the election algorithm must always guarantee:
  - **Safety**: For all non-faulty processes $p$:
    - $p$ has elected:
      - (q: a particular non-faulty process with the best attribute value)
    - or Null
  - **Liveness**: For all election runs:
    - election run terminates
    - & for all non-faulty processes $p$: $p$'s elected is not Null

- At the end of the election protocol, the non-faulty process with the best (highest) election attribute value is elected.
  - Common attribute: leader has highest id
  - Other attribute examples: leader has highest IP address, or fastest cpu, or most disk space, or most number of files, etc.
System Model

- $N$ processes.
- Messages are eventually delivered.
- Failures may occur during the election protocol.
- Each process has a unique id.
  - Each process has a unique attribute (based on which Leader is elected).
  - If two processes have the same attribute, combine the attribute with the process id to break ties.
Classical Election Algorithms

• Ring election algorithm

• Bully algorithm
Classical Election Algorithms

- Ring election algorithm
- Bully algorithm
Ring Election Algorithm

- $N$ processes are organized in a logical ring
- All messages are sent clockwise around the ring.
Ring Election Protocol (basic version)

• When Pi start election
  • send election message with Pi’s <attr_i, i> to ring successor.
• When Pj receives message (election, <attr_x, x>) from predecessor
  • If (attr_x, x) > (attr_j, j):
    • forward message (election, <attr_x, x>) to successor
  • If (attr_x, x) < (attr_j, j)
    • send (election, <attr_j, j>) to successor
  • If (attr_x, x) = (attr_j, j) : Pj is the elected leader (why?)
    • send elected message containing Pj’s id.
• elected message forwarded along the ring until it reaches the leader.
Ring Election: Example

Goal: Elect highest id process as leader
Ring Election: Example

Initiates the election

Goal: Elect highest id process as leader
Ring Election: Example

Goal: Elect highest id process as leader

Election: 32

Initiates the election
Ring Election: Example

Election: 80

Goal: Elect highest id process as leader
Ring Election: Example

Election: 80
Initiates the election

N12
N3
N6
N32
N80
N5

Goal: Elect highest id process as leader
Ring Election: Example

Goal: Elect highest id process as leader
Ring Election: Example

Goal: Elect highest id process as leader
Ring Election: Example

Initiates the election

N12 → N3 → N6 → N32 → N80 → N5

Elected: 80

Goal: Elect highest id process as leader
Ring Election: Example

Initiates the election

Elected: 80

elected = 80

Goal: Elect highest id process as leader
Ring Election: Example

Goal: Elect highest id process as leader
Goal: Elect highest id process as leader

Ring Election: Example

Initiates the election

N12

N6

N3

N32

N80

N5

elected = 80
Ring Election Protocol (basic version)

- When Pi starts election
  - send \texttt{election} message with Pi’s \texttt{<attr}_i, i> to ring successor.
- When Pj receives message (\texttt{election}, \texttt{<attr}_x, x>) from predecessor
  - If (\texttt{attr}_x, x) > (\texttt{attr}_j, j):
    - forward message (\texttt{election}, \texttt{<attr}_x, x>) to successor
  - If (\texttt{attr}_x, x) < (\texttt{attr}_j, j)
    - send (\texttt{election}, \texttt{<attr}_j, j>) to successor
  - If (\texttt{attr}_x, x) = (\texttt{attr}_j, j): Pj is the elected leader (why?)
    - send \texttt{elected} message containing Pj’s id.
- \texttt{elected} message forwarded along the ring until it reaches the leader.

What happens when multiple processes call for an election?
Ring Election: Example

Election: 80 sent twice.
Elected: 80 also sent twice.
Ring Election Protocol [Chang & Roberts’79]

• When Pi start election
  • send election message with Pi’s <attr_i, i> to ring successor.
  • set state to participating

• When Pj receives message (election, <attr_x, x>) from predecessor
  • If (attr_x, x) > (attr_j, j):
    • forward message (election, <attr_x, x>) to successor
    • set state to participating
  • If (attr_x, x) < (attr_j, j)
    • If (not participating):
      • send (election, <attr_j, j>) to successor
      • set state to participating
  • If (attr_x, x) = (attr_j, j) : Pj is the elected leader (why?)
    • send elected message containing Pj’s id.

• elected message forwarded along the ring until it reaches the leader.
  • Set state to not participating when an elected message is received.
Ring Election: Example

Initiates the election

Election: 80 and Elected: 80
sent only once.
Performance Analysis

- Let’s assume no failures occur during the election protocol itself, and there are $N$ processes.
- Let’s also assume that only one process initiates the algorithm.
- **Bandwidth usage:** Total number of messages sent.
- **Turnaround time:** The number of serialized message transmission times between the initiation and termination of a single run of the algorithm.
Worst-case

When the initiator is the ring successor of the would-be leader.
Worst-case

- (N-1) messages for Election message to get from N6 to N80.
- N messages for Election message to circulate around ring without message being changed.
- N messages for Elected message to circulate around the ring.
- No. of messages: (3N-1)
- Turnaround time: (3N-1) message transmission times
Best-case

When the initiator is the would-be leader.
Best-case

When the initiator is the would-be leader.

No. of messages: 2N

Turnaround time: 2N message transmission times
Performance Analysis

• Let’s assume no failures occur during the election protocol itself, and there are $N$ processes.

• Let’s also assume that only one process initiates the algorithm.

• Bandwidth usage (total number of messages)
  • $O(N)$: Worst case = $3N - 1$; Best case = $2N$.

• $O(N)$ turnaround time.
Performance Analysis

• Let’s assume no failures occur during the election protocol itself, and there are $N$ processes.

• When each process initiates the algorithm?
  • $O(N)$ messages in best-case.

  • $N_1$ election messages generated at the start of the algorithm.
  • Only one survives, and completes a full round.
    • $N$-1 more messages.
  • One round for the elected message
    • $N$ messages.
  • Total: $3N-1$ messages
Performance Analysis

• Let’s assume no failures occur during the election protocol itself, and there are $N$ processes.

• When each process initiates the algorithm?
  • $O(N)$ messages in best-case.
  • $O(N^2)$ in worst-case.

• $N$ election messages generates at the starts of algorithm.
• $N - 1$ survive the next time step.
• $N - 2$ survive the next time step.
• ….
Performance Analysis

- Let’s assume no failures occur during the election protocol itself, and there are $N$ processes.

- When each process initiates the algorithm?
  - $O(N)$ messages in best-case.
  - $O(N^2)$ messages in worst-case.
  - $O(N)$ turnaround time.
Correctness

• Assuming no process fails.

• Safety:
  • Process with highest attribute elected by all nodes.

• Liveness:
  • Election completes within $3N - 1$ message transmission times.
Handling Failures

Initiates the election

Election: 80

Crash
Handling failures

• Use the failure detector.
• A process can detect failure of N80 via its own local failure detector:
  • Repair the ring.
  • Stop forwarding *Election*:80 message.
  • Start a new run of leader election.
Handling Failures

Election: 80

Initiates the election

N6

Initiates re-election

elected = 32

N12

N3

N32

N5

N80

Crash

elected = 32

elected = 32

elected = 32

elected = 32
Handling failures

• Use the failure detector.
• A process can detect failure of N80 via its own local failure detector:
  • Repair the ring.
  • Stop forwarding Election:80 message.
  • Start a new run of leader election.
• But failure detectors cannot be both complete and accurate.
  • Incomplete FD => N80’s failure might be missed.
What happens if a process failure is undetected?

Initiates the election

Election: 80

Crash
What happens if a process failure is undetected?

No "elected" message generated.

Crash

Election: 80

Algorithm does not terminate.

Liveness violated.

Initiates the election

N12

N3

N6

N32

N5

N80
Handling failures

- Use the failure detector.
- A process can detect failure of N80 via its own local failure detector:
  - Repair the ring.
  - Stop forwarding *Election:80* message.
  - Start a new run of leader election.
- But failure detectors cannot be both complete and accurate.
  - Incomplete FD => N80’s failure might be missed
    - violation of liveness.
  - Inaccurate FD => N80 mistakenly detected as failed
What can happen if an alive process is detected as failed?

Election: 80

Initiates the election
What can happen if an alive process is detected as failed?

Election: 80
What can happen if an alive process is detected as failed?

Elected: 80

elected = 80
What can happen if an alive process is detected as failed?

Elected: 80

elected = 80
What can happen if an alive process is detected as failed?

Elected: 80

Inaccurately detects N80 has failed

Initiates re-election

Safety has been violated.
Fixing for failures

• Use the failure detector.

• A process can detect failure of N80 via its own local failure detector:
  • Repair the ring.
  • Stop forwarding Election:80 message.
  • Start a new run of leader election.

• But failure detectors cannot be both complete and accurate.
  • Incomplete FD => N80’s failure might be missed
    • violation of liveness.
  • Inaccurate FD => N80 mistakenly detected as failed
    • new ring will be constructed without N80.
    • a process with lower attribute will be selected.
    • violation of safety.
Classical Election Algorithms

- Ring election algorithm
- Bully algorithm
Bully algorithm

• Faster turnaround time than ring election.

• Explicitly build in the notion of timeouts into the algorithm.

• Let's assume (for simplicity of exposition) that the attribute based on which leader is elected is the process id.

• Before discussing Bully algorithm, let’s first discuss a simpler (related) algorithm...
Multicast-based algorithm

- Start an election
  - Multicast <election, my ID> to all processes
  - If receive <agree> from all processes, then elected
    - Multicast <coordinator, my ID>
  - If receive <disagree> from any process
    - Give up election

- Receive <election, ID> from process p
  - If ID > my ID
    - Send <agree> to p (unicast)
  - If ID < my ID
    - Send <disagree> to p
    - Start election (if not already running)

- What about failures?
Multicast-based algorithm

• Start an election
  • Multicast <election, my ID> to all processes
  • If receive <agree> from all processes or timeout, then elected
    • Multicast <coordinator, my ID>
  • If receive <disagree> from any process
    • Give up election

• Receive <election, ID> from process p
  • If ID > my ID
    • Send <agree> to p (unicast)
  • If ID < my ID
    • Send <disagree> to p
    • Start election (if not already running)

• Can we improve on this?
Multicast-based algorithm

- Start an election
  - Multicast <election, my ID> to all processes
  - If receive <agree> from all processes or timeout, then elected
    - Multicast <coordinator, my ID>
  - If receive <disagree> from any process
    - Give up election

- Receive <election, ID> from process p
  - If ID > my ID
    - Send <agree> to p (unicast)
  - If ID < my ID
    - Send <disagree> to p
    - Start election (if not already running)

- Can we improve on this?
Bully Algorithm

• All processes know other process’ ids.

• Do not need to multicast election to all processes.

• Only to processes with higher id.
Bully Algorithm

• When a process wants to initiate an election
  • if it knows its id is the highest
    • it elects itself as coordinator, then sends a *Coordinator* message to all processes with lower identifiers. Election is completed.
  • else
    • it initiates an election by sending an *Election* message
    • (contd.)
Bully Algorithm (2)

• **else** it initiates an election by sending an *Election* message
  • Sends it to only processes that have a *higher id than itself*.
  • **if** receives no answer within timeout, calls itself leader and sends *Coordinator* message to all lower id processes. Election completed.
  • **if** an answer received however, then there is some non-faulty higher process => so, wait for coordinator message. If none received after another timeout, start a new election run.

• A process that receives an *Election* message replies with *disagree* message, and starts its own leader election protocol (unless it has already done so).
Bully Algorithm: Example

P2 initiates election after detecting P5’s failure.

1. P2 initiates election
2. P2 receives "replies"
3. P3 & P4 initiate election
4. P3 receives reply
5. P4 receives no reply
5. P4 announces itself

What if P4 fails after step 3?
Bully Algorithm: Example

P2 initiates election after detecting P5’s failure.

1. P2 initiates election
2. P2 receives "replies"
3. P3 & P4 initiate election
4. P3 receives reply
5. P4 receives no reply
5. P4 announces itself

What if P4 fails after step 4?
Bully Algorithm (2)

- **else** it initiates an election by sending an *Election* message
  - Sends it to only processes that have a *higher id than itself*.
  - **if** receives no answer within *timeout*, calls itself leader and sends *Coordinator* message to all lower id processes. Election completed.
  - **if** an answer received however, then there is some non-faulty higher process => so, wait for coordinator message. If none received after another *timeout*, start a new election run.

- A process that receives an *Election* message replies with *disagree* message, and starts its own leader election protocol (unless it has already done so).

*Next class: how to set timeouts and analyze Bully algorithm.*