Recap: Mutual Exclusion
• Centralized
• Ring-based
• Ricart and Agrawala’s
• Maekawa’s

Why Election?
• Example 1: sequencer for TO multicast
• Example 2: leader for mutual exclusion
• Example 3: group of NTP servers: who is the root server?

What is Election?
• In a group of processes, elect a leader to undertake special tasks.
• What happens when a leader fails (crashes)
  – Some process detects this (how?)
  – Then what?
• Focus of this lecture: election algorithms
  – 1. Elect one leader only among the non-faulty processes
  – 2. All non-faulty processes agree on who is the leader
• We’ll look at 3 algorithms

Assumptions
• Any process can call for an election.
• A process can call for at most one election at a time.
• Multiple processes can call an election simultaneously.
  – All of them together must yield a single leader only
  – The result of an election should not depend on which process calls for it.
• Messages are eventually delivered.

Problem Specification
• At the end of the election protocol, the non-faulty process with the best (highest) election attribute value is elected.
  – Attribute examples: CPU speed, load, disk space, ID
  – Must be unique
• Each process has a variable elected.
• A run (execution) of the election algorithm should ideally guarantee at the end:
  – Safety: $\forall$ non-faulty p: (p’s elected $\neq$ q; a particular non-faulty process with the best attribute value) or $\bot$
  – Liveness: $\forall$ election: (election terminates) & $\forall$ p: non-faulty process, p’s elected is eventually not $\bot$. 

CSE 486/586 Distributed Systems
Leader Election

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Recap: Mutual Exclusion
• Centralized
• Ring-based
• Ricart and Agrawala’s
• Maekawa’s

Why Election?
• Example 1: sequencer for TO multicast
• Example 2: leader for mutual exclusion
• Example 3: group of NTP servers: who is the root server?

What is Election?
• In a group of processes, elect a leader to undertake special tasks.
• What happens when a leader fails (crashes)
  – Some process detects this (how?)
  – Then what?
• Focus of this lecture: election algorithms
  – 1. Elect one leader only among the non-faulty processes
  – 2. All non-faulty processes agree on who is the leader
• We’ll look at 3 algorithms

Assumptions
• Any process can call for an election.
• A process can call for at most one election at a time.
• Multiple processes can call an election simultaneously.
  – All of them together must yield a single leader only
  – The result of an election should not depend on which process calls for it.
• Messages are eventually delivered.

Problem Specification
• At the end of the election protocol, the non-faulty process with the best (highest) election attribute value is elected.
  – Attribute examples: CPU speed, load, disk space, ID
  – Must be unique
• Each process has a variable elected.
• A run (execution) of the election algorithm should ideally guarantee at the end:
  – Safety: $\forall$ non-faulty p: (p’s elected $\neq$ q; a particular non-faulty process with the best attribute value) or $\bot$
  – Liveness: $\forall$ election: (election terminates) & $\forall$ p: non-faulty process, p’s elected is eventually not $\bot$. 

CSE 486/586 Distributed Systems
Leader Election

Steve Ko
Computer Sciences and Engineering
University at Buffalo
Algorithm 1: Ring Election [Chang & Roberts’79]

- N Processes are organized in a logical ring
  - \( p_i \) has a communication channel to \( p_{(i+1) \mod N} \)
  - All messages are sent clockwise around the ring.

- To start election
  - Send \( \text{election} \) message with my ID

- When receiving message \( (\text{election}, \text{id}) \)
  - If \( \text{id} > \text{my ID} \): forward message
  - Set state to \( \text{participating} \)
  - If \( \text{id} < \text{my ID} \): send \( (\text{election}, \text{my ID}) \)
  - Skip if already \( \text{participating} \)
  - Set state to \( \text{participating} \)
  - If \( \text{id} = \text{my ID} \): I am elected (why?) send \( \text{elected} \) message
  - \( \text{elected} \) message forwarded until it reaches leader

Correctness?

- Safety: highest process elected
- Liveness: complete after \( 3N-1 \) messages
Example: Ring Election

Election:

P1
P2
P3
P4
P0
P5

1. P2 initiates election after old leader P5 failed

2. P2 receives "election", P4 dies

3. Election: 4 is forwarded forever?

May not terminate when process failure occurs during the election!
Consider above example where attr= highest id

Algorithm 2: Modified Ring Election

- **election** message tracks all IDs of nodes that forwarded it, not just the highest
  - Each node appends its ID to the list
- Once message goes all the way around a circle, new **coordinator** message is sent out
  - Coordinator chosen by highest ID in election message
  - Each node appends its own ID to coordinator message
- When coordinator message returns to initiator
  - Election a success if coordinator among ID list
  - Otherwise, start election anew

Modified Ring Election

- How many messages?
  - 2N
- Is this better than original ring protocol?
  - Messages are larger
- What if initiator fails?
  - Successor notices a message that went all the way around (how?)
  - Starts new election
- What if two people initiate at once
  - Discard initiators with lower IDs

What about that Impossibility?

- Can we have a totally correct election algorithm in a fully asynchronous system (no bounds)
  - No! Election can solve consensus
- Where might you run into problems with the modified ring algorithm?
  - Detect leader failures
  - Ring reorganization (member failures)
Algorithm 3: Bully Algorithm

- Assumptions:
  - Synchronous system
  - attr = id
  - Each process knows all the other processes in the system (and thus their ids)

- 3 message types
  - election – starts an election
  - answer – acknowledges a message
  - coordinator – declares a winner

- Start an election
  - Send election messages only to processes with higher IDs than self
  - If no one replies after timeout: declare self winner
  - If someone replies, wait for coordinator message
    - Restart election after timeout

- When receiving election message
  - Send answer
    - Start an election yourself
    - If not already running

Example: Bully Election

1. P2 initiates election
2. P2 receives replies
3. P3 & P4 initiate election
4. P3 receives reply
5. P4 announces itself

Analysis of The Bully Algorithm

- Best case scenario?
  - The process with the second highest id notices the failure of the coordinator and elects itself.
    - N-2 coordinator messages are sent.
    - Turnaround time is one message transmission time.

- Worst case scenario?
  - When the process with the lowest id in the system detects the failure.
    - N-1 processes altogether begin elections, each sending messages to processes with higher ids.
    - The message overhead is O(N^2).
**Turnaround time**

- **T**: Message bound—all messages arrive within \( T \) units of time (synchronous)
- \( T_{\text{process}} \): Processing bound—bound on the processing time at each process
- Turnaround time:
  - \( \text{election} \) message from lowest process (T)
  - Timeout at 2\(^{nd}\) highest process (X)
  - \( \text{coordinator} \) message from 2\(^{nd}\) highest process (T)
- How long should the timeout be?
  - \( X = 2T + T_{\text{process}} \)
  - Total turnaround time: \( 4T + 3T_{\text{process}} \)

**Summary**

- Coordination in distributed systems sometimes requires a leader process
- Leader process might fail
- Need to (re-) elect leader process
- Three Algorithms
  - Ring algorithm
  - Modified Ring algorithm
  - Bully Algorithm

**Acknowledgements**

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).