CSE 486/586 Distributed Systems
Leader Election

Slides by 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 from among only the non-faulty processes
  2) All non-faulty processes agree on which 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 such elections combined must yield a single leader
  - Election results should not depend on which process called for it
- Messages are eventually delivered.
Problem Specification

- An election protocol selects the non-faulty process with the “best” selected election attribute at termination.
  - Attribute examples: CPU speed, load, disk space, ID
  - Must be unique
- Each process maintains a variable \textit{elected}.
- An execution of the election algorithm should ideally guarantee at the end:
  - \textbf{Safety}: For each non-faulty process \( p \): \( p \)'s \textit{elected} = \( q \): a particular non-faulty process with the best attribute value, or \( \emptyset \)
  - \textbf{Liveness}: election terminates and for each non-faulty process \( p \), \( p \)'s \textit{elected} is eventually not \( \emptyset \)
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 \textit{election} message with my ID
- When receiving message \((\text{election, id})\)
  - If \( id > \) my ID: forward message
    - Set state to \textit{participating}
  - If \( id < \) my ID and state is not \textit{participating}: send \((\text{election, my ID})\)
    - Set state to \textit{participating}
  - If \( id = \) my ID: I am \textbf{elected} (why?) send \textit{elected} message
    - \textit{elected} message forwarded until it reaches leader again
• The election was started by process 17.
• The highest process identifier encountered so far is 24
• (final leader will be 33)
Ring-Based Election: Example

• The worst-case scenario occurs when?
  – the counter-clockwise neighbor (@ the initiator) has the highest attribute.

• In the example:
  – The election was started by process 17.
  – The highest process identifier encountered so far is 24
  – (final leader will be 33)
Ring-Based Election: Analysis

• In a ring of N processes, in the worst case:
  – N-1 *election* messages to reach the new coordinator
  – Another N *election* messages before coordinator decides it’s elected
  – Another N *elected* messages to announce winner

• Total Message Complexity = 3N-1

• Turnaround time = 3N-1
Correctness?

- **Safety**: highest process elected
- **Liveness**: complete after 3N-1 messages
  - What if there are failures during the election run?
Example: Ring Election

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 the message goes all the way around a circle, a new *coordinator* message is sent out
  - Coordinator is chosen by the highest ID in the *election* message
  - Each node appends its own ID to *coordinator* message
- When coordinator message returns to initiator
  - Election is a *success* if the new coordinator is in the ID list
  - Otherwise, start the election anew
Example: Ring Election

1. P2 initiates election

2. P2 receives "election", P4 dies

3. P2 selects 4 and announces the result

4. P2 receives "Coord", but P4 is not included

5. P2 re-initiates election

6. P3 is finally elected
Modified Ring Election

• How many messages?
  – 2N

• Is this better than original ring protocol?
  – Messages are larger

• Reconfiguration of ring upon failures
  – Can be done if all processes "know" about all other processes in the system

• What if the 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?
  – Detecting leader failures
  – Ring reorganization
Algorithm 3: Bully Algorithm

• Assumptions:
  – Synchronous system
  – Election attribute is process ID
  – Each process knows all the other processes in the system (and thus their IDs)
Algorithm 3: Bully Algorithm

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

- Start an election
  - Send election messages 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 receives no reply

6. P4 announces itself

answer=OK
The Bully Algorithm

$p_4$ fails, and $p_1$ detects this

$p_3$ fails

Eventually.....
Analysis of The Bully Algorithm

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

• Worst case scenario?
• When the process with the **lowest id in the system** detects the failure.
  – N-1 processes ultimately begin elections, each sending messages to all processes with higher IDs.
  – The message overhead is $O(N^2)$. 
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
References

• Textbook Section 15.3. Required Reading.
Acknowledgements

- These slides by Steve Ko, used and lightly modified with permission by Ethan Blanton
- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).