CS4/MSc Distributed Systems
Tutorial 5: Leader Elections

Stephen Gilmore
stg@dcs.ed.ac.uk

Tutorial: M 12, Room 5325
Office hour: Th 2, Room 1608

Yussuf Abu-Shaaban
yna@dcs.ed.ac.uk

Tutorial: M 12, Room 6207
Office hour: M 4, Room 3311

Leader Elections

- A **leader election** is defined as getting a set of processes to agree on a **unique** leader. The leader (coordinator) typically knows all the other processes (participants) in the group.
- A leader election may be initiated when an existing leader is detected or suspected to have failed.
- The election of a leader must be announced to all processes in the group.
- Leader elections can be based on
  - a global priority – extrema finding
  - individual preference – a voting scheme
- The main concern is usually the fast and successful completion of the election process.

Safety and Liveness Properties

We assume that each process P_i $i = 1, \ldots, N$, has a variable $elected_i$ which contains the process identifier of the coordinator process. A special value $\perp$ is used to mark when this variable is undefined, i.e. when there is an election in progress.

The key properties of a useful leader election algorithm are:

Safety: A participating process P_i has variable $elected_i = \perp$ or $elected_i = P$ is chosen as the non-crashed process at the end of the run with largest identifier.

Liveness: All processes P_i participate and eventually $elected_i \neq \perp$ or crash.

As with mutual exclusion algorithms, performance is judged on the basis of the number of messages necessary to complete the algorithm, which is proportional to the total network bandwidth used.

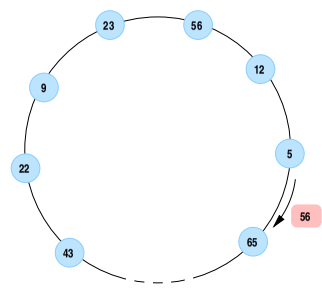
Ring-based Election Algorithm (1)

- Processes are assumed to be arranged in a logical ring: each process P_i has a communication channel to the next process in the ring $P_{(i-1) mod N}$. Each process has an identifier.
- The values of the identifiers must be unique and totally ordered.
- Identifiers may be dynamic, for example, reflecting the current computational load each process P_i might have $\langle 1/load_i, i \rangle$ as its identifier.
- All messages are sent clockwise round the ring and communication is reliable.
- The objective is to elect the process with the largest identifier as the **coordinator**.
- Any process can initiate the election by placing its identifier in an **election** message and sending it to its neighbour. It will also change its state from **non-participant** to **participant**.

Ring-based Election Algorithm (2)

- On receiving an **election** message a process compares the identifier it contains with its own identifier
  - if the message identifier is greater than its own and its state is **participant**, it simply passes the message on.
  - if the message identifier is greater and the process state is **non-participant**, it changes its state to **participant** and forwards the message on unchanged.
  - if the process identifier is greater (it must be non-participant) it changes its state to **participant** and forwards a new election message with its own identifier inserted.
- On receiving an **election** message with its own identifier in a process has won the election. It sets it state to **non-participant** and sends an **elected** message containing its own identifier.
- On receiving an **elected** message each process set its state to **non-participant** and sets its variable recording the coordinator.

Ring-based Election Algorithm Example



Both safety and liveness properties are satisfied. In the worst case (the process with the highest identifier is the last to be consulted) $3N - 1$ messages need to be sent.

Bully Algorithm (1)

- The bully algorithm is more robust than the ring-based algorithm because it can continue even if a process crashes during the election. In order to detect such failures the system must be assumed to be synchronous: **timeouts** are used.
- We assume that there is a set of processes $P_1, P_2, \ldots, P_N$, each of which has knowledge of the complete set and each of which can communicate with each other process directly.
- Moreover the set is ordered by the identifiers of the processes.
- The algorithm uses three types of messages:

  election messages used to announce an election;
  answer messages sent in reply to election messages;
  coordinator messages sent when the election is complete to announce the election result—the new coordinator.
- An election can be instigated by any process which detects (by message timeout) that the current coordinator has failed.

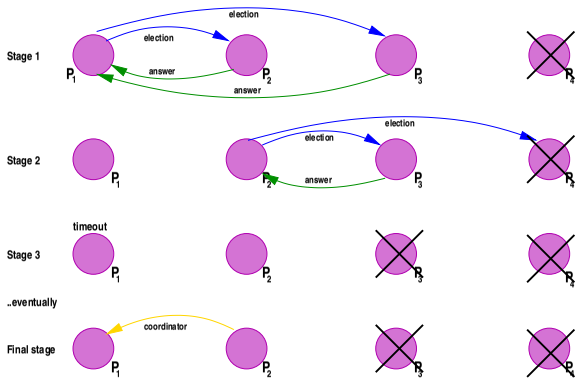
Bully Algorithm (2)

- If the process detecting the failure has the highest remaining identifier it can simply send a **coordinator** message, announcing itself as the new coordinator, circumventing the election.
- If a lower process detects the failure it announces the need for an election by sending an **election** message to each process with a higher identifier. It sets a **timeout** T and waits for replies.
- If no response arrives within time T the process assumes that all higher processes have failed and takes on the role of coordinator itself. It sends a **coordinator** message to all lower processes announcing itself as the new coordinator.
- If it receives an **answer** message from a higher process it then sets a second **timeout** T' and waits for a coordinator message.
- If no coordinator message arrives it assumes that there has been a subsequent failure and initiates another election by sending an **election** message to each higher process.

Bully Algorithm (3)

- If a process receives an **election** message, it sends back an **answer** message, and begins another election (sending **election** messages to all higher processes) if it has not already started one. It is possible for several elections to be started at once because several processes detect the failure of the current coordinator.
- If a process P_i receives a **coordinator** message, it sets its variable $elected_i$ to the identifier of the coordinator contained within the message.
- The algorithm is intended to work in a dynamically changing group of processes. As well as crashing, processes may also be restarted. When a process is inserted into the group to replace a failed process it starts an election.
  - If it has the highest identifier it can simply announce itself as coordinator even though there is a coordinator currently working.

Bully Algorithm Example



Bully Algorithm Characteristics

- If we assume that message delivery is reliable and the failure detector is reliable, the liveness condition is satisfied for the algorithm and all process will eventually reach agreement on the coordinator or will have crashed.
- If failed processes are not restarted the safety property is also satisfied. However if a crashed process can be replaced by another with the same identifier then the safety property is not met.
- If the replacement process has the highest identifier but joins the group, sending a **coordinator** message, just as another process is sending its **coordinator** message, the messages may arrive at different orders with different members of the group. This could lead to subsets of processes having different records of who is the current coordinator.
- In the best case the number of messages is just $N - 2$ coordinator messages; in the worst case $O(N^2)$ messages are required.