Elections

Pedro F. Souto (pfs@fe.up.pt)

April 17, 2021

Leader Election

Why Many distributed algorithms rely on a process that plays a special role – **coordinator/leader**. Such algorithms usually are:

- Simpler
- More efficient

What Upon completion of the algorithm all non-faulty nodes agree on who the coordinator is.

- Only one node is elected the coordinator
- All nodes know the identity of the coordinator

Garcia-Molina's Algorithms: Introduction

- The algorithms were proposed in the scope of **system reorganization** upon failure/recovery of system components. But, elections are also useful:
  - At initialization;
  - To add/remove nodes (to a less extent).
- GM observes that we can ensure fault-tolerance by means of two approaches:

  By masking failures i.e. by using algorithms that continue to work correctly, even if some system components fail:
    - This is the only approach if we need continuous operation
    - Also likely to be the more appropriate, if failures are common

  By reorganizing the system i.e. by halting normal operation and take some time to reorganize the system
    - Likely to be allow simpler algorithms
- We abstract the leader election problem from this context
  - This leads to simpler versions of GM's algorithms

Some notes on the paper

This paper is really worth the reading

- It is very well written
- It is an early paper on distributed algorithms and GM explains the issues at length
- It touches on several recurrent issues in distributed systems/algorithms:
  - Fault-tolerance
  - Synchronous vs asynchronous systems
  - Failure detection (and its impossibility in asynchronous systems)
  - Groups of processes
  - RPCs
- GM is very careful/rigorous:
  - Assumptions
  - Specifications
  - Algorithms
- And, in spite of all that, the specification for asynchronous systems is buggy

System Model/Assumptions

1 All nodes cooperate and use the same algorithm

4 All nodes have some **stable(/safe)** storage

5 When a node fails, it immediately halts all processing.
   - Crashed nodes may recover
   - Data on stable storage is not lost, i.e. is as before the crash

3 The communication subsystem does not spontaneously generate messages

6 There are no transmission errors (but messages may be lost)

7 Messages are delivered in the order in which they are sent

8 The communication system does not fail and has an upper bound on the time to deliver a message, T

9 A node always responds to incoming messages with no delay

Observation Assumptions 8 and 9 mean the system is synchronous.
   - The author claims that they are reasonable both for a LAN or a high-connectivity network
   - They will be dropped below

Specification: State

- Virtually all distributed algorithms may be described by state machines:
  - Describing the operation of each node (process)
  - Changing their state in response to reception of messages or to the passage of time

$S(i).s$ state of the node i: one of DOWN, ELECTION and NORMAL [a]

- When a node crashes its state changes automatically to DOWN

$S(i).c$ the coordinator according to node i

[a] G-M considers an additional state, but here we are presenting election algorithms independently of their application

Specification of Leader Election

Assertion 1 At any time instant, for any two nodes, if they are both in NORMAL state, then they agree on the coordinator:

$$\forall_{i,j} : S(i).s = S(j).s = \text{NORMAL} \Rightarrow S(i).c == S(j).c$$

Assertion 2 If no failures occur during the election, the protocol will eventually transform a system in any state to a state where:

a) there is a node i such that $S(i).s = \text{NORMAL}$ and $S(i).c = i$

b) all other non-faulty nodes $j \neq i$ have $S(j).s = \text{NORMAL}$ and $S(j).c = i$

Safety vs. Liveness Properties (Parenthesis)

- Any specification can be expressed in terms of **safety** and **liveness** properties (Lamport 77):

  Safety property states that something (bad) will not happen
    - Proving such a property involves proving an invariant
    - Once an execution violates a safety property, there is nothing that can be done to fix that

  Liveness property states that something (good) must happen
    - Proving such a property involves a different technique
    - Any "partial" execution can always be extended so that eventually something good happen
      - If that is not possible, then something bad must have happened, i.e. some safety property must have been violated
- What about the properties of the leader election specification?

Specification of Leader Election

Assertion 1 At any time instant, for any two nodes, if they are both in NORMAL state, then they agree on the coordinator:

$$\forall_{i,j} : S(i).s = S(j).s = \text{NORMAL} \Rightarrow S(i).c == S(j).c$$

Assertion 2 If no failures occur during the election, the protocol will eventually transform a system in any state to a state where:

a) there is a node i such that $S(i).s = \text{NORMAL}$ and $S(i).c = i$
b) all other non-faulty nodes $j \neq i$ have $S(j).s = \text{NORMAL}$ and $S(j).c = i$

Leader Election vs. Mutual Exclusion

1. In an election fairness is not important
   - All we need is that one node becomes the leader
2. An election protocol must deal properly with the failure of the leader
   - Usually, mutual exclusion protocols assume that a process in a critical section does not fail
3. All nodes need to learn who the coordinator is

The Bully Election Algorithm (1/3)

Idea A node wishing to become a leader:

- Looks around to ensure stronger nodes are not up (**phase 1**)
- If does not see any, it
  1. imposes itself as leader; (**phase 2**)
  2. brags about it.

Convention The smaller a node's identifier the **stronger** it is [a]

[a] G-M uses the inverse order

The Bully Election Algorithm (2/3)

Phase 1 A node wishing to become leader checks if stronger nodes are around sending them an ARE-U-THERE message

- If present, a stronger node responds with a YES message and initiates a new election itself
  - By checking if stronger nodes are around
  - What if the challenged node is already in the middle of an election?
- A candidate whose challenge is answered, backs off
  - But should start a timeout to detect a possible failure of the challenger

The Bully Election Algorithm (3/3)

Phase 2 Node i begins phase 2, if it does not receive any response to its probe within $2T$. It comprises two steps:

1. Sends a HALT message to weaker nodes
   - Upon receiving HALT, a node

   1.1 Sets its state to ELECTION

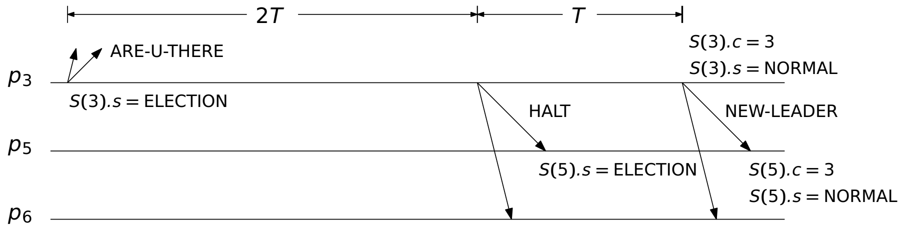
   1.2 Cancels any election it has already started
2. T time units later, node i sends a NEW-LEADER message to weaker nodes, and sets $S(i).c$ to i and $S(i).s$ to NORMAL
   - Upon receiving that message, node k sets $S(k).c$ to i and $S(k).s$ to NORMAL

Comment The HALT message (1st step) is required to ensure that **Assertion 1** is **not** violated

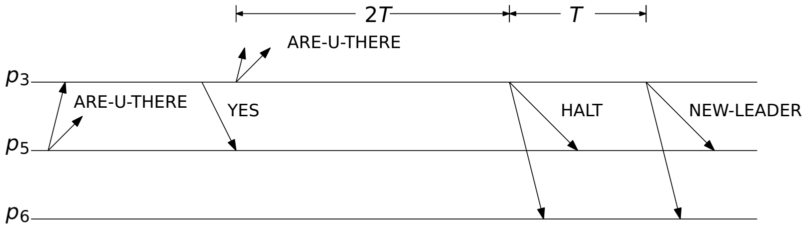
Failure of the candidate during the second phase will trigger a new election

Bully Algorithm: Example Execution

- Strongest node starts upon failure detection



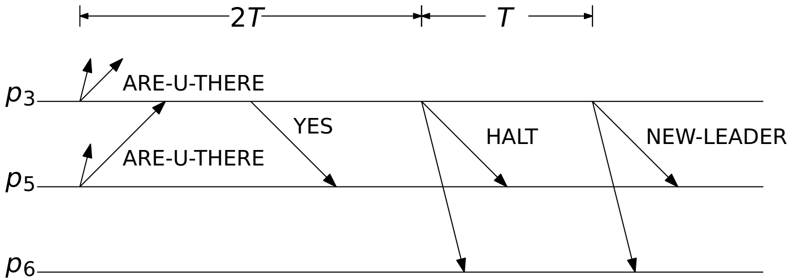
- Strongest node starts upon challenge



Food for thought Is it possible to remove the HALT message?

Bully Algorithm: Concurrent Executions (1/3)

- Two nodes start the election more or less simultaneously



Stronger nodes will reply to weaker ones

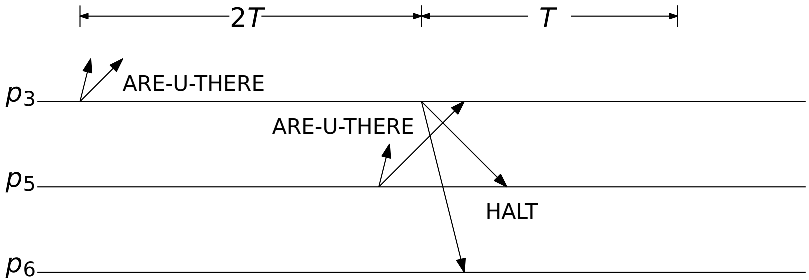
Strongest node will finish its election

- unless it fails

Weaker nodes will back off

Bully Algorithm: Concurrent Executions (2/3)

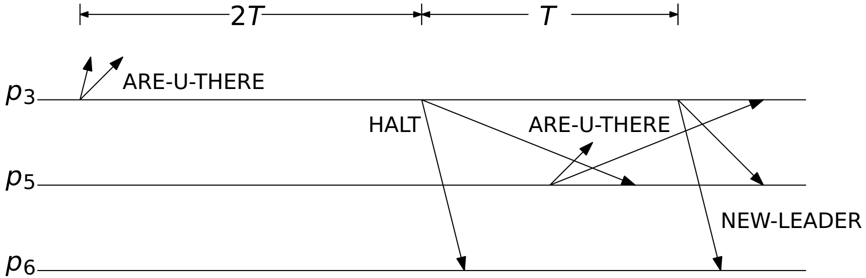
- What if the HALT and ARE-U-THERE messages are concurrent? I.e.
  - What if ARE-U-THERE is received between HALT and NEW-LEADER?



  - Should p_3 send YES?
  - Should p_3 start another election?
  - Imagine a run where p_5 has just recovered, and did not receive HALT

Bully Algorithm: Concurrent Executions (3/3)

- Is it possible that the ARE-U-THERE and NEW-LEADER be concurrent?



The Bully Election Algorithm and Failures

What about node failures? Depends on the node:

Leader Upon detection of failure of the current leader

- A process initiates an election

Candidate Upon detection of failure of the candidate

- A process initiates an election

Other processes it does not matter

- GM's algorithm starts a new election on such an event, because its focus is on reorganization

What about recovery of a node (after a failure)?

- The node initiates an election

What if a another node has already initiated an election?

- This is just another case of a concurrent execution – see above
  - Execution depends on which node is stronger

The Actual Bully Algorithm (1/3)

RPC rather than messages, with some interesting features

- Syncronous caller is suspended until it receives the reply from remote process or there is a timeout
- ONTIMEOUT clause (argument is time bound to receive reply, T)
  - ARE-U-THERE, HALT just sends the next RPC (synchronous)
  - NEW-LEADER restart election (reorganization)
- Immediate assume special implementation (without context switch nor RPCs on remote side) for low latency
  - ▶ Used for RPCs that use timeouts (all of them)

Failure Suspector each node has a failure suspector that triggers a TIMEOUT event, if the node does not receive a message from the **last known leader** for some time

- ▶ Upon a TIMEOUT event
  - If $s == NORMAL$ the node calls ARE-U-THERE
    - ▶ And starts a new election if ARE-U-THERE times out
  - Otherwise starts a new election immediately

The Actual Bully Algorithm (2/3)

ELECTION is a procedure that executes the entire protocol by calling appropriate RPCs (?sequentially?). E.g.:

ARE-U-THERE is a null RPC, i.e. it has no statements

- Receiver does not initiate election if stronger.
- Candidate gives up, if ARE-U-THERE returns

HALT nodes that respond are added to the candidate's Up set, which belongs to the process state

NEW-LEADER Is sent only to nodes that are in Up

- If some node does not reply, candidate starts new election

Recovery is a procedure that is executed automatically upon recovery of a node, and starts a new election (call ARE-U-THERE RPC ...)

- If it is stronger than the current leader, the latter will not receive ARE-U-THERE, but it will move to the ELECTION state upon reception a HALT RPC and

Question How is a weaker recovering node integrated in the system?

- See the CHECK procedure below

The Actual Bully Algorithm (3/3)

Reorganization (not covered earlier)

Check is a procedure executed periodically by the leader

- It scans every other node (using the ARE-U-NORMAL RPC) to find out if there are new processes
- It has a dual purpose:
  1. Find new nodes
  2. Allows other nodes to detect failure of the leader

ARE-U-NORMAL RPC checks the state of a remote node

- The response is: $s ==$ NORMAL ? YES : NO
- Upon reception of a NO, the caller starts a new election

Leader Election without Assumptions 8 and 9

If we drop assumptions:

8 The communication system does not fail and has an upper bound on the time to deliver a message, T

9 A node always responds to incoming messages with no delay

Then assertions:

Assertion 1 At any time instant, for any two nodes, if they are both in NORMAL state, then they agree on the coordinator

Assertion 2 If no failures occur during the election, the protocol will eventually transform a system in any state to a state where there is a coordinator

cannot be satisfied **always**:

1. Assume node i is the coordinator, has not crashed but it does not respond to other nodes because it is too slow
2. From the point of view of other nodes, it has crashed, so to satisfy Assertion 2, they must elect a new coordinator
3. But, if they elect a new coordinator, Assertion 1 will be violated

Specification without Assumptions 8 and 9 (1/2)

Groups

Definition Is a set of nodes with a **group id**, i.e. an identifier

- All messages are tagged with the group id
- Not all messages with foreign group ids can be ignored

Node state Includes also the following pieces of information:

$S(i).g$ the current group id;

Specification without Assumptions 8 and 9 (2/2)

Assertion 3 At any time instant, for any two nodes i and j in NORMAL state and in the same group, then they must agree on the coordinator: $S(i).s = \text{NORMAL} \wedge S(j).s = \text{NORMAL} \wedge S(i).g = S(j).g \Rightarrow S(i).c = S(j).c$

- This alone is weak, as it can be satisfied by a singleton group

Assertion 4 Suppose that:

1. there is a set of operating nodes R which all have **two way communication with all other nodes** in R. That is Assumptions 8 and 9 hold for nodes in R
2. there is no superset of R satisfying the previous property
3. no node failures occur during the election

then the election algorithm will eventually transform the nodes in set R from any state to a state where there is i in R such that for every node j in R:

$$S(j).s = \text{NORMAL} \wedge S(j).c = i \wedge S(j).g = S(i).g$$

Note Assertions 1 and 2 are special cases of Assertions 3 and 4

The Invitation Algorithm (1/2)

Idea Rather than imposing itself as a leader, a node wishing to become a coordinator invites others to join in a group where it is the coordinator

- Initially, each node creates a singleton group, of which it is the coordinator
- Periodically, coordinators try to merge their group with other groups in order to form larger groups

Description

Failure detection a node that is not a leader periodically checks if its leader is still alive (using ARE-U-THERE messages)

- If not, it creates a singleton group of which it is the leader

Group merging a node that is a leader periodically probes all other nodes for leadership (using ARE-U-LEADER messages)

- If one or more nodes reply, node *i* initiates the merging protocol after a delay inversely proportional to its priority
  - The variable delay helps preventing different nodes to initiate the merging concurrently

The Invitation Algorithm (2/2)

1. Node *i*, leader and **candidate**, sends an INVITATION message:
   - to all leaders that have responded
   - to the members of its current group
2. When a leader *j* receives an INVITATION, it forwards it to the other group members
3. All nodes that receive an INVITATION, directly or indirectly, respond with an ACCEPT message to the candidate (to leader)
4. The candidate adds the sender of each ACCEPT message as group member
5. After **enough(?)** time to receive ACCEPT messages from all group members, the new leader sends a READY message to all of them
   - Note that assumptions 8 and 9 hold for *R*
6. Upon receiving the READY message to a previously sent ACCEPT, node *k* joins the new group and sends an ACK message
   - If a node does not receive a READY message after a timeout, it initiates a new election

The Invitation Algorithm: Concurrency

- A process moves to the ELECTION state, before:
  - Sending the INVITATION message, if candidate
  - Sending the ACCEPT message, otherwise
- A process moves to the NORMAL state, after:
  - Sending the READY message, if candidate
  - Receiving the READY message, otherwise
- A process responds to:
  - ARE-U-LEADER
  - INVITATION

  messages only if its state is NORMAL, i.e. it is not participating in an election:
  - A process does not participate in more than one election at a time

The Invitation Algorithm: Liveness

- Process failure is **suspected** using timeouts while waiting for the reception of some messages in response to:
  - ARE-U-THERE, sent to the coordinator
  - INVITATION, sent to other leaders (or by the leaders to their group members)
  - ACCEPT, sent to candidate
  - READY, sent to the new group members
- Upon timeout, there are several possible actions:

  Advancing to the next phase of the protocol
  - E.g. when waiting for responses to INVITATION

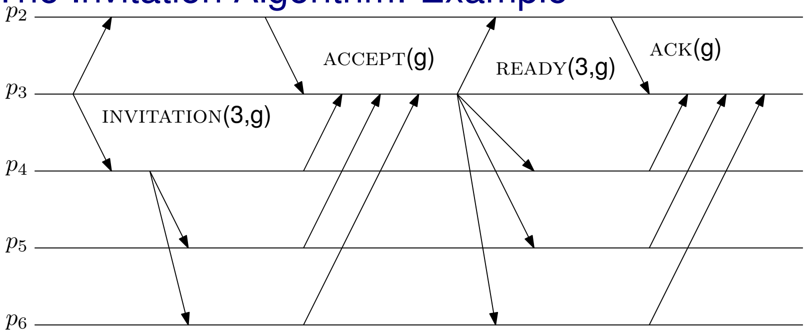
  Initiating recovery procedure
  - E.g. when waiting for responses to ACCEPT

  that creates a singleton group
  - No need for communication

The Invitation Algorithm: Example

- Consider a group of 6 nodes with ids from 1 to 6, with node 1 as leader
- Let node 1 fail
- Each of the other members forms a singleton group
- Assume that nodes 3 and 4, send invitations to the other nodes and that the conditions on the system are such that:
  - Node 2 accepts the invitation of node 3, leading to one group coordinated by node 3 and members 2 and 3;
  - Nodes 5 and 6 accept the invitation of node 4, leading to one group of coordinated by node 4 and members 5 and 6;
- Some time later, one of the nodes invites the other coordinator to join it in a group
- For an informal proof of correctness check the paper

The Invitation Algorithm: Example



- p_4 forwards the INVITATION to its group members
- To ensure liveness we need several timeouts:

  Candidate cannot wait indefinitely for:

    accept
    ack

  Other processes cannot wait indefinitely for:

    ready
- Is ACK really needed?

Is the Invitation Algorithm Correct?

It appears correct

But Scott Stoller has shown that it does not satisfy Assertion 4

Suppose that:

1. there is a set of operating nodes R which all have **two way communication with all other nodes** in R. That is Assumptions 8 and 9 hold for nodes in R
2. there is no superset of R satisfying the previous property
3. no node failures occur during the election

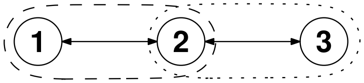
then the election algorithm will eventually transform the nodes in set R from any state to a state where there is i in R such that for every node j

$$S(j).s = \text{NORMAL} \wedge S(j).c = i$$

...when the connectivity is not transitive

Problematic Scenario

1. Node 1 crashes
2. Nodes 2 and 3, each forms a singleton group



3 Node 1 recovers, but communication between nodes 1 and 3 has been lost. Communication between all other pairs of nodes works normally

4 Node 1 forms a singleton group, and invites node 2

5 Nodes 1 and 2 become a group, whereas node 3 becomes a singleton group.

Observation 1 If no more failures occur, these groups will not change

Observation 2 The set $\{2,3\}$ satisfies the hypotheses on set R in Assertion 4

Contradiction

- Assertion 4 requires that node 2 be coordinator of group $\{2,3\}$
- Node 2 is a member of group $\{1,2\}$

Solution (1/2)

Fix the specification The specification is too strong

- It requires processes that are connected to belong to the same group
- If one of them is not the leader, that is not going to happen

Weaken the requirements But that requires a more complex definition

Two nodes are disconnected in a time interval if all messages sent between them during that interval are lost

Stable system in a time interval if, during that interval, no crashes or recoveries occur and every pair of nodes is either connected or disconnected

Connectivity graph, when a system is stable is the undirected graph whose vertices correspond to the nodes and with an edge between vertices i and j iff nodes i and j are connected

Clique cover of a graph is a partition of that graph's nodes into cliques, i.e. fully connected subgraphs

E^* reflexive and transitive closure of relation E

Solution (2/2)

Let $\langle V, E \rangle$ be the system connectivity graph

Assertion 4' For a given system, there is a constant τ such that if the system is **stable** for a time interval of duration at least τ, then by the end of that interval, the system reaches a state such that

a) $S(i).s = \text{NORMAL} \wedge S(i).g = S(S(i).c).g \wedge (\langle i, S(i).c \rangle \in E^*)$

b) the number of groups is at most the size of a minimum-sized clique cover of $\langle V, E \rangle$

Note A clique cover is a partition of a graph's vertices in cliques. E.g. the sets

$$\{\{1,2\},\{3\}\},\{\{1\},\{2,3\}\},\{\{1\},\{2\},\{3\}\}$$

are clique covers of the problematic graph above.

Theorem The Invitation Algorithm satisfies **Assertion 4'**

Proof Check Scott Stoller's paper

Further Reading

- Subsection 6.5, Tanenbaum and van Steen, *Distributed Systems*, 2nd Ed.
- Hector Garcia-Molina, *Elections in a Distributed Computing System*, IEEE Transactions on Computers, Vol. C-31, No. 1, January 1982, pp. 48–59
- Scott Stoller, *Leader Election in Asynchronous Distributed Systems*, IEEE Transactions on Computers, Vol. C-59, No. 3, March 2000, pp. 283–284