**CSC-360 Distributed Algorithms Paper**

Frank Bedekovich  
Robert Krency  
Regina Thase

With the proliferation of computing in every segment of society, the need for efficient algorithms continues to grow. Some tasks however cannot be handled by a single processor or node, and thus must be distributed across multiple interconnected systems. Algorithms designed to handle this type of computing are known as distributed algorithms. This classification is a subset of parallel algorithms, where multiple independent processes are utilized to complete a task with minimal knowledge of the other sub-processes.

It then follows that there must be a master process that handles delegation and collection for the sub-processes in order to produce the final result. A major issue that distributed algorithms face is resiliency. In other terms, what happens to processes that crash or disconnect, and what happens when the process that is no longer viable was the initial leader?

This problem has been formalized as the Coordinator Election Problem, or Leader Election Problem, characterized by Howard Hamilton in his work *Introduction to Operating Systems*. This is especially relevant for developing distributed algorithms that will have a single entry point, and work will then be subdivided.

Marc Brooker’s article *Leader election in Distributed Systems* further breaks this down into terms that are easier to relate: “Leader election is the simple idea of giving one thing (a process, host, thread, object, or human) in a distributed system some special powers. Those special powers could include the ability to assign work, the ability to modify a piece of data, or even the responsibility of handling all requests in the system.”

A simple form of this is to assign one processing-node as the leader for the algorithm, from which the work and data will be distributed to other nodes. Other nodes in the system are then assigned tasks to carry out, with minimal knowledge or concern for the work of other nodes. Solutions to the leader election problem vary in terms of efficiency and complexity.

One such example algorithm has been termed the ring election algorithm. The initial leader traverses each node, one at a time, connecting them in a circular or ring fashion such that the final node connects with the initial node.

This algorithm relies on two assumptions. First, processes only send messages to the next process in the ring. Second, passed messages will continue along the ring in the event of a process crash. As the algorithm is processed, information continues to flow from each process to the coordinator. In the event that process *Pi* doesn’t receive a response in a predetermined amount of time, process *Pi* initiates a new leader election as the communication ring has broken down. Hamilton breaks down this algorithm into the following steps:

1. Initialize an active process list to empty
2. Send an “Elect( *Pi* )” message to the next node, and add *Pi* to the active list
3. If a process receives an “Elect( *j* )” message
   1. If this is the first message seen
      1. Initialize this process’s active list to [*i, j*]
      2. Send messages “Elect( *i* )” and “Elect( *j* )” to the next
   2. If ( *i != j* ), add *i* to the active list and forward “Elect( *j* )” to each of the active list
   3. Otherwise ( *i = j* ), so process *Pi* has a complete set of active processes
      1. Choose the highest process ID *Px*
      2. Send “Elected( *Px* )” to neighbor
4. If a process receives an “Elected( *Px* )” message
   1. Set its coordinator to *Px*

There are varying refinements and implementations of the ring algorithm. A synchronous ring algorithm will check nodes in rounds instead of individually, reducing the time spent sending messages around the ring. ID ring algorithms utilize identifiers for each node and assume that the node with the highest ID is the coordinator. An asynchronous ring algorithm will send an identifier around the process chain from the initial node, alerting each other process of a new coordinator and assigning it as a subprocess.

Consider the logical arrangement of the processes in this algorithm. Richard Anthony, in his work *Election Algorithm - an Overview*, explains the ring algorithm as “the nodes are arranged in a logical ring, and the nodes only communicate with their logical neighbors.” It then follows that this solution to the leader election problem is only viable when processes can be logically shaped circularly. A different approach is required when processes take a different form.

A second approach to tackling the leader election problem is the Bully Algorithm, according to Hamilton. Unlike the ring algorithm, the bully algorithm chooses the next node in the system based on the highest weight, without care for the previous node. This continues until the lowest weighted node is connected last. Similarly to the ring election, when a process *Pi* has sent a message to the current coordinator and does not receive a message after a predetermined amount of time, process *Pi* starts a new election process with itself as the candidate. Hamilton details this in the following steps:

1. Process *Pi* sends an “Election” message to every higher priority process
2. If no response
   1. Process *Pi* assumes the coordinator role
   2. It sends an “Elected” message to all lower priority processes with the new coordinator
3. If an “Election” response is received by a higher priority process *Pj*
   1. Reinitiate the election process using process *Pj*
4. If an “Elected( *Pi* )” response is received by a lower priority process
   1. Update its coordinator to *Pi*
   2. Continue the message to lower priority processes

The bully algorithm does not rely on any arrangement of the processes, only the ability to send to other processes. As with developing any algorithm, it’s important to understand the tradeoffs between differing solutions to the same problem.

The leader election problem remains to be an active area of interest in the area of Distributed Algorithms. As distributed algorithms are processed, faults are expected and the systems require a degree of resiliency. Being able to adapt to faults is key, with the main interest being ensuring a master process that coordinates all other processes. This paper has discussed two solutions for finding a new coordinator process, the ring algorithm and bully algorithm. Utilizing either algorithm is dependent on the ability to manipulate the logical arrangement of processes and provides a degree of resilience for distributed algorithms that rely on a coordinator process.

**References**

Anthony, Richard John. “Election Algorithm in Systems Programming.” *Election Algorithm - an Overview | ScienceDirect Topics*, 2016, https://www.sciencedirect.com/topics/computer-science/election-algorithm.

Brooker, Marc. *Leader Election in Distributed Systems*. amazon.com, <https://aws.amazon.com/builders-library/leader-election-in-distributed-systems/>.

Hamilton, Howard J. “Introduction to Operating Systems.” *Distributed Algorithms*, http://www2.cs.uregina.ca/~hamilton/courses/330/notes/distributed/distributed.html.