Simulation Environment for Leader Election Protocols in Mobile Ad-Hoc Networks

Tiago Sousa
up201907205@up.pt

Hugo Cardoso
up201907840@up.pt

***Abstract*—Distributed Systems need to make decisions and take actions in a uniform, quick and decisive manner. However, independently coordinating a large number of nodes has inherent problems, such as straining the network and high decision-making times. Leader Election is a solution to these problems. This project intents to create a simulation environment where different Leader Election protocols may be tested using the infrastructure presented in this report. The implementation was developed in Python, resulting in a simulation environment that mimics and Ad-Hoc network where nodes run in separate processes and can be crashed to simulate network topology changes. Furthermore, the simulation environment also provides crucial data, such as the number of messages exchanged and the time it took to elect a leader. Validating the simulator required the implementation of two protocols, the Bully election and the Invitation election protocol [1]. The results obtained during testing demonstrate the success of this simulation environment and showcase the differences between the two protocols.**

I. Introduction

Leader election is a fundamental problem in distributed systems, where nodes must agree on a single coordinator to manage tasks and handle failures. During the election process a node is elected as the coordinator for that system and can then be used as a reference for a variety of contexts such as the slave-master methodology of clock synchronization or decision making. Having a single leader node can bring several advantages to a network such as reduce complexity in coordination for replication or synchronization, simplify conflict resolution (i.e. one node wants to take a different action from a different node), load balancing and centralizing decision making which greatly improves the response time of systems. This problem becomes more complex in dynamic networks where nodes can join, leave or fail. The Leader election solution has been extensively studied, which resulted in a variety of different protocols, each with their advantages and disadvantages. Testing and validating these protocols becomes crucial before they are deployed as often time distributed systems are placed in critical infrastructures. Moreover, the kind of network it is deployed in may constrain it's abilities or reduce it's performance. Typically Ad-Hoc networks are very challenging to work in as nodes change their position relative to other nodes, which results in a smaller bandwidth and longer transmission times, with longer transmission times being particularly detrimental for time sensitive leader election protocols such as the Invitation protocol. This project implements both the Bully election and the Invitation election algorithms, with the goal of validating the simulator. Utilizing the developed simulation environment, this report intents to share the results provided by the simulator, which include the time taken to complete the leader election under different scenarios and network topologies, as well as the number of messages exchanged. The paper is structured as follows: Section II defines the problem of study. Section III describes the simulation environment and the election algorithms structure. In Section IV the results are presented and discussed. Section V lists the project management tools and addresses possible future developments. This report concludes in Section VI, where an overview is detailed.

II. Problem Definition

In distributed systems, the need for coordination among nodes arises with frequency, as having a designated leader is essential to ensure the system operates smoothly. However, electing a node in a distributed environment is not a trivial task. It's also worth to note that network issues are not negligible and can complicate the election process, such as delays, packet loss and node failures. Knowing that the problem of leader election is a vastly studied topic with various proposed approaches and solutions, this project targets to build a simulation environment capable of testing, in an uniform way and under similar circumstances, so that the algorithms can be studied and compared.

III. Proposed Methodology

The present section will detail the simulation environment that allows the user to generate multiple processes with their own sockets, Internet Protocols (IPs), ports and memory spaces. It is important that processes are used and not threads so that each node does not share memory as this would be the case in the real world. The user can set up any kind of network using graphs or trees or any other kind of data structure, provided that the neighbors of each node are passed onto each process responsible for the node.

A. Node Class

As mentioned before, the simulation environment will create multiple processes - one per node - passing each process their node ID, their neighbors and additional control flags (i.e. number of iterations the node should run before exiting, which election protocol should be tested and whether or not the leader node should crash at the end of each iteration). To achieve this, a class was created. In Figure 1 the class diagram for the class

Node can be seen, where the attribute ”Arguments” is a list of control flags - the ”Counter” attribute determines the number of times the election process is called which is done so in a periodical manner. The attributes ”In Election”, ”Leader” and ”Halt” are inherent to the Bully Election protocol and the attributes ”Group”, ”Is Leader” and ”Leaders” are inherent to the Invitation Election protocol. The attribute ”Crashed” determines whether the node is to be crashed or not - the node is crashed if it is not communicating with the rest of the network - which was implemented in two different ways: one is where the socket of that node is closed and the node does not get reintroduced into the network, the other way is to set that process to sleep, which will reintroduce the node into the system once it is done sleeping. This solution allows simulating the crash of each node and the simulation of an environment where the crashed node recovers, which is important so that the protocol's reaction to changes in the network topology can be understood. The ”Node ID” attribute not only identifies the respective node, but also determines its priority during the leader election protocol - the higher the ID, the more suitable that node is to be a leader. The ”Buffer Size” and the ”Encoding Format” represent the maximum length of the message received and the format in which messages are encoded into bytes, respectively. The available functions are: the ”*send_msg(int target_id,string msg)*” which encodes the string passed to it in ”*msg*” and sends it to the node whose ID is the ”*target_id*”. The ”*receive_msgs_thread()*” which is called at node initialization as a thread, so that the nodes do not miss any messages due to poorly allocated resources. The ”*node_loop()*”, firstly calls the ”*create_socket()*” function, which creates a socket and binds it to the respective node and stores it in the attribute ”Node Socket”. Afterwards, the function runs a periodic loop (the tested period was of 40 seconds, but it can be increased or decreased to comply with user needs) for ”Counter” amount of iteration and it crashes the leader node (and only the leader node) if the ”CRASH” argument was passed to the process. This function also calls the ”*start_invitation_election()*” or ”*start_bully_election()*”, depending on the argument that was passed to the process, once per iteration.

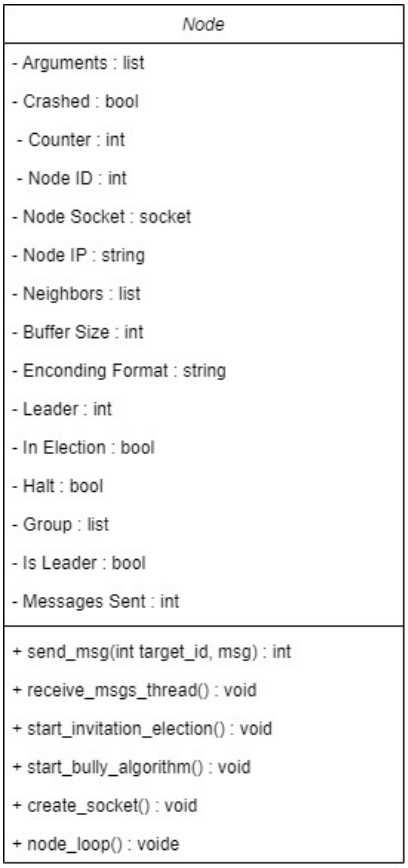

Fig. 1: Class diagram of the Node class

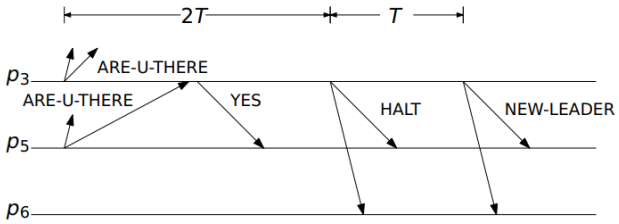

Fig. 2: Bully Protocol

B. Leader Election Protocols implemented

In order to test the simulation environment built, two protocols were implemented: the Bully Election Protocol and the Invitation Protocol. Both of these algorithms were originally proposed by Garcia-Molina [1] and were fundamental in the leader election space with significant impact in multiple different industries. Furthermore, the simulation environment is implemented to impose all assumptions referred in Garcia-Molina's work. Since this project's focus isn't the protocol themselves, this report does not discuss the issues of the protocols or their details.

The Bully Election Protocol is based on the idea that a node will probe the network to check if there are higher priority nodes and if, after a period of time, there are no responses to the probe it will halt other elections and declare itself as leader, after a given period of time. Figure 2[1] illustrates the normal function of the algorithm, note that the priorities are inverted.

Initially the **Invitation Election Protocol** sets every node as a singleton group, where the node is the leader of its own group. Secondly, the protocol probes the network for a leader, where only leaders are allowed to respond to this probe. If multiple leaders respond, it starts an election by sending the ”INVITATION” message to every group leader that responded, to which the leaders respond with ”ACCEPT” and forward that invitation to the members of their group. Thirdly, the node waits an inversely proportional amount of time to its priority (i.e. the lowest priority node waits the

[1]This image was taken from the Distributed Systems course, at the Faculty of Engineering of University of Porto

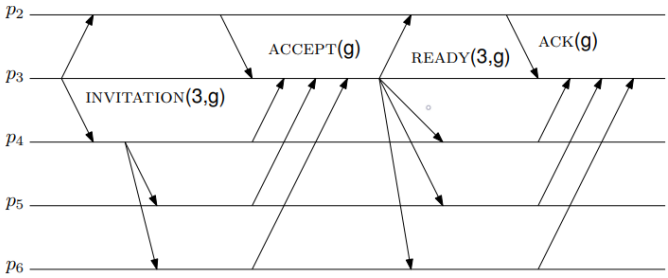

Fig. 3: Invitation Protocol

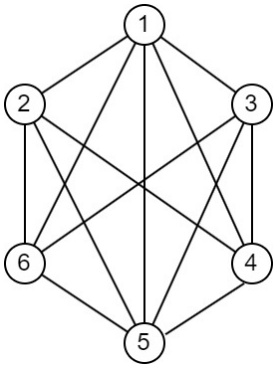

Fig. 4: Six Node fully connected Graph Example

longest amount of time for responses) - this is an important step due the possibility of two leaders initiating the election, where only one should be considered leader. Afterwards, the highest priority node adds all the nodes that answered "ACCEPT" to their group and sends out a "READY" message letting every node know that they are no longer leaders (and therefore can not send invitations or add other nodes to their groups, stopping any previously started elections), to which other nodes respond with "ACK" to acknowledge the new leader. Figure 3[2] illustrates the process, note that the priorities are inverted.

IV. Results

Firstly, it is important to recognize Python allows you to create more processes than there are CPU cores. To achieve this, Python inserts a layer that allocates CPU time for each process, which can become a problem in this context since the protocols rely on precise timings to elect the leader. However, analyzing this is beyond the scope of the project, but for this reason and because the execution times directly effect the leader election times, it is important to specify that the processor used during testing was the Intel Core i5 9600k @ 3.7 GHz, which is a 6-core, 6 threads processor with no overclocking. The presented results were measured without any prints so as to not effect the leader election times and each election was run five times for each topology. Furthermore, these results do not account for leader crashes, although regardless of protocol a consensus was consistently reached about the leader and even if the leader crashed after election a new leader was successfully elected. Regarding the Invitation election, the consensus was considered reached and timed once the last "ACK" was received by the leader. For the network topology a fully connected graph with 4, 6 and 12 nodes was chosen, as it encompasses multiple case scenarios and represents different levels of strain in the simulation system. An example of the network topology with 6 nodes can be seen in 4.

In Table I the results obtained for both the Bully protocol and the Invitation protocol can be seen. In general sense, the larger the network the more messages are exchanged and electing takes longer, which is excepted and validates the simulation environment. On the other hand, the number of messages generated is also as excepted and can be verified through the respective theoretical concepts behind the protocols. Note that since no edge cases were forced during elections the number of messages exchanged was consistent throughout all tests and did not change from the theoretical number. The difference between the Bully election algorithm and the Invitation election algorithm becomes very apparent under the microscope provided by the simulation environment. The Bully algorithm takes longer as every node has to wait the same amount of time for every response of all the other nodes during the election process, whereas during the Invitation algorithm the leader has to wait a shorter amount of time compared to other nodes to assume leadership and communicate it to others, which makes the system elect a leader significantly faster than the Bully protocol. However, the Bully Election protocol requires less messages to reach a consensus which is useful for networks with less bandwidth. Considering these results, the simulation environment is successfully validated.

TABLE I: Election Protocol Statistics

Protocol	No. of Nodes	Mean	Std. Dev.	Avg. No. Exchanged Msgs
Bully Election	4 Nodes	4.2ms	2.1ms	27
	6 Nodes	26.2ms	14.1ms	40
	12 Nodes	226.3ms	130.6ms	242
Invitation Election	4 Nodes	0.91ms	0.48ms	109
	6 Nodes	45.7ms	7.7ms	751
	12 Nodes	192.3ms	60.8ms	15973

V. Project Management and Tools

This project was accomplished using Python. For theoretical concepts we resorted to the contents provided in the Distributed Systems course, at the Faculty of Engineering of University of Porto, coupled with the paper presented by Garcia-Molina [1], which greatly contributed to the success of this project, which we are thankful for.

The development of this project allowed the practical experimentation and analysis of the concepts of leader election lectured in the course. The division of grades is as follows:

- Tiago Sousa - 60%
- Hugo Cardoso - 40%

[2]This image was taken from the Distributed Systems course, at the Faculty of Engineering of University of Porto

A. Future Work

Three features were planned but not implemented. Firstly a Graphical User Interface (GUI) were the user could see messages sent and received, the status of the system and protocol and topology of the network as well as the leader elected in real-time. Secondly, implementing a weighed graph were the weights would be representative of the distance between nodes, which would negatively impact the time it would take to receive and send messages between nodes, simulating a real world problem where these distances could even decrease or increase by adjusting the weight in the graph. Thirdly, probability node crashes would also better translate the real world.

VI. Conclusion

The simulation environment proved to be capable of demonstrating how various factors can impact a leader election process. The primary focus was on analyzing the time taken to complete the election process under specific conditions and with different algorithms and the number of exchanged messages as a means to validate the deployment of the simulation environment that proved to be successful. By running simulations in a controlled environment, relevant measurements of election times could be gathered and key impactful factors that influence efficiency identified. With both algorithms reliably reaching consensus, even in the presence of node failures, the simulation environment allowed to compare them, while performing under consistent conditions, ensuring accuracy, repeatability and providing scalability.

References

[1] Garcia-Molina, "Elections in a distributed computing system," *IEEE Transactions on Computers*, vol. C-31, no. 1, pp. 48–59, 1982.