Report

Leader Election

Review, Implementation and Analysis

Project in Distributed Systems

Spring'23

Team:

Aparna Agarwal (2021121007)

Dhruv Hirpara (2020102029)

Harshita Gupta (2020101078)

Abstract

Leader election is an important problem in distributed computing, where nodes in a network elect one of themselves as a leader. The leader is responsible for performing certain tasks that require coordination and consensus. There are several algorithms for leader election, each with different characteristics and performance trade-offs. The goal of this project is to implement and experiment with different leader election algorithms for various topologies and study their performance in terms of the number of messages exchanged.

Keywords: Distributed Systems, Leader Election, bully algorithm, echo with extinction, Lehann-Chang Robots, FloodMax, Franklin, Hirschberg-Sinclair.

Contents

1. Introduction

In distributed systems, a leader election algorithm is used to choose a single process from a group of processes to act as the leader. The leader is responsible for coordinating the actions of the other processes and ensuring that the system operates correctly. In this project, we present an experimental approach based on an MPI (Message Passing Interface) implementation, with the goal to characterize the relevant evaluation metrics based on statistical processing of the results.

1.1 Problem Statement

We consider a set of N processes p1, ..., pN that (i) have no shared variables, and (ii) can communicate by message passing. From this set, we want to select one process to play a special role (e.g. coordinator), such that (i) every process pi should have the same coordinator, (ii) if the elected process fails a new election round is started. In order to be able to make this choice, we assume that each process has a unique ID, and the problem is to elect the process with the largest ID. This ID could e.g. be related to the inverse of the load on the process, in order to elect a relatively unloaded process. This problem statement implies that IDs are unique and that IDs are totally ordered.

1.2 Objectives

- To implement and test different leader election algorithms, including Bully Algorithm, LeLang Chang Roberts (LCR) and Hirschberg–Sinclair.
- Explore and analyse Synchronous and Asynchronous algorithms.
- To experiment with the implemented algorithms on different network topologies, including random, ring, and mesh topologies, and record the number of messages exchanged during the leader election process.

2. Related Work

The first leader election algorithm was designed for the ring network. In 1977, G. Le Lann proposed this algorithm which is also known as the Ring algorithm. It works on an asynchronous and unidirectional ring network. In 1979, Chang et al. improved the Ring algorithm and proposed the LCR algorithm. This algorithm also works on the unidirectional logical ring network. Though this algorithm reduces the message overhead of the election process and its time overhead is high. Hirschberg and Sinclair proposed the HS algorithm for a bidirectional ring network. This algorithm reduces the message complexity of the election process from $O(N^2)$ to $O(NlogN)$, but its time complexity is very high. Biswas and Dutta proposed the Timer-based Leader Election Algorithm for the unidirectional synchronous logical ring network.

No leader election algorithm has been designed considering the failure-recovery failure model for the ring network to elect a reliable leader. Most of the existing

algorithms considered that node or link failures or recovery do not occur during the election which is not the case in practical scenarios.

In 1982, Gracia Molina introduced the Bully algorithm that works on a complete mesh network. Kordafshari et al. presented the Modified Bully algorithm to overcome the drawbacks of the Bully algorithm. Mega-Merger is one of the earliest leader election algorithms designed for arbitrary network topology. Robert Gray Gallager developed it at MIT in 1983. The Yo-Yo is a leader election algorithm designed for arbitrary network topology. Unlike the Mega-Merger algorithm, Yo-Yo has simple specifications, and its correctness is simple to establish.

3. Evaluation Metrics

There are two important metrics to characterize the performance of algorithms for leader election:

3.1 Turnaround time

The time needed for the election (measured from the moment the election process is started until all processes have agreed on a common coordinator).

3.2 Bandwidth/Communication Complexity

The number of messages used to elect a new coordinator.

4. Ring Topology

Rings are a simple starting point to analyse leader election algorithms in distributed systems. The lower bounds and impossibility results for ring topology also apply to arbitrary topologies. There is no leader election algorithm for anonymous rings, even if the algorithm knows the ring size (non-uniform) and in the synchronous model.

4.1 LeLang Chang Roberts(LCR) Algorithm

Describe what you did in order to use the data for analysis.

4.2 Franklin Algorithm

Describe what you did in order to use the data for analysis.

4.3 Hirschberg–Sinclair(HS) Algorithm

Assumptions

- Bidirectional

- Don't know ring size
- Comparisons only for UID's

Types of Messages

- *send_data_tag*
- *return_data_tag*

Algorithm

Each process operates in phases 0, 1, 2... In each phase k, process i sends messages with *send_data_tag* with its UID in both directions to travel distance 2^k and return back to it. If both tokens return then process i continues in phase k+1.

- A process checks the received UID with its own when it gets an message with *send_data_tag* UID.
- If the received UID is smaller, then it discards it
- If the received UID is greater then
  - it passes it to the next process, if it is not the end of its path.
  - else it returns it back to the previous one and travels back to the originating process with tag *return_data_tag*
- If it is equal, then the process declares itself the leader

Complexity Analysis

A processor initiating messages along paths of length 2 i causes messages to emanate in both directions, and return

$$TOTAL = 4.(1.n + 2.(n/2) + \cdots 2^i.(n/(2^{i-1} + 1)) \dots) = O(n \log n)$$

Time Complexity: $O(n)$

5. Complete topology

a complete network refers to a network topology where each node is directly connected to every other node in the network. In other words, it is a fully connected network, where every node can communicate directly with every other node without the need for any intermediate nodes or links.

5.1 Bully Algorithm

Assumptions

- Each node has a unique ID.
- System is synchronous
- Every process knows the process number of all other processes
- Processes do not know which processes are currently up and which processes are currently down
- In the election, a process with the highest process number is elected as a coordinator which is agreed by other alive processes

Types of Messages

- *ELECTION*
- *OK*
- *COORDINATOR*

Algorithm

1. A process with ID i initiates the election
2. It sends *ELECTION* messages to all process with ID > i
3. Any process upon receiving the *ELECTION* message returns an *OK* to its predecessor and starts an election of its own by sending *ELECTION* to higher ID processes.
4. If it receives no *OK* messages, it knows it is the highest ID process in the system. It thus sends *COORDINATOR* message to all other processes
5. If it received *OK* messages, it knows it is no longer in contention and simply drops out and waits for an *COORDINATOR* message from some other process.
6. Any process that receives *COORDINATOR* message treats the sender of that message as coordinator

Complexity Analysis

When the process having the lowest priority number detects the coordinator's failure and initiates an election, in a system of n processes, altogether (n-2) elections are performed. For each election ever process sends the election message to each process greater than him and sends OK to each process lesser than him

$$NUM_{OK} = 0 + 1 + 2 \ldots n - 3 = \frac{(n-3)(n-2)}{2}$$

$$NUM_{Elections} = 0 + 1 + 2 \ldots n - 2 = \frac{(n-3)(n-2)}{2}$$

Message Complexity = $O(n^2)$
Time Complexity = O(3)

6. Arbitrary topology

6.1 FloodMax Algorithm

Assumptions

- Process is Arbitrarily connected
- Each Process Knows the diameter of the Graph
- System is synchronous
- In the election, a process with the highest process number is elected as a coordinator

Types of Messages

- *MAX_UID*

Algorithm

1. A process is made a simulator that calculates the Diameter using BFS traversals and sends it to the processes involved.
2. Each Process sends/floods max *MAX_UID* it knows so far to each process it is connected to.
3. Each time a process receives a UID it updates its *MAX_UID* to the maximum it has seen so far.
4. Above Two steps is repeated for the diameter number of rounds.
5. After completing the diameter number of rounds, each node will have the max UID it has seen so far which will also be the global maximum; thus every node will know who is the leader of the network.

Complexity Analysis

To elect a leader it takes

6.2 Echo with Extinction

Each process in a distributed system is allowed to initiate a run of the Echo Algorithm with its own process ID as the tag. The process with the smallest ID among all the initiators will complete its wave, and the process that initiated this wave becomes the leader.

Types of Messages

- *TOKEN*
- *LEADER*

Algorithm

- Each initiator starts a wave, tagged with its id by sending *TOKEN,ID* messages
- At any time, each process takes part in at most one wave.
- If a process p in wave caw_p is hit by a wave r:
  - if $caw_p > r$, then p changes to wave r (Reinitialize the algorithm);
  - if $caw_p < r$, then p continues with wave q (the message is ignored – extinction);
  - if $caw_p = r$, then the incoming message is treated according to the echo algorithm of wave caw_p
- I If wave p executes a decide event (at p), p becomes leader

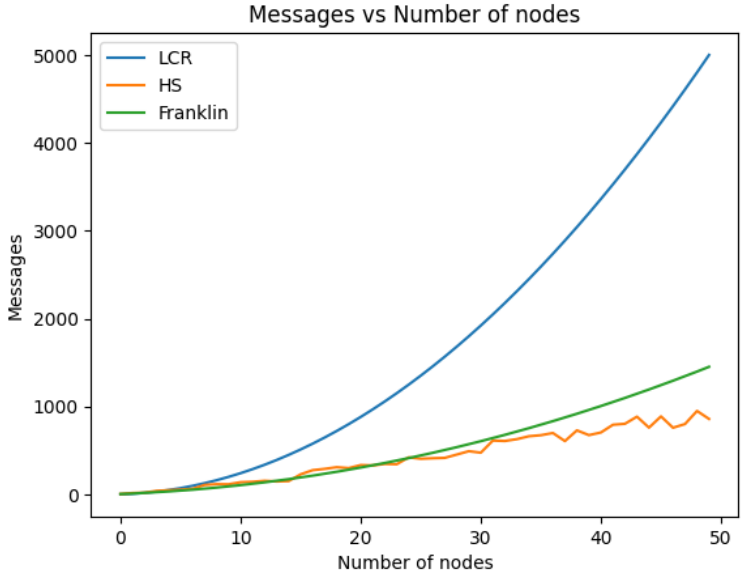
Complexity Analysis

When everyone initiates their own wave. Worst-case message complexity: $O(N.E)$

7. Results

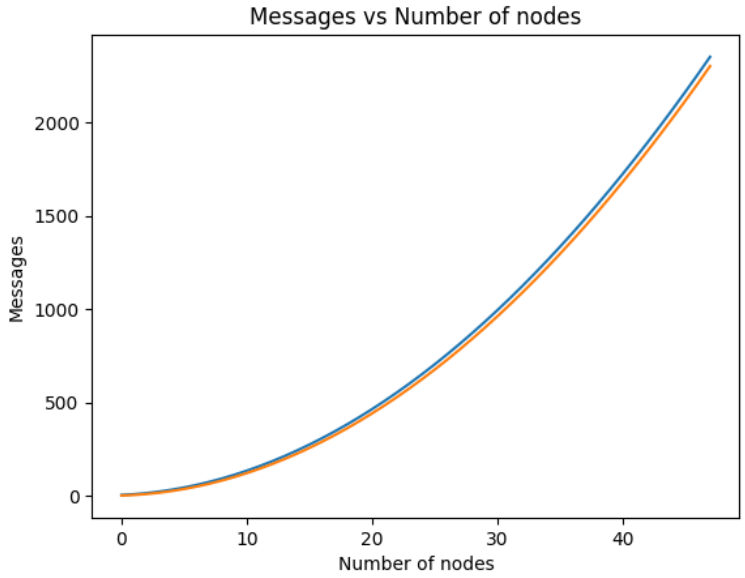
7.1 Evaluating the HS,LCR for ring Network

We have plotted num_processes vs num_messages graph for different algorithms from our simulations



7.2 Complete Network

In this we have implemented bully for which we have plotted graph for worst possible



case

7.3 Arbitrary Topology

We have choosed random graphs and compared the number of message transmitted

Nodes	Echo	Flood_max
7	52	56(diameter:4)
8	83	72(diameter:3)
9	131	120(diameter:4)

8. Limitations and Challenges

What could have been investigated if given more time? What have been difficult when solving the problem and getting answers for your research questions/hypotheses?

9. Conclusion

Write the conclusion. What did you gain from the project assignment? Briefly explain your questions/hypotheses, findings, and meaningful discussion points in relation to the data collection, data management, data analysis, visualization and interaction concepts, and evaluation of your tool. What additional investigations need to be performed (or what is the limitation) in order to say that your solution is a good one for the problem?

10. References

- Le Lann, G. (1977). Distributed systems-towards a formal approach. In IFIP congress, volume 7, pages 155–160. Toronto.
- Chang, E. and Roberts, R. (1979). An improved algorithm for decentralized extrema finding in circular configurations of processes. Communications of the ACM, 22(5):281–283.
- Hirschberg, D. S. and Sinclair, J. B. (1980). Decentralized extrema-finding in circular configurations of processors. Communications of the ACM, 23(11):627–628.
- Biswas, A. and Dutta, A. (2016). A timer-based leader election algorithm. In Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016 Intl IEEE Conferences, pages 432–439. IEEE.
- Garcia-Molina, H. (1982). Elections in a distributed computing system. IEEE transactions on Computers, 31(1):48–59.
- Gallager, R. G., Humblet, P. A., and Spira, P. M. (1983). A distributed algorithm for minimum-weight spanning trees. ACM Transactions on Programming Languages and systems (TOPLAS), 5(1):66–77.
- MPI-based Evaluation of Coordinator Election Algorithms
- Wikipedia Leader election
- Some observations on leader election algorithms for distributed systems