An Augmented Version of The Chang and Roberts Algorithm

David Brown
School of Computing Science
University of Glasgow
Glasgow, United Kingdom
2137184b@student.gla.ac.uk

Joseph Cameron
School of Computing Science
University of Glasgow
Glasgow, United Kingdom
2117625c@student.gla.ac.uk

Yiannis Kathidjiotis
School of Computing Science
University of Glasgow
Glasgow, United Kingdom
2082011k@student.gla.ac.uk

Maria-Luiza Koleva
School of Computing Science
University of Glasgow
Glasgow, United Kingdom
2127555k@student.gla.ac.uk

Bryan Mutai
School of Computing Science
University of Glasgow
Glasgow, United Kingdom
2131791m@student.gla.ac.uk

***Abstract*—Leader election algorithms are crucial mechanisms which distributed systems use to achieve coordination and agreement. Several mutual exclusion algorithms need to elect a leader and use a leader election algorithm to accomplish this. One such algorithm is the Chang and Roberts algorithm. In this paper, an augmented version of the Chang and Roberts algorithm is proposed that achieves significantly better performance in a single-election worst-case scenario. The implementation of the two versions is then discussed, using Java RMI. Evaluation has been carried out that illustrates the predictions of the algorithms' complexity. The augmented algorithm sends $2N$ messages for a single election in both the worst case and best case scenarios, while the original algorithm has a worst-case complexity of $3N-1$.**

***Index Terms*—Algorithm, Leader Election, Ring-Based, Complexity, Distributed Computing Systems**

I. Introduction

The goal of distributed computing systems is to connect users and resources transparently. To achieve this goal, the possibility of many individual processes located in various geographical locations must be accounted for. The widespread nature of distributed systems proposes many challenges, one of which is coordination. It is essential to enable distributed processes with the ability to coordinate their actions, particularly when these processes may be accessing shared resources within the distributed system. Without any system coordination, shared variables would likely become corrupt.

One way in which distributed systems achieve coordination is through ring-based algorithms or the central server algorithm. These algorithms aim to achieve mutual exclusion (thus coordination), however, they must usually select an initiating process to start the algorithm. For example, the central server algorithm must first elect a central server by utilizing a leader election algorithm. Leader election algorithms are distributed algorithms that elect a single process to become a coordinator (or 'leader') of a distributed task, thus solving the leader election problem. The leader election problem is considered solved if the following requirements are met:

- Safety: The process with the largest identifier at the end of the run (that has not crashed) must be elected as the leader. All other processes must know which process is the leader.
- Liveness: All processes must participate and one process will be elected as leader eventually.

Furthermore, leader election algorithms are considered valid if they achieve:

- Termination: The algorithm finishes execution within a finite amount of time.
- Uniqueness: Only one leader is elected.
- Agreement: The leader is known to all the other processes in the system.

One of the most familiar leader election algorithms is the Chang and Roberts algorithm [2]. The Chang and Roberts algorithm is a uni-directional ring-based algorithm that complies with the safety and liveness requirements of the leader election problem. It also achieves termination, uniqueness and agreement.

However, upon analysis of this algorithm's complexity, performance issues can be observed. The best case complexity of the Chang and Roberts algorithm with respect to the number of messages sent is $O(2N)$ where N represents the number of nodes that can take part in an election. This best case scenario occurs when the true leader starts an election. Conversely, the worst case scenario occurs when the true leader is situated behind the node that starts an election. The worst case complexity is $O(3N-1)$. For a large number of nodes, the difference in message complexity can significantly compromise the overall performance of the algorithm. Furthermore, the extra messages sent in the worst case may be unnecessary. To address this issue, an augmented algorithm was designed, which is the subject of this paper.

II. Algorithm Design

One of the most popular leader election algorithms is the ring-based election algorithm known as the Chang and Roberts algorithm [2]. The Chang and Roberts algorithm assumes that all processes within a distributed system have a form of unique identification and that all processes can organize themselves into a ring structure with a clockwise direction [3].

The pseudocode for the Chang and Roberts Algorithm is detailed in Algorithm 1. An election is started by a node sending an election message to its clockwise neighbour and communicating which node it believes should be the leader. Each subsequent node forwards the message until the node with the highest ID receives a message saying it should be the leader. At this point, the node with the highest ID decides it is the leader and sends an elected message to its next neighbour. The elected messages notify each node in the ring who the leader is and establish the property of Agreement.

Algorithm 1: Chang and Roberts Algorithm

```
Input: n processes arranged in a logical ring
Output: n processes arranged in a logical ring which
        have agreed on an elected leader
for each process Pk do
    participant = false;

    \\To start an election
    if not Pk.participant then
        Pk.participant=true;
        sendElectionMessage(Pk.id);
    end

    \\When receiving an election message msg
    if not Pk.participant then
        Pk.participant=true;
        sendElectionMessage(max(pk.id,
          msg.leaderId));
    end
    else if msg.leaderId == Pk.id then
        Pk.participant=false;
        Pk.leaderId=msg.leaderId;
        sendElectedMessage(Pk.leaderId);
    end
    else
        if msg.leaderId >Pk.id then
            sendElectionMessage(msg.leaderId);
        end
    end

    \\When receiving an elected message msg
    Pk.leaderId=msg.leaderId;
    Pk.participant=false;
    if msg.leaderId!=Pk.id then
        sendElectedMessage(msg.leaderId)
    end
end
```

The Chang and Roberts algorithm as explained above is used as a basis for the augmented version presented in this report. The augmented algorithm makes the same assumptions

Algorithm 2: Augmented Chang and Roberts Algorithm

```
Input: n processes arranged in a logical ring
Output: n processes arranged in a logical ring which
        have agreed on an elected leader
for each process Pk do
    participant = false;

    \\To start an election
    if not Pk.participant then
        Pk.participant=true;
        sendElectionMessage(Pk.id, Pk.id);
    end

    \\When receiving an election message msg
    if not Pk.participant then
        Pk.participant=true;
        sendElectionMessage(msg.starterId, max(pk.id,
          msg.leaderId));
    end
    else if msg.starterId == Pk.id then
        Pk.participant=false;
        Pk.leaderId=msg.leaderId;
        sendElectedMessage(Pk.id, Pk.leaderId);
    end
    else
        if msg.leaderId >Pk.id then
            sendElectionMessage(msg.starterId,
              msg.leaderId);
        end
    end

    \\When receiving an elected message msg
    Pk.leaderId=msg.leaderId;
    Pk.participant=false;
    if msg.starterId!=Pk.id then
        sendElectedMessage(msg.starterId,
          msg.leaderId)
    end
end
```

as the original. Both versions of the algorithm assume no failures since if any node in the ring fails, the communication channel between all nodes would be broken [4]. This follows from the fact that the communication is uni-directional [4].

Similarly to the original, the augmented algorithm uses election and elected messages. Election messages for the augmented version are of the following format:

- starterId - ID of the process that started the election.
- leaderId - ID of process that is the current suggested leader.

Elected messages are sent when it is established which node should be the leader. This is where our augmented algorithm differs from the original. They are of the following format:

- starterId - ID of process which started the election.
- electedId - ID of process elected to be leader.

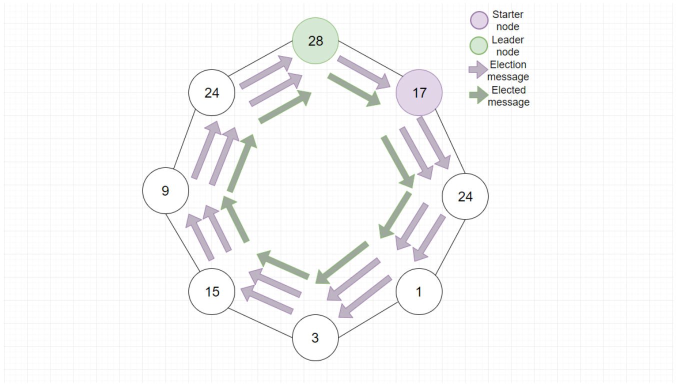


Fig. 1. Chang and Roberts algorithm, single election, worst-case

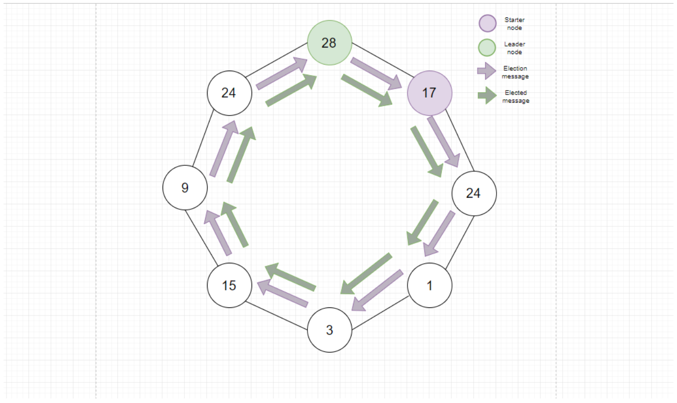


Fig. 2. Augmented Chang and Roberts algorithm, single election, worst-case

Any process in the ring can initiate an election, provided that it is not already a participant in one. Each process has a flag that marks whether it is a participant in an on-going election. If a node is aware of an election in progress, then it knows a leader will be elected.

The pseudocode of the augmented algorithm is detailed in Algorithm 2. There are several differences between the augmented and original version, they have been highlighted. Firstly, each message sent carries more information. The processes now need to communicate not only the leaderId but the starterId as well. The second change is that the process that started the election now decides who the leader should be. The starter process is then responsible for communicating this decision around the ring as the leader does in the original version of the algorithm.

These differences were introduced as part of the design of the augmented algorithm because they positively affect the complexity in relation to the total number of messages sent. The identified changes contribute to the augmented algorithm having a lower complexity in a subset of cases detailed below, while retaining the original version's complexity in all others.

Assuming a single election, the Chang and Roberts algorithm has a worst-case complexity of $(3N-1)$, where N is the number of nodes in the ring [4]. In the worst case, the node which should be elected as the next leader is the direct predecessor of the node which initiates the election. Thus, it takes $N-1$ election messages to reach the node with the highest ID, another N election messages until the node in question receives its own ID as an incoming message [2]. After the leader is determined, N elected messages are sent to communicate to all other nodes who the leader is [4]. This case is illustrated for a ring consisting of eight nodes in Fig.1.

In contrast, the augmented version of the algorithm has a constant total number of messages sent. The number of election messages sent is N as the starter node determines who the next leader should be. N elected messages are then sent to notify all other nodes of this decision. Thus, the worst-case complexity of the augmented algorithm is $O(2N)$. This scenario is illustrated in Fig. 2.

The best cases for both versions of the algorithm are the same, since this scenario involves the node which should be the next leader initiating an election itself. Thus, the best-case complexity for a single election is $O(2N)$.

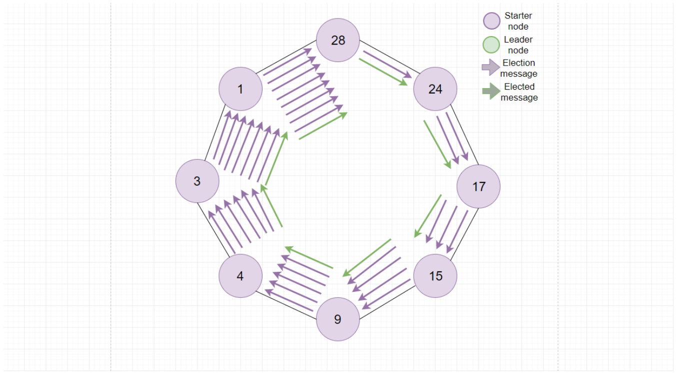


Fig. 3. Chang and Roberts algorithm, multiple concurrent elections

For multiple elections, the augmented algorithm performs the same as the original algorithm. The worst-case for multiple concurrent elections is if all nodes in the ring initiate an election at the same time. Moreover, the worst performance would be observed in a ring where each node is a predecessor to the node with the next highest ID. In such a ring, when travelling from the node with the highest ID in the direction of communication, the IDs would be descending until the node with the lowest ID, which would be the predecessor to the starting node. This configuration is illustrated in Fig.3.

Each node sends the next a message saying that it thinks it should be the leader, starting an election. These are the first N election messages. At the second iteration, each node receives a message from its predecessor that claims it should be the leader. Since all nodes except node 28 have a predecessor with a higher ID, they forward the election message they just received. These messages are $N-1$ because node 28 doesn't send one. In the third iteration, all nodes except node 24 receive an election message. Again, all nodes except node 28 forward the message. The number of messages in this iteration is $N-2$. The election process continues and at each iteration, the number of messages sent is smaller by 1, until node 28 decides that it should be the leader. Thus, the total number of election messages sent is the sum of the numbers from $1 to N$. This is equal to $N(N+1)/2$ [7]. N elected messages are then sent as in the previous scenarios discussed. It can be concluded that the worst-case complexity for multiple concurrent elections is $O(N^2)$ [5].

III. Implementation

Both versions of the algorithm have been implemented to facilitate the comparison between them. Each node is represented by a remote object, adhering to the remote object interface Node. There are two implementations - one for each version of the algorithm, namely *AugmentedAlgorithm* and *OriginalAlgorithm*.

Helper classes were then created to assist with forming the ring of nodes and starting elections. The *InitializeRing* class is used to create and register a single node. It requires an input of a unique ID and indication of the leader election algorithm the node should follow. The ID of the next node in the ring can be optionally specified.

It is expected that the first node created will not be able to obtain a reference to its neighbour as it will not have been instantiated. Thus, to connect the complete ring, *ManualStartElection* is used. It takes as inputs the IDs of two existing nodes and sets the second one to be the next to the first one. The first node then starts an election to demonstrate the execution of the algorithm it implements.

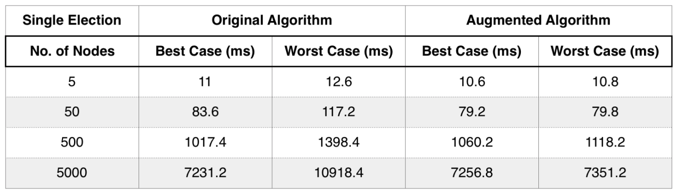
The above code base is sufficient to run a number of nodes on different machines as separate processes and observe the behaviour of the two algorithms. It was recognized that to properly evaluate the performance of the original and augmented versions, rings with a large number of processes would have to be created. Thus, it was concluded that simulating the execution of the algorithms and comparing them against each other would be a suitable approach to evaluation. This was achieved by the *TestAlgorithm*, which is responsible for creating and registering all nodes in a ring.

To fully evaluate the different versions of the algorithm, multiple concurrent elections need to be examined. *StartElectionRound* builds upon *ManualStartElection*. It focuses solely on starting elections by taking as inputs an arbitrary number of node IDs which should all start concurrent elections. Each election was started in a separate thread through the use of a *Runnable*. The benefit of this approach is that per election round, only one more 'thread' is created. For five hundred nodes with two election rounds, it would only require two additional threads. This decision allows scaling the number of nodes well into the hundreds without worrying about the host system adding a bottleneck to testing.

When introducing multiple threads, the problem of synchronization must be addressed. All nodes use a shared integer variable *messageCount* to count the total number of messages sent in the course of the election. The thread-safe update of this variable was achieved by initializing it as *AtomicInteger* [1]. This class guarantees synchronization of the updates and reads of the integer it contains.

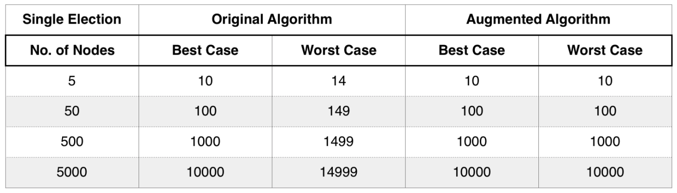
IV. Evaluation

After the program had been implemented, testing was performed to demonstrate that the algorithm had actually been improved. Both algorithms were tested over an increasing number of nodes n, in best-case as well as worst-case scenarios as described in section Algorithm Design. A single election was executed five times for every n, and the execution time along with the number of messages sent was noted. For every set of five results, an average was calculated. The results for single elections can be seen in Figures 4 and 5.



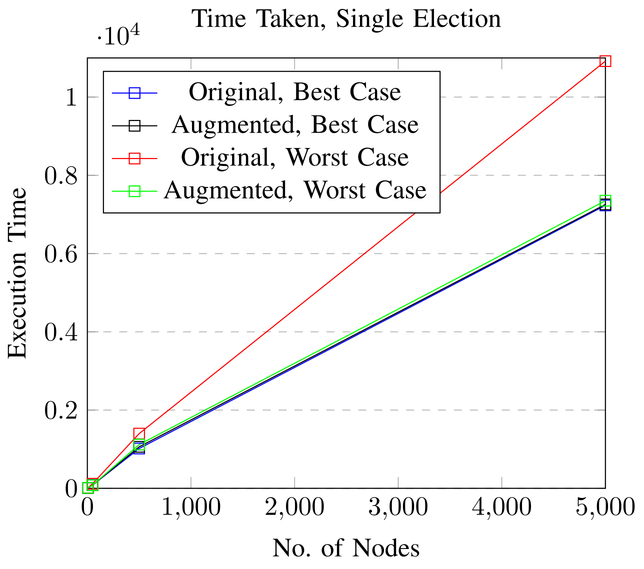
Single Election	Original Algorithm		Augmented Algorithm	
No. of Nodes	Best Case (ms)	Worst Case (ms)	Best Case (ms)	Worst Case (ms)
5	11	12.6	10.6	10.8
50	83.6	117.2	79.2	79.8
500	1017.4	1398.4	1060.2	1118.2
5000	7231.2	10918.4	7256.8	7351.2

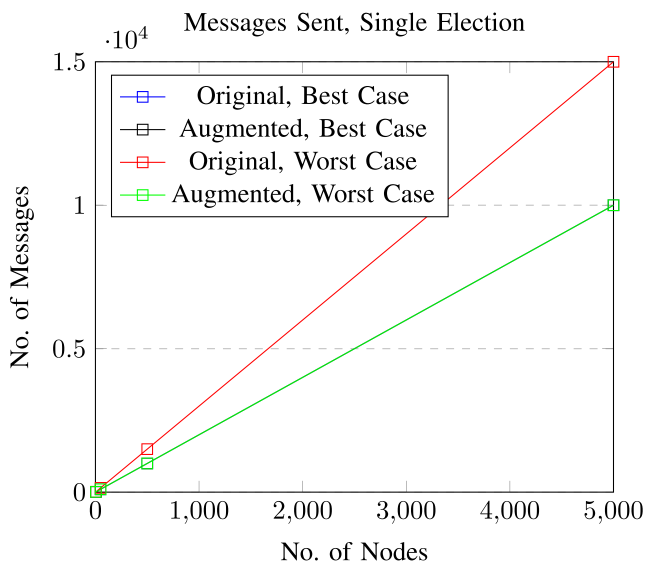
Fig. 4. Original Algorithm vs. Augmented Algorithm (Single Elections), Time Taken



Single Election	Original Algorithm		Augmented Algorithm	
No. of Nodes	Best Case	Worst Case	Best Case	Worst Case
5	10	14	10	10
50	100	149	100	100
500	1000	1499	1000	1000
5000	10000	14999	10000	10000

Fig. 5. Original Algorithm vs. Augmented Algorithm (Single Elections), Messages Sent





In Figure 4 the times of the two algorithms are compared between the best case and the worst case. While the best-case execution times between the two algorithms are very similar, the augmented algorithm's worst-case execution times improved. This improvement is due to the augmented algorithm's number of messages sent being constant from best to worst case, as demonstrated in Figure 5.

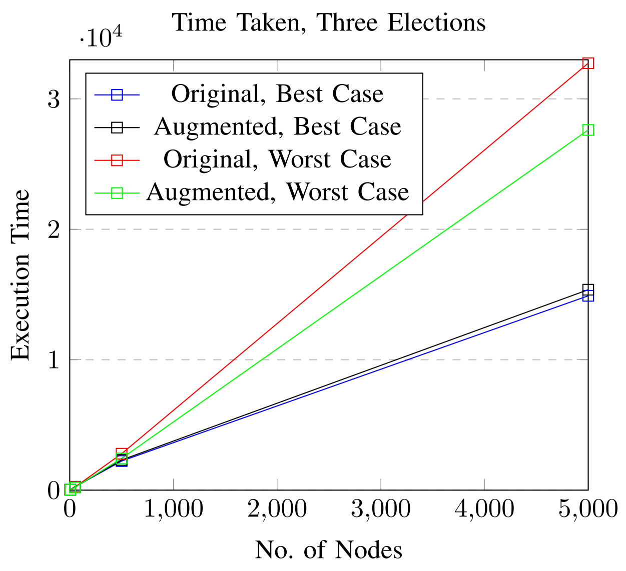
In the scenario where multiple elections are initiated simultaneously, Figures 6 and 7 demonstrate the behaviour of the original algorithm against the augmented algorithm. The evaluations were carried out with three nodes starting concurrent elections. In this setting, the best case is when nodes with IDs n, $n-1$ and $n-2$ start an election while the worse case is when nodes with IDs 1, 2 and 3 start an election. Regarding the augmented algorithm, it can be seen that while the number of messages sent differs between the best-case and the worst-case scenarios, there is still an overall improvement in both time and message complexity.

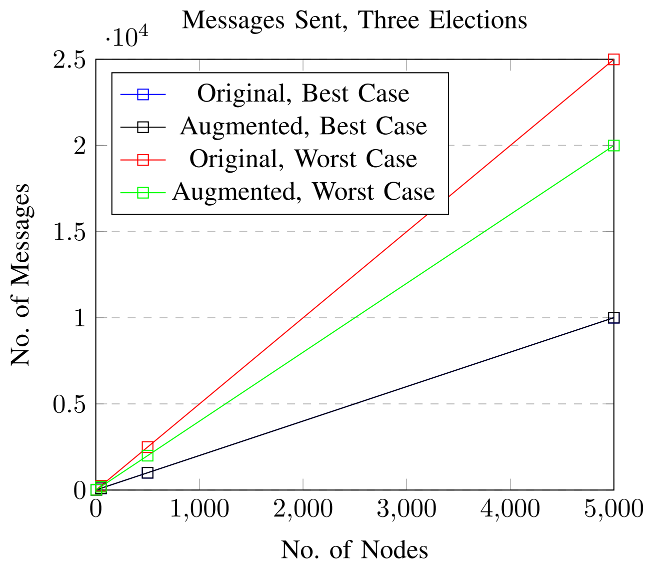
Three Elections	Original Algorithm		Augmented Algorithm	
No. of Nodes	Best Case (ms)	Worst Case (ms)	Best Case (ms)	Worst Case (ms)
5	28.4	28.8	30.4	30
50	252.8	228.4	258.6	208.2
500	2231.8	2792.4	2312.4	2430.8
5000	14899.8	32734	15368	27618

Fig. 6. Original Algorithm vs. Augmented Algorithm (Multiple Elections), Time Taken

Three Elections	Original Algorithm		Augmented Algorithm	
No. of Nodes	Best Case	Worst Case	Best Case	Worst Case
5	13	19	13	17
50	103	244	103	196.6
500	1003	2494	1003	1995.8
5000	10003	24994	10003	19995.4

Fig. 7. Original Algorithm vs. Augmented Algorithm (Multiple Elections), Messages Sent





V. Related Work

This section highlights two other leader election algorithms that exist.

A. *The Bully Algorithm*

The Bully algorithm uses time-outs to detect process failure in a synchronous system, where the proposed augmented algorithm does not require the system to be synchronous. The Bully algorithm requires each process to know which processes have the highest identifiers and which is the current coordinator, whereas, in the augmented algorithm, only the starterID and the process with the highest ID at each point of the election is needed. The Bully algorithm will initiate an election when the coordinating process is known to have failed. A process with a lower identifier will send an election message to the processes known to have higher identifiers and wait for an answer message, if it does not receive one, it declares itself the coordinator and sends a coordinator message to all processes with a lower identifier. If it receives an answer message, it waits for a coordinator message, where if it won't arrive it will begin another election. This results in a $O(N^2)$ complexity for a worst-case scenario. If the process declaring an election is the one with the highest identifier, it declares itself as the coordinator and sends a coordinator message to the other processes that have not failed. This results in $O(N-2)$ complexity for a best-case scenario. Therefore the augmented algorithm achieving the same time and message complexity between best and worse-case scenario, leads to a better worse case complexity than the Bully algorithm but not a better best case complexity. In addition, safety may be at risk if the timeout values for detecting failure are broken, where in the proposed algorithm this can't be an issue as timeouts do not exist.

B. The HS Algorithm

[8] This leader election algorithm functions on a bi-directional synchronous ring. This algorithm operates in phases, starting at phase 0. In each phase k, a process sends its unique ID in both directions to travel distance 2^k and return back. If both tokens return, then proceed to phase $k+1$. When receiving a message, a process compares the receiving ID with its own:

- if ID is smaller, the sending process is discarded and won't send out its ID anymore.
- if ID is greater, it will be passed on to the next process as long as it is not the end of the path, where it is passed back.
- if the two IDs are equal, process declares itself the leader and sends a final round of messages informing every other process if it's leader status.

The use of a bidirectional ring causes a better overall complexity, time complexity being $O(N)$ and the message complexity $O(nlog(n))$ [6]. Each node in the HS algorithm has to store references to both its neighbours, resulting in a higher space complexity than a uni-directional ring.

VI. Conclusion

The goal of this research was to improve on the Chang and Roberts algorithm performance by proposing a fundamental change in its design, augmenting how the election process works and enhancing its performance under select cases. The modifications involved including a starterId in messages sent between the processes, adding a mechanism for the process that initiated the election to determine the leader instead of the leader itself doing so, as well as communicating this decision around the circle. These changes contribute to the augmented algorithm having a lower complexity in a subset of cases while retaining the original version's complexity in all other cases. These changes positively affect the complexity regarding the total number of messages sent which positively influenced the time taken for an election to take place. These improvements were reflected in the evaluation results in figures 4 and 5. All while still satisfying the safety and liveness requirements as well as maintaining the leader election validity checks for termination, uniqueness and agreement.

A primary concern is the lack of fault tolerance while transmitting messages across the processes. Significant changes would have to be applied to achieve this goal, as the lack of fault tolerance is a fundamental property of uni-directional rings. Finding a solution incorporating fault tolerance could be included in future work on the Chang and Roberts algorithm.

The design, implementation and evaluation detailed in this report have shown how simple changes in algorithm design can significantly affect performance.

References

[1] *Class AtomicInteger. AtomicInteger (Java SE 9 & JDK 9),* 2017, docs.oracle.com/javase/9/docs/api/java/util/concurrent/atomic /AtomicInteger.html.

[2] Chang, Ernest, and Rosemary Roberts, *An Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processes. Communications of the ACM, vol. 22, no. 5,* Jan. 1979

[3] Chen, Wei-Mei. *Cost Distribution of the Chang-Roberts Leader Election Algorithm and Related Problems. Theoretical Computer Science, vol. 369, no. 1-3,* 2006, doi:10.1016/j.tcs.2006.08.019.

[4] Coulouris George, Dollimore Jean, Kindberg Tim, Blair Gordon, *Distributed Systems: Concepts and Design (Fifth Edition),Pages 644-646,* 2012

[5] Dolev, Danny, et al. *An O(n Log n) Unidirectional Distributed Algorithm for Extrema Finding in a Circle. Journal of Algorithms, vol. 3, no. 3,* 1982, pp. 245260., doi:10.1016/0196-6774(82)90023-2.

[6] Lynch Nancy A, *Distributed algorithms,Lecture Notes for 6.852, Pages 25-27,* 1993.

[7] Franklin, Randolph. *On an Improved Algorithm for Decentralized Extrema Finding in Circular Configurations of Processors. Communications of the ACM, vol. 25, no. 5,* Jan. 1982, doi:10.1145/358506.358517.

[8] Hirschberg Daniel S, Sinclair James Bartlett. *"Decentralized Extrema-Finding In Circular Configurations Of Processors",Communications of the ACM, vol.23, pages 627-628* Nov. 1980, doi:10.1145/359024.359029.