CHAPTER 3
MESSAGE EFFICIENT RING LEADER ELECTION IN DISTRIBUTED SYSTEMS

3.1 INTRODUCTION

Leader Election is a vital and fundamental problem in distributed systems and in any communication network. Distributed Systems is a collection of heterogeneous systems which interact with each other through messages. The main objective of Distributed System is, though there are heterogeneous systems in the network, it creates a single system image or uniprocessor image to the user, through various transparency metrics. The communication between the processes is achieved by exchanging messages. The software of the Distributed System is tightly coupled and the processes of the system coordinate with each other. They have lots of resources in common and so mutual exclusion algorithms are used to take care of the critical regions. While they wait for the common resources, they might end up in a deadlock. Deadlock detection and prevention algorithms should keep an eye on the resources and if there is a deadlock wound wait or wait-die algorithms are used to kill the eldest or the youngest process to remove the deadlock.

The replicated data management, group communication, atomic commit protocols, etc needs the process coordination. All the above stated protocols need a particular process among the group, to be the leader to have the control over the situation. In general, the process with the highest process-id is the coordinator or the leader. Any process, which satisfies the rule, can become the leader and the only issue in distributed system is that at any point of time there should be a unique process available as the leader to do the coordination and all the other processes should agree up on the present leader, without any confusion.
If the processes of the distributed system never fail, then the leader process can be decided at the time of the process group gets generated. But, there are systems where processes keep coming and leaving in the group and processes do crash. Specially, in Wireless networking like Wireless LAN, Satellite oriented Services cellular phones, the mobile systems are subject to loss of messages or the data and the mobile host can crash or can be down for some time. Electing a leader process is a basic operation which happens in the system very often. Many researchers have contributed a lot of paradigms to simply this task. Different kinds of leader election algorithms have been proposed and most of them are widely in usage.

3.2 BACKGROUND

3.2.1. Election Algorithm

Election algorithm is a special purpose algorithm, which is run for selecting the coordinator process among N number of processes. This coordinator or leader process plays an important role in the distributed system to maintain the consistency through synchronization. For example, in a system of client and server, the mutual exclusion algorithm is preserved by the server process $P_s$, which is chosen from among the processes $P_i$ where $i=1, 2, ..., N$ that are the group of processes which would use the critical region of a particular resource with mutual exclusion. Election Algorithm is needed in this situation to choose the server process among the existing processes. Eventually all the processes must agree upon the leader process. If the coordinator process fails due to various reasons, then immediately the election should happen to choose another alive process as leader to take up the job of the failed leader.

In a group of processes, any process can initiate the election algorithm whenever it encounters the failure of leader process. A process comes to know the failure of the leader when there is no reply from the leader process for any
synchronization queries. Clocks of such processes should be in synchronization and the network with which the messages are exchanged should also be reliable.

There can be situations that all N processes could call for an election from its side leading to N concurrent elections. At any time, process P_i is one among the following two states: when an election happens:

**Participant** refers to the process is directly or indirectly involved in election algorithm.

**Nonparticipant** refers to the process in not engaged with the election algorithm currently.

The goal of Election Algorithm is to choose and declare one and only process as the leader even if all processes participate in the election. And at the end of the election, all the processes should agree upon the new leader process without any confusion.

Without loss of generality, the elected process should be the process with the largest process identifier. This may be any number representing the order / time of birth / priority / energy of the process which should be unique in nature. Each process has a variable called LEAD, which contains the process identifier of the current leader. When the process participates in the election, it sets this LEAD to NULL.

Any Election Algorithm should satisfy the following two properties [93].

**Safety:** Any process P, has LEAD = NULL if it is participating in the election; If there is no election happening at present, then it’s LEAD = P, where P is the highest PID and it is alive at present.

**Likeness:** All the processes should agree on the chosen leader P after the election. That is, LEAD = PID_x for all the processes P_i where i=1,2,...,N and x is the process with the highest process identifier.
The Bully Election Algorithm [94] of Garcia Molina in 1982, elects the leader process uniquely which satisfies the safety and likeness requirements.

The numbers may be allocated in simply numerical order of the Ethernet address or some other numbers such as priority, the mere process id, etc. The Ring Election Algorithm [95] is based on the ring topology with the processes ordered logically and each process knows its successor in a unidirectional way, either clockwise or anti-clockwise.

When an alive process comes to know that the coordinator process has crashed, it creates an election message say ELMSG by inserting its own PID in it and sends the message to its immediate next node. If the successor is also down, the message would skip that process and goes to the next process of the successor or to the next etc., till it reaches a process which is not dead, along the ring network. When the ELMSG is received by any process, it appends its PID to the process list in the message. Like this, all the available processes in the ring would insert their respective PID in the list. Finally, the ELMSG comes back to the process which initiated the message and the process too would recognize that it only had initiated that message, by finding its own PID in the list.

When it receives the ELMSG filled with all the participants of the election, the election initiator process analyses the message and finds the highest PID among them, converts the ELMSG into coordinator message say COMSG and removes all the PIDs from the list but the highest PID. This COMSG message is circulated along the ring for a full circulation to inform the running processes about who the new coordinator is. Seeing this message, all the processes change their LEAD variable with this identifier of COMSG. After the full circulation, when it comes to the process initiator, it discards the COMSG and resumes to its work, The Election Algorithm ends here.

When the ELMSG comes back to the process that started it:
• The process sees its identifier in the list and so it stops the circulation of the message.
• It checks all the PIDs and decides the coordinator by selecting the process with the highest identifier.
• It changes the message type to COMSG and enters the PID of coordinator process in the message.
• COMSG is circulated along the ring.
• When it comes back to the process that started it and it gets discarded there.

3.2.2 Limitation of Ring Election Algorithm

During the worst case, multiple election messages may happen in parallel when more than one process detects the failure of the coordinator process.

Figure 3.1. Ring Election Algorithm with simultaneous elections

Figure 3.1 shows a typical scenario of such election. Here there are two elections happening simultaneously started by the processes 5 and 2. Though process 5 has already started the election, it still participates in the election of process 2. This leads to a lot of overhead to each of the process due to the creation and servicing of each and every election message. This also causes heavy network traffic and sometimes congestion in the network.

In the best case, when a single process detects the crashing of the leader, the message overhead due to election $N_E$ is obtained with $O(n)$ as follows:
\[ N_E = n_e + n_c. \] (1)

Where \( n_e \) refers to the number of ELMSG and \( n_c \) refers to the number of COMSG.

In the average and worst case, when all the N processes start the election message, \( N_E \) is obtained with \( O(n^2) \) from the following equation.

\[ N_i = N(n_e + n_c). \] (2)

This message complexity will drastically bring down the entire system's performance, as all the processes spend quite a lot of amount of time in the creation and servicing of these messages. In order to bring up the performance even during election, we need to have exactly only one complete election happening instead of simultaneous redundant elections. All the redundant election messages except the original one need to be killed. Through some way if the system differentiates the ELMSG as original and the redundant ones, then it is very simple to destroy the unwanted ELMSGs. For solving this problem we have proposed an improved election algorithm which uses the synchronized clock time to identify and kill the duplicate ELMSGs and thus yields better performance for the system. The following section discusses the proposed election model.

### 3.3 MESSAGE EFFICIENT RING ELECTION ALGORITHM (MEREA)

In the previous section we have seen that in the average and the worst case scenarios, the number of messages that are exchanged between processes is high in the original Ring Algorithm. Therefore it imposes heavy traffic on the network. The proposed algorithm tries to intensively eliminate the redundant Election messages.

#### 3.3.1 Assumptions
1. All the processes in the distributed group should have their clocks synchronized to each other. We have logical clock and physical clock synchronization algorithms namely Lamport’s algorithm for the logical clock synchronization and Cristian’s and Bekeley algorithms for the physical clocks or the real time clocks.

2. The Network is perfect. (i.e.) when any message is sent, it won’t be lost or modified. It would reach the destination in the predetermined time. If the destination process is alive, it can see the message which was sent to it. Here too, we have reliable primitives to keep the network perfect.

Table 1.1: MEREA Algorithm

```
Begin
set COORD = NULL
set FLAG = TRUE
set T_{elec} = current time
Step1: call Build ELMMSG function
Step2: for k: = 1 to n – 1
          send ELMMSG to SUCC_i
          call update ELMMSG
        next i
end for
Step3:
        build COMSG
        set COORD = LEAD
```
Step4: for \( k = 1 \) to \( n-1 \)

Send COMSG to SUCC\(_i\)
call process COMSG

\textit{next} \( i \)

\textit{end for}

\textit{end}

The proposed Message Efficient Ring Election Algorithm is given in Table 1.1 through 1.5. When process \( P_i \) realizes that the coordinator has crashed, it initiates the Message Efficient Election Algorithm by building the ELMMSG.

**Table 1.2: Function: Build ELMMSG**

```plaintext
input Current Time, PID
begin
Create ELMMSG
Set \( T = \) Current Time
begin
\textbf{if} (FLAG = TRUE)
\textbf{then} \textbf{if} (T > T_{\text{elec}})
then destroy ELMMSG
return
\textbf{end if}
\textbf{else}
set FLAG = TRUE
set COORD = NULL
append PID to Participants
\textbf{end if}
\textbf{end}
```
Each process has three significant log information with it like COORD, FLAG and $T_{elec}$. COORD has the PID of the present coordinator process and the FLAG which is of Boolean data type has the value of either True or False. If it is False it means either there is no election presently initiated by the particular process or it has not yet participated in any election recently. If it is set to True, then it is involved in electing the coordinator process either by initiating it or by participating in it. If so the COORD value will be set to NULL automatically as there is no coordinator presently available. When election process ends once again this data member is set to False. The last data item $T_{elec}$ has set to NULL initially and whenever the process initiates the ELMMSG, it would be set to the time of creation of that ELMMSG.

The data structure $ELMSG$ has two main components namely

- Creation Time of the election message. This is of constant data type which gets assigned in the build ELMMSG function
- List of Participants. This is of array data type of N members, initially with one element and keeps appended as ELMMSG gets serviced along the ring.

As $P_i$ initiates the election process, it first sets its COORD to NULL, its FLAG to True and $T_{elec}$ to the current time of the system. It then calls a function to build the ELMMSG by setting $T$ to the current time of the system which is a synchronized one. It also inserts its PID to the Participants list in the first place and circulates the message along the ring by throwing it to its immediate successor.
All the processes in the ring receive this ELMMSG and service it by checking the FLAG value of it. If it is already true and if the time of creation of its ELMMSG is earlier than the received ELMMSG, the process immediately recognizes this message as duplicate and destroys it instantly. If not, it sets its FLAG to true, COORD to NULL and appends its PID in the list.

Table 1.4: Function: Build COMSG

```
begin
LEAD = highest PID in the list of ELMSG
delete all the PID except LEAD
change ELMMSG to COMSG
return
end
```

Table 1.5: Function: Process COMSG

```
begin
set COORD = LEAD
set Flag = FALSE
set T_elec = NULL
return
end
```

When all active processes enter their PIDs in the participants list, the ELMMSG comes back to the election initiator process. It identifies its PID in the list and finds the highest PID in the list and makes it as the coordinator. It then builds the coordinator message say COMSG. The PID of the new coordinator would be the content of COMSG. This message is sent to each of the processes of the
ring and while receiving the COMSG, the respective process sets the COORD to the PID of new coordinator, FLAG to FALSE and $T_{elec}$ to NULL. When this message completes one full circulation, the election initiator destroys this COMSG and resumes to its execution.

According to our assumption, all the processes in the group have their clocks synchronized, and so all the alive processes have the same time in their clock. When any process Q receives the ELMSG, it reads its log, to check whether it has created any ELMSG recently. Any one of the following 3 scenarios is possible.

(i) Q would have initiated the ELMSG before P.

(ii) Q would have initiated the ELMSG just after P.

(iii)Q did not create any ELMSG at all.

Here the fourth option, which is, Q and P would have initiated the ELMSG at the same time is never possible. In a system, if two events are related to each other, their clock time which is their time of event will not be same. (i.e.) if a and b are related events, then $C(a) \neq C(b)$ where C refers to their clock time. No two events will happen at the very same time as they are concurrent. This is proved by Lamport in his logical clock synchronization algorithm.[96]

The first two scenarios should be given more importance, as they lead to simultaneous elections. One among the two only is the original election and the other concurrent elections should be stopped as soon as they are encountered as duplicate elections. This is done by our new ELMSG data structure where it has the entry of the message creation time. This time plays a vital role in the destruction of duplicate ELMSGs and so it should be perfectly synchronized within the system.
When any process say P, receives the ELMSG, it checks whether it has already started any election, and if so, it compares the time of both the election messages. If P’s time is earlier than of the incoming message, then it understands that the received message is a duplicate one and it destroys it immediately otherwise it enters its PID in the list and circulates it along the ring. This model ensures that there will be exactly one ELMSG which completes a full circulation wherein the others get destroyed along the way. This leads to exactly only one circulation of COMSG for the ring unlike the original ring algorithm wherein there would be as many as number of messages as the number of elections happening which was of $o(n^2)$ message complexity.

Figure 3.2: Passing and Destruction of Election Message in MEREA
Figure 3.2 (a) and (b) represent the message passing during the execution of MERE. Figure 3.2 (a) depicts 7 processes and the leader process 7 is down due to which election has been initiated by processes 3 and 2 at time 6 and 8 respectively. As the ELMMSG of process 3 coming to process 2, it checks for its time of ELMMSG creation which is greater than of process 3’s and so it services the message by entering its PID there. Eventually it would be destroyed by some other process if it is a duplicate one.

On the other hand, in figure 3.2 (b), processes 2 and 3 start the election at time 5 and 2 respectively and when the ELMMSG of process 2 goes to process 3 for servicing, it finds that it is a duplicate one as its clock 2 is less than the message’s which is 5 and so it destroys the message there itself. This eliminates the further processing of this ELMMSG by other processes unnecessarily and also stops the creation of COMSG on this ELMMSG behalf.

3.4. MATHEMATICAL ANALYSIS

3.4.1. Best Case Analysis

Let N be the number of processes in the ring. In the Best case, only one process detects the failure of coordinator. Then, altogether, there will be 2N messages sent, one full circulation of ELMMSG for a maximum of N processes and one full circulation COMSG for a maximum of N processes.

\[ n_e + n_c = 2n \]  

leading to O(n) message complexity. The Ring Election Algorithm and the our proposed Ring Election Algorithm have the same time complexity as in equation (1).

3.4.2. Worst Case Analysis
Original Ring Algorithm’s worst case message complexity is $O(n^2)$ from equation (2). In our Algorithm, the worst case of the modified Election Algorithm is further divided into three more cases of Worst-Best, Worst-Average and Worst-Worst case.

1) Worst-Best Case: This case is when the processes initiate the Election in the anticlockwise direction according to time when the ring network is of clockwise direction. (i.e.) When the Election Algorithm gets invoked by various processes of the ring in the opposite direction of the ring according to time. The Election Messages get destroyed in the very first go itself, that is when Process $i$ sends ELMSG to its immediate neighbour Process $j$, and since the time of message of Process $j$, is earlier, it destroys the $i$’s ELMSG. Likewise, except the very earlier message, all the other ELMSGs would be killed by the immediate successor processes.

Number of ELMSG = $1 + 1+ ...+ 1$ (n times)

Number of COMSG = $1 + 1+ ...+ 1$ (n times)

ELMSG destroyed by successor process $\leq n – 1$

Therefore,

Total Messages = $n_c + n_c + n-1_{DUPLI}$ which is approximately equal to $3n$.

DUPLI represents the number of duplicate election messages. This leads to a message complexity of $O(n)$.

2) Worst-Average and Worst-Worst Case:

It is when the processes initiate the Election in the clockwise direction according to time when the ring network is of clockwise direction. (i.e.) when the Election Algorithm gets invoked along the direction of the ring according to time. Let Process $i$ creates the first ELMSG and passes it on along the ring. In a
fraction of time its successor creates the ELMSG and passes to its successor and so on. In this case, only Process i can kill all the duplicate ELMSGs created by all the other processes. The duplicate election messages would take n-1, n-2, n-3 ... 3, 2, 1 hops to reach process i right from the successor to the immediate predecessor. The number of messages are,

Number of ELMSGs= 1 + 2 + ... + n- 1 + n

Number of COMSGs= n

Total number of messages =1 + 2 + ... + n- 1 + n + n

\[ = \sum_{k=1}^{n} k + n \]

\[ = (1/2) n^2 + (1/2) n + n \]

\[ = n + n + n (n - 1) / 2 \]

\[ = n^2 - n/2 + 2n \]

(4)

The message complexity is $O(n^2)$ for the worst-worst case. In average case, the ELMSG would be destroyed somewhere in the middle and most of the times by the immediate successor itself. Moreover in our model, when a process receives the ELMSG from others, it won’t initiate the Election later at all. And so, the message complexity would be heuristically $O(n)$. The probability of having $O(n^2)$ message complexity is very rare.

We have focused more on the worst case as processes in the real world keep communicating with each other and try sharing the common resources, many processes would identify the leader's crash and intern initiate the Election. Therefore, best case would be very rare and the average case to worst case are only possible. Our algorithm tries to reduce the number of messages in the worst case to a great extent.

3.5 EXPERIMENTS AND SIMULATION RESULTS
We have simulated the existing ring election algorithm and our MEREA in Java. We have created number of processes such as 100, 200 and so on. The clock time of each process has been synchronized internally. We kept 3 seconds message propagation time to reach from one process to another and 1 second to process the message. We also made the process with the highest PID to be down which in turn invoked the Election activity automatically. Figure 3.3 till Figure 3.18 show the screenshots of the execution of both the algorithms. Our algorithm creates a GUI wherein we can input the number processes that need to be created at runtime, and the algorithm that should be executed presently.

![Image](image.png)

**Figure 3.3.** Election Algorithm Simulation First Round

There are three options for this, namely Worst case of Ring algorithm, MEREA Worst-Worst case, and MEREA Worst-Best case. After selecting the algorithm, the leader process gets crashed externally and the election starts. Our algorithm

![Image](image.png)

**Figure 3.4.** Ring Election Algorithm in First Round
also displays the number of election messages, number of coordinator messages and the number of total messages exchanged between the processes during election.

Figure 3.5: MERE A Worst –Best Case Execution

Figure 3.6: MERE A Worst –Worst Case Execution

Figure 3.7: Second Round Number of Processes = 200
Figure 3.8: Ring Election Algorithm in Second Round

Figure 3.9: MERE A Worst–Best Case Execution in Second Round

Figure 3.10: Ring Election Algorithm in Round Three

Figure 3.11: MERE A Worst–Worst Case log in Round Three
Figure 3.12: MEREA Worst –Best Case log in Round Three

Figure 3.13: Ring Election Algorithm in Round Four

Figure 3.14: MEREA Worst –Worst Case log in Round Four

Figure 3.15: MEREA Worst -Best Case log in Round Four
Figure 3.16: Election Algorithm Simulation Fifth Round

Figure 3.17: Ring Election Algorithm in Fifth Round

Figure 3.18. MEREA Worst–Worst Case log in Fifth Round
Figure 3.19: MERE/A Worst – Best Case Execution in Fifth Round

Figure 3.20 shows the message comparison of the Ring algorithm and MERE/A during the worst case performance. It is very clear that MERE/A performs better than the Ring algorithm during the worst case and as the number of processes increases, our algorithm shows better performance.

3.6 SUMMARY

In this chapter, we have proposed Message Efficient Ring Election Algorithm (MERE/A). The proposed algorithm identifies the duplicate election message
with the help of time of creation of the message and terminates them all and allows only the original election message to complete one full circulation. It drastically reduces the number of election messages and their respective coordinator messages. Since MEREA kills all the election messages and due to that further creation of coordinator messages are also stopped, it not only reduces the message complexity but also reduces the time required for processing the unnecessary messages. Simulation results show that the worst case performance of our algorithm is better than the existing algorithm.