International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 3, ISSN 2249 –071X

Comparison of Various Leader Election Algorithms on Basis of Time and Message Complexity

Mala Saraswat
RJIT, Tekanpur

Abstract **- A distributed system is a collection of processors that do not share memory or a clock. Each processor has its own memory, and the processors communicate via communication networks. A distributed algorithm is an algorithm, run on a distributed system, that does not assume the previous existence of a central coordinator .The main problem in distributed algorithm is the coordinator election algorithm. The coordinator election problem is to choose a process from among a group of processes on different processors in a distributed system to act as the central coordinator.**

Key words **– election algorithm, unidirection and bidirection algorithm, mobile adhoc networks**

I. INTRODUCTION

Many Distributed algorithms require one process to act as coordinator, initiator, sequencer to perform some special role like coordinator in centralized mutual exclusion or deadlock detection etc. In general, it does not matter which process takes on this special responsibility, but one of them has to do it. If all processes are exactly same ,with no distinguishing characterstics, there is no way to select one of them to be special. We assume each process has unique id. The goal of election algorithm is to ensure that when an election starts, it concludes with all processes agreeing on who the new coordinator is to be. An **election algorithm** is an algorithm for solving the coordinator election problem. By the nature of the coordinator election problem, any election algorithm must be a distributed algorithm.
The main assumption in election algorithms is every process has a unique identity.

II. ELECTION ALGORITHMS ON RINGS

In ring algorithm the processes form a ring. Each process only sends messages to the next process in the ring. The ring election algorithms can be classified into two categories

1) Unidirectional
2) Bidirectional

In unidirectional ring algorithms messages are sent only in one direction where as in bidirectional ring algorithms messages are sent in both the directions.

a) Unidirectional Lelann Algorithm

The first election algorithm for rings was proposed by LeLan. In LeLann algorithm, every initiator send a token with their identity around the whole ring. Nodes are not allowed to initiate after they receive a token. Whenever a node receives back its id, it has seen every other initiators id by assuming FIFO channels [6].

Algorithm LeLann's

1) For all initiators with id set state to initiator id and send <token, id> to Next process
2) While the receiving process does not receives its own id do steps 3-5
3) Add Next process id to the List
4) Send again <token, id> to Next process
5) If the received process has the maximum value in list, mark it as leader else received process is lost.

Table I. Comparison of no of Message and Time

No of processes	Messages	Time
2	4	0.513
4	16	0.520
8	64	0.538
16	256	0.572

Message Complexity

Every initiator sends N messages. So the worst case time complexity is N^2.

This algorithm is implemented using MPI [4] and the message complexity and time complexity given by the MPI program is shown in *Table I.*

b) Unidirection Chang Roberts

The next unidirectional algorithm with small modifications was proposed by Chang-Roberts [7]. The idea behind his algorithm was when a node receives a token with smaller id than itself, then it has no chance of winning the election so we can drop such packets without forwarding them.

Algorithm Chang Roberts

1) For all initiators with id set state to initiator id and send <token ,id> to Next process
2) When the process receives <token, id> it sends it to the next process only if it's id is greater than the id which it received.
3) When the receiving process gets its own id it becomes the leader.
4) It sends a message that it's the leader to all the processes.

Message Complexity

The message complexity of this algorithm depends on the arrangement of the nodes in the ring. The best case time complexity is 2N-1. The process with largest id sends N messages and other N-1 processes send one message each. The worst case time complexity is N(N+1)/2. The process with largest id sends N messages and other N-1 processes send messages from 1…N-1.

This algorithm is implemented using MPI [5] and the message complexity and time complexity given by the MPI program is shown in *Table II.*

Table II. Comparison of no of Message and Time

No of processes	Messages (best case)	Time (best case)
2	3	0.515
4	7	0.521
8	15	0.544
16	31	0.576

c) Bidirectional Ring Algorithms

In bidirectional ring algorithms messages can be exchanged in any direction. Processor maintains a register ID that contains the name(identifier) of a "large" processor and a (Boolean) register DIR that contains a "direction" on the ring in which there are processors that still have a "smaller" processor up for election. Messages are generated that contain the name of a "large" candidate, and are sent out in the direction where a "smaller" candidate is known to be still alive. The idea of the chase is to eliminate the smaller candidate, and force agreement on the larger candidate. Processors that initiate a chase are termed *active*, and the remaining processors are termed *observant*. After the current active processors have initiated a chase, the observant processors basically relay messages onwards unless they notice an "unusual" situation on the ring. There are two "unusual" situations that can arise at the site of an observant processor as the algorithm proceeds .They are

(i) The processor receives a message of the current phase, say via its left link, that contains a value that is less than the current value in its ID register. The processor turns active, increments its phase number by 1, and initiates a chase with the value its current ID in the direction of the message that was received, i.e., out over its left link.

(ii) The processor receives two messages of the same phase from opposite directions .The processor turns active, increments its phase number by 1, and initiates a chase with the largest value contained in the two messages in the direction of the smallest. The number of active processors that can arise in a phase rapidly decreases as the algorithm proceeds, and that in the end precisely one processor will be left. This processor will know that it is elected because either it receives a message of the current phase with a value identical to the one it sent out (and stored in its ID register) or it receives two messages of the same phase from opposite directions that hold identical values.

Algorithm for Bidirectional Ring

The algorithm describes the actions of an arbitrary processor on a bidirectional ring with half-duplex links as required for electing a leader.[1]

Initialization

a) Let u be the executing processor's identifiation number. Send message <u,o> to both neighbors and phase number Pnum:=0;

b) Wait for corresponding messages<u1,o> and <u2,o> to come in from two neighbors

c) Compare u1 and u2 and set ID to max(u1,u2)and Dir to the min(u1,u2) and goto **Active** state else **Observant** state.

Election

A processor performs in either active or observant state.

Active A processor enters the active state with some value v stored in its ID-register and a phase number p. The phase number p is either stored in Pnum or it is an update stored in temporary register. The phase number Pnum is incremented by 1 and a message <v, Pnum> is sent in Dir direction and goes to observant state.

Observant In this state a processor receives messages and passes them on, unless an "unusual" situation is observed that enables it to initiate a new phase.

Inauguration

A transfer to this final phase occurs when the algorithm terminates and the ID register contains the identity of the unique leader

Message complexity

The message complexity of this bidirectional algorithm is 1.44NlogN+ O(N).

This algorithm is implemented using MPI [4] and the message complexity and time complexity given by the MPI program is shown in *Table III*

Table III. Comparison of no of Message and Time

No of processes	Messages	Time
2	5	0.513
4	11	0.522
8	23	0.540
16	47	0.575

III. LEADER ELECTION ALGORITHMS FOR MOBILE ADHOC NETWORK

Mobile Adhoc Network is a network wherein a pair of nodes communicate by sending messages over direct wireless links or over a series of wireless links including one or more intermediate nodes. Only pairs of node that lie within one another's transmission radius can directly communicate with each other. Wireless failure occurs when previously communicating nodes move such that they are no longer within transmission range of each other. Likewise wireless link formation occurs when nodes that were too far separated to communicate move such that they are within transmission range of each other. Developing distributed algorithm for adhoc networks is a very challenging task since topology may change frequently and unpredictably [2].

In this approach the largest identity node is selected as the leader using minimum wireless messages. A mobile ad hoc network can be considered as a static network with frequent link or node failures, which can be thought of as a mobile node of an adhoc network going out of reach. We use the diameter concept to cover all the nodes in the network. A diameter is defined as the longest distance between any two nodes in the network where the distance is defined as the shortest path between the nodes. The distance metric is measured in number of hops. It is assumed the network gets stabilized after a single change occurs during leader election process and also that there are only a finite number of changes in the network.

Consider a network of N nodes. This algorithm [3] takes more than diameter rounds to terminate since the topological changes are considered during the leader election. If, however, the topological changes are not considered, then it takes diameter rounds to elect the leader.

Leader Election

For each round, each node propagates its unique identifier to its neighbors and a maximum identifier is elected as a leader. This maximum identifier is propagated in the subsequent rounds. All the rounds need to be synchronized. idlist (i) identifies identifier list for nodei, which consists of all the neighbors for nodei. Lid(i) =max(idlist(i))

Termination

At (rounds >= diameter), for each nodei If all identifiers in idlist (i) are the same, then the nodei stops sending the maximum identifier further and elects the maximum identifier in the idlist (i) as the leader. The termination may not be at the end of the diameter rounds, the algorithm gets terminated if for each nodei the elements in idlist (for each node) are the same

Algorithm

Each node i in the network has two components

a) idlist - identifier list

b) lid (i) - leader id of node i.

1) Each node say node i transmits its unique identifier in the first round and Lid(i) in the subsequent rounds to their neighbors and all these ids will be stored in idlist.
   Lid(i) = max (idlist(i));
2) A unique leader is elected in diameter rounds, if there are no topological changes in the network. The algorithm is modified to incorporate topological changes in between the rounds and below is the description of how the algorithm is modified.

Case 1:

If a node has no outgoing links then lid(i) = i;

Case 2:

If a node leaves between the rounds, then the neighbors would know this. Suppose nodei leaves the network after round r and let its neighbors be j and k neighbors of i (i.e. j, k).

1) Delete (ilist, idlist(j & k)) // delete ilist from idlist
2) Repeat while (round > = diameter), // Termination condition
   i) Compare all the identifiers present inidlist(i) for each nodei
   ii) If all the identifiers in idlist(i) are equal, nodei stops propagating its maximum identifier and elects the maximum identifier as the leader.

Case 3:

If a new node i joins the network in between the rounds say round r then the neighbours will update its idlist.

1) If neighbours of i say node j is the neighbour for nodei add (i, idlist(j)).The normal algorithm continues (the ids are propagated), nodes keep exchanging the information till diameter rounds.

2) Repeat while (round > = diameter),

For all nodes in the network (nodej) receives an identifier i at diameter round.

If i is greater than the maximum identifier nodej has propagated in the previous round (diameter-1).

a) Propagate nodei to all the neighbours of j.

b) Also propagate the nodei information to all the neighbours of neighbour i until the whole network is covered, if the above condition satisfies.

Else do not propagate the information. nodes in the network

3) Compare all the identifiers present in idlist(i) If all the identifiers in idlist(i) are equal, nodei stops propagating itsmaximun identifier and elects the maximum identifier as the leader.

4) All nodes in the network follow this process and a unique leader is elected connected component.

The time taken for the algorithm to elect a leader will be O (diam + Δt) where Δt is the time taken for all the nodes to converge and Δt depends on the topology changes.

Message complexity

The message complexity of this algorithm depends on the number of rounds. In each round it sends 2N messages if we consider a ring topology as every node has 2 neighbours. So message complexity is 2N* No. of rounds. This algorithm is implemented using MPI [5] and the message complexity and time complexity given by the MPI program is shown in *Table IV*

Table IV. Comparison of no of Message and Time

No of processes	Messages	Time
2	8	0.530
4	24	0.533
8	80	0.545
16	288	0.579

IV. EXPERIMENTAL RESULTS

We have implemented all the algorithms in MPI[4] using different number of processors. The algorithms are

1) LeLann's Algorithm
2) Chang Roberts Algorithm
3) Leader algorithm for bidirectional ring
4) Leader algorithm for mobile adhoc networks

The message and time complexity of the implemented programs are shown in *Tables V-VIII* wrt to number of processor. Fig1 shows the graph for message complexity with respect to number of processors for all five algorithms including both Chang's Best and Worst case

Table V. Comparison of no of Message and Time for N=2

S. No.	Algorithm	Time	No of message transferred
1)	Unidirection Lelann's Ring Algorithm	.513	4
2)	Unidirection ChangRoberts Algorithm	.515	5
3)	Bidirection Algorithm	.514	5
4)	Mobile Adhoc	.530	8

Table VI. Comparison of no of Message and Time for N=4

S. No.	Algorithm	Time	No of message transferred
1)	Unidirection Lelann's Ring Algorithm	.520	16
2)	Unidirection ChangRoberts Algorithm	.521	10
3)	Bidirection Algorithm	.5 17	11
4)	Mobile Adhoc	.533	24

Table VII. Comparison of no of Message and Time for N=8

S. No.	Algorithm	Time	No of message transferred
1)	Unidirection Lelann's Ring Algorithm	.538	64
2)	Unidirection ChangRoberts Algorithm	.544	36
3)	Bidirection Algorithm	.540	23
4)	Mobile Adhoc	.545	80

Table VIIII. Comparison of no of Message and Time for N=16

S. No.	Algorithm	Time	No of message transferred
1)	Unidirection Lelann's Ring Algorithm	.572	256
2)	Unidirection ChangRoberts Algorithm	.576	136
3)	Bidirection Algorithm	.574	47
4)	Mobile Adhoc	.579	288

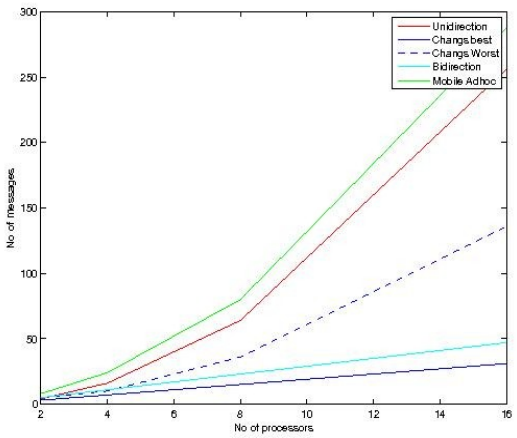

Fig.1. Graph Showing Simulations for Message Complexity with Respect to Number of Processors

V. CONCLUSIONS

If we compare the four algorithm the Lelann's algorithm is the basic algorithm and requires large number of message exchanges. Chang's and Robert algorithm made some modifications to Lelann's algorithm but in the worst case that algorithm also requires $O(N^2)$. These are the two unidirectional algorithms for leader election in ring topology.

The bidirectional algorithm requires less messages than worst case Chang's and Roberts algorithm. It requires O(N logN) messages. Since the messages are sent in both the directions it will take less time to find out the leader when compared to unidirectional algorithms. The last algorithm is implemented for mobile adhoc networks. The algorithm is run in many rounds. The messages complexity depends on number of rounds. It requires more number of messages but it will handle the partitions in the network and guarantees that there is only one leader at a time.

REFERENCES

[1] J.Van Leeumen and R.B Tan, "A improved upper bound for distributed election in bidirectional rings of processors" in Distributed Computing Volume 2, Number 3 (1987)2:149- 160

[2] E.Gafni and D.Bersekas, "Distributed algorithms for generating loopfreeroutes in Networks with frequently changing topology". IEE Transactionson Communications. C-29(1) : 11-18, 1981

[3] Pradeep Parvathipuram1, VijayKumar1, and Gi-Chul Yang 2 "An Efficient Leader Election Algorithm for Mobile Adhoc Networks" ICDCIT 2004. LNCS 3347, pp.32-4 2004

[4] Message passing tutorial http://www.mcs.anl.gov/research/projects/mpi/

[5] Message passing tutorial https://computing.llnl.gov/tutorials/mpi/

[6] http://www.sics.se/~ali/teaching/dalg/l06.ppt

[7] faculty.cs.tamu.edu/welch/papers/dialm.ps

[8] Vincet D.Park and M.Scott Corson "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks.", Proc IEEE Infocom April 7-11 1997

AUTHOR'S PROFILE

Mrs. Mala Saraswat
received her Bachelors degree in Engg (CSE) from MITS Gwalior, M.Tech. (CSE) from IIT Kanpur. She has a ten years of teaching experience. Currently, Asst Prof in Department of Information Technology in Rustam Ji Institute of Technology, Tekanpur , and life member of CSI and ISTE,
Email Id- malasaraswat@gmail.com