Performance analysis of election algorithm computer science

Distributed systems are the systems consisting of multiple processors that connect through a network to communicate. To manage the communication between different nodes and the exchange of data between them, a leader among them is required.

In our project we implement the various Election algorithms for choosing the leader in Distributed Systems thus solving the coordinator election problem. We are also comparing the performance of each of these election algorithms. First we implemented the Election algorithms using the message passing interface(MPI). Then we measured and compared the performance of each of these election algorithms and simulated the results. Finally we modified the distributed leader election algorithm to suit the mobile ad-hoc networks.

Key Words: Distributed Systems – Election algorithms – Unidirectional ring algorithm – Lelann’s algorithm – Chang Roberts algorithm – Bidirectional ring – Leader election – Mobile Adhoc Networks

IntroductionDistributed system

It is a group of processors where the memory or a clock is not shared. Every processor has its own associated memory and the information is exchanged through communication networks.

Distributed algorithm

A distributed algorithm is an algorithm run on such a distributed system assuming the non-existence of central coordinator in these systems. So these algorithms require one process to act as a coordinator. There is no way to select one of them to be leader if all the processes are alike without

different characteristics. One of the processes has to take this special responsibility, no matter which process takes it.

This problem in which a leader has to be elected is termed as the coordinator election problem that is how to choose a process among the different processors to make it a central coordinator.

Election algorithm

An election algorithm is used to solve the coordinator election problem in these distributed systems. Any election algorithm must be a distributed algorithm by the nature of the coordinator election problem. The most important feature in election algorithm is it assumes every process has a Unique ID. It votes a process from among the different processors which can act as the initiator, sequencer and monitor in detecting and solving situations like Deadlock, Mutual Exclusion etc. Thus electing a leader process has become a major issue in wired and ad hoc networks.

The goal of election algorithm is to see that when an election begins it ends with all processes by an agreement as who has to be the new coordinator.

ELECTION ALGORITHM ON RINGS :

A ring is formed by the processes in ring algorithm. In this each process sends only messages to the next process in the ring.

It can be classified into two categories

Unidirectional

Bidirectional

The messages are sent only in one direction in unidirectional and in both directions in Bidirectional ring algorithms.

To compare the performance of these algorithms, the different criteria taken into consideration are

Total number of messages passed

Complexity of the messages used

Time elapsed by the algorithm

2. Implementation Software

We have used the message passing interface (MPI) for implementing our algorithms which are discussed below. It is a standard specification for communication through messages among different processes. It is independent of any language. It is used in parallel computing to write programs for group and point to point communication between nodes.

We used the ‘ C’ language to implement the election algorithms. The MPI’s routines are directly callable in ‘ C’. The main MPI calls used in our program are

MPI_Init: Before communicating, all instances of the code should call this so as to prepare the MPI implementation for the communications environment.

MPI_Finalize: For exiting the communication, this is called by all the instances of the code.

MPI_Comm_size: To learn about the number of processors which are using MPI environment to communicate, this routine is called.

MPI_Comm_rank: Each of this process assigns an integer to the communicating process.

MPI_Send: To send a message to another process, this is called.

MPI_Recv: This call allows to receive a message from a process.

3. Unidirectional Ring Algorithms

The ring algorithm consists of processes arranged in the form of a ring consisting of a token. The token is passed between processes and the process which has the ring can send a message.

The election problem can be implemented using the ring algorithms

Lelanns algorithm

Chang Roberts algorithm

3. 1 LeLanns algorithm

In this we assume that all the processes are physically and logically ordered.

In LeLanns algorithm whenever the coordinator is lost, the initiator sends a token to the other processes in the ring by adding its id. Nodes cannot

initiate any messages once they receive the token. After circulating the token, if the process receives back its id then it is chosen to be the leader since it knows that others cannot become leaders as it knows all the id’s of the other processes and it has the least id. The message complexity of LeLanns algorithm is O(N2).

ALGORITHM:

Step 1: begin

Step 2: send the token to neighbours with id of current process as i

Step 3: add current process id j and forward to neighbours

Step 4: if process P receives back its id then

Step 5: leader is P

Step 6: else return null

Step 7: end

Message Complexity:

Every initiator sends N messages. So the worst case time complexity is N2. The algorithm is implemented using MPI and the message complexity and time complexity given by the MPI program is

No. of processes

Messages

Real time

User time

System time

5

25

1. 195

0. 025

0. 023

10

100

1. 292

0. 027

0. 024

15

225

1. 446

0. 030

0. 027

20

400

1. 551

0. 034

0. 030

25

625

1. 654

0. 036

0. 030

Table 1: LeLann’s algorithm

3. 2 Chang Roberts algorithm

This is similar to lelanns algorithm but with a little change. When a process receives a token with an id greater than the current process id, it drops that particular token as that process cannot be a leader . Hence it forwards the token with an id less that itself. In this way it saves time by discarding the unwanted messages. The worst case message complexity of Chang Roberts algorithm is O(N2) and the average case message complexity is O(N logN).

ALGORITHM:

Step 1 : send message with identifier = I to other processes

Step 2 : if identifier J of current process > I then send the message to neighbours with identifier I

Step 3 : else drop message with identifier I and send the message with identifier J to neighbours

Step 4 : continue this process until a particular process receives back a message with its identifier.

Step 5: if a process receives a message with its id then process= leader.

Step 6: else return null

Step 7: end

Message Complexity:

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 algorithm is implemented using MPI and the message complexity and time complexity given by the MPI program is given in the table 2.

No. of processes

Messages

Real time

User time

System time

5

9

1. 189

0. 024

0. 023

10

19

1. 299

0. 027

0. 024

15

29

1. 412

0. 029

0. 026

20

39

1. 531

0. 033

0. 028

25

49

1. 650

0. 036

0. 031

Table 2: Robert Chang’s Best Case Algorithm

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 1A? a,¬A¦N-1.

No. of processes

Messages

Real time

User time

System time

5

15

1. 186

0. 024

0. 023

10

55

1. 301

0. 027

0. 025

15

120

1. 414

0. 030

0. 027

20

210

2. 511

0. 034

0. 029

25

325

1. 654

0. 035

0. 030

Table 3: Robert Chang’s Worst Case Algorithm

4. Bidirectional Ring Algorithms4. 1 Leader election algorithm for Bidirectional Ring

In these bidirectional ring algorithm messages can be sent or exchanged in any direction. We have used the algorithm mentioned in “[2] An improved upperbound for distributed election in bidirectional rings of processors. J. Van Leeumen and R. B Tan. Distributed Computing(1987)2: 149-160” for implementing it with the MPI.

The name (identifier) of a “ large” processor is contained in the register ID which is maintained by the processor and a (Boolean) register DIR that has a

“ direction” on the ring in which there are processors that still have a “ smaller” processor up for election.

A “ smaller” candidate which is still alive when the messages(the ones having the name of a “ Large” candidate) are created, have them being sent out in its direction. Processors that begin a chase are known as active, and the left over processors are observant. To get rid of the smaller candidate and force agreement on the larger candidate is the main idea behind a chase. After the current active processors have begun the chase, the observant processors basically relay messages onwards unless they notice an “ unusual” situation on the ring only. As the algorithm proceeds, there are two “ unusual” situations that can arise at the location of an observant processor. They are

(i) The processor receives a message of the current phase, say through its left link, that contains a value which 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) Two messages of the same phase are received by the processor 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. As the algorithm proceeds, several active processors that can arise in a phase rapidly decreases, and at the end a single processor will be left precisely. This processor will be familiar that it receives two messages of the same phase from opposite directions that hold

same values and is elected because either it receives a message of the current phase with a value exactly alike 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 same values. ALGORITHM [2]:

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 identification number. Send message to both neighbors and phase number Pnum:= 0;

b) Wait for corresponding messages and 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.

2) Election

A processor performs in either active or observant state.

a) 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 is sent in Dir direction and goes to observant state.

b) 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.

Receive messages from one or both directions. Discard any message received with p less than Pnum.

i) If the number of messages left are zero then go to observant state.

ii) If the number of messages left is one then { Let the one message received be where necessarily p>= PNUM.}

if p= PNUM then

v = ID: goto inaugurate;

v < ID: begin

DIR:= direction from which the message was received;

goto active state

v > ID: begin

goto observant

else

PNUM = p;

ID =: v;

DIR:= the direction in which the message was going Send message to direction

DIR;

goto observant

iii) If the number of messages left is one then{Let the two messages

received be and , necessarily from opposite directions and with p>= PNUM}

if v1= v2

Pnum := p;

goto inaugurate

else

v1!= v2;

ID:-= max(v1, v2);

DIR:= the direction of min(v1, v2);

goto active

3) 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 the bidirectional algorithm is 1. 44NlogN + O(N).

MPI is used implementing the algorithm. The Time and message complexity

given by the MPI program is

No. of processes

Messages

Real time

User time

System time

5

14

1. 186

0. 024

0. 022

10

29

1. 302

0. 027

0. 024

15

44

1. 417

0. 030

0. 026

20

59

1. 534

0. 033

0. 028

25

74

1. 661

0. 036

0. 030

Table 4: Leader election algorithm for Bidirectional Ring

4. 2 Leader election algorithm for Mobile Adhoc Networks

A mobile ad hoc network is dynamic in nature. It is composed of a set of peer-to-peer nodes, that exchanges the information within the network through some wireless channels directly or through a series of such links. A node is independent to move around as there is no fixed final topology. The nodes move freely in a geographical area and are loosely bounded by the transmission range of these wireless channels. Within its transmission range, a mobile node communicates with a set of nodes thus implying all of them have to be in a network. These set of nodes are also known as the neighbors of the communicating node. The mobile nodes act as intermediary routers to direct the packets between the source and the destination nodes (i. e., the set of neighbors). A node is designated as a leader to coordinate the information that needs to be exchanged among nodes and to be in charge of their data requirements. The identification problem of a leader is termed as the “ leader election problem”.

Why do we need to select this leader? When the nodes are set out, they form an adhoc network between them within which the whole communication happens. If the topology of the network changes dynamically, a node may suspend its communication with the previous node, just like in distributed networks. So there has risen a need for a leader so that the maintenance of the network and the clock synchronization within it can be done. Also a new leader has to be chosen every time the members of the group are getting updated while communication is taking place.

When the communicating nodes move freely and if they are not within the transmission range of each other, then the wireless network fails . Similarly the formation of wireless links happen only when the nodes which are separated and are too far and to communicate, move within the transmission range of one another. The network topology may change rapidly and unpredictably over time since the nodes are mobile. So developing efficient distributed algorithm for adhoc networks is a challenging work to be done. The largest identity node is chosen to be the leader using minimum wireless messages in this approach. 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. To cover all the nodes in the network we use the diameter concept. While distance is described as the shortest path between the nodes, diameter is defined as the longest distance between any two nodes in the network. The number of hops will be the taken for measuring the distance and the assumption is that the network becomes stable after a change happens during leader election process and there are only a limited number of changes in the network.

A network having N nodes are considered here. Since the topological changes are considered during the leader election, this algorithm takes more than diameter rounds to terminate. If however, the topological changes are not considered diameter rounds are taken to elect the leader. We have used the algorithm mentioned in “[3]An Efficient Leader Election Algorithm for Mobile Adhoc Networks Pradeep Parvathipuram1, Vijay Kumar1, and Gi-Chul Yang2” for implementing it with the MPI.

Leader Election

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

Termination

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

ALGORITHM [3]:

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 node i 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 ilist contains the group of identifiers that node i has sent to its neighbors before round r along with i The ilist information is also deleted from all the neighbors of j and k if the ilist identifiers have been propagated in the previous rounds. This process continues until all the nodes in the network are covered.

2) Repeat while (round > = diameter), //

Termination condition

Compare all the identifiers present in idlist(i) for each node i. If all the identifiers in idlist(i) are equal, node i 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 neighbors will update its idlist.

1) If neighbors of i say node j is the neighbor for node i. 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 (node j) receives an identifier i at diameter round. IF i is greater than the maximum identifier node j has propagated in the previous round (diameter-1).

a) Propagate node i to all the neighbors of j.

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

Else do not propagate the information to nodes in the network

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

ii) 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 + AZaˆ? t) where AZaˆ? t is the time taken for all the nodes to converge and AZaˆ? t 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 neighbors. So message complexity is 2N* No. of rounds. This algorithm is implemented using MPI and the message complexity and time complexity given by the MPI program is

No. of processes

Messages

Real time

User time

System time

5

30

1. 187

0. 023

0. 022

10

120

1. 301

0. 026

0. 024

15

240

1. 421

0. 030

0. 027

20

440

1. 541

0. 032

0. 029

25

650

1. 752

0. 037

0. 031

Table 5: Leader Election Algorithm for Mobile Adhoc Networks

5. SimulationsAA

Message Complexity with respect to number of processes

AA

Time

No. of Messages Transferred

Sno

Algorithm

N= 5

N= 10

N= 15

N= 20

N= 25

N= 5

N= 10

N= 15

N= 20

N= 25

1

LeLann’s

1. 195

1. 292

1. 446

1. 551

1. 654

25

100

225

400

625

2

Chang Roberts

1. 189

1. 299

1. 412

1. 531

1. 65

9

19

29

39

49

3

Bidirectional Ring

1. 186

1. 302

1. 417

1. 534

1. 661

14

29

44

59

74

4

MobileAdhoc

1. 187

1. 301

1. 421

1. 541

1. 752

30

120

240

440

650The message and time complexity of each of these 4 algorithms for different number of processes is implemented in our programs and the results are as shown in table 6.

All the above simulations are plotted on the graph so as to analyze the way different algorithms message complexity varies with the number of processes on which it executes.

6. Conclusions

Table 6: Simulation ResultsComparing the results, we can conclude that the Lelann’s algorithm is the most fundamental algorithm and requires large number of message exchanges among the four algorithms. Chang’s and Robert algorithm made

considerable changes to Lelann’s algorithm however in the worst case that algorithm also requires O(N2). For leader election in ring topology these are the two unidirectional algorithms that are to be considered.

The bidirectional algorithm requires less messages than the worst case Chang’s and Roberts algorithm. It requires O(N logN) messages. It takes less time to discover the leader when compared to unidirectional algorithms since the messages are sent in both the directions. The final algorithm is put into effect for mobile adhoc networks and is run in many rounds. The messages complexity depends on number of rounds. It guarantees that there

is only one leader at a time but however it handles the partition in the network and requires more number of messages .