A Comparative Analysis of Election Algorithm in Distributed Systems

Heta Jasmin Jhaveri
Student of Government Engineering College,
Sector 28
Gandhinagar

Sanjay Shah
Professor of Government Engineering College,
Sector 28
Gandhinagar

ABSTRACT
In distributed system, an important challenge faced is the adoption of efficient algorithms for coordinator election. The main role of an elected coordinator is to manage the use of a shared resource in an optimal manner. Among all the algorithms reported in the literature, the Bully and Ring algorithms have gained more popularity. This paper proposes a comparative analysis of the various election algorithms in distributed system and also presents a new approach for effective election.

General Terms
Distributed Systems, Election algorithms

Keywords
Election, Coordinator, Priority, Status table.

1. INTRODUCTION
A distributed system is a collection of processors interconnected by a communication network in which each processor has its own local memory and other peripherals and the communication between them is held by message passing over the communication network [1].

Features of Distributed System:

1. Inherently distributed applications
2. Information sharing among distributed users
3. Resource sharing
4. Better price performance ratio
5. Shorter response times and higher throughput
6. Higher reliability
7. Extensibility and incremental growth
8. Better flexibility in meeting users needs

Several distributed algorithms require that there be a coordinator node in the entire system that performs some type of coordination activity needed for the smooth running of other nodes in the system. As the nodes in the system need to interact with the coordinator node, they all must unanimously who the coordinator is. Also if the coordinator node fails due to some reason (e.g. link failure) then a new coordinator node must be elected to take the job of the failed coordinator [1].

Leader election is a fundamental problem in the distributed systems. The election node starts when one or more nodes discover that the leader has failed, and it terminates when the remaining nodes know who the new leader is.

Election algorithms are based on the following assumptions:

1. Each process in the system has a unique priority number.
2. Whenever an election is held, the process having the highest priority number among the currently active process is elected as the coordinator.
3. On recovery, a failed process can take appropriate actions to rejoin the set of active processes.

2. EXISTING ALGORITHMS
Among the prominent algorithms are those listed below:

(1) Hector Garcia-Molina,(1982; also known as Bully Algorithm).

(2) Silberschatz and Galvin (1994)

(3) Sandipan Basu (2010)

2.1 Hector Garcia-Molina,(1982; also known as **b**ully algorith**m**).[3]

2.1.1 Assumptions
- Every node in the system has a unique priority number.
- Every node in the system knows the priority of the other nodes.
- Whenever an election is held. The node having the highest priority number among the currently live nodes is elected as the coordinator.
- On recovery, a failed process can take appropriate actions to rejoin the set of active processes.

2.1.2 Algorithm
When a node (say n1) sends a request message to the coordinator and does not receive a reply within a fixed timeout period, it assumes that the coordinator has failed. It then initiates an election by sending an election message to every other node with a higher priority number than itself. If node n1 does not receive any response to its election message within a fixed timeout period, it assumes that among the currently active nodes

it has the highest priority number. Therefore it takes the job of the coordinator and sends a broadcast message: coordinator message to all the nodes in the system that has lower priority than itself declaring that it is the new coordinator. If n1 receives a response for its election message, this means that there are nodes live that have higher priority than itself, so n1 does not take any action and waits to receive the final result of the election.

When a node n2 receives an election message from a node with lower priority than it, it sends a response message: alive message to the sender informing that it is alive and will take over the election activity. Now n2 holds an election if it is not already holding one. In this way, the election activity moves on to the nodes that has the highest priority number among the currently active processes and eventually wins the election and becomes the new coordinator.

A failed node n must initiate an election after a recovery. If the current coordinators priority number is higher than the node n then the current coordinator will win the election initiated by node n. On the other hand, if n's priority is higher than the current coordinator, it will not receive any response for its election message. So it wins the election and takes over the coordinator's job from the currently active coordinator. Therefore, the active process having the highest priority number always wins the election. Hence the algorithm is called the "bully" algorithm.

2.1.3 Operation

Initially there are 6 alive nodes in the system and node 6 with the highest priority is the coordinator. But node 6 has crashed which is realized by node 2, so it sends an election message to nodes 3,4,5,6 with higher priority than node 2.

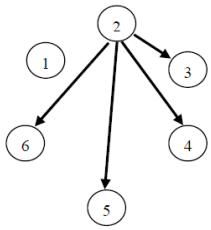


Fig 1.1

As node 6 has crashed, so node 2 receives OK message only from nodes 3, 4,5 and discovers that there are nodes which are live with higher priority than itself.

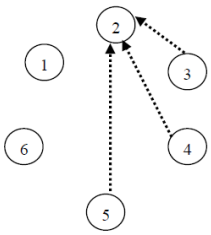


Fig 1.2

Now node 3 sends election message to nodes 4, 5, 6. Similarly, node 5 and 5 sends message to nodes with higher priority than theirs.

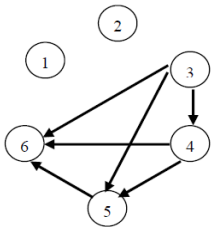


Fig 1.3

Nodes 4, 5 sends OK message to node 3 and 3,4 respectively. Node 5 discovers that among the currently live nodes, it has the highest priority.

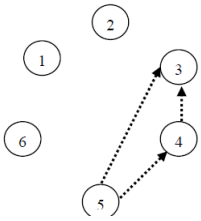


Fig 1.4

Node 5 broadcasts coordinator message to all the nodes.

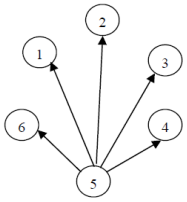
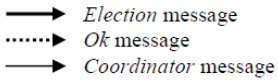


Fig 1.5



2.2 Silberschatz And Galvin (1994)

2.2.1 Assumptions

1. All the nodes in the system are organized as a logical ring.
2. The ring is unidirectional in the nodes so that all the messages related to election algorithm are always passed only in one direction.

2.2.2 Algorithm

While the message circulates over the ring, if the successor of the sender nodes is down the sender can skip over successor, or the one after that until an active member is located.

When a node n1 sends a request message to the current coordinator and does not receive a reply within a fixed timeout period, it assumes that the coordinator has crashed. So it initiates an election by sending an election message to its successor. This message contains the priority of node n1. On receiving the election message, the successor appends its own priority number to the message and passes it on to the next active member in the ring.

In this manner, the election message circulates over the ring from one active node to another and eventually returns back to node n1. Node n1 recognizes the message as its own election message by seeing that in the list of priority numbers held within the message the first priority number is its own.

Among this list, it elects the node with the highest priority as the new coordinator and then circulates a coordinator message over the ring to inform the other active nodes. When the coordinator message comes back to node n1, it is removed by node n1.

When a node n2 recovers after failure, it creates an inquiry message and sends it to its successor. The message contains the identity of node n2. If the successor is not the current coordinator it simply forwards the enquiry message to its own successor. In this way, the inquiry message moves forward along the ring until it reaches the current coordinator. On receiving the inquiry message, the current coordinator sends a reply to node n2 informing that it is the current coordinator.

2.3 Sandipan Basu Algorithm

2.3.1 Assumptions

The following assumptions are made for this algorithm:-

(1) All nodes in the system are assigned a unique identification numbers from 1 to N.
(2) All the nodes in the system are fully connected.
(3) On recovery, a failed process can take appropriate actions to rejoin with the set of active processes.
(4) When a process wants some service from the coordinator, the coordinator is bound to response within the fixed time out period; besides its other tasks.
(5) (5) We assume that a failure cannot cause a node to deviate from its algorithm and behave in an unpredictable manner.
(6) (6) Lamport's concept of logical clock is used in distributed system that we are considering.

2.3.2 Algorithm

When a process (say) P**i** sends a message (any request) to the coordinator and does not receive a response within a fixed timeout period, it assumes that the coordinator has somehow failed. Process P**i** refers to its process status table, to see who is process having the second highest priority number. It then initiates an election, by sending an *ELECTION* message to the process (say) P**j**, having priority just below the failed coordinator; i.e. process with the second highest priority number.

2.3.2.1 Case 1

When P**j** receives an election message (from P**i**), in reply, P**j** sends a response message *OK* to the sender, informing that it is alive and ready to be the new coordinator. Therefore, P**j** will send a message

COORDINATOR to all other live processes (having priority less than P**j**) in the system. Hence, P**i** starts its execution from the point where it was stopped.

Number of messages in this case = 2 + (n-1)

2.3.2.2 Case 2

If P**i** does not receive any response to its election message, within a fixed timeout period; it assumes that process P**j** also has somehow failed. Therefore, process P**i** sends the election message to the process (say, P**k**) having the priority just below the process P**j**. This process continues, until P**i** receives any confirmation message *OK* from any of the process having higher priority than P**i**. It may be the case that, eventually P**i** has to take the charge of the coordinator. In that case, P**i** will send the *COORDINATOR* message to all other processes having lower priority than P**i**.

2.3.2.3 Case 3

Consider process P**m** recovers from its failed state. Immediately, it sends a REQUEST message to any of its live neighbors. The purpose of the REQUEST message is to get the process status table from its neighbor. So, as soon as any of P**m**'s live neighbors receives a REQUEST message, it sends a copy of the current process status table to P**m**. After receiving the process status table, P**m** checks whether its own priority number is less than the process having the highest priority (i.e. current coordinator's priority) or not.

Number of messages in this case = 2

2.3.2.3.1 Case1

If the current coordinator's priority is higher than P**m**'s priority, in that case, P**m** will send its priority number and an *UPDATE* messages to all other processes in the system, to tell them to update P**m**'s status (from CRASHED to NORMAL) in their own process status table.

Number of messages in this case = (n-1)

2.3.2.3.2 Case 2

If P**m**'s priority is higher than the current coordinator's priority; then P**m** will be the new coordinator and update the process status table and sends the COORDIANTOR message to all other processes in the system, and takes over the coordinator's job from the currently active coordinator.

Number of messages in this case = (n-1)
So the efficiency of the algorithm in any case is O (n)

3. COMPARISON

In Bully algorithm, 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. All the processes except the active process with the highest priority number and the coordinator process that has just failed perform elections. So in the worst case, the bully algorithm requires $O(n^2)$ messages. When the process having the priority number just below the failed coordinator detects failure of coordinator, it immediately elects itself as the coordinator and sends n-2 coordinator messages. So in the best case, it has O(n) messages.

During recovery, a failed process must initiate an election in recovery. So once again, Bully algorithm requires $O(n^2)$ messages in the worst case, and (n-1) messages in the best case.

In ring algorithm, on the contrary, irrespective of which process detects the failure of coordinator and initiates an election, an election always requires 2(n-1) messages. (n-1) messages needed for one round rotation of the ELECTION message, and another (n-1) messages for the COORDINATOR message. The algorithm proposed by Sandipan Basu has O(n) message efficiency.

During recovery, a failed process does not initiate an election on recovery, but just searches for the current coordinator. So ring algorithm only requires n/2 messages on average during recovery.

In the algorithm proposed by Sandipan Basu, the number of ELECTION messages made when the coordinator fails is 2 in the worst case. And it requires (n-1) coordinator messages. In the best case, it requires only (n-2) coordinator messages as there is no need to make any ELECTION message.

During recovery, in the best and the worst case, a failed process requires 2 ELECTION message are required to know the current coordinator and (n-1) messages to send its own priority to other nodes. So in all, 2 + (n-1) messages are requires. Thus it requires O(n) messages.

Another approach for an effective election is as follows:

With a distributed system of n nodes, the approach assumes that all the nodes have unique id and each node in the system knows the id of the all other nodes. Also that all the nodes in the system are properly synchronized in time. When a node say n1 discovers the coordinator failure of node C, it sends a broadcast message to all the other nodes with three parameters:

(C failed, n1 new Coordinator, TS1)

declaring that the coordinator C has failed, and now n1 is the new coordinator, along with the timestamp TS1. It may happen that another node n2 also discovers the coordinator failure and sends a broadcast message with timestamp TS2. The nodes receiving the two messages checks the two timestamp, and considers the node as new coordinator which has less timestamp or in other words, which discovered the coordinator crash earlier.

So in all (n-1) Coordinator messages are required in any case, and during recovery, a failed node sends an inquiry message to any neighboring node to know the current coordinator which requires two messages and (n-1) messages to let the other nodes know that it is alive again.

4. CONCLUSION

The paper makes an analysis of the three algorithms discussed and efficiency in terms of number of messages exchanged in each case. Also it presents an effective approach to perform election when a coordinator node has crashed.

5. ACKNOWLEDGMENTS

Our special thanks to Prof. Sanjay Shah who has contributed largely towards development of this paper.

6. REFERENCES

[1] Sinha P.K, Distributed Operating Systems Concepts and Design, Prentice-Hall of India private Limited, 2008.

[2] Tanenbaum A.S Distributed Operating System, Pearson Education, 2007.

[3] Garcia – Molina "Elections in a distributed computing system", IEEE transactions on computers, vol C-31, No 1,pp 48-59., 1982.

[4] Fredrickson and Lynch, Fredrickson, and Lynch, "Electing a Leader in a synchronous Ring", journal of the ACM, Vol 34, pp 98-115, 1987.

[5] An Efficient Approach of Election Algorithm in Distributed Systems" - Sandipan Basu / Indian Journal of Computer Science and Engineering (IJCSE), Vol 20, No 1, 2010