ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

Optimal Distributed Leader Election Algorithm

[I]Dinesh Kumar Yadav, [II]Vijay Kumar Sharma
[I,II]Dept. of CS, RIET, Jaipur, Rajasthan, India

Abstract
Leader election is the most critical part of any distributed system and also challenging one. By optimizing the performance of leader election, performance of system can be improved. There is already number of algorithms but in this paper there is a proposal of a new Leader Election algorithm. In this proposal we are trying to tradeoff among time, message and space complexity. Space is available easily as compare to other two factors, so proposal is to minimize the time as well as message complexity by using little bit extra space.

Keywords
Distributed algorithms, Leader election, algorithms, complexity, Distributed Computing

I. Introduction

Application of distributed computing is expending very rapidly. For improvement of the performance of any application, it is must to improve the performance of distributed system. Leader Election is already said the most important part of any distributed system.

[1]The problem of leader election is fundamental in the operation of distributed systems. A distributed system is a collection of autonomous computing nodes which can communicate with each other and which co-operate on a common goal or task.

[1]A leader performs a centralized co-ordination after being selected. This may be necessary in some problems where a completely distributed solution is either not available or offers less attractive performance. Whenever some failure occurs it is necessary for the nodes to adapt to the new conditions so that they can continue working. This requires some kind of re-organization. Leaders are elected to manage the re-organization

The leader election problem is supposed to be one of the most prominent ones in distributed computing. It consists of appointing a leader process from an initial configuration in which all processes are in the same state, that is, they all are candidates and can become a leader [10]. In distributed computing leader election is the process of designating a single process as the organizer of some task distributed among several computers (nodes). In many cases we need a coordinator in the network for coordination tasks. When this coordinator crashes, we have to select another process as the substitute

Garcia-Molina's classical Bully Algorithm is a prime solution to leader election in synchronous systems. In this paper there is a proposal of a new algorithm for leader election using priority queue . Priority queue is created using the max heap structure and it keeps the ID of all nodes of network in sorted form . Priority Queue will be dedicated with each node (process). Each priority queue is managed by its node(process) separately. It takes fewer messages as well as less time as compare to other popular leader election algorithm like Bully and Ring.

II. Related Work

There are lot of work already have done in the field of leader election. Several leader election algorithms such as Ref. [6] the Bully algorithm, [7]Ring algorithm, [8]Chang and Robert' s algorithm, [9] Peterson's algorithm, [10] LeLann's algorithm, and [11] Franklin's algorithm have been proposed over the years. These algorithms, however, require nodes to be directly involved in leader election. Information is exchanged between nodes by transmitting messages to one another until an agreement is reached. Once a decision is made, a node is elected as the leader and all the other nodes will acknowledge the role of that node as the leader. [2]A new approach has been proposed for leader election using heap tree method. In this approach formal heap tree method is used.

III. Proposed Approach

A. System Assumptions

The assumptions of our algorithm are:

1. We assume also that communications under the synchronous system is reliable.
2. The nodes do not know which ones are currently up and which ones are down.
3. Each node in the system has a unique node ID.
4. Each node knows the ID of all nodes.
5. Each node is linked with other nodes.

B. Proposed Approach of Leader Election

In this proposal , we assume that each node knows about other nodes of the network same as[6] Bully algorithm, so it is certain that each node has stored the information about all the nodes of the network. So in this proposal we assume that information (ID) of each node is stored with each node in a data structure, in sorted form that is a Priority Queue.

The main part of this approach is priority queue which is used to store the information of other nodes ID in sorted form. It also supports our assumption that each node knows about the other node . In this way each node have information of other node.

C. Priority Queue

[17]The heap data structure itself has enormous utility. one of the most popular applications of a heap: its use as an efficient priority queue. As with heaps, there are two kinds of priority queues: max-priority queues and min-priority queues Implementation of max-priority queues are based on max-heaps. Here array implementation of Heap is used.

A priority queue is a data structure for maintaining a set S of elements, each with an associated value called a *key(ID)*. A max-priority queue supports the following operations.

INSERT(S, x) inserts the element x into the set S. This operation could be written

as $S \leftarrow S \cup \{x\}$.

MAXIMUM(S) returns the element of S with the largest key.

EXTRACT-MAX(S) removes and returns the element of S with the largest key.

INCREASE-KEY(S, x, k) increases the value of element x's key to the new value k,
Which is assumed to be at least as large as x's current key value.
One application of max-priority queues is to schedule jobs on a shared computer. The max-priority queue keeps track of the jobs to be performed and their relative priorities. When a job is finished or interrupted, the highest-priority job is selected from those pending using EXTRACT-MAX. A new job can be added to the queue at any time using INSERT.

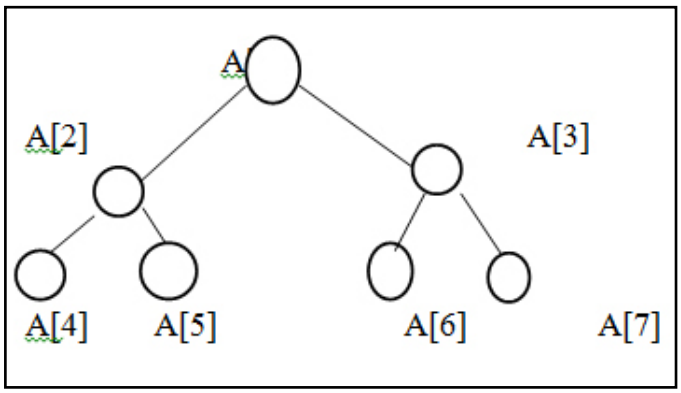

Fig. 1: Priority Queue(A Max Heap)

D. Total space required by single node (process or machine) of a network for proposed approach

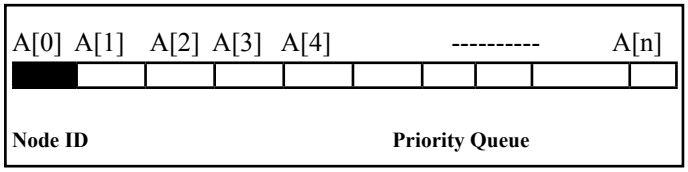

Fig. 2 : Basic structure for node

So total space required by any node in proposed approach is (n+1)
Above structure is proposed structure for every node ,in this way each machine or process will store its ID as well as other nodes ID which are linked in the network.
For example if any node let's say $node_i$, then A[0] will store $node_i$'s ID and from A[1] to A[10](priority queue) PQ_i It will store all nodes including $node_i$'s id in sorted form.

E. Proposed Algorithm

In this proposal suppose we have total n nodes and have to elect a leader of largest ID among these n nodes .As per assumption each node knows about other node and their ID .So ,The proposal is that each node have a data structure Heap ,Which is used as a priority Queue Or In other word every nodes have a priority Queue of n nodes ID and for this [17]Priority Queue , Heap data structure is used.

In this way in proposed algorithm every node is utilizing (n+1) space

$n+1$= node(process) own ID + Priority Queue
So,Total space used= $n(n+1)$

In the priority Queue the Max heap property is used.So the Id with Maximum value will be on top.
Signal ={0,1}
*Signa*l is a binary variable ,It is either 1 or 0. 1 for Yes and 0 for no.
Initially the *signal* will be 0.
Start (When leader election starts)
Any node suppose *ni* who wants to send ELECTION message.First change the value of *signal;*
1.*Signal=1 (now no one can send the message except ni);*
2.If n_i *crashed*
then after some delay *td, signal=0* and some new process will start from beginning *;*
else
3.n_i send message ELECTION to all other nodes.
Apply EXTRACT-MAX() to the priority queue and elect node of this ID as LEADER and broadcast this message to every node.
4.*Signal =0;*
5.Whosoever node received this, will apply Extract-MAX() on own priority queue and verify this leader with extract value.(In this way every node have a list of candidate nodes in sorted form)

When Leader has crashed, then…
Whosoever came to know that leader has crashed do the following steps.
Let us assume that process n_b came to know about leader failure then it will perform following steps

1 *Signal*=1

2 If(nb fails after changing the signal value) then

after time t_b the value of signal will be reset to 0 and new process will follow from step 1

else

3. Send the message to all nodes, ELECTION that Leader has crashed and election has started now.
4.Apply EXTRACT-MAX() to its priority queue and elect this ID node as leader and broad cast it as LEADER.
5. *Signal*=0;
6. Now all other node will receive this message, will apply EXTRACT-MAX() function on their priority queue and verify this value with elected leader's ID.

Case1. Leader election starts (all nodes are candidate nodes)
When leader election starts , suppose any node nodei starts leader election then it will first change the value of signal variable from 0 to 1 (Now ,no node can send the election message except nodei). After this, nodei will send message ELECTION to every node of the network and also calls EXTRACT-MAX() on its priority queue lets say *PQ*i .Whatever value comes , *nodei* elect the node of this id as leader and broadcast this as a leader to each member of the network.
After this every node knows that who is leader and then each node call EXTRACT-MAX() to their respective priority queues. By extracting the largest id each node verify the leader by comparing it with leader's ID and also update their priority queue. Now each priority queue is a sorted list of candidate nodes ID.

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

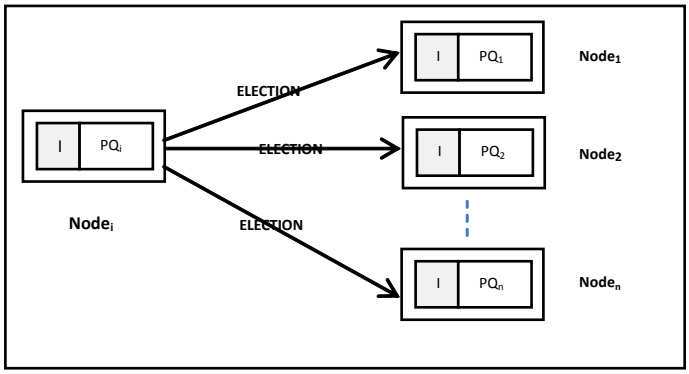

Fig. 3 : Leader election starts

Case2. Leader has crashed

Suppose leader has crashed and more than one node came to know about this crash then both will try to initiate leader election ,so first of all they race for signal variable and whosoever be successful to change the variable signal from 0 to 1 will send the Fig.4 ELECTION message. After this each node knew that leader has crashed and election process has started.

The $node_b$ which sent above message, now call EXTRACT-MAX() on own priority queue PQ_b. .Whatever value comes , $node_b$ elect the node of this id as leader and broadcast this as a new leader to each member of the network.

Now the $node_b$ again changes the value of signal from 1 to 0.

After this every node knows that who is new leader and then each node call EXTRACT-MAX() to their respective priority queues. By extracting the largest id each node verify the leader by comparing it with leader's ID and also update their priority queue. Now each priority queue is a sorted list of candidate nodes ID.

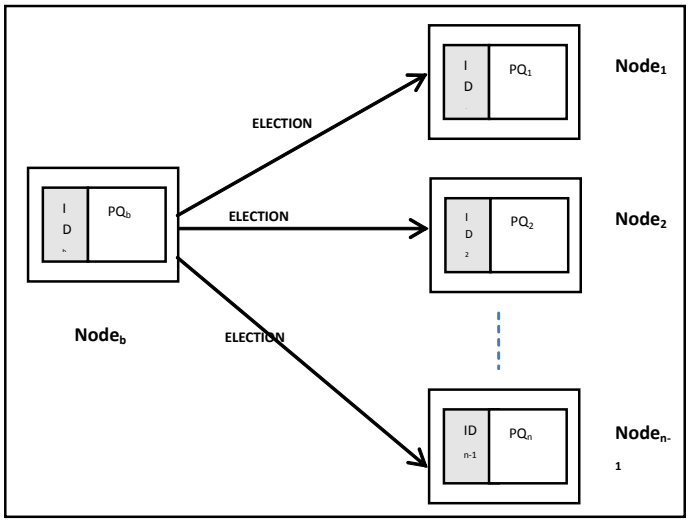

Fig. 4: New Leader election When Leader has crashed

A. Analysis of Proposed algorithm

Total space required by this approach will be $n(n+1)$

No of message used in leader election= $(n-1)$

In any situation total no of message used in leader election will be $(n-1)$.

No of message used when leader has crashed= n_a-1

Where n_a= number of active nodes after the leader has crashed

If i is the no of crashed leader at any time t, then active nodes $n_a = (n-i)$

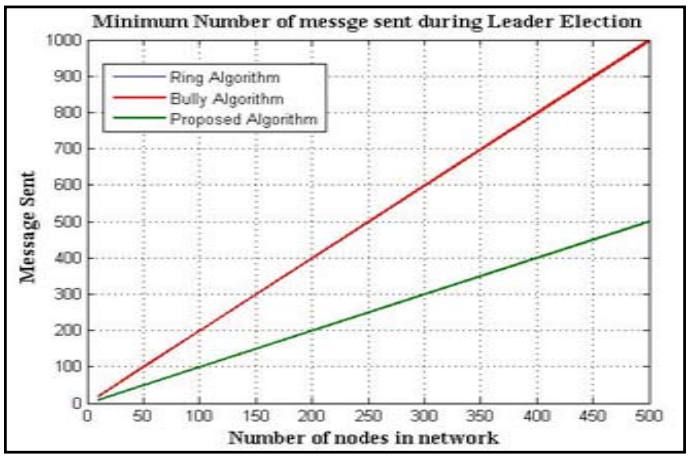

Fig.5: Minimum no of message sent during leader election

V. Conclusion and Future Work

Selection of leader election algorithm in a distributed system plays a vital role in the performance of the system and there should be a proper trade-off among space, time and message complexity. The proposed algorithm in this paper is an attempt to improve leader election algorithm (using priority queue). Lesser time complexity and less number of message sent in leader election using priority queue method shows that it will perform better than earlier proposed algorithm in distributed computing. By using proposed algorithm proper balance among space, time and message can be obtained. Proposed algorithm sends *n-1* message which is quite less from earlier proposed algorithms.

In proposed method Build heap procedure can be further optimized so that priority queue take lesser time and this will improve the performance of proposed algorithm of leader election. In future we tend to adopt this approach in ad hoc and sensor environment.

Table 1 : Performance of Various Algorithms in Leader Election

Algorithm	Space Used	Order Of algorithm	Minimum Message	Maximum Message
Priority Algorithm	n(n+1)	nlogn	n-1	n-1
Bully algorithm	n*n	n*n	2n-2	n*n
Ring algorithm	n*n	n*n	n-1	n*n

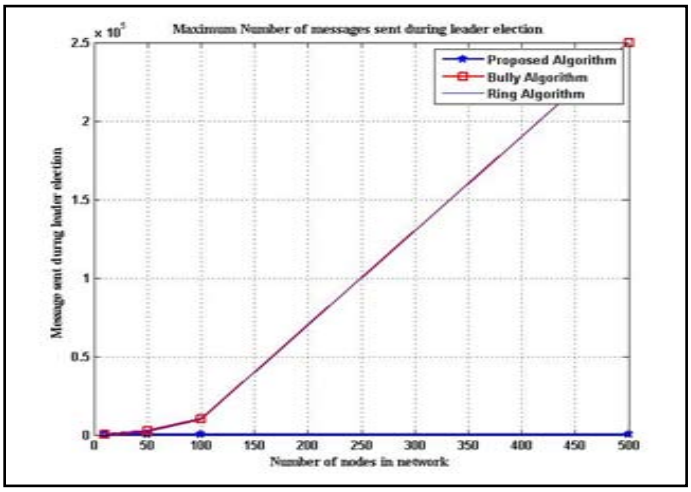

Fig.6: Maximum no of message sent during leader election

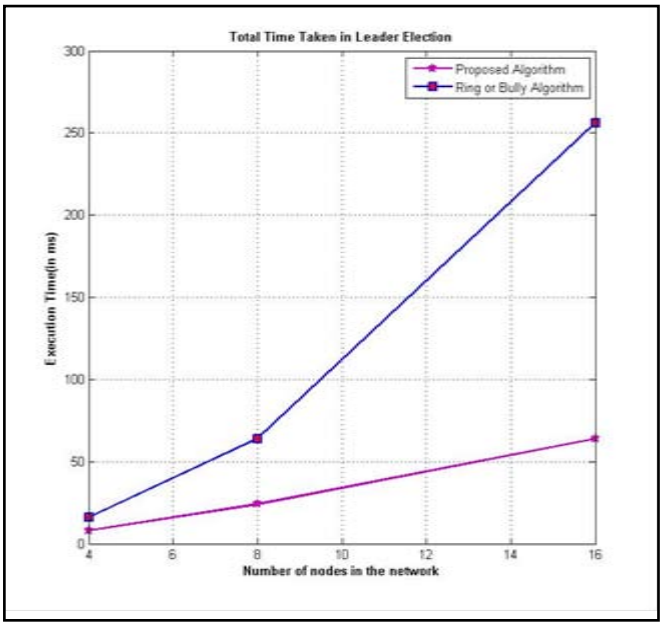

Fig.7: Execution time during leader election

minor words such as “a”, “an”, “and”, “as”, “at”, “by”, “for”, “from”, “if”, “in”, “into”, “on”, “or”, “of”, “the”, “to”, “with”.
Author details must not show any professional title (e.g. Professor), any academic title (e.g. Dr.) or any membership of any professional organization.
To avoid confusion, the family name must be written as the last part of each author name (e.g. Antony S Llyod).
Each affiliation must include, at the very least, the name of the company and the name of the country where the author is based (e.g. XXX Private Ltd, Netherlands).

References

[1] *Princy Francis and Sanjeev Saxena, IEEE conference, 1998,Optimal Distributed Leader Election Algorithm For Synchronous complete Network .*
[2] *Mohammad Reza EffatParvar, Nasser Yazdani, Mehdi EffatParvar , Aresh Dadlani and Ahmad Khonsari,IEEE conference,2010, Improved Algorithms for Leader Election in Distributed Systems .*
[3] *Vandana Sharma, Parvinder S. Sandhu, Satwinder Singh, and Baljit Saini, World Academy of Science, Engineering and Technology 42, 2008,Analysis of Modified Heap Sort Algorithm on Different Environment*
[4] *Xio Dong Wang, Ying Jie Wu, Journal of Computer Science and Technology. 22(6): 898-903 An improved heap sort algorithm with nlogn –0.788928n comparisons in worst case*
[5] *McDiarmid C J H, Journal of Algorithms, 1989, 10(3): 352~365,Reed B A. Building Heaps Fast.*
[6] *H. Gracia-Molina, IEEE Trans. on Computers, vol. C-31, no. 1, Jan. 1982 “Elections in a distributed computing system”*
[7] *N. Fredrickson and N. Lynch,Journal of ACM, vol. 34, no. 1, pp. 98-115”, 1987 “Electing a leader in a synchronousring”*
[8] *E. Chang and R. Roberts, Communications of the ACM, vol. 22, no. 5, pp.281-283, May 1979 “An improved algorithm for decentralized extrema-finding in circular configurations of processes”.*
[9] *G. L. Peterson, ACM Trans. Programming Languages and Systems, pp. 758-762, Oct. 1982 “An O(n log n) unidirectional algorithm for the circular extrema problem”.*
[10] *G. LeLann, Information Processing Letters, pp. 155-160, 1977 “Distributed systems - towards a formal approach”.*
[11] *W. R. Franklin, Communication of the ACM, pp. 336-337, 1982 “On an improved algorithm for Misra and Kate N. Nagaraj,”Security in Wireless Ad Hoc*
[12] *H. GaFcia-Molina." IEEE Trans. on Computers, Vol C-31, Jan 1982, 48-59 "Elections in a Distributed Computing System”*
[13] *G. Fredrickson and N. Lynch, in Proc. 16th ACM Symp. on Theory of Computing, Washington, USA, pp. 493-503, 1984 “The impact of synchronous communication on the problem of electing a leader in a ring”*
[14] *E. Korach, S. Moran, and S. Zaks, in Proc. 3rd ACM Symp. on Principles of Distributed Computing, Vancouver, Canada, pp. 199-207,Aug. 1984, “Tight lower and upper bounds for some distributed algorithms for a complete network of processors”.*
[15] *Gonnet G H, Munro J I, 1986, 15(6): 964-971, Heaps on Heaps. SIAM Journal on Computing.*
[16] *P. M. B. Vitanyi, “Distributed election in an Archimedean ring of processors”, USA, pp. 542-547, 1984,in Proc. 16th ACM Symp. on Theory of Computing, Washington*
[17] *Thomas H. Cormen,Charles E. Leiserson, Ronald L. Rivest, Clifford Stein,”Introduction to Algorithms, Second Edition”*

Author's Profile and Image

Dinesh Kumar Yadav B.E. in CS & Engg and area of interest is distributed algorithm.