An Efficient Overhead-Aware Leader Election Algorithm for Distributed Systems

Muneer Bani Yassein
Department of Computer Science
School of Computer and Information Technology
Jordan university of Science and Technology
Irbid 22110, Jordan

Ala'a N. Alslaity
Department of Computer Science
School of Computer and Information Technology
Jordan university of Science and Technology
Irbid 22110, Jordan

Sana'a A. Alwidian
Department of Computer Science
School of Computer and Information Technology
Jordan university of Science and Technology
Irbid 22110, Jordan

ABSTRACT
In the area of distributed computing, the leader election process is meant with selecting a single node as a leader or a coordinator for a particular task that is distributed among other members. In such environments, if the leader got crashed, all other nodes have to elect another leader. In the literature, many leader election algorithms have been proposed. Most popular is the Garcia Molina's Bully algorithm. In this paper, we propose a new leader election algorithm that is based on sending a lower number of messages to perform leader election. The results show that our proposed algorithm reduces the overhead associated with the classical Garcia's Bully algorithm and efficiently outperform it in terms of reducing latency and message complexity.

General Terms
Leader Election Algorithms, Distributed Systems.

Keywords
Distributed System, Leader Election, Leader, Bully Algorithm

1. INTRODUCTION
In recent years, distributed systems are growing rapidly. Therefore, managing and controlling these systems becomes a challenging issue. In distributed systems, each node must efficiently and accurately cooperate with other nodes to perform a specific job [1]. In such systems, particular nodes are selected to handle the responsibility of leadership and coordination. Examples of these nodes are file servers, time servers and central lock coordinators. These servers are called leaders **[2]**. Algorithms through which leaders are elected called: leader election algorithms.

In leader election algorithms, a single node is designated as the organizer (or leader) of some task that is distributed among several nodes **[3].** Generally, it does not matter which node should take over the leader's job, but one of them has to do it. After a leader is elected however, each node throughout the network should respect the elected node and recognize it as the group leader.

When the leader is crashed, other nodes must elect another leader. Many algorithms have been proposed for electing leaders in distributed systems such as Bully algorithm [4] and ring algorithm [5].

Bully algorithm is one of the most widely applicable algorithms for electing leaders in distributed computing systems [6]. It was proposed by Garcia Molina in 1982. In this algorithm, when a node *N* detects that the leader is crashed, it sends an *ELECTION* message to all nodes with higher *ID*. If no one of these nodes responds, node *N* will win the election and becomes the leader. If one of the higher nodes responds however, it will take over, and node *N*'s job is done. When a higher-*ID* node receives a message, it sends an *OK* message back to the sender declaring that it is available and will take over. The receiver then holds an election, unless it is already holding one. All nodes give up but one and that one is the new leader **[4].** The available and absolutely higher-*ID* node announces its victory by sending a *COORDINATE* message back to all nodes telling them that it is the new leader. Figure 1 illustrates the steps of Bully algorithms in details. Bully algorithm is simple in terms of its concept and implementation. However, its main drawback is the high number of message passing which is of order $O(n^2)$ which causes a heavy traffic and overhead on the network.

In this paper, we propose a new leader election algorithm that performs the task of electing leaders with lower number of messages, and therefore, reduces the overhead associated with Garcia's Bully algorithm.

The rest of this paper is organized as follows: Section 2 provides a set of related work in leader election, Sections 3 and 4 illustrates our algorithm assumptions and the proposed algorithm respectively. A mathematical analysis is presented in section 5. And finally, Section 6 concludes the paper

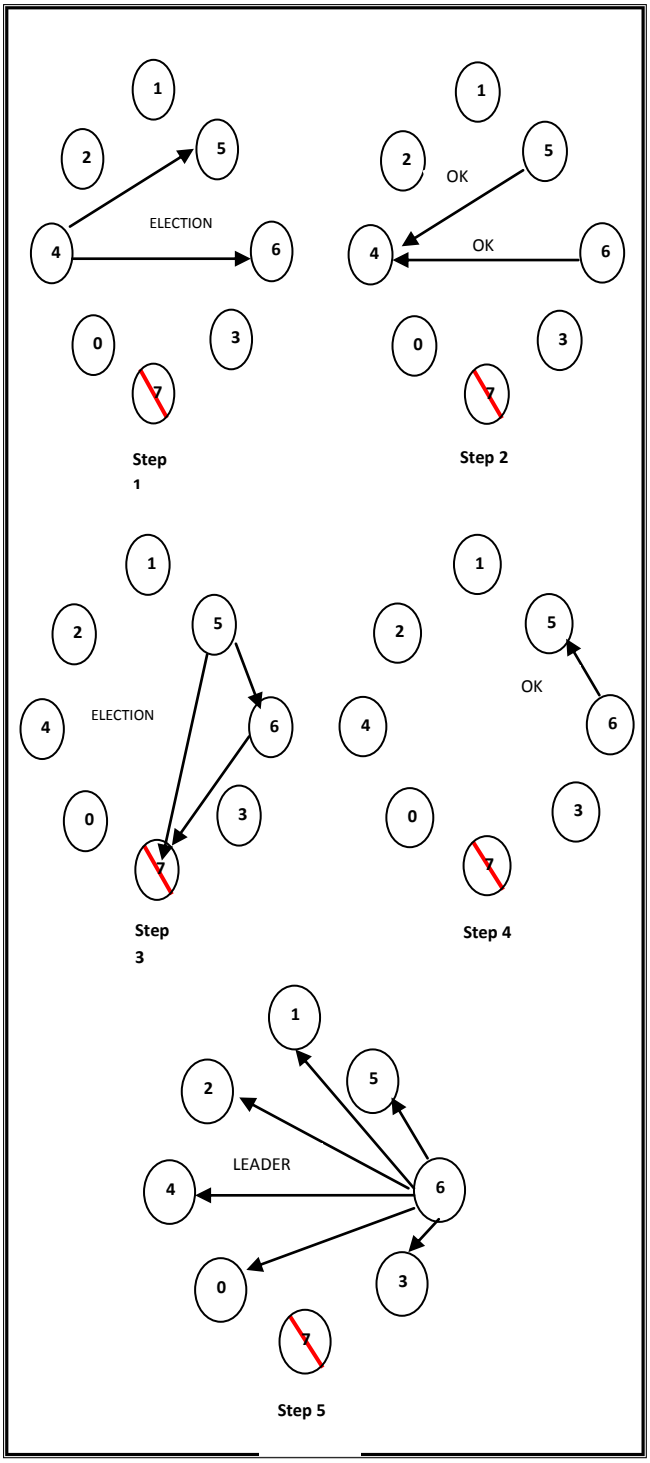

Fig1: The Traditional Bully Algorithm

2. RELATED WORK

Bully algorithm is one of the most important election algorithms that were proposed for solving the leader election problem. In the literature, many modifications for the original Bully algorithm were proposed for the purpose of enhancing its performance. Among these modifications is the algorithm proposed by Kordafshari et al [7]. In this work, the authors discussed the drawbacks of the original bully algorithm and enhance it using an optimal message algorithm.

When a process P detects the leader crash, it starts election by sending an ELECTION message to all processes that has higher process IDs. If P got no response, it announces itself as the new leader. Otherwise, all other alive processes with higher IDs respond by sending their own IDs along with the OK messages. After that, process P selects the process with higher ID and sends back a GRANT message to that elected process. Then, the later sends a COORDINATOR message to all other processes declaring that it is the leader. For the purpose of reducing concurrent election, when process P detects that the current leader is down, it starts election. If a process A receives an ELECTION message from process P and from another process, B which has a lower-ID ,it waits for a short period of time and replies to B and stops its own algorithm. However, if P got neither ELECION message nor a response from any other process with lower ID, it declares itself as a leader. Although this method reduces message passing complexity to some extent, it produces redundant elections which in turn, consume resources, increase total message passing and increase network traffic.

Another modification for Garcia's Bully algorithm was proposed by Gholipour et al [9], where they proposed an algorithm that decreases the number of messages that should be exchanged between processes. In their work, the authors proposed the use of "Coordination group" which is a set of several ordered nodes that includes the current coordinator and *K* alternatives to that coordinator. The IDs of alternative process are sent to all other processes. When a process P detects the crash of the coordinator, it sends a "crash-leader" message to the first alternative process. If the later is alive, it re-checks the availability of the coordinator before it announces itself as the new leader. Otherwise, if the first alternative process is crashed, the process P repeats the same previous operation with all other alternatives constituting the "Coordination group". In the case where all alternatives are crashed, process P initiates another election algorithm and generates another "coordination group" (which consists of all available processes with IDs higher than P) and repeats the same previously mentioned steps. After this long operation, if P still didn't receive a response, it is selected as a coordinator. In Gholipour's algorithm, nodes depend on the "Coordination group" as a base for the leader election operations, and this group should be updated continuously. Unfortunately, maintaining such group need extra message passing operations which in turn increases the overhead.

Sung-Hoon Park [10] proposed a safety-strengthened leader election protocol that allows processes to elect a new leader only when all of them agree upon the current leader's crash. The election made by this protocol is much stronger than the election performed by the classical Bully algorithm since if no set of processes agree upon the leader's crash, no election will happen, yet, no progress is made. The proposed algorithm is based on a standard three asynchronous phases: the prepare phase, in which a particular process proposes a leader to the other processes which are required to agree on. The ready phase in which all processes that agree on the newly elected leader acknowledge the reservation of the potential leader. And the commit phase where the elected leader is announced and all other process accept it. The algorithm of [10] is fully distributed. However, since all processes must admit the leader's crash, progress of the algorithm can be guaranteed only in case of minimal violating a safety property. In addition, leader election is performed with an unreliable failure detector. This is because it does not extend the asynchronous model of concurrent computation to include

global failure. Moreover, this algorithm needs more messages to ensure its condition (i.e. the entire processes should agree upon the current leader's crash to elect new leader). Hence, traffic load and the overhead will increase.

In [11], Mahdi proposed an election algorithm based on electing a leader and an assistant to that leader, such that in the case of the current leader's crash, the assistant will take over the responsibilities of the leader and becomes the new leader. The assistant-based election algorithm proposed in [11] works as follows: if a any process detects that the current leader fails, the detector process will send a message to the assistant of the crashed leader. Once the assistant receives the message, it communicates first with the leader by sending a message to check if the leader is really crashed or still alive. If the leader is crashed, the assistant becomes the new leader and sends "I am a leader" message to all other processes. In turn, the assistant (which becomes the new leader), and based on the request of all nodes, elects a new assistant to be the "vice-leader" for the future. The problem of this approach becomes evident when the assistant itself got crashed. Here, the processes need to initiate a complete election process to elect a new leader and an assistant to that leader, thus, increases the message complexity

3. ALGORITHM ASSUMPTIONS

For our proposed algorithm, we assume that the distributed system is consisting of a set of processes or nodes, and each node has a unique ID. Such that the set N= {1, 2, 3, …, n-1, n} where N[1] is the node with ID=1 and N[2] is the node with ID=2, and so on. This ID could either represent the node's network address (this is applicable in cases where all nodes are exactly the same with no distinguishing characteristics), or the ID could represent the capacity of nodes (this is applicable in unreliable environments such ad hoc networks, where nodes are varying in their capacities, and the node with higher capacity will be assigned a higher ID). In all cases, our election algorithm attempts to locate the node with higher ID and elect it as a leader. Therefore, in a set of n nodes, N[n] is usually the leader, and N[n-1] is the next candidate leader and so on.

In addition, we assume that each node knows the IDs of all other nodes. However, it is not mandatory to know which nodes are currently up and which ones are currently down. For the purpose of our proposed algorithm, we assume also that each node not only knows the IDs of other nodes (as in the case of Bully algorithm), but also order them in a descending-ordered list. To distinguish this list from the list in the original Bully algorithm, we will call our descending-ordered list as: *MAP*.

After the completion of election task, we assume that all nodes should be informed about and agree upon the newly elected leader.

4. THE PROPOSED ALGORITHM

As we have mentioned before, the number of messages that should be exchanged between nodes in Bully algorithm is high, imposing heavy traffic on the network. To overcome this problem, we present an optimized algorithm that significantly decreases the number of messages (ELECTION and OK messages) that should be exchanged between nodes to perform leader election task. Moreover, the amount of time each node should wait in order to know about the elected leader is also decreased.

Moreover, in the descending-sorted list (the MAP) that we propose, the first item in the list represents the node with the highest ID, the successor item is the node with the next higher ID, and so on. Based on the main concept of Bully algorithm (that we also follow in our work), which states that the bigger guy wins the war, it is easy to conclude that the first item in the MAP is the leader node, the second item is the next candidate node to be the leader and so on. This way of organizing nodes in the nodes' MAP assists in decreasing (or even preventing) the global election that takes place in the classical Bully algorithm.

In this algorithm, when a node A detects that the leader is crashed, A does not send an ELECTION message to all nodes with higher ID (as the case of Bully algorithm), rather, it looks its MAP up and checks the next candidate node whose ID is right after the crashed node and sends it an ELECTION message. We refer to the next candidate node as *C*. If *C* is available (alive), it sends an OK message back to *A* to indicate that it is alive and will take over. Then, *C* sends a COORDINATOR message to all other nodes (including *A*) telling them that it is the newly elected leader.

Note that in our algorithm, and after the candidate node C receives the ELECTION message, it needs not to hold another election (as in the case of Bully algorithm). This is because this node (i.e. C) is chosen from the beginning with a guarantee that it will be the next candidate leader (unless it is crashed).

It is worthwhile to mention that node A waits for a period of time T for receiving an OK message from C. If node A does not receive any OK messages after the duration *T*, it will realize that the candidate node, *C*, is not available or it is currently crashed, and therefore, it sends an ELECTION message to the successor node. If after *T* time the successor does not reply, *A* will skip over the successor and goes to the next node along the MAP, or the one after that until a running node is found. The worst case happens when all nodes with higher IDs are all crashed, in this situation, node *A* itself will be the leader.

If two or more nodes, say *A* and *B* simultaneously detect the crash of the leader, both of them will do the same previously mentioned job. In this case, the candidate node will consider the ELECTION message that has been received first and send an OK message to the sender that initiate the election first to inform it that it is alive. At the same step, it sends a STOP message to the second node that initiates the election to inform it to stop waiting and to indicate that the election process is running. In our algorithm, we assume that all nodes maintain an up-to-date MAP; therefore, there is no chance that any two nodes will send an ELECTION message to two different candidates.

It is worthy to mention here that during the election process, if the previously crashed leader wakes up and becomes available again, it resumes the leadership without the need to initiate election. This is done by directly sending LEADER message to all other (lower-ID) nodes.

From here, we can conclude that our algorithm depends on the advantageous assumptions of Bully algorithm in addition to proposing further new assumptions that enhance the performance of our work.

Figure 2 represents the pseudo code which is triggered when any node detects the leader crash and Figure 3 represents the working of our algorithm where node 4 detects that the current leader (node 7) is crashed. Node 4 checks its MAP to see that node 6 is the successor and the most prior node to be the leader. Therefore, node 4 sends node 6 an ELECTION message. Since node 6 is alive, it replies back with an OK message to node 4 and immediately sends all other nodes (including node 4) a LEADER message declaring that it is the winner of the leadership.

```
//When any node A detects the leader crash:

For i=n-1 to 1 // n is the total number of nodes

{

Send ELECTION message to MAP[i].

Wait for T time to receive OK message

If OK message received

        {

        MAP[i] is the new leader

        Send LEADER message to all n-(n-i) nodes

        }

else

i=i-1

}
```

Fig 2: Pseudo code that is triggered when any node detects the crash of the leader

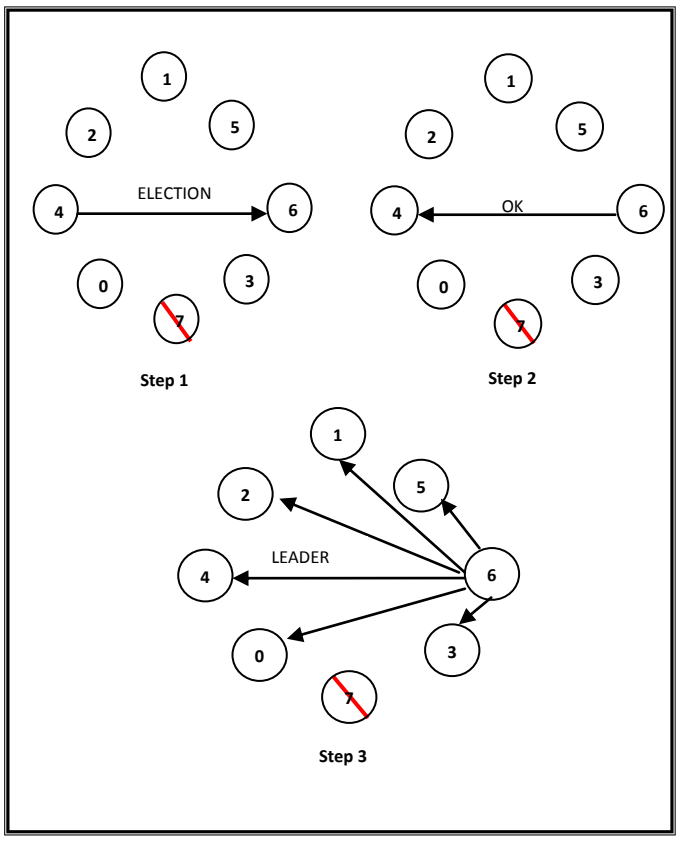

Fig 3: The Proposed Algorithm

5. MATHEMATICAL ANALYSIS

In this section, we analyze the performance of our algorithm and show its efficiency in terms of reducing the number of sent messages. In order to provide a comprehensive analysis, we investigate all the possible cases that might occur in the leader election process.

We assume that the number of nodes in the distributed system is *n*, the number of sent messages by a particular node *i* is *S* and is referred to as S_i, and the number of received messages by a particular node *i* is *R* and is referred to as R_i. We consider the number of sent messages(*ELECTION*) messages and the number of received messages (*OK*)messages as the main criteria to measure the algorithm overhead. Note that we will not consider the number of COORDINATOR messages (*LEADER* in our algorithm) since these messages are sent as a final stage in the election process to inform about the new leader and the number of these messages will be the same in all algorithms (provided that the number of nodes is the same).

The message overhead can be denoted by this formula:

$$\boldsymbol{Overhead = S_i + R_i}$$

In the discussion that follows, we discuss the possible cases that might occur in the environment of leader election. But before we get into the details of the mathematical analysis for each case, it is important to pinpoint the different cases that take place for both the detector node (i.e. the node that detects the crash of the leader), and the number of elections that might be involved (based on how many nodes got crashed). In our comprehensive analysis, we found out that a single node might be crashed; yet, a single election will take place. Or there might be more than one crashed nodes, therefore, there will be several elections. We study the effect of n/4 and n/2 elections in an environment of n nodes. In addition, we assume that a single node detects the crash, and this node might be the lowest node in the system, i.e. N[1], the middle node, N[n/2], or the next higher node, which is the node right after the crashed leader, i.e. N[n-1].

5.1. CASE (I): A single election happened, regardless to the detector node:

If any node *i* detects that the current leader is crashed, it looks up its MAP and sends an ELECTION message to the node that is on the top of the MAP (i.e. the next higher ID node), which is the candidate node to be elected. In this case we assume that the candidate node is alive and therefore it will handle the responsibility of leadership. In this scenario, the number of sent ELECTION messages S_i in our algorithm will be equal to 1 and the number of received OK messages by the detector node will also be1. So the overall overhead is 1+1. This means that we need a maximum of 2 messages for holding leader election.

The situation is different for Bully algorithm. If we assume, as above, that the next higher ID node is available and that the lowest ID node, N[1], detects the leader crash, then the message overhead will be:

$$overhead = 2(n-1) + 2(n-2) + 2(n-3) + \cdots + 2 \qquad (1)$$

Whereas the number of messages transmitted when the middle node N[n/2] detects the crash will be:

$$overhead = 2\left(\frac{n}{2}\right) + 2\left(\frac{n}{2} - 1\right) + 2\left(\frac{n}{2} - 2\right) + \cdots + 2 \quad (2)$$

In both cases, the message complexity for Bully is in the order of $O(n^2)$, where it is of a constant complexity in our work.

5.2. CASE (II): n/4 elections happened, regardless to the detector node:

This scenario happened when the current candidate node N[n-1] is not available and the other next candidate nodes N[n-2], N[n-3], … might not be available too. In this case, we will assume that in a system of n nodes, n/4 nodes are crashed, and therefore, n/4 elections will take place. Regardless to the index of node that detects the crash, the number of sent ELECTION messages in our algorithm will be n/4+1, and the number of OK messages will be 1, thus, the message complexity is given in the following formula:

$$overhead = \left(\frac{n}{4}\right) + 2 \quad (3)$$

For Bully algorithm, if n/4 elections required and either N[1] or N[n/2] detects the crash, the message overhead will be given in formulas 4 and 5, respectively:

$$overhead = \left((n-1) + \frac{3n}{4}\right) + \left((n-2) + (\frac{3n}{4} - 1)\right) + \left((n-3) + (\frac{3n}{4} - 2)\right) + \cdots + \frac{n}{4} \quad (4)$$

$$overhead = \left((\frac{n}{2}) + \frac{n}{4}\right) + \left((\frac{n}{2} - 1) + (\frac{n}{4} - 1)\right) + \left((\frac{n}{2} - 2) + (\frac{n}{4} - 2)\right) + \cdots + \frac{n}{4} \quad (5)$$

From the formulas above, it's easy to conclude that the message complexity of our algorithm is in the order of O(n), where in the Bully algorithm, it's of order $O(n^2)$

5.3. CASE (III): n/2 elections happened and the node N [1] detects the crash

If the half of the nodes is not available, then the number of elections needed is equal to n/2. In this case, if the lowest ID node, N [1] detects the crash, the message complexity for our algorithm will be as follows:

$$overhead = \left(\frac{n}{2}\right) + 2 \quad (6)$$

In the same case, the number of messages required by the Bully algorithm is much more than that in our algorithm and this is clear in formula 7:

$$overhead = \left((n-1) + \frac{n}{2}\right) + \left((n-2) + (\frac{n}{2} - 1)\right) + \left((n-3) + (\frac{n}{2} - 2)\right) + \cdots + \frac{n}{2} \quad (7)$$

5.4. CASE (IV): n/2 elections happened and the node N[n/2] detects the crash

This scenario is similar to CASE (III) except that the node that detects the crashes is the middle one with the index N [n/2]. In this case it is clear that, in both our algorithm and the bully algorithm, the node N[n/2] will send n/2 ELECTION messages before it realizes the crash of all the elected nodes, and then, chooses itself as a leader. From the time latency perspective, it's fair to illustrate here that our algorithm will be slower than Bully algorithm and it will consume more time waiting to hear from the nodes that are already crashed. This time is minimized in Bully algorithm since the node that detect the crash sends ELECTION messages to the whole higher n/2 nodes in one shot, and then announces itself as a leader after knowing the crash of them all. Table 1 summarizes the previously mentioned message complexities for all cases.

5.5. Special Cases:

In this section, we demonstrate some of the unusual cases that might occur in a distributed system during the election process, namely: when the lowest ID node, N[1], detects the leader crash, and in the meanwhile, it is the only available node. In this case, the number of ELECTION messages required for election will be n-1 (for both our algorithm and Bully algorithm). With a difference that our algorithm requires more delay time waiting for a response from the n-1 crashed nodes.

The other case happens when the next higher ID node (i.e. the node right after the crashed leader) detects the crash. This is the optimal case for both algorithms since neither ELECTION message nor OK messages will be sent. In this case, the node N[n-1] will directly elect itself as a leader and send LEADER messages to all other nodes announcing its leadership.

Table 1: Mathematical analysis summary results

			# of elections		
			1	**n/4**	**n/2**
Detector node	**N[1]**	Ours	2	O(N)	O(N)
		Bully	$O(N^2)$	$O(N^2)$	$O(N^2)$
	N[n/2]	Ours	2	O(N)	O(N)
		Bully	$O(N^2)$	$O(N^2)$	O(N)

6. CONCLUSION

In the literature, many algorithms were proposed to perform leader election; one good example is Bully algorithm. Our proposed algorithm relies on the good assumptions of Bully algorithm and introduces enhancements on the leader election operation by sorting the IDs of the nodes and choosing the

node with the next higher ID after the leader got crashed. This mechanism reduces the number of election messages and, in some cases, the number of steps required to complete the election process, therefore, time complexity is also reduced as compared to the Bully algorithm. The complexity of our proposed algorithm is at most O(n) compared to that in Bully which is $O(n^2)$. As a future work, we will extend our proposed algorithm to be applicable in more critical environments, particularly, in the mobile ad hoc distributed systems. Since these systems are more prone to failures than conventional distributed systems. Selecting a leader should become more aware of the aliveness and efficiency of the leader to be elected so as to survive despite failures or disconnections of mobile nodes.

7. REFERENCES

[1] Rebecca Ingram, Patrick Shields, Jennifer E. Walter and Jennifer L. "An Asynchronous Leader Election Algorithm for Dynamic Networks". IPDPS '09 Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing. pp. 1-12, 2009.

[2] Mehdi Mirakhorli, Amir AzimSharifloo, MaghsoudAbbaspour, "A Novel Method for Leader Election Algorithm," The 7th IEEE International Conference on Computer and Information Technology (CIT 2007), pp.452-456, 2007.

[3] Mina Shirali, Abolfazl Haghighat Toroghi, Mehdi Vojdani, "Leader Election Algorithms: History and Novel Schemes". Third International Conference on Convergence and Hybrid Information Technology (ICCIT), vol. 1, pp.1001-1006, 2008.

[4] H.Garcial Molina "Election in a distributed Computing System". IEEE Trans. Comp, 1982, vol31, no. 1, pp. 48-59. 1982

[5] N.Fredrickson and Lynch "Electing a Leader in a Synchronous Ring" ACM, , vol.34 no. 1, pp. 98-115, 1987.

[6] M. Gholipour, M.S. Kordafshari, M. Jahanshahi, A.M. Rahmani, "A New Approach for Election Algorithm in Distributed Systems". The Second International Conference on Communication Theory, Reliability, and Quality of Service, pp.70-74, 2009.

[7] M. S. Kordafshari, M. Gholipour, M.Jahanshahi, A.T. Haghighat "Modified bully election algorithm in distributed systems". CCOMP'05 Proceedings of the 9th WSEAS International Conference on Computers, 2005

[8] Junqueira, F., Reed, B., Sera_ni, M.: Zab: High-performance broadcast for primary-backup systems. In: Proc. of the IEEE Int'l Conf. on Dependable Systems and Networks (DSN-DCCS), 2011.

[9] M. Gholipour, M. S. Kordafshari, M. Jahanshahi, A. M. Rahmani "A New Approach For Election Algorithm in Distributed Systems". 2009 Second International Conference on Communication Theory, Reliability, and Quality of Service, 2009

[10] Sung-Hoon Park "A Stable Election Protocol based on an Unreliable Failure Detector in Distributed Systems". 2011 Eighth International Conference on Information Technology: New Generations. pp. 979 – 984, April 2011

[11] Zargarnataj, M. "New Election Algorithm based on Assistant in Distributed Systems". IEEE/ACS International Conference on Computer Systems and Applications, 2007. AICCSA '07. pp. 324 – 331, May 2007.