Improved Algorithms for Leader Election in Distributed Systems

Mohammadreza Effatparvar*, Nasser Yazdani*, Mehdi Effatparvar†, Aresh Dadlani*‡ and Ahmad Khonsari*‡
*School of Electrical & Computer Engineering, University of Tehran, Tehran, Iran
†Electrical & Computer Engineering Department, Islamic Azad University - Ardabil Branch, Ardabil, Iran
‡School of Computer Science, Institute for Studies in Theoretical Physics and Mathematics, Tehran, Iran
{effatparvar, yazdani}@ut.ac.ir, effatparvar@iauardabil.ac.ir, {a.dadlani, ak}@ipm.ir

Abstract

An important challenge confronted in distributed systems is the adoption of suitable and 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. In this paper, we describe novel approaches towards improving the Bully and Ring algorithms and also propose the heap tree mechanism for electing the coordinator. The higher efficiency and better performance of our presented algorithms with respect to the existing algorithms is validated through extensive simulation results.

1. Introduction

Designating a single node as an organizer in distributed systems is a challenging issue that calls for suitable election algorithms. In distributed systems, nodes communicate with each other using shared memory or via message passing. The key requirement for nodes to execute any distributed task effectively is coordination. In a pure distributed system, there exists no central controlling node that arbitrates decisions and thus, every node has to communicate with the rest of the nodes in the network to make an apt decision. Often during the decision process, not all nodes make the same decision. Not only is the communication between nodes time-consuming, but also is the decision-making process. Coordination among nodes becomes difficult when consistency is needed among all nodes. Centralized controlling nodes can be selected from the group of available nodes to reduce the complexity of decision making. Many distributed algorithms require one node to act as coordinator, initiator, or otherwise perform some special role. In general, it does not matter which node takes on this special responsibility, but one of them has to do it.

Leader election is a technique that can be used to break the symmetry of distributed systems [1]. In order to determine a central controlling node in a distributed system, a node is elected from the group of nodes as the leader to serve as the centralized controller for that decentralized system [2].

Some applications of leader election include finding a spanning tree with the elected leader as root [3], breaking a deadlock, reconstructing a lost token in a token ring network, and adopting leader election in ad hoc networks [4][5]. Leader election algorithms for static networks have been proposed in [6]. These algorithms work by constructing several spanning trees with a prospective leader at the root of the spanning tree and recursively reducing the number of spanning trees to one. However, these algorithms work only in cases where the topology remains static and hence, cannot be used in a mobile setting [3][7].

The purpose of leader election is to choose a node that will coordinate activities of the system [8]. In any leader election algorithm, a leader is usually chosen based on some criterion such as choosing the node with the largest identifier. Once the leader is elected, the nodes reach a certain state known as *terminated* state. In leader election algorithms, such states are partitioned into *elected states* and *non-elected states* [9]. When a node enters either state, it always remain in that state. Every leader election algorithm must be satisfied by the safety and liveness condition for an execution to be admissible [10]. The liveness condition states that every node will eventually enter an elected state or a non-elected state. The safety condition for leader election requires that only a single node can enter the elected state and eventually, become the leader of the distributed system.

Several leader election algorithms such as the Bully algorithm [11], Ring algorithm [12], Chang and Roberts' algorithm [13], Peterson's algorithm [14], LeLann's algorithm [15], and Franklin's algorithm [16] 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.

In this paper, we present improved modifications on the existing Bully and Ring algorithms in Sections 2 and 3, respectively. We also introduce the heap tree method in electing the leader node by imposing lesser complexity in message passing and bandwidth usage in Section 4. In Section 5, we compare our algorithms with the existing algorithms followed by concluding remarks in Section 6.

2. Modified Bully algorithm

As it has been mentioned, the number of messages exchanged between nodes in Bully algorithm is very high. In [17], we presented a new approach based on a sort mechanism to reduce the number of messages. This modified Bully algorithm may, however, consume more time with regard to the actual Bully algorithm in finding and electing the leader. In this section, we introduce another approach to facilitate the Bully algorithm with fault tolerance capabilities.

In this algorithm, when a node, say N, notices that the leader has crashed, it sends an election message to all nodes with higher ID numbers. Each node that receives the election message sends its ID as a response to N. If no node responses to N, it will broadcast a coordinator message to all nodes. If some nodes respond to N, it will select the node with the highest ID number as coordinator and will send a new message with the selected ID number to all nodes, informing them about the new leader. One drawback of this approach is that the message carrying the highest node ID may get lost before reaching N. Therefore, a fault tolerant mechanism is required to prevent such fault. As illustrated in Figure 1, when node N selects the highest ID number, it sends the selected ID to the rest of the nodes. The newly elected leader then sends an election message to nodes with greater ID numbers to ensure that there is exists no node with a greater ID number than itself. If the leader receives a response from nodes with ID numbers greater than its own, it introduces the greatest one as the new leader. Otherwise, it remains unchanged.

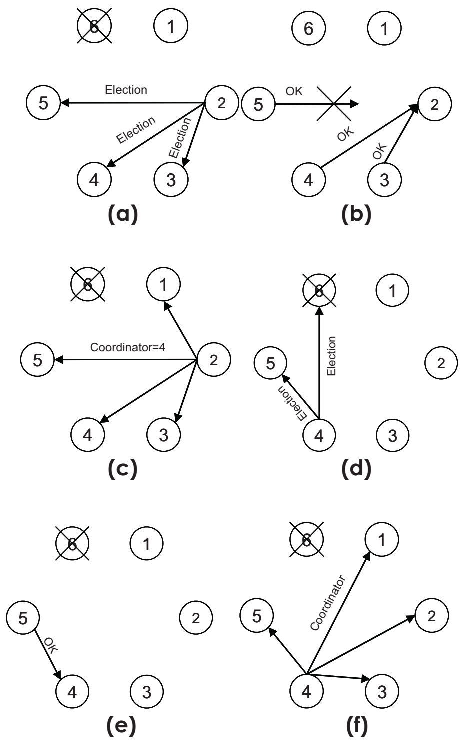

Figure 1. The modified Bully algorithm (a) Node 2 notices the coordinator has crashed & sends an election message to nodes 3,4 & 5 (b) 2 receives OK messages from 3 and 4, but not 5 (c) 4 is elected as leader (d) 4 sends an election message to 5 (e) 5 responds with an OK to 4 (f) 4 introduces 5 as the new leader to the rest.

3. Modified Ring algorithm

In this section, we tend to introduce an appropriate method to modify the Ring algorithm by reducing the number of message passing and additional messages being sent to the elected leader.

As in Figure 2, when a node notices that the leader has crashed, it sends its ID number to its neighboring node in the ring. Thus, it is not necessary for all nodes to send their IDs into the ring. At this moment, the receiving node compares the received ID with its own, and forwards whichever is the greatest. This comparison is done by all the nodes such that only the greatest ID remains in the ring. Finally, the greatest ID returns back to the initial node. If the received ID equals that of the initial sender, it declares itself as the leader by sending a coordinate message into the ring.

It can be observed that this method dramatically reduces

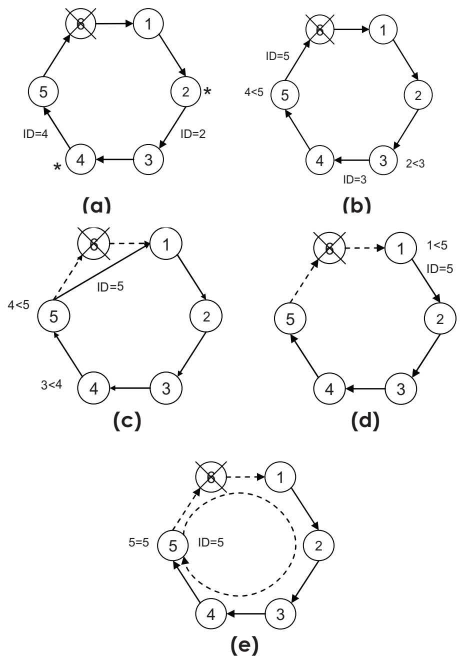

Figure 2. The modified Ring algorithm (a) Nodes 2 & 4 notice that the coordinator has crashed simultaneously (b) They send their IDs into the ring (c,d) The greatest ID always remains in the ring (e) 5 is declared as the leader.

the overhead involved in message passing. Thus, if many nodes notice the absence of the leader at the same time, only the message of the node with the greatest ID circulates in the ring thus, preventing smaller IDs from being sent.

4. Leader election with heap tree

In this section, we describe a novel heap tree-based algorithm for leader election. In this approach, each node of the tree corresponds to an element of the array that stores the value in the node. The tree is completely filled on all levels except possibly the lowest, which is filled from the left up to a point. An array A that represents a heap is an object with two attributes: `length`$[A]$ and `heap-size`$[A]$, which are the number of elements in the array and in the heap stored within array A, respectively. Although A`[1..length`$[A]$`]` may be valid, no element past A`[heap-size`$[A]$`]` is an element of the heap, where `heap-size`$[A] \leq$ `length`$[A]$.

The root of the tree is A`[1]`, and given the index i of a node, the indices of its parent `PARENT`(i), left child `LEFT`(i), and right child `RIGHT`(i) can be computed easily. Based upon the type of heap being used, the values in the nodes satisfy a heap property. The property of the max-heap is that for every node i other than the root, A`[PARENT`(i)`]` $\geq A[i]$, i.e, the value of a node is at most the value of its parent. Thus, the largest element in a max-heap is stored at the root, and the sub-tree rooted at a node contains values no larger than that contained at the node itself. A min-heap is organized in the opposite manner; the min-heap property is that for every node i other than the root, A`[PARENT`(i)`]` $\leq A[i]$. Hence, the root is always the smallest element.

The `MAX-HEAPIFY()` procedure, which runs in $O(\log n)$ time, is the key to maintaining the max-heap property. The `BUILD-MAX-HEAP()` procedure, which runs in linear time, produces a max-heap from an unordered input array. The `MAX-HEAP-INSERT()`, `HEAP-EXTRACT-MAX()`, `HEAP-INCREASE-KEY()`, and `HEAP-MAXIMUM()` procedures, which run in $O(\log n)$ time, allow the heap to be used as a priority queue.

Hereafter, we intend to describe our method using the heap tree characteristics. We adopt the max-heap to explain our algorithm and unlike the Ring algorithm, our approach does not require any connectivity in the group. Also, the nodes need not possess complete information regarding the other nodes. In our method, each node that joins the group records the information about its parent and children. The data item stored in each node is greater than or equal to the data items stored in its children. For the case of simplicity, we implement heaps using arrays rather than linked list-like structures. We simply number the nodes in the heap from top to bottom, numbering the nodes on each level from left to right and store the i^{th} node in the i^{th} location of the array. The height of an n-element heap based on a binary tree is $\log n$, which is good during tree reconstruction. The basic operations on heaps run in time at most proportional to the height of the tree and thus, take $O(\log n)$ time.

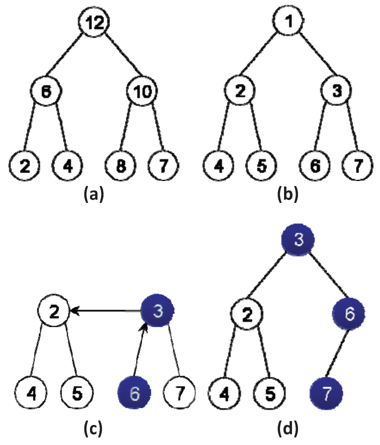

Figure 3. The heap tree algorithm (a) The original nodes (b) Nodes with their indices (c,d) Node comparison according to the algorithm.

In the new heap tree approach, nodes are added to the tree by joining each node and then comparing its ID with its father's ID. If its ID is greater than that of its father's, the node swaps position with the father thus, resulting in a tree reconstruction. In this structure, when the root is deleted from the tree, we say that the leader has crashed. As shown in Figure 3, when a node realizes that the leader has crashed, it sends the election message to its father. This message traverses up to the children of the deleted root where the left and right children of the deleted root compare their IDs with each other to determine the new leader. Thus, it is not necessary for all nodes to start sending their own IDs or election message in the tree. The election message sent by the node reaches its direct father in the tree and at this moment, the receiving node analyzes this message to determine whether it is a duplicate message or not. If duplicate, it is dropped by that node, otherwise it is sent to the next father in the tree. By doing so, the leader can be selected in $O(\log n)$ time at the expense of a comparably reduced number of messages. In this method, each node should save the information of its father, left and right children, and its sibling. Therefore, unlike the Bully algorithm which requires a memory space of n^2, our approach requires a smaller memory space of only $4n$. More details on these algorithms is provided in Table 1.

5. Simulations and evaluation results

In this section, we compare and evaluate the algorithms based on their message passing complexity.

Table 1. Comparison of leader election algorithms

Method	Total Memory Needed	Order	Min. Messages	Max. Messages	Apprx. No. Messages when leader has crashed
Max-Heap	$4n$	$\log n$	$\log n$	$\log n + (n-1)$	$\sum_{i=1}^{k} \lfloor \log(C_i) \rfloor - \left[\sum_{i,j=1; i \neq j}^{k} \lfloor \log(\max(A_i \cap B_j)) \rfloor \right]$; $A_i = \{f(C_i) = [C_i, f(\lfloor C_i/2 \rfloor)] \mid 0 < C_i\}$ and $B_j = \{f(C_j) = [C_j, f(\lfloor C_j/2 \rfloor)] \mid 0 < C_j\}$
Bully	n^2	n^2	$2n-2$	n^2	$N_{(i)} = (n-i+1)(n-i) + (n-1)$
Ring	n^2	n^2	n	n^2	$\sum_{i=1}^{n} (n-i) = 1/2\,[(n-i)(n-i+1)]$

5.1. Analysis of the modified Bully algorithm

In the modified Bully algorithm, if a single node detects the crashed coordinator, $N_{(i)}$ is obtained with an order of $O(n)$ as follows:

$$N_{(i)} = 2(n-i) + (n-2), \tag{1}$$

With fault tolerance, the order of message passing increases to $O(n^2)$ as follows:

$$\begin{aligned} N_{(i)} &= 2(n-i) + (n-2) + 2(n-i') + (n-2), \\ &= 2\left[(n-i) + (n-2) + (n-i')\right], \end{aligned} \tag{2}$$

where i' is the selected leader ID number in the first step. Figure 4(a) plots the Bully algorithm against the modified Bully algorithm for the case when only one node notices that the coordinator has crashed. The number of messages passed in terms of the number of nodes that realize the absence of the coordinator is depicted in Figure 4(b) for the fault-tolerant Bully algorithm, the Bully algorithm based on the sort mechanism, and the heap tree approach.

Table 2 indicates the number of faults, sent and received messages in the fault-tolerant Bully algorithm. For instance, if the 4^{th} node found that the coordinator has crashed, it would send 145 messages to the nodes with greater IDs. However, due to the occurrence of 12 faults in the network, the number of messages received reduces to 133. After a initially being designated, the selected coordinator sends messages to nodes with IDs greater than itself, which is 7 in this case. Since it receives 7 responses from nodes with higher IDs, it declares the final coordinator to the rest.

5.2. Modified Ring Algorithm

In this sub-section, we examine the message complexity of the Ring algorithm with its modified version. If $n_{\{i_1, i_2, \cdots, i_m\}}$ is the number of nodes that concurrently detect the absence of the crashed coordinator and n is the number of nodes in the ring, then the total number of messages passed with an order of $O(n^2)$ is as follows:

$$T = n_{\{i_1 i_2, \cdots, i_m\}} \times n. \tag{3}$$

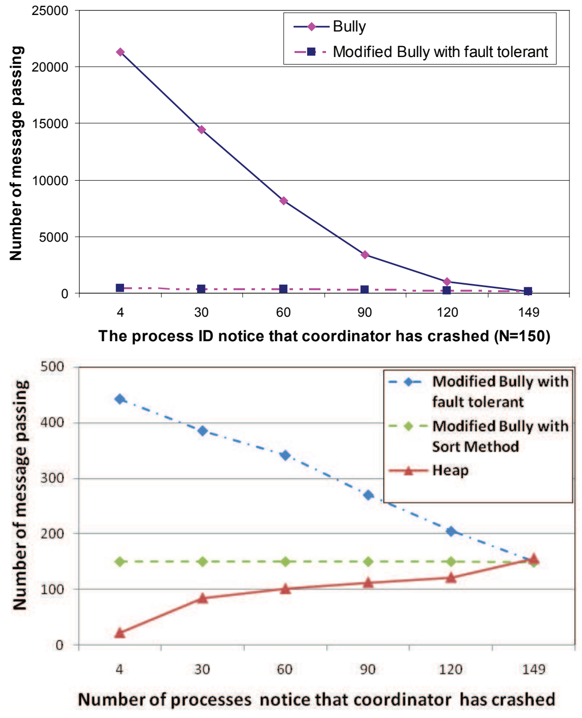

Figure 4. Message passing in terms of nodes that realize that the coordinator has crashed (a) Bully versus fault-tolerant Bully algorithm (b) Fault-tolerant Bully versus sort-based Bully versus heap tree.

Similarly, for the modified Ring algorithm, we have:

$$\sum_{i=0}^{n} i = n(n-1)/2 = 1/2(n^2 - n), \tag{4}$$

with an order of $O(n^2)$ and reduced number of messages passed. Thus, the complexity of modified Ring is much lower than the Ring algorithm. Figure 5 compares the Ring, modified Ring, and heap tree algorithms where the number of nodes in the ring is assumed to be 10.

6. Conclusion and Future Works

Leader election algorithms play a vital role in distributed environments. In this paper, we presented improved modi-

Table 2. Number of Message passing in the fault-tolerant Bully algorithm and the heap tree method

Node ID	Messages in Heap Tree	Messages in Fault-tolerant Bully	Sent Messages	Received Messages	No. of Faults	Messages sent to higher ID	Messages received from higher ID	Coordinator Messages
4	23	442	145	133	12	7	7	149
30	85	385	119	110	9	3	3	149
60	102	341	89	32	57	35	35	149
90	113	270	59	51	8	5	5	149
120	122	205	29	20	9	3	3	149

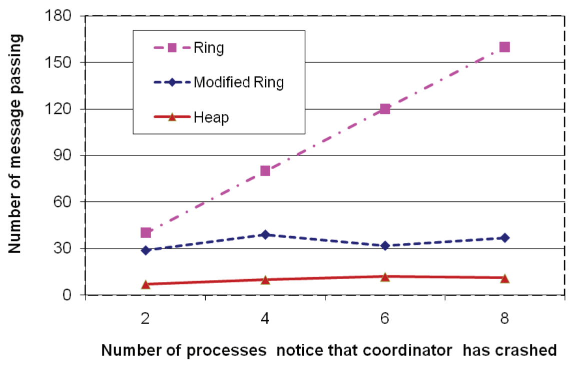

Figure 5. Message passing in Ring, modified Ring and heap tree when several nodes notice that coordinator has crashed simultaneously.

fications to the well-known Bully and Ring algorithms. In addition, we proposed a novel approach known as the heap tree method towards leader election based on the max-heap data structure. The lesser complexity in message passing exhibited by this method is justified through obtained simulation results. In future, we tend to adopt these appraoches in ad hoc and sensor environments.

References

[1] G. Fredrickson and N. Lynch, "The impact of synchronous communication on the problem of electing a leader in a ring", in Proc. 16^{th} ACM Symp. on Theory of Computing, Washington, USA, pp. 493-503, 1984.

[2] R. G. Gallager, "Choosing a leader in a network", Internal Memorandum, Laboratory for Information and Decision Systems, MIT, 1977.

[3] R. G. Gallager, P. Humblet and P. Spira, "A distributed algorithm for minimum weight spanning trees", ACM Trans. on Programming Languages and Systems, vol.4, no.1, pp. 66-77, Jan. 1983.

[4] N. Malpani, J. Welch and N. Vaidya, "Leader election algorithms for mobile ad hoc networks", 4^{th} Intl. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Boston, USA, pp. 96-103, Aug. 2000.

[5] P. Basu, N. Khan and T. Little, "A mobility based metric for clustering in mobile ad hoc networks", In Proc. 21^{st} Intl. Conference on Distributed Coputing Systems, Washington, USA, pp. 413, Apr. 2001.

[6] D. Peleg, "Time optimal leader election in general networks", Journal of Parallel and Distributed Computing, vol.8, no.1, pp. 96-99, Jan. 1990.

[7] P. Humblet, "Selecting a leader in a clique in $O(n \log n)$ messages", Internal Memorandum, Laboratory for Information and Decision Systems, MIT, 1984.

[8] J. Welch and H. Attiya, *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*, 2nd ed. London, UK: McGraw-Hill Publishing Company, 2001.

[9] E. Korach, S. Moran, and S. Zaks, "Tight lower and upper bounds for some distributed algorithms for a complete network of processors", in Proc. 3^{rd} ACM Symp. on Principles of Distributed Computing, Vancouver, Canada, pp. 199-207, Aug. 1984.

[10] P. M. B. Vitanyi, "Distributed election in an Archimedean ring of processors", in Proc. 16^{th} ACM Symp. on Theory of Computing, Washington, USA, pp. 542-547, 1984.

[11] H. Gracia-Molina, "Elections in a distributed computing system", IEEE Trans. on Computers, vol. C-31, no. 1, Jan. 1982.

[12] N. Fredrickson and N. Lynch, "Electing a leader in a synchronous ring", Journal of ACM, vol. 34, no. 1, pp. 98-115, 1987.

[13] E. Chang and R. Roberts, "An improved algorithm for decentralized extrema-finding in circular configurations of processes", Communications of the ACM, vol. 22, no. 5, pp. 281-283, May 1979.

[14] G. L. Peterson, "An $O(n \log n)$ unidirectional algorithm for the circular extrema problem", ACM Trans. Programming Languages and Systems, pp. 758-762, Oct. 1982.

[15] G. LeLann, "Distributed systems - towards a formal approach", Information Processing Letters, pp. 155-160, 1977.

[16] W. R. Franklin, "On an improved algorithm for decentralized extrema finding in circular configurations of processors", Communication of the ACM, pp. 336-337, 1982.

[17] M. Effatparvar, M. R. Effatparvar, A. Bemana, and M. Dehghan, "Determining a central controlling processor with fault tolerant method in distributed system," In Proc. of ITNG'07, May 2007.