Heap Base Coordinator Finding with Fault Tolerant Method in Distributed Systems

Mehdi Effat Parvar 1, Amir Masoud Rahmani2, Mohammad Reza Effat Parvar3, Mehdi Dehghan4

1 ECE Department, Islamic Azad University- Ardabil Branch, Ardabil, Iran
2 ECE Department, Islamic Azad University- Tehran Science and Research Branch, Tehran, Iran
3 ECE Department of Tehran University, Tehran, Iran
4 Computer and IT Eng Department of AmirKabir University of Technology, Tehran, Iran

Abstract

Coordinator finding in wireless networks is a very important problem, and this problem is solved by suitable algorithms. The main goals of coordinator finding are synchronizing the processes at optimal using of the resources. Many different algorithms have been presented for coordinator finding. The most important leader election algorithms are the Bully and Ring algorithms. In this paper we analyze and compare these algorithms with together and we propose new approach with fault tolerant mechanisms base on heap for coordinator finding in wireless environment. Our algorithm’s running time and message complexity compare favorably with existing algorithms. Our work involves substantial modifications of an existing algorithm and its proof, and we adapt the existing algorithms to the noisy environment base on fault tolerant mechanisms.

1. Introduction

Election of a leader is a fundamental problem in distributed computing. It has been the subject of intensive research since its important was first articulated by Gerard Lelann [1]. The practical importance of election in a distributed computing is further emphasized by Garcia Molina’s Bully algorithm [3].

In a pure distributed system, there is no central controlling processor that arbitrates decisions. Without a central authority or coordinator, any processor has to communicate with all processors in the network to make decision. Often during the decision process, not all processors make the same decision. Communication between processors takes time and further more, making the decision takes time. Coordination among processors becomes difficult when consistency is needed among all processors.

Leader election is a technique that can be used to break the symmetry of distributed Systems [5].

Some applications of leader election include finding a spanning tree with the elected leader as root[18], breaking a deadlock, reconstructing a lost token in a token ring network, using leader election in Ad Hoc network [16,17].

Leader election algorithms for static networks have been proposed in [19]. 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 if the topology remains static and hence cannot be used in a mobile setting [8,9].

Leader election is a useful building block in distributed systems, whether wired or wireless, especially when failures can occur. For example, if a node failure causes the token to be lost in a mutual exclusion algorithm, then the other nodes can elect a new leader to hold a replacement token. Leader election can also be used in group communication protocols, to choose a new coordinator when the group membership changes. The standard definition of the leader election problem for static networks [4] is that:

1. Eventually there is a leader and
2. There should never be more than one leader.

In general election algorithm attempt to locate the process with the highest process number and designate it as coordinator the algorithm differ in the way they do the location. Furthermore we also assume that every process know the process number of every other process. What the processes do not know is which ones are currently up and which ones are currently down. The goal of an election algorithm is to ensure that when an election starts, it concludes wit all processes agreeing on who the new coordinator is to be.

Several leader election algorithms have been proposed over the years [5-12]. Some of the grand election algorithms that we can mention to them are Bully algorithm, Ring algorithm, Chang and Roberts’ algorithm [13], Peterson’s election algorithm [15], Lelann’s algorithm [1], Franklin’s algorithm [14]. Such leader election algorithms proposed until now require processors to be directly involved in leader election. Information is exchanged between processors by transmitting messages to each other. The processors exchange messages with each other and try to reach an agreement. Once an agreement is reached, a processor will be elected as leader and all other processors will acknowledge the presence of the leader.

The rest of paper organized as follow: in section 2 we describe our modified Bully algorithm, section 3 propose the heap base election, in section 4 we explain the fault tolerant method base on heap tree, section 5 discusses...
about analytical simulation and section 6 presents the conclusion and future works.

2. Modified Bully Algorithm with Fault Tolerant Mechanism

As it has been mentioned, in Bully algorithm the number of messages that should be exchanged between processes is very high. In previous section we presented the new approach with sort mechanism and we decreased the number of messages. Bully with the sort mechanism has a little message passing, but it may consume more time in contrast with Bully algorithm to find the leader, we describe another approach to modify Bully. In this algorithm [2] when process P notices that the leader has crashed, it sends an election message to all processes with higher ID number.

![Diagram of Bully Algorithm](image)

Figure 1: Process 2 notices the coordinator has crashed so sends an election message to processes 3,4,5(a), and receives OK message from processes 3,4 it means that Ok message of process 5 is failed (b), then process 4 is selected as a coordinator (c), and process 4 according to algorithm sends an election message and selects process 5 as a coordinator (d,e,f).

Each process that receives election message sends its ID as a respond to process P. If no process responses to process P, it will broadcast one coordinator message to all processes. If some processes response to process P, it will select the process with the highest ID number as coordinator and that will send a new message with selected ID number to all processes. In this manner all processes know the new leader. This approach is a suitable way to select the leader, but when the process with the highest ID number is sent, its message to process P maybe lost. So we want to present a fault tolerant mechanism to prevent this fault. To do this, when process P selected the highest ID number it sends the selected ID to all processes, now the new leader sends election message to processes with greater ID number to be sure that there is no process with great ID numbers. If a message is received from processes with great ID numbers, it introduces the greatest one as leader. Otherwise it remains the leader again.

In the rest of this paper we compare the number of message passing of these methods with together and also we compare them with Modified Ring algorithm [2].

3. Leader Election with Heap Tree

In this section we will describe other leader election algorithm. In this method we use the heap tree for selecting the leader. 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], which is the number of elements in the array, and heap-size [A], the number of elements in the heap stored within array A. That is, although A[1 .. length[A]] may contain valid numbers, no element past A[heap-size[A]], where heap-size[A] ≤ length[A], is an element of the heap.

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 simply.

On most computers, the LEFT procedure can compute 2i in one instruction by simply shifting the binary representation of i left one bit position. Similarly, the RIGHT procedure can quickly compute 2i +1 by shifting the binary representation of i left one bit position and adding in a 1 as the low-order bit. The PARENT procedure can compute \[\lfloor i / 2 \rfloor\] by shifting i right one bit position.

There are two kinds of binary heaps: max-heaps and min-heaps.

The values in the nodes satisfy a heap property, the specifics of which depend on the kind of heap. In a max-heap, the max-heap property is that for every node i other than the root, A[PARENT(i)] ≥ A[i], that is, 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 way; the min-heap property is that for every node i other than the root, A[PARENT(i)] ≤ A[i].

The smallest element in a min-heap is at the root. 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 data structure to be used as a priority queue.

After explaining the heap we intend to describe our method by using the heap tree characteristics. We use the max heap to explain our algorithm and unlike the Ring algorithm we don’t need any ring for connect to group; also the node doesn’t need to have complete information about the other nodes.

In this method each node that joins to group should save 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. We could implement heaps using a linked list like structure, like we did with binary trees but in this instance it is actually easier to implement heaps using arrays. 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 $\log n$; so it is good when we want to reconstruct the tree. The basic operations on heaps run in time at most proportional to the height of the tree and thus take $O(\log n)$ time.

In new method by heap tree, node can join to tree by joining to each node, then it will be compare its ID with fathers ID and then if its ID is greater than the fathers ID, node change its locate with its father and the tree reconstruct again. When the leader crashed it means that our root has been deleted in tree.

When a process notices that the leader has crashed, it starts sending the elect message to its father. This message goes up to deleted roots children and then the receiving process analysis this message to find out the duplicate message. If this message was duplicate then it drops by the node but if it was not duplicate the message send to next father in the tree. By doing this method the leader will be selecting in $O(\log n)$ and with suitable number of message. This method unlike the bully algorithm doesn’t need a lot of buffer. In this method each node should save the 4 node information; father, left and right children and its brother. So the memory needed of this method is $4n$ but in bully it was $n^2$. In table (1) you can see more information about these algorithms.

![Figure 2](image_url)

**Figure 2:** (a) shows the original node, (b) shows these nodes with index, (c) and (d) show the algorithm by comparing nodes with together.

<table>
<thead>
<tr>
<th>Method</th>
<th>Total Memory need for all nodes</th>
<th>Order</th>
<th>Min Messages</th>
<th>Max Messages</th>
<th>Number of Message for processes notice that coordinator has crashed (Approximately)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Heap</td>
<td>$4n$</td>
<td>$\log_2 n$</td>
<td>$\log_2 n$</td>
<td>$\log_2 n + (n-1)$</td>
<td>$\sum_{i=1}^{k}</td>
</tr>
<tr>
<td>Bully</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$2n - 2$</td>
<td>$n^2$</td>
<td>$N_{(i)} = (n - i + 1)(n - i) + (n - 1)$</td>
</tr>
<tr>
<td>Ring</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n$</td>
<td>$n^2$</td>
<td>$\sum_{i=1}^{n}(n - i) = 1/2(n^2 - (n + 1))$</td>
</tr>
</tbody>
</table>

4. Fault Tolerant Base on Heap Tree Method

The main problem of this new method is occurred when two nodes drop in the same time at the network.

---

**Table 1:** Comparison of the Leader Election algorithms

<table>
<thead>
<tr>
<th>Method</th>
<th>Total Memory need for all nodes</th>
<th>Order</th>
<th>Min Messages</th>
<th>Max Messages</th>
<th>Number of Message for processes notice that coordinator has crashed (Approximately)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Heap</td>
<td>$4n$</td>
<td>$\log_2 n$</td>
<td>$\log_2 n$</td>
<td>$\log_2 n + (n-1)$</td>
<td>$\sum_{i=1}^{k}</td>
</tr>
<tr>
<td>Bully</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$2n - 2$</td>
<td>$n^2$</td>
<td>$N_{(i)} = (n - i + 1)(n - i) + (n - 1)$</td>
</tr>
<tr>
<td>Ring</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n$</td>
<td>$n^2$</td>
<td>$\sum_{i=1}^{n}(n - i) = 1/2(n^2 - (n + 1))$</td>
</tr>
</tbody>
</table>
This problem maybe happens because of power limited nodes speed and other general network problems.

When these dropped nodes have a same father, or when the father and one of its child crashed, we cannot finding the coordinator by this method; because we lost the property of the tree and the tree will not remain connective.

![Diagram of nodes and trees](image)

**Figure 3:** (a) shows the original node, (b) shows these nodes with index, (c) shows the drop crashed nodes, (e) the children with lower index determine as a new root, (f) after doing the replacement nods knew by their new index, (g) all nodes from top to down doing this method, (h) the final state of the tree, (i) when one new node join to the tree, it will go to best position.

We apply our method in binary tree instead of the heap tree; so of first we construct the binary tree. When the new node want to add the tree. It sends a join message to the root and the root finds the best place for it. The root try to build a complete tree so it insert the new node in the tree; then the new node sends the Ack message to its father and the Ack message raise in the tree until it receives by the root.

Briefly we can say that in any add or delete process the root update its information about the tree and it can hold the tree as a complete tree in others insertion process.

In this method after the heap tree constructed the node positions number released instead of node ID’s; so the root was the first node and its children are the second and third node in the heap.

The root determines the nodes ID by their positions in the tree; not only by their ID number. So after that the nodes know their father by the positions of them. When the node in position i notices that its father is not exist in [i/2] it determines the new father between itself and its brother, and the others sub tree will update.

By doing this method when more than one node crash in the network, we can hold the system in stable state and finding the coordinator will be do as before.

### 5. Simulation and Evaluation Results

After describing the represented algorithm, in this section we will compare and evaluate the gained results. We will also compare the complexity of message passing between the algorithms and show the improvement of them.

In Modified Bully with Fault Tolerant Mechanism we have previous parameters, so if only one process detects crashed coordinator we have:

\[ N_{(i)} = (n - i) + (n - i) + (n - 2) \]

\[ = 2(n - i) + (n - 2) \]  \hspace{2cm} (1)

Which has Order \( O(n) \). When we use the fault tolerant mechanism according to figure (4) message passing will increase.

\[ N'_{(i)} = 2(n - i) + (n - 2) + 2(n - i) + (n - 2) \]

\[ = 2(n - i) + (n - 2) + (n - i) \]  \hspace{2cm} (2)

In which \( i' \) is a selected leader ID number in first step. The Order of this method is \( O(n^2) \). Figure (4) shows the comparison between Bully Algorithm and modified Bully with fault tolerant mechanism. Figure 5 shows the messages number between modified Bully with fault tolerant, Bully with sort method, and Heap.

![Graph of message passing](image)

**Figure 4:** The comparison of message passing in Bully and Modified Bully with fault tolerant (If only one process notices that the coordinator has crashed

The complexity of message passing in modified Ring Algorithm is [2]:

\[ \sum_{i=0}^{n} i = n(n - 1)/2 = 1/2n^2 - n \]  \hspace{2cm} (3)

Which has Order \( O(n^2) \).

So the complexity of modified Ring is much lower than the Ring Algorithm.
faults have occurred. One of these faults is the root crash. In average, 13% of times root and another node crash with root. In average, 21% of times two other nodes else of root crash in the tree; that these nodes are may be two brothers or one father and one child.

Figure 5 shows the comparison between Ring, modified Ring and Heap Tree algorithms. In our simulation we assume that the number of existing processes in the ring is 10 and the topology of the Ring has been products randomly. The number of message passing in different status is shown when several processes notice that coordinator has crashed concurrently.

In the simulation of fault tolerant heap tree we considered 10 nodes with 13 times run in each state.

In average 13% of times root and another node crash with root, and in 21% of times two other nodes else of root crash in the tree; that these nodes are may be two brothers or one father and one child.

Figure 6 shows 20 times of run in the network.

Table 2 indicates the number of fault, sent and received message in modified Bully with fault tolerant mechanism. For example, if the 4-th process found that the coordinator has crashed, it will send 145 messages to the processes with larger IDs. But, because of the occurrence of the fault in the network, it is impossible to receive all the messages, therefore, it will receive 133 messages. So, 12 faults have occurred. After the primary determination of the coordinator by a message the selected coordinator again sends messages to processes with greater ID than itself and this number is produced at random and it is 7. It also receives 7 messages from these processes and finally the main coordinator will be selected and all the processes receive its ID.

6. Conclusion and Future Works

Election Algorithms in distributed system play an important role in the system operation. The important algorithms for this kind of work are Bully and Ring Algorithms. The approaches presented in this paper have improved these two algorithms and we presented the new other method for selecting the leader that we call it Heap tree method. In Bully with sort mechanism [2], the message complexity reduces significantly, but the time spent to select leader is a little large because we need a timer.

Another method to improve Bully algorithm was used simultaneously with fault tolerant mechanism. This algorithm reduces the number of message passing to select the leader, and we compare it with Bully algorithm. In this paper a new method is also introduced to improve Ring Algorithm. Modified Ring Algorithm is a suitable approach to leader election and it reduces the message complexity when several processes concurrently notice that the coordinator has crashed. So the bandwidth and message passing will be improved. Finally we create the other algorithm that we call it Heap tree method. In this method we use the max heap to select the leader; and we shown this method has minimum order in compare the other algorithms.

In future works we will use these algorithms to leader election in ad hoc and sensor network as a distributed form. It is also possible to use the election algorithms in dynamic environments.

<table>
<thead>
<tr>
<th>Process ID</th>
<th>Number of Message in Heap Tree</th>
<th>Number of Message in Bully with fault tolerant</th>
<th>Sent messages</th>
<th>Received messages</th>
<th>Number of fault</th>
<th>message to process with larger ID</th>
<th>Received message from process with larger ID</th>
<th>Coordinator messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>23</td>
<td>442</td>
<td>145</td>
<td>133</td>
<td>12</td>
<td>7</td>
<td>7</td>
<td>149</td>
</tr>
<tr>
<td>30</td>
<td>85</td>
<td>385</td>
<td>119</td>
<td>110</td>
<td>9</td>
<td>3</td>
<td>3</td>
<td>149</td>
</tr>
<tr>
<td>60</td>
<td>102</td>
<td>341</td>
<td>89</td>
<td>32</td>
<td>57</td>
<td>35</td>
<td>35</td>
<td>149</td>
</tr>
<tr>
<td>90</td>
<td>113</td>
<td>270</td>
<td>59</td>
<td>51</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>149</td>
</tr>
<tr>
<td>120</td>
<td>122</td>
<td>205</td>
<td>29</td>
<td>20</td>
<td>9</td>
<td>3</td>
<td>3</td>
<td>149</td>
</tr>
<tr>
<td>149</td>
<td>156</td>
<td>150</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>149</td>
</tr>
</tbody>
</table>
7. References