Leader Election Algorithm Using Heap Structure

Sepehri, M., Goodarzi, M.

Sepehri, M.
Department of Computer Engineering & IT, Islamic Azad University
Fars Sciences and Research Branch, Fars
IRAN

Goodarzi, M.
Department of Civil Engineering, Iran University of Science and Research,
Tehran
IRAN

Abstract: Several distributed algorithms require that there be a coordinator process in the entire system. Since all other processes in the system have to interact with the coordinator, they all must agree on who the coordinator is. Furthermore, if the coordinator process fails due to the failure of the site on which it is located, a new coordinator process must be elected to take up the job of the failed coordinator. Election algorithms are meant for electing a coordinator among currently running processes. This paper deals with the distributed leader election algorithm for a set of processes connected by a tree network. In this paper, we propose a linear time algorithm for leader election using heap structure and describe it in details.

Keywords: Leader Election, Distributed Algorithm, Heap Structure, Tree.

1 Introduction

Electing a leader is a fundamental problem in distributed systems and it is studied in a variety of contexts. The election problem is standard and many solutions are known [8, 13]. It has been solved under many different assumptions: the graph can be a ring, a tree, a complete graph or a general connected graph. The leader election problem in a distributed system is to find exactly one processes as a coordinator for performing some special tasks. It is a fundamental problem for achieving fault tolerance and reducing resource utilization under distributed networks
and is studied extensively in different computation models [14]; see [6, 13] for more information. Consider any network of $N$ nodes; a leader node is defined to be any node of the network identified by some characteristics (unique from all other nodes). A leader election process is defined to be a uniform algorithm (code) executed at each node of the network; at the end of the algorithm execution, exactly one node is elected the leader and all other nodes are in the non-leader state [15]. A large body of research has already focused on finding efficient solutions to elect one process among an $n$-process under various assumptions [12, 13].

1.1 The model

This paper deals with model follows the standard models for distributed systems given in [7, 5, 13]. The communication model is a point-to-point communication network which is represented as a simple connected undirected graph, where the vertices represent processes and two vertices are linked by an edge if the corresponding processes have a direct communication link. In this paper, we assume that the graph is a tree: a connected graph without cycle. In this model, each process in the system has a unique priority number. Processes communicate by message passing. Whenever an election is held, the process having the highest priority number among the currently active processes is elected as the coordinator. On recovery, a failed process can take appropriate actions to rejoin the set of active processes. Our algorithm is based on heap structure and for simplicity; in the description of the algorithms we will assume that there is only one process on each node of the distributed system.

1.2 Related works

Garcia-Molina [4] has proposed the Bully Algorithm for a leader election which in the worst case requires $O(n^2)$ messages. However, in the best case, the Bully Algorithm requires only $n-2$ messages. Tanenbaum [12] and Silberschatz et al.[10] have described A Ring Algorithm. In the Ring Algorithm, irrespective of which process detects the failure of the coordinator and initiates an election, an election always requires $2(n-1)$ messages. Gallager et al.[3] have developed a leader election algorithm for arbitrary networks whose message complexity is $O(N \log N + E)$ where $N$ is the number of nodes and $E$ is the number of edges in the network. Santoro in [9] has studied how the knowledge of topology of the network and the orientation of the links in the network affect the message complexity of leader election algorithms in networks. Subsequently, many authors have studied leader election algorithms for various kinds of networks.

1.3 Outline of the paper

The paper is organized as follows. In Section two we briefly review basic concepts which are use in our algorithm. In section three a leader election algorithm using heap structure is studied. In section four time complexity analysis of the mentioned algorithm is introduced. Section five is devoted to conclusion.

2 Basic concepts

First, we should define some basic concepts which are use in our algorithm. Each of these
basic concepts is discussed briefly in the sections that follow:

2.1 Tree network

A tree network with $P$ processes provides communication between every pair of processes with a minimal number of links ($P-1$). One advantage of a tree is its low connectivity; the failure of any one of its links creates two subsets of processes that can not communicate with each other [1].

2.2 The heap structure

The heap data structure is an array object that can be viewed as a nearly complete binary tree. Each node of the tree corresponds to an element of the array that stores the value in the node. There are two kinds of binary heaps: max-heap and min-heap. In this paper, we use max-heap. In a max-heap, the max-heap property is that 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 [9].

2.3 Single node broadcast

In the first communication task we want to send the same packet from a given process to every other process (we call this a single node broadcast). Clearly, to solve the single node broadcast problem, it is sufficient to transmit the given node’s packet along a spanning tree rooted at the given node. With an optimal choice of such a spanning tree, a single node broadcast takes $O(r)$ time, where $r$ is the diameter of network [1].

3 Leader election algorithm using heap structure

Input: All processes $p_1, p_2... p_n$ from an array $A$

Output: The process with the maximum priority number is stored at the root of tree $T$ declares itself the leader

Elect Leader (T)
{
    Build-Max-Heap (A); // this procedure goes through the remaining nodes of the tree and run Max-Heapify on each one
    Leader = T->info; // the largest element in a max-heap is stored at the root
    Leader-Status=True; // the process which is stored at the root will take over the election activity
    Single node broadcast (); // leader sends a coordinator message to all processes, informing them that from now on it is the new coordinator
}

For all $p_i$ do
    If $p_i$ sends a request to leader and does not receives a reply with a fixed timeout
    {
        Leader-Status=False; // the process which is stored at the root will not take over the election activity
        Number=Leader priority number; // leader number will be used after recovery action
        Leader priority number=0; // allocate zero to leader so it can not take over the election activity
    }
}
Elect Leader (T); // call Elect Leader with the new input configuration, so the process whose its priority number is equal to the root will take over the election activity 

} If recovery action performs 

{  
   Leader priority number=Number; // a failed process rejoin the set of active process with its previous number 
   Elect Leader (T); // call Elect Leader with the new input configuration 
}

In the above algorithm, each process has a unique priority number. Whenever an election is held by calling Elect Leader() procedure, Max-Heap was built and the highest priority number among the currently active processes which is stored at the root of tree is elected as the coordinator. When a process (say \( p_i \)) sends a request message to the leader and does not receive a reply within a fixed timeout period, it assumes that the leader has failed. Then the priority number of the leader became zero. Therefore it does not take over the leader activity in the next call. Leader broadcasts a coordinator message to all processes, informing them that from now on it is the new coordinator. As part of recovery action, this method requires that the failed process must initiate an election on recovery.

Thus the running time of our proposed leader election algorithm using heap structure is linear and equal to \( O(p) + \Theta(\log p) \).

5 Conclusion

We proposed an algorithm for leader election in tree network. The leader election algorithm we introduce and study here is based on heap structure. A tree \( T \) of size processes \( P \) is initially built and when the algorithm terminates, there is a unique process at the root of the tree that is strictly greater than the others. Therefore, the process which is stored at the root will take over the coordinator activity and broadcast the others that from now on it is the new coordinator. We analyzed the algorithm and reached the total cost of \( O(p) + \Theta(\log p) \).

4 Time complexity analysis

The above algorithm consists of two main procedures such as Build-Max-Heap () and Single node broadcast (), so we should analyze the running time of these procedures. The running time of optimal algorithm for the basic communication problem using tree network with \( p \) processes for Single node broadcast () is \( \Theta(\log p) \) [1]. The total cost of Build-Max-Heap () is \( O(n) \) [17].

References


