Leader Election Algorithm in Wireless Environments Using Fibonacci Heap Structure

Arihant Kumar Jain¹
Department of Computer Engineering,
University College Of Engineering, Rajasthan Technical University, Kota (INDIA)
¹erarihant@rediffmail.com

Ramashankar Sharma²
²sharma_rs@yahoo.com

Abstract

Today is an era of wireless communication systems. The day is not so far when the whole distributed system of the world will goes wireless. Leader in a distributed system plays a vital role in various computing environments like wireless networks etc. The main focus of researchers is on electing a new leader with lesser time consumption and with transmitting minimum number of messages in the network. Election of a leader using binary heap tree mechanism has garnered attention recently. But there are many other variants of heaps available in the literature which has better running time as compared to binary heaps. This paper presents design of leader election algorithm using a better variant called Fibonacci heap structure. Our design elects a new leader in O(1) time by passing lesser number of messages. The design is suitable in wireless environments.

1. Introduction

In any leader election algorithm, a leader is usually chosen based on some criterion such as choosing the node with the largest identifier such as highest battery life, computing power etc. in case of mobile ad hoc networks or with the lowest one such as the clock time of the first node when it enters into the system. This criterion is called its unique identifier (UID) [4, 8]. We assume that each node will have its own UID unique throughout the network and the comparison of this UID will account for leader election. Say, there are N nodes in a network, so after the termination of the algorithm, exactly one node will be designated as the new leader and remaining N-1 nodes will be informed of the new leader. For simplicity in the description of our algorithm, we assume that there is only one process at each node of the distributed system. These N-1 nodes are called the candidate nodes for leader election. However, at some later stages one node from these N-1 nodes can be elected as the new leader depending on the criterion. The only thing is that after the execution of the algorithm there should be at least and at most one leader which is the standard condition and it should always be satisfied.

This paper follows the standard models for distributed systems given in [9-11]. Leader election has been solved under many different assumptions: the graph can be a ring [3, 7], a tree [12-14], a complete graph [15] or a general connected graph. In this paper, we assume that the graph constructed will gains a shape of Fibonacci heap [16]. 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. Processes communicate by message passing through these edges. 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.

Leader election in wireless environments can also be read in the previous literature [17, 18]. Our design works much better in case of faulty environments. Suppose there is a faulty system in which a node becomes leader and before the role of this leader has been done, it stops working. So for what purpose this leader was elected? If the time to elect a new leader is greater than the time to complete its job in the network, then the algorithm designed for electing that leader should again needs to be reviewed. Our works involves substantial modifications of an existing algorithm and its proof. Our design exploits the properties of Fibonacci Heaps and its operations [16, 19]. After comparing with other variants of heaps, we find Fibonacci heap better candidate for leader election algorithms. The higher efficiency and better performance of our presented algorithms with respect to the existing algorithms is validated through extensive simulation results.

2. Leader Election using Fibonacci Heap

In this section, we will describe our leader election algorithm. In this method, we use the Fibonacci heap tree for selecting the leader. Each node of the tree corresponds to an element that stores the value (taken as the UID) in the node. We have to find the minimum key of the Fibonacci heap and designate it as the leader.

A Fibonacci heap [19] is a collection of heap-ordered trees. It consists of a forest of trees. A given Fibonacci heap \( H \) is accessed by a pointer \( \text{min}[H] \) to the root of the tree containing a minimum key; this node is called the minimum node of the Fibonacci heap and this will be elected as the leader node. The pointer \( \text{min}[H] \) thus points to the node in the root list whose key is minimum. If a Fibonacci heap \( H \) is empty, then \( \text{min}[H] = \text{NIL} \). The roots of all the trees in a Fibonacci heap are linked together using their left and right pointers into a circular, doubly linked list called the root list of the Fibonacci heap. The siblings at each level are also connected in such manner. The order of the trees within a root list is arbitrary. The advantage of circular, doubly linked list used in Fibonacci heap is that first we can remove a node from a circular, doubly linked list in \( O(1) \) time and second, given two such lists, we can concatenate them (or "splice" them together) into one circular, doubly linked list in \( O(1) \) time. We are trying to exploit this property of Fibonacci heap.

The root of the tree is \( H[1] \), and given the index \( i \) of a node, the indices of its parent \( \text{PARENT}(i) \), child \( \text{CHILD}(i) \), left sibling \( \text{LEFT}(i) \), and right sibling \( \text{RIGHT}(i) \) can be computed easily. In our method, each node that joins the group records the information about its parent, child, left and right sibling. The nodes need not possess complete information regarding the other nodes of the graph. The values in the nodes (taken as UID) satisfy the heap property. The property of the min-heap is that for every node \( i \), \( H[\text{PARENT}(i)] < A[i] \), i.e., the value of a node should always be greater than the value of its parent. Thus, the smallest element in a Fibonacci heap is stored at the root, and the subtree rooted at a node contains values no smaller than that contained at the node itself. Hence, the root is always the smallest element.

Now, to make an empty Fibonacci heap, the \( \text{FH-Make}(H) \) is used. This procedure allocates and returns the Fibonacci heap object \( H \), where heapsize\( [H] = 0 \) and \( \text{min}[H] = \text{NIL} \) as there are no trees in \( H \). Thus, the cost of \( \text{FH-Make}(H) \) is equal to \( O(1) \) actual cost.

To insert a new node, the procedure \( \text{FH-Insert}(H, x) \) is called which will insert a node \( x \) in Fibonacci heap \( H \). Node \( x \) is added to the left of \( \text{min}[H] \) node creating a new heap with only one element and then its value is compared with \( \text{min}[H] \). If the value of node (cannot be equal as this value is UID) \( x \) is greater than that of \( \text{min}[H] \), then it is simply added to the root list. If its value is smaller than that of \( \text{min}[H] \), then \( \text{min}[H] \) pointer will starts pointing the value \( x \) and it will be elected as the new leader and all other nodes will be informed of the new leader. In both cases, it does not make any effect on the other parts of the heap and it takes only \( O(1) \) time to insert a new node and elect a leader as there is always a pointer that points to the root of the heap i.e. \( \text{min}[H] \).

When a node is inserted into an empty Fibonacci heap, then \( \text{min}[H] \) will points to that node as there are no other nodes in the network. New nodes are inserted into the root list according to the \( \text{FH-Insert}(H, x) \) and leader will be elected accordingly. At that time, suppose nodes are only inserted in sequence into the network and no deletion e.g. a crash has been done yet, then our Fibonacci heap will works like a bidirectional ring. This is because the root list is connected in a circular, doubly linked list way. The Fibonacci heap will gain its shape of a tree or restructure itself only when a node has been deleted from the system. So our design works much better in wireless networks as they are unstable and nodes need not stay for long in the system. It just come, completes its work and leaves the system. So frequency of insertions and deletions in the system is high.

In our design, when the root is deleted from the tree, we say that the leader has crashed. Its children will become roots of new trees. The trees with same degree (here degree means the number of children) are merged with each other using procedure \( \text{FH-Union}(H1,H2) \). In
this procedure, min[H1] and min[H2] are compared. The smaller key remains at the root and the other becomes child of that root. This justifies the heap property also. This is repeated until every root has a different degree. At last, we check each of the remaining roots and find the minimum and elect it as the new leader. Therefore the amortized running time of this procedure is $O(\log n)$.

3. Analysis of Leader Election using Fibonacci Heap

In this section, we will analyse our design of leader election algorithm. The procedures FH-MAKE(H), FH-Insert(H, x), FindMin(), DecreaseRoot(), FH-Union(H1,H2) runs in $O(1)$ time and DeleteMin() runs in $O(\log n)$ amortized time. In our method, each node should not save the information about all the nodes of the network. It just stores information of its parent, left and right sibling, and its child. Therefore, unlike the Bully algorithm which requires a memory space of $n^2$, our approach requires a smaller memory space of only $4n$. In addition, two other fields at each node will be of use. The number of children in the child list of node $x$ is stored in degree[$x$] and the boolean-valued field mark[$x$] indicates whether node $x$ has lost a child since the last time. The maximum degree $D(x)$ of any node in an $n$-node Fibonacci heap is $O(\log n)$. The maximum number of messages transmitted when the leader has crashed is of order $O(\log n)$.

4. Conclusion and Future Works

In this paper, we presented a novel approach towards leader election using Fibonacci heap by electing minimum node as leader. Fibonacci heaps offer a good example of data structure designed with amortized analysis in mind. However the resulting structure is a little complicated, but it can be made useful in practical cases of leader election algorithms. The design is suitable in wireless networks in spite of the fact that they are unstable and prone to faults. We concluded that higher the system is immune to the faults, the better our design works. The lesser complexity in message passing exhibited by this method is justified through obtained simulation results. In future, we tend to adopt these approaches in mobile ad hoc and sensor environments.

5. References