An Optimal Distributed Algorithm for Failure-driven Leader Election in Bounded-Degree Networks

Yuan-Chieh Chow Kenneth C.K. Luo Richard Newman-Wolfe

Department of Computer and Information Science
University of Florida

Abstract

We consider the leader election problem in point-to-point network where the number of communication links for each node is bounded. Instead of focusing on electing a leader as an initialization step, this paper emphasizes the election of the leader in a fault-tolerant distributed system where the election is triggered by the failure or the abdication of the current leader; it is assumed that the number of initiators that detect the failure or the abdication and start the election algorithm is bounded. A leader election algorithm is presented. The algorithm is based on a new approach that is sharply different from those of previous works: timestamped multiple-sourced flooding. The worst-case time and communication complexities of the algorithm are both optimal; the former is $O(D)$ and the latter is $O(E)$ equivalent to $O(N)$ in our network model.

1. Introduction

For various reasons, it is desirable to elect a leader among all the nodes in a network. For instance, a network may need a lock-coordinator to handle concurrency control for a distributed database, or a leading file server to keep master copies of the files. As each node is considered to be an autonomous entity, the process of election is done by exchanging messages. When a node wishes to be a candidate for the leadership, it initiates the election by broadcasting a message to other nodes for their voting support. Thus, nodes competes for the leadership and eventually only one prevails.

A closely-related problem is the construction of minimum spanning trees in a network, one of the most intensively studied problems in distributed computings [2][5][6][10][11]. The problem of leader election and that of constructing minimum spanning trees can be reduced to each other easily; an efficient distributed algorithm for the former can be transformed into one for the latter with only minor modifications, and the reverse is true. The pioneering work of distributed minimum-spanning-tree algorithms was done by Gallager, Humbelt and Spira [11]. Their algorithm, the GHS algorithm for short, has had a profound impact on subsequent works.

The GHS algorithm is based on the idea of *searching*-and-*combining*. It works by merging fragments, starting from each node being a zeroth-level fragment, level by level while in the meantime these fragments continue to find the best edges for the tree edges. The algorithm proceeds in a bottom-up fashion by combining fragments until a final fragment results which is made of the edges of a minimum spanning tree. The communication complexity of the GHS algorithm is $O(E+N\log N)$ while its time complexity is $O(N\log N)$.

Gafni [10], Chin and Ting [6] have proposed algorithms that improve the time complexity to $O(N\log^* N)$, where $\log^* N$ is the number of times that the log function has to be applied to N to yield a result less than or equal to 1. In [5], Awerbuch proposes an algorithm with $O(N)$ time complexity and $(E+N\log N)$ communication complexity, which is one of the best algorithms available in the sense that the time complexity and communication complexity are well-balanced, Lien [12] gave an algorithm that has an average time complexity of $O(D)$ and average communication complexity of $O(E)$, based on his simulations. The algorithm provided by Ahuja and Zhu [2] also achieves the same average performance. Recently, Peleg [14] proposed a fast algorithm for leader election. The worst-case time complexity of the algorithm is $O(D)$ but its communication complexity is very large, i.e., $O(DE)$, in the worst case it can be large as $O(DV^2)$. It is widely believed that for arbitrarily general networks, $\Omega(D)$ is the time complexity lower bound and $\Omega(E+N\log N)$ is the lower bound on communication complexity.

While it remains theoretically interesting to design an algorithm to achieve both time and communication complexities, there are two important practical

0-8186-2755-7/92 $03.00 © 1992 IEEE

issues in leader election that are largely neglected: re-election and network topologies. Except for the first leader, all the subsequent leaders will be re-elected. Thus, an algorithm tailored to re-electing a leader may be able to take advantage of the current situation in the system and perform better than those algorithms designed to elect a leader from the scratch. For the second issue, it can be observed that the performance of an election algorithm may depend on the network topology to which the algorithm is applied. For instance, Attiya etc [3] designed an election algorithm on a chordal ring with $O(N)$ communication complexity.

In this research, we consider networks in which the degree of each node is bounded by an integer constant. This so-called "bounded-degree" network model consists of a wide spectrum of networks. Examples of this network model include *k-regular* networks, chordal rings with k chordal. It has been shown [9] that *k-regular* graphs is, in some sense, optimally reliable. That is, it can survive large degree of link and node failures and still remain connected. It is important to note that each node in a network has only a limited number of communication channels and finite buffer space. Thus the degree of each node can not be arbitrarily large. Therefore, the star-like network is impractical to constructed in a large-scale network. This explains why the majority of existing large-scale networks fit into the bounded-degree network model (e.g., ARPAnet and CSnet).

We present a distributed algorithm for electing leader in networks with the *bounded-degree* property. The algorithm is invoked when the current leader fails to continue its responsibilities. It is based on a strategy that is sharply different from those of the previous works. Most of the algorithms applicable to this problem work in a bottom-up manner adopted in GHS algorithm. In contrast, our algorithm works in a top-down fashion. The algorithm is based on flooding [4][7][8][13][15] and is very efficient. In the worst case, its time complexity is $O(D)$, an optimal time complexity. The communication complexity is bounded by $O(E+k \cdot N)$ where k is the number of nodes initiating the election process. In distributed computing systems where the leader election is triggered by the failure or the abdication of the current leader or by other conditions making it a necessity to elect a new leader, the number of nodes observing the conditions and thus initiating the election process is bounded by the number of neighboring nodes. From this, we show that our algorithm can achieve an optimal worst-case communication complexity of $O(E)$ which is equivalent to $O(N)$ in bounded-degree networks.

It is important to note that in all the previous algorithms, every initiator is expected to "wake up" every node to compete for leadership. By doing this, the communication complexities of these algorithms remain bounded by $O(E+N\log N)$ (see [11]), regardless of how many initiators coexist in the network. Thus in bounded-degree networks, our algorithm is superior to all the previous algorithms in terms of communication complexity.

2. The Model and Assumptions

It is assumed that the leader election proceeds under an environment in which the following conditions hold: (i) The network is connected and contains N nodes and E edges; each node has a unique id and has a finite and bounded number of ports which are used to establish communication links. (ii) Due to network reconfigurations, the topological information available to each node is strictly local; each node is only aware of its neighbors. (iii) The time for a message to be sent correctly over a communication link is finite; the time required for node processing is negligible; so is the message delay due to the traffic in a node. (iv) The network functions properly as a connected component during the execution of the algorithm. The failure of the current leader does not result in a partition of the network; as leader election is required whenever the current leader is lost, fails to function properly, or wishes to abdicate its position as a leader, only the neighbors of the current leader are able to detect this situation and initiate the election; this implies that the number of nodes spontaneously initiating the election algorithm is bounded. (v) There are no global memory and global time, and nodes exchange information by strictly using message passing.

3. The Approach

3.1 Description of The Algorithm

In this section, we present our leader election algorithm. The algorithm is divided into two stages. In the first stage, the process of selecting a leader can be divided into two operational phases: the broadcast phase and the reply phase. The broadcast phase begins when one or more nodes spontaneously initiate the leader election process when they detect the failure of the current leader. Each of these initiators starts by flooding the network with its timestamped "Campaign-For-Leader" (CFL) message. The timestamp of each message can be generated by reading the clock of the node and attached with its node id so that a tie can be avoided. As it is assumed that no global time is available, an initiator with lower timestamp does not necessarily imply that it starts earlier. However, practically it is likely that these clocks can be synchronized to some extend.

As messages are flooding, a spanning forest with each tree rooted at an initiator is built up. The trees are established in a manner similar to those in [8][13][15]. Each node, except for those initiators, chooses as its parent node the node from which the first CFL message arrives. Each node acknowledges its parent with a message notifying this relationship. The CFL messages are flooded across the network as far as possible. Eventually, a flood comes to an end, i.e., it reaches a node which either has no neighbors to which this message can be forwarded or has no neighbor which has not received the message. This node, called a leaf node, will then initiate the reply phase.

In the reply phase, a leaf node replies to its parent with a *voting message*. A voting message is merely a short acknowledgement to a CFL. A parent node will send its voting message to its parent after all its children's voting messages have been collected. Once a node finishes its reply, its part is done; it will receive no further CFL messages. Since there are usually several initiators during this stage, two or more message floods may collide at a node. In this case, the receiving node will choose the sender node whose CFL has the lowest timestamp as its parent. If there is a tie, it is broken by the order of the initiators' ids (e.g., by the lexical order of the initiators' ids). If several identical CFL messages arrive that share the same lowest timestamp, then the first node from which one of these messages was received is chosen as the parent. After determining the lowest-timestamped message and choosing its parent, the node proceeds to flood its neighbors with the message. Note that a node only acknowledges a neighbor as its parent as long as no lower timestamped message arrives. Once such a message arrives, the current parent node will be replaced.

In some sense, we may consider the flood of the CFL message with lowest timestamp to "win the fight" and to continue spreading. The winning flood will proceed in two ways: 1) it takes over the ground of the losing floods, eventually taking over their initiators. 2) it tracks down the unreplied children (i.e. the children who have not yet replied with voting messages). Fortunately, this doesn't have to be done by flooding in both directions because the tree has already been set up by the losing flood. When a CFL message arrives at a node and the timestamp value of this CFL message is lower than that of the message the node receives from its parent, the parent of this node is replaced by the node from which this message comes and becomes a child of its former child. Then this CFL message is sent to all of its unreplied children, including its "new" child (its former parent). A node may switch its parent several times before it receives all the replies from its children and goes into its reply phase. It is important to note that CFL messages are sent only to those nodes who have not yet replied. This enables us to reduce the number of messages exchanged and thereby shorten the election process. Notice again that once a message with a lower timestamp arrives at a losing initiator, this initiator is conquered and now has a parent. The conquered initiator will behave like an ordinary node and will reply to its parent with its voting message whenever it receives the replys from all of its children.

The first stage ends when all the nodes, except for the leader, are in the reply phase. That is, only the leader can get all the voting messages from its children. By then the network is spanned by an ultimate tree which has the leader as its root. Each tree edge implies a child-parent relationship. Once the leader is elected, the result is distributed by the leader. This is easily achieved by disseminating the information along the tree from parent to child.

It is worth mentioning that the use of timestamps is the key factor in the performance improvement of this algorithm over the previous results. The conflict between CFLs from different initiators can be effectively resolved without involving the calculations of the sizes of the trees for which the initiators are the roots. When merging partial spanning trees, it is undesirable to permit a large tree to be absorbed by a smaller tree because this will lead to unnecessary message exchanges. Instead of using sophisticated mechanisms to keep track of the sizes of the trees dynamically, timestamps can be used to decide which tree is cheaper to subsume whenever two or more CFL messages of different initiators meet.

In summary, we can see that the messages with the lowest timestamp will spread across the network without being blocked. While all of the other initiators are building trees, they contribute to the whole process by helping to reduce the number of nodes that the lowest-timestamped messages have to visit and thus speed up the election as a whole.

The Leader Election Algorithm

1. *Notations*

a. PR_i : the parent node of i.

b. SB_i : the set of sibling nodes of i.

c. CR_i : the set of child nodes of i.

d. $NEIB_i$: the set of neighboring nodes of i.

e. UNK_i : neighbors of node i and whose relationships to i are still unknown (e.g. they could be either siblings or children of i); UNK_i is initialized to be $NEIB_i$.

f. CFL : a Campaign-For-Leader message from an initiator, CFL.time represents the timestamp contained in this message.

g. VOTE : a voting message to be sent to a parent to acknowledge its acceptance of a candidate.

h. ACK-parent(i): the parent-notifying message from node i to its parent.

i. ACK-sibling(i): the sibling-notifying message from node i to its sibling.

j. \ : difference, a set operation.

2. *Node Algorithm* for *node i*:

```
For a message CFL from j:
  a.1)   if $i$ is not "VISITED",
  a.2)   begin
             /* $j$ is the first node sending a CFL to $i$ */
  a.3)       Mark $i$ VISITED;
  a.4)       Current-CFL $\leftarrow$ CFL;
  a.5)       send ACK-parent($i$) to $j$;
  a.6)       $PR_i \leftarrow j$;
  a.7)       $UNK_i \leftarrow UNK_i \setminus j$;
  a.8)       if $NEIB_i \setminus j = \varnothing$,
                /* $i$ is a leaf node */
  a.9)          send VOTE to $j$;
  a.10)      else for every $k \in NEIB_i \setminus \{j\}$,
                      send CFL to $k$;
  a.11)  end
  a.12)  else begin /* $i$ has been visited before */
  a.13)      if CFL.time < Current-CFL.time,
  a.14)      then begin
  a.15)         Current-CFL $\leftarrow$ CFL;
  a.16)         send ACK-parent($i$) to $j$;
  a.17)         $CR_i \leftarrow CR_i \cup \{PR_i\}$;
         /* the current parent becomes a new child of $i$ */
  a.18)         for every $k \in CR_i \setminus REPLIED_i$,
  a.19)             send CFL to $k$;
  a.20)         end
  a.21)      else if CFL.time = Current-CFL.time
  a.22)         begin
  a.23)           send ACK-sibling($i$) to $j$;
  a.24)           $SB_i \leftarrow SB_i \cup \{j\}$;
  a.25)           $UNK_i \leftarrow UNK_i \setminus \{j\}$;
  a.26)         end
  a.27)      else if CFL.time > Current-CFL.time
  a.28)           begin
  a.29)           send Current-CFL to $j$;
  a.30)           $CR_i \leftarrow CR_i \cup \{j\}$;
  a.31)           end
  a.32)  end
For an ACK-parent($j$):
  b.1)   $CR_i \leftarrow CR_i \cup \{j\}$;
  b.2)   $UNK_i \leftarrow UNK_i \setminus \{j\}$;
For an ACK-sibling($j$):
  c.1)   $SB_i \leftarrow SB_i \cup \{j\}$;
  c.2)   $UNK_i \leftarrow UNK_i \setminus \{j\}$;
For a VOTE from $j$:
  d.1)   mark $j$ REPLIED;
  d.2)   $REPLIED_i \leftarrow REPLIED_i \cup \{j\}$;
  d.3)   if every $k$ in $CR_i$ is REPLIED,
  d.4)   begin
  d.5)       if $i$ has no parent
             /* $PR_i = \varnothing$, $i$ is an initiator */
  d.6)           stop; $i$ is the leader;
             /* $i$ has lowest timestamp */
  d.7)       else send VOTE to $PR_i$;
  d.8)   end
```

4. Correctness and Computational Complexity

Theorem 1. *The algorithm is correct for electing a unique leader.*

Proof. As the network remains connected, every non-initiating node has a parent. Every initiator, except for the leader, will also have a parent because it will eventually be invaded and conquered by messages with lower timestamps. Hence, once a tree is built that has the leader as its root, it follows immediately that a unique leader can be elected. □

Lemma 1. *Let v be the node that two CFL messages from different initiators i_1 and i_2 encounter. The paths via which CFL messages from i_1 to v and from i_2 to v arrived are both minimal.*

Proof. For the CFL from i_1 to v, let $P = \langle i_1, v_1, v_2, \ldots, v_{i-1}, v_i, v\rangle$ be the path through which CFL arrives at v using the flooding protocol, where for all $1 \le j \le i$ and v_j is the parent of v_{j+1} and i_1 is the parent node of v_1. We show, by contradiction, that the delay for v to receive CFL from i_1 is minimized if the CFL travels along P. Suppose that $P' = \langle i_1, v'_1, \ldots, v'_j, v\rangle$ is a shorter path than P to deliver CFL from i_1 to v. Let (v'_k, v'_{k+1}) be the first pair of nodes in P' path such that v'_k is not the parent node of v'_{k+1}, and that there is another node w in this network that is the parent node of v'_{k+1}. Since the parent node is chosen as the first node from which the CFL arrives, there must be a shorter path from v via w to v'_{k+1}, and then from v'_{k+1} to t, which implies that P' is not minimum, a contradiction. Hence, P is a shortest path. Similarly, we can show that the CFL from i_2 arrives at v via a shortest path. □

Theorem 2. *The time complexity of the algorithm is $O(D)$.*

Proof. Let k be the number of initiators concurrently existing in the network and let i_w be the leading initiator with the lowest timestamp in its CFL. We start counting time when i_w begins its broadcast of CFLs.

The following two cases are considered:

Case 1: v_w is the initiator that starts earliest. In the first stage, each initiator uses flooding to broadcast CFL messages. When two CFLs from different initiators, i_1 and i_2 respectively, meet at a node v, from Lemma 1 we know that the distance between v and i_1 is minimal, so is the distance between v and i_2. Suppose the CFL from i_1 is the winner. This CFL will move forward to i_2 by following the path established by the CFL from i_2. Let the length of the path for CFL from i_1 to reach i_2 be $D_{1,2}$; we have $D_{1,2} \leq 2D$. Therefore, we can see that, in the worst case, the CFL messages from the leading initiator i_w will visit each initiator by the longest path of length less than $2kD$. As k is bounded, the path length is in the order of $O(D)$. Since the CFL messages from i_w have the lowest-timestamp and will never be terminated or blocked by other CFL messages, it takes at most $O(D)$ time units for the last node to receive a CFL message. Hence, the total time needed for the first stage is $O(D)$.

Case 2: v_w is the initiator that starts latest. Since it takes $O(D)$ for a CFL message arrives from any other initiator to v_w, the worst case happens when v_w starts immediately before a CFL originated at i_1 with higher timestamp arrives at i_w such that i_1 has second lowest timestamp. This will add at most another $O(D)$ time units to the total time units. As there are at most k initiators, the total time units can not exceed kD. Again, this yields $O(D)$ time units for the first stage.

From the above two cases, it can be seen that the time complexity in the first stage is $O(D)$ regardless when v_w initiates. As announcing the elected leader in the second stage takes no more than $O(D)$ time units, time complexity for the algorithm is $O(D)$. □

Theorem 3. *The communication complexity of this algorithm is* $O(E)$.

Proof. In the broadcast phase of the first stage, each edge is traveled at most twice by the first CFL message as a result of flooding. However, each node may be visited by messages from different initiators. Note that, except for the first visit, subsequent CFL messages are delivered only to those children who have not yet replied to their parent. If there are k initiators, each node can be visited by at most k distinct CFL messages. Following the paths of the tree, each CFL message, in its worst case, can travel across less than $(N-1)$ edges for which this message is not the first CFL message. As there are k initiators, the total number of CFL messages exchanged over the edges can not exceed $k \cdot (N-1)$.

In the reply phase, each node will send a voting message to its parent, which implies that $(N-1)$ voting messages are sent. By adding another $N-1$ messages for broadcasting the identity of the new leader, the total number of messages is no greater than $2E+k \cdot (N-1)+2(N-1)$, which yields a communication complexity of $O(E+N) = O(E)$ as k is bounded by an integer constant and N is dominated by E. □

Corollary. *The communication complexity of this algorithm is* $O(N)$ in bounded-degree networks.

5. Conclusion

We present a new distributed algorithm for leader election in bounded-degree networks. The practical significance of the algorithm is that it takes advantage of the the limited number of initiators which might detect the failure of a leader. The new approach avoids waking up all the nodes to compete for the leadership. In this way, the algorithm can achieve optimal time complexity as well as communication complexity.

References

1. A. Aho, J. Hopcroft and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.
2. M. Ahuja and Y. Zhu, "A distributed algorithm for minimum weight spanning trees based on echo algorithms", *Proc. of the 9th International Conf. on Distributed Computing systems*, pp. 2-8, June 1989.
3. H. Attiya, J. van Leeuwen, N. Santoro, and S. Zaks, "Efficient Elections in Chordal Ring Networks", *Algorithmica* 4: pp. 437-446, 1989.
4. B. Awerbuch and S. Even, "Reliable broadcast protocols in unreliable networks", *Networks*, Vol. 16, pp. 381-396, 1986.
5. B. Awerbuch, "Optimal distributed algorithms for minimum weight spanning tree, counting, leader election and related problem", *Proc. of the 19th ACM Symp. on Theory of Computing*, pp. 230-240, May 1987.
6. F. Chin and H. F. Ting, "An almost linear time and $O(n \log n + e)$ Messages Distributed algorithm for minimum weight spanning trees", *Proc. of* 1985 *FOCS Conference*, pp. 257-266, October 1985.
7. Y. Dalay. "Broadcast protocols in packet switched computer networks". Tech. Rep. 128, Dep. of Electrical Engineering, Stanford University. April 1977.
8. Y. Dalal and R. Metcalfe, "Reverse path forwarding of broadcast packets," *Commun. Ass. Comput. Mach.*, Vol. 21, pp. 1040-1048, Dec. 1978.
9. T. Evans and D. Smith, "Optimally reliable graphs for both edge and vertex failures", *Networks*, Vol. 16, pp. 199-204, 1986.
10. E. Gafni, "Improvements in time complexity of two message-optimal algorithms", *Proc. of* 1985

PODC Conference, pp. 175-185, August 1985.

11. R. Gallager, P. Humblet, and P. Spira. "A distributed algorithm for minimum-weight spanning trees", *ACM Trans. on Programming Languages* and *Systems*, Vol. 5, No. 1, January, 1983, pp. 66-77.
12. Y. N. Lien, "A New node-join-tree distributed algorithm for minimum-weight spanning trees", *Proc. of the 8th International Conf. on Distributed Computing systems*, pp. 334-340, June 1988.
13. K. Luo, Y.C. Chow and R. Newman-Wolfe, "An efficient Broadcast Protocols in Networks with Changing Topologies", *2nd IEEE Workshop on Future Trend of Distributed Computing Systems.* pp. 88-93, 1990.
14. D. Peleg, "Time-Optimal Leader Election in General Networks", *Journal of Parallel* and *Distributed Computing* **8**, pp. 96-99, 1990.
15. A. Segall and B. Awerbuch, "A reliable broadcast protocol," *IEEE Trans. on Commun.*, Vol. COM-31, pp. 895-901, 1983.
16. A. Tanenbaum. *Computer Networks*. Addison-Wesley, Reading, 1980.