September 1984 LIDS-P-1407

SELECTING A LEADER IN A CLIQUE IN O(N log N) MESSAGES*

by

Pierre A. Humblet**

ABSTRACT

This paper presents an extremely simple algorithm for all processors in a completely connected network to agree on a unique leader. It requires O(N log K) messages, where N is the number of processors, and K is the number of processors that independently start the algorithm.

---

*See page 8.

**Room No. 35-203, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts 02139.

SELECTING A LEADER IN A CLIQUE IN O(N log N) MESSAGES

Pierre A. HUMBLET

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

Paper to be presented at the 23rd Conference on Decision and Control
LAS VEGAS HILTON, Las Vegas, Nevada
December 12-14, 1984

1) INTRODUCTION

Consider a network of N processors (nodes) with communication lines (edges) between any two of them, so that the underlying graph is complete. The nodes have distinct identities that are ordered and they locally number their adjacent edges from 1 to N-1, but they initially ignore the identities of the edge destinations. The nodes are initially asleep. An arbitrary non empty subset wake up spontaneously and start executing a common asynchronous distributed algorithm that consists of sending and receiving messages over edges and processing them. Messages arrive without error, after an arbitrary but finite delay.

We are interested in designing an algorithm for all nodes to agree on a common leader, using as few messages as possible. This is equivalent (up to O(N) messages) to the problems of finding the node with largest identity, or of making sure that all nodes awake, or of finding a spanning tree. It is clear that O(N**2) messages are enough. However [1] have shown that O(NlogN) messages are both necessary and sufficient in the worst case.

Similar problems arise in networks that are not complete. For dense networks a high cost is paid by the unability to guarantee that all nodes will be reached without exploring most edges. To that effect O(E) messages are required in the worst case, where E is the number of edges. [2] gives a simple algorithm that requires O(NlogN) messages to find the leader; this is an average over a class of random graphs and it assumes that N is initially known. [3] offers a method with at most O(NlogN) + O(E) messages to find the leader. The same cost is incurred in [4] to solve the more complicated problem of finding a minimum spanning tree.

The special case of a ring network has also been studied. Worst case communication costs of O(NlogN) can be achieved readily [4],[5], [6],[7] and are also necessary [8].

This paper offers an extremely simple algorithm to find a leader in a complete network using O(NlogN) messages, each containing at

most logN + i bits, where i is the length of the representation of a node identity. The algorithm clearly illustrates the essential feature necessary to get a small number of messages, namely to give priority to nodes that have already done much work. The next section presents the algorithm. Its communication cost is analyzed in the third section.

2) DESCRIPTION OF THE ALGORITHM

The algorithm works by having nodes attempt to capture other nodes, enlarging their domains. The node which capture all other nodes becomes the leader. A captured node keeps its domain, without trying to augment it. A node can be part of many domains but remembers which of its edges leads to its "master", i.e. the last node by which it has been captured.

When node A sends a TEST message to node B to attempt to capture it, B forwards the message to its master C which may possibly be B itself. If the size of A's domain is larger than that of C (or they are equal but the identity of A is larger) then C stops its capture process (WITHOUT becoming part of A's domain) and sends the message WINNER(A) to B. B then becomes part of A's domain and forwards WINNER(A) to A. A continues the capture process. To insure that C does not receive messages from B after having lost the fight, B is restricted to forward only one message at a time to its master. Other messages that may arrive while the

issue of a fight is uncertain are queued at B. They are forwarded to B's master when the result of the previous fight is known.

If C wins it sends WINNER(C) to B. A does not receive any reply and so is inhibited from increasing its domain.

We now proceed with a more formal description of the algorithm. Each node maintains four variables, one array and two message sets:

STATE: its state, with values "active" or "stopped".

SIZE: the number of nodes it has captured

MASTER: the identity of its current master

PENDING: the number of messages to forward to the master

EDGE_TO(id): the number of the edge leading to id (if known).

INPUT_SET: set in which arriving messages are placed

PENDING_SET: set of messages waiting to be forwarded to the master

Initially all sets are empty, the STATEs are "active", the SIZEs are 0, the MASTERs are set to ID (the identity of the local node) the PENDINGs are 0 and the EDGE_TO()s are undefined except that EDGE_TO(ID) is set to -1. Edge -1 is an artificial "self loop" that we introduce to simplify the description of the algorithm.

The algorithm starts when a high level protocol awakes one or many nodes by placing the message WINNER(ID) in their INPUT_SETs.

A node waits until a message is placed in its INPUT_SET. It then

processes the message completely and either it becomes the leader or it waits for more messages.

Node ID receiving the message WINNER(id) does as follows:

```
If ID = id then  \* start or response to a TEST that originated here *\
     { SIZE = SIZE + 1;
       If SIZE = N then STOP;                    \* node ID is the leader *\
       If STATE = "active " then \* no fight has been lost, continue *\
              send TEST(SIZE,ID) on edge SIZE }
else { If MASTER <> id then
            { MASTER = id;              \* record and notify new master *\
              send WINNER(id) on EDGE_TO(id) }
       PENDING = PENDING - 1;          \* forward waiting TEST messages *\
       if (PENDING > 0) then
              send a message from PENDING_SET on EDGE_TO(MASTER) }
```

Node ID receiving the message TEST(size,id) on edge e does as follows:

```
If e < SIZE  then                    \* message comes from node in domain *\
     { If (size,id) > (SIZE,ID) then      \* lexicographical ordering *\
            { STATE = "stopped";
              send WINNER(id) on edge e }
       else   send WINNER(ID) on edge e }
else            \* message comes from outside of domain, tell master *\
     { EDGE_TO(id) = e;
       PENDING = PENDING + 1;
```

```
        If (PENDING = 1) then send TEST(size,id) on EDGE_TO(MASTER)
                         else put TEST(size,id) in PENDING_SET )
```

3) CORRECTNESS AND COMPLEXITY ANALYSIS

The algorithm must terminate, in the sense that all nodes either stop or wait for a message but no message is in transit, because no node can generate more than N - 1 TEST messages, each TEST message can cause at most three other messages to be sent, and message propagation times are finite.

Define the "domain" of node ID at time t as the set of nodes from which it has received WINNER(ID), including ifself if it has been awaken by the higher level protocol. Note that the cardinality of a domain does not decrease with time and that as time increases each node belongs to more and more domains. After a node becomes part of a new domain the size of the old one cannot increase by more than one (as the master has been defeated and will not issue new TEST messages but may still receive an answer to an outstanding TEST) while the size of the new one becomes at least as large as what the size of the old one will ever be.

The following fact is critical to the analysis of the algorithm: If at times T1, T2, ...Tk respectively domains D1, D2 ...Dk have the same size s then they are almost disjoint, in the sense that

at most k - 1 nodes belong to two of them, and no node belongs to more than two of them.

Here is the outline of a proof: If a domain Di has won a node that already belonged to another domain Dj the size of Dj was not more than that of Di and the size of Di has increased to at least what the size of Dj will ever be, i.e. at least s. We conclude that if a node is common to two domains of size s, a battle must have taken place when both domains had size s - 1 and thus its issue was decided on the basis of node identities. The domain Di with smallest identity cannot win such a battle and thus at most k - 1 of the Di's may each absorb at most one node from another one.

Rank the nodes in order of decreasing order of SIZE at termination, breaking ties arbitrarily and denote by Sk the final SIZE of the kth ranked node. The node with SIZE S1 and largest ID cannot have lost a fight thus must have become the leader. Sk is not greater than (N + k - 1)/k because at most k - 1 nodes belonged to two of the domains of the nodes ranked from 1 to k at the times they reached size Sk, thus there cannot be two leaders.

A total of at most 4 messages can be transmitted for each TEST message generated, and a node generates no more TESTs than its final SIZE. The final SIZE of the nodes that were never awaken by the high level protocols is zero. For the other nodes we can use the bound on Sk derived above and we conclude that the total

number of messages is no more than 4 * ((N - 1) * (1 + 1/2 + 1/3 + ... + 1/K) + K) = O(N log K), where K is the number of awaken nodes. This number can be tightened by noticing that the first TEST message received by a node generates at most one other message on a true edge and also that the algorithm could stop as soon as SIZE is greater than (N + 1)/ 2.

4) ACKNOWLEDGEMENTS AND HISTORICAL NOTES

This research was conducted at the M.I.T. Laboratory for Information and Decision Systems with partial support provided by NSF under Grant NSF-ECS-8310698 and by DARPA under Contract ONR/N00014-84-K-0357.

Paris Kanellakis introduced us to the leader problem in 1978. This algorithm was developed in the fall of that year and was immediately extended to general graphs in [3]. However our interest in those algorithms quickly vanished as the much more powerful and elegant [4] was discovered. Shmuel Zaks, visiting MIT in the fall of 1983, introduced us to [1] and revived our interest in algorithms for complete graphs.

REFERENCES

[1] E. Korach, S.Moran and S. Zaks, "Tight Lower and Upper Bounds for Some Distributed Algorithms for a Complete Network of

Processors", Technical Report 124, IBM Scientific Center, Haifa, Israel, November 1983.

[2] R.G. Gallager, "Choosing A Leader in a Network"', Internal Memorandum, Laboratory for Information and Decision Systems, MIT, undated.

[3] R.G. Gallager, "Finding a Leader in a Network with O(E) + O(N log N) Messages", Internal Memorandum, Laboratory for Information and Decision Systems, MIT, undated.

[4] R.G. Gallager, P.A. Humblet and P.M. Spira, "A Distributed Algorithm for Minimum Spanning Tree", ACM Transactions on Programming Languages and Systems, Vol 5, No 1, January 1983, pp 66-77.

[5] D.S. Hirschberg and J.B. Sinclair, "Decentralized Extrema-finding in Circular Configurations of Processes", CACM, Vol 23, No. 11, 1980, pp.627-628.

[6] D. Dolev, M. Klawe and M. Rodeh, "An O(NlogN) Unidirectional Distributed Algorithm for Extrema Finding in a Circle", J. of Algorithms, Vol 3, 1982, pp 245-260.

[7] G.L. Peterson, "An O(NlogN) Unidirectional Algorithm for the Circular Extrema Problem", ACM Transactions on Programming Languages and Systems, Vol 4, 1982, pp 758-762.

[8] J.E. Burns, "A Formal Model for Message Passing Systems", TR-91, Indiana University, September 1980.