Information Processing Letters 25 (1987) 103–109
North-Holland

6 May 1987

A TIME-OPTIMAL MESSAGE-EFFICIENT DISTRIBUTED ALGORITHM FOR DEPTH-FIRST-SEARCH *

K.B. LAKSHMANAN ** and N. MEENAKSHI

Department of Computer Science and Engineering, Indian Institute of Technology, Madras 600 036, India

K. THULASIRAMAN

Department of Electrical Engineering, Concordia University, Montréal, Canada H3G 1M8

Communicated by E.C.R. Hehner
Received 17 July 1986
Revised 7 October 1986

In this paper we study the problem of distributed construction of a depth-first-search tree for an asynchronous communication network. First, we point out that any algorithm requires at least $2n-2$ time units and $2m$ messages in the worst case, where n and m are the number of nodes and the number of edges in the network, respectively. We then provide a modification to a recent algorithm due to Awerbuch (1985), and show that the new algorithm is time-optimal, while requiring less then $4m-(n-1)$ messages.

Keywords: Distributed system, asynchronous network, communication graph, depth-first-search, message and time complexities

1. The model

Consider a distributed computing system consisting of a number of autonomous processors interconnected through communication links. The processors do not share a common memory, have only local information and hence communicate frequently to coordinate any computation to be accomplished. The interconnection network can be modeled by an undirected communication graph $G = (V, E)$ where nodes correspond to the processors and the edges to bidirectional communication links. The processors have distinct identities, but each processor knows only the identities of its neighbors. Each processor performs a variety of local tasks, besides receiving messages from its neighbors, performing some computation and sending messages to its neighbors. The exchange of messages between two neighboring processors is asynchronous in that the sender always hands over the message to the communication subsystem and proceeds with its own local task. The communication subsystem, we assume, will deliver the message at its destination, without loss or any alteration, after a finite but undetermined time lapse. The messages sent over any link also follow a first-in-first-out rule. The messages received at any processor are stamped with the identity of the sender and transferred to a common queue before being processed one by one. Messages arriving at a node simultaneously from several neighbors may be placed in any arbitrary order in the queue. Since several computations may be in progress concurrently, we assume that the network has suitable mechanisms so that, at the receiving end, messages corresponding

* This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant A0890 at McGill University, Montréal, Canada, and Grant A4680 at Concordia University, Montréal, Canada.

** Presently visiting the Department of Electrical Engineering, McGill University, Montréal, Canada H3A 2A7.

to any particular computation initiated by a particular node can be distinguished and separated out.

A distributed algorithm consists of the collection of similar node algorithms residing at the processors. These node algorithms specify the actions to be taken in response to the messages that may arrive at a node. It is assumed that the actions necessary for processing a message can all be performed in negligible *computation* time, uninterrupted by the arrival of other messages. Hence, the complexity measures used to evaluate the performance of distributed algorithms only relate to the *communication* aspect. The message complexity is the total number of messages transmitted during the execution of the algorithm. The time complexity is the time that elapses from the beginning until the termination of the algorithm, assuming that the delay in any link is exactly one unit of time. It must be recognized that this assumption of unit time delay in communication links is made only for the purpose of timing analysis and the algorithm is expected to operate correctly under the previous assumption that the delay is finite, but cannot be bounded. Also, given a communication graph with n nodes and m edges, the actual performance of any distributed algorithm, in terms of its message and time complexities, will depend upon the structure of the graph, the degree and other characteristics of the node initiating the algorithm, the delays encountered in the links, etc., and hence we use only the worst-case analysis in comparing two algorithms, as well as in discussing the optimality of any algorithm.

2. The problem and lower bounds on complexities

Given an asynchronous communication network as described above and a starting node s, we want to construct a depth-first-search (DFS) tree, rooted at s, for the communication graph $G = (V, E)$. All messages are required to be of fixed length independent of the size of the graph. At the end of the computation, the DFS tree will be available in a distributed fashion, each node except the root knowing its father in the tree. We assume that the algorithm is initiated by node s and that no processor or link failure takes place during its entire execution.

It is well known that a DFS algorithm partitions the edges of an undirected graph into tree edges and back edges. This requires an exploration of the graph with the center of activity moving from one node to another in a systematic way [4,6]. Initially, the start node s is the center of activity. When a node becomes the center of activity for the first time, it marks itself as *visited*. Also, whenever a node becomes a center of activity, it tries to identify a neighbor who is not visited yet and transfers the center of activity to that node. But if no such neighbor exists, i.e., if the node is completely scanned, then it shifts the center of activity to the father node in the tree, or simply terminates of the node happens to be the start node itself.

If there are n nodes in the graph, then it is clear that there must be $2n - 2$ shifts of the center of activity. It is also easily seen that graphs for which the DFS tree constructed has a linear chain of n nodes are obvious cases requiring all shifts of the center of activity to proceed sequentially, no matter what algorithm is employed. Since each shift of the center of activity has to be accomplished by a passage of a message, any distributed algorithm for DFS should have a worst-case time complexity of at least $2n - 2$.

We have also seen that a node can shift the center of activity to the father node or terminate the algorithm only if it has ensured that each one of the neighbors has been visited. If all the messages are required to be of fixed length, and cannot contain the number of nodes in the graph, the node identities, etc., then each node requires at least one message to arrive from each of its neighbors before it recognizes that it has been completely scanned. Thus, any distributed algorithm for DFS should have a message complexity of at least 2m, where m is the number of edges in the graph.

3. Existing solutions

Cheung [3] has presented an algorithm for finding a DFS tree whose message complexity is 2m.

Thus, his algorithm is message-optimal. However, in his algorithm, the messages are all transmitted one after another in sequence and hence the time complexity is also 2m. Therefore, Awerbuch [1] recently considered the problem of constructing a DFS tree in O(n) time. His algorithm requires 4m messages and achieves a time complexity of $4n - 2 - 2n_1$, where n_1 is the number of nodes of degree one in the graph. The performances of both Cheung's and Awerbuch's algorithms remain the same uniformly for all cases—best to worst.

Awerbuch's algorithm requires four kinds of messages—DISCOVER, RETURN, VISITED, and ACK. DISCOVER messages are used to shift the center of activity from a visited node to an univisited one. RETURN messages are used to shift the center of activity from a node to its father in the tree. VISITED messages are used by a node to inform all its neighbors, except the father, that it has been visited. ACK messages are sent in response to VISITED messages. In fact, when a node becomes the center of activity, VISITED messages are sent out to all neighbors, except the father, and only after ACK messages have been received from these neighbors, the center of activity is shifted to another node. The basic idea is that by the time the center of activity is shifted to any node, every node knows exactly which of its neighbors have been visited. This ensures that DISCOVER messages are never sent to an already visited node. Thus, the algorithm requires $n - 1$ DISCOVER, $n - 1$ RETURN, $2m - (n - 1)$ VISITED, and $2m - (n - 1)$ ACK messages, all adding up to 4m. More importantly, the VISITED and ACK messages add two units of time at each node of degree greater than one to the time complexity. The DISCOVER and RETURN messages need $2n - 2$ time units. As a result, Awerbuch's algorithm has a time complexity of $4n - 2 - 2n_1$, where n_1 is the number of nodes of degree one.

In [2], Chang has proposed a graph traversal scheme called pure traversal which can be used to construct an arbitrary rooted spanning tree of the communication graph. The worst-case message and time complexities of his algorithm are $4m - 2n + 2$ and 2d, respectively, where d is the distance of the farthest node from the starting one in the graph. Chang's algorithm requires two types of messages —EXPLORER and ECHO. Segall [5] has also studied this problem independently and has constructed an algorithm whose worst-case message and time complexities are 2m and 2d, respectively. Interestingly, Segall's algorithm uses only one type of message and does not require ECHO messages as in [2]. It is this work that provided the motivation for us to see if the performance of Awerbuch's algorithm can be improved.

4. Our modification

The time complexity of Awerbuch's algorithm can be improved and made optimal by simply eliminating the ACK messages and ensuring that VISITED messages are always transmitted in communication-time parallel to DISCOVER or RETURN messages. But this creates a new problem—more than one DISCOVER message could get sent to a node.

In our modified algorithm, a node marks itself as visited when it receives a DISCOVER message for the first time. It also tries to identify a neighbor which has not been visited yet and to send a DISCOVER message to it, at the same time informing, through VISITED messages, its status to all other neighbors except, of course, the father. Since VISITED messages could suffer long delays in the communication links, a node trying to identify an unvisited neighbor may not have the correct and complete information regarding the status of all of its neighbors. It may, therefore, choose an already visited neighbor and send a DISCOVER message to it, simply bacause it has not received any message from that neighbor at that stage. But, luckily, it is possible to recover from such a mistake because a DISCOVER or VISITED message sent by that neighbor will eventually arrive at its intended destination. In order to enable this kind of recovery, the sender of a DISCOVER message always records the identity of the neighbor to whom such a message is sent, so that if a DISCOVER or VISITED message is received from that neighbor, an alternative neighbor, if any, can be found to shift the center of activity. This strategy also means that a DISCOVER message received at an already visited node can simply be ignored, except for the purpose of

recognizing that the sender has also been visited already. The detailed algorithm is presented in Appendix A.

Observe that the depth-first-search tree built as a result of execution of the algorithm is really not dependent on the pattern of delays encountered in the communication links. It is only dependent on the order in which the neighbors are selected by an already visited node to send DISCOVER messages. On the other hand, the number of messages exchanged during the execution of the algorithm is dependent on the delays encountered in the communication links. The best-case situation arises when the communication delay in any link is one time unit, the same assumption to be made for the time complexity analysis. Recall that VISITED messages are always sent in communication-time parallel to DISCOVER or RETURN messages. The above assumption implies that by the time a node becomes a center of activity, all VISITED messages sent by its neighbors should have been received and processed. Thus, there will never be a mistake made of sending a DISCOVER message to an already visited node. In other words, exactly $n-1$ DISCOVER messages will be sent in the best case, consuming exactly $n-1$ units of time. Also, exactly $n-1$ RETURN messages will be sent, consuming an additional $n-1$ units of time. Hence, the time complexity of our algorithm is $2n-2$, which is optimal.

Now, in order to determine the number of messages exchanged in the best case we still have to account for VISITED messages. Every node i of degree d_i clearly sends out at least $d_i - 2$ VISITED messages. But, the start node s and those nodes which finally appear as leaf nodes in the DFS tree will send out one more VISITED message each. Thus, the total number of VISITED messages exchanged in the best case is $2m - 2n + \ell + 1$, where ℓ is the number of leaf nodes in the DFS tree constructed. Including the DISCOVER and RETURN messages, the total number of messages exchanged in the best case is $2m + \ell - 1$. Clearly, here $\ell \geq 1$. If the graph G is complete, then $\ell = 1$ so that in such a case the number of messages exchanged during the execution of our algorithm could be as low as 2m, the optimum value.

In order to evaluate the number of messages exchanged by the algorithm in the worst case, we observe that the pattern of delays in communication links could be such that every DISCOVER or VISITED message that can prevent the mistake of sending a DISCOVER message to an already visited node is received at its destination only after such a DISCOVER message has been dispatched in the opposite direction. In other words, every node in the graph may have to send a DISCOVER message to every one of its neighbors other than the father. Thus, the start node s will send as many as d_s DISCOVER and $d_s - 1$ VISITED messages. Any other node i, of degree $d_i \geq 2$, will send $d_i - 1$ DISCOVER, $d_i - 2$ VISITED, and one RETURN messages. A node of degree one can only send a RETURN message. Summing up, the total number of messages in the worst case is $4m - 2(n-1) + (n_1' - 1)$, where n_1' is the number of nodes of degree one in the graph, excluding the start node s. Thus, the message complexity of our algorithm could vary between 2m and $4m - (n-1) - 1$, depending on the structure of the graph and the pattern of delays in the communication links. Also, observe that if the communication graph G is a tree by itself, then $m = n - 1$ and $\ell = n_1'$, and thus in this case our algorithm will require exactly $2m + n_1' - 1$ messages independent of the nature of delays in the communication links.

5. Concluding remarks

In this paper, we have presented a modification to an algorithm proposed by Awerbuch for the distributed construction of a DFS tree in an asynchronous communication network, with gains in three dimensions—number of different types of messages, message complexity, and time complexity. The modified algorithm presented in Appendix A uses only three types of messages. But observe that RETURN and VISITED messages basically play the same role and that all RETURN messages can be replaced by VISITED messages in the algorithm. Thus, the algorithm really requires only two types of messages. However, if, at the end of computation, each node should know not only its father in the DFS tree but also its children, all three types of messages are needed. Besides,

the worst-case message complexity of the algorithm is less than $4m-(n-1)$. More importantly, the algorithm is time-optimal. The fact that the ACK messages of Awerbuch's algorithm can be eliminated helps us recogize that it is the VISITED messages that really enable us to achieve an O(n) time complexity. Moreover, the fact that no node will receive more than one DISCOVER message if the communication delay in any link is exactly one time unit shows that if the communication delays in all links are nearly the same, the message complexity of our modified algorithm will be very close to the optimal value of 2m.

In Section 3, we referred to Segall's algorithm for pure traversal. This algorithm for building an arbitrary spanning tree with a specified root, which is reliably informed of the completion of the traversal before the termination of the algorithm, can be shown to be both message- and time-optimal. On the other hand, whether there exists an algorithm for constructing a DFS tree which is both message- and time-optimal is an open question.

Acknowledgment

The authors thank the unknown referees for pointing out an error in the earlier version of the algorithm and for suggesting improvements to the presentation.

Appendix A. The formal presentation of the algorithm

Messages of the algorithm

DISCOVER	– sent to a neighbor who is not known to have been visited, for the purpose of visiting,
RETURN	– returns the center of activity to the father,
VISITED	– informing neighbors of the status.

Variables kept at node i

neighbors(i)	– set of neighbors of node i (input),
father(i)	– father of i in the DFS (output); initially, father(i) = i for all nodes; finally, father(i) = i only for the start node,
nomessage(i)	– subset of neighbors(i) including those neighbors not known to have been visited, i.e., no VISITED, DISCOVER or RETURN messages have been received from them; initially, nomessage(i) = neighbors(i) for all nodes.
visited(i)	– boolean flag set to true once visited, i.e., on receiving the DISCOVER message for the first time; initially, visited(i) is false for all nodes,
explore(i)	– the identity of a neighbor in nomessage(i) to whom a DISCOVER message has been sent; initially, explore(i) = i for all nodes.

To trigger the algorithm, node s delivers a DISCOVER message to itself. This message is not counted in the complexity.

Algorithm at node i

```
for DISCOVER message from j do
    begin
        delete j from nomessage(i);
```

```
            execute procedure recover;
            if node i has already been visited
              then do nothing
              else begin
                        set visited(i) to true;
                        set father(i) to j;
                        execute procedure shift-center-of-activity;
                        for all p ∈ neighbors(i) and p ≠ father(i) and p ≠ explore(i) do
                          send VISITED to p
                   end
        end;


    for VISITED message from j do
        begin
            delete j from nomessage(i);
            execute procedure recover
        end;

    for RETURN message from j do
        begin
            delete j from nomessage(i);
            execute procedure shift-center-of-activity
        end;


    procedure recover;  /* initiate recovery, if necessary */
      begin
            if explore(i) = j
              then execute procedure shift-center-of-activity
              else do nothing
      end;


    procedure shift-center-of-activity;
      begin
            if there exists k ∈ nomessage(i)
              then begin
                        set explore(i) to k;
                        send DISCOVER to k
                   end
              else begin
                        set explore(i) to i;
                        if father(i) = i
                          then TERMINATE  /* start node */
                          else send RETURN to father(i)
                   end
      end;
```

References

[1] B. Awerbuch, A new distributed depth-first-search algorithm, Inform. Process. Lett. 20 (3) (1985) 147–150.
[2] E.J.H. Chang, Echo algorithms: Depth parallel operations on general graphs, IEEE Trans. Software Engrg. SE-8 (4) (1982) 391–401.
[3] T. Cheung, Graph traversal techniques and the maximum flow problem in distributed computation, IEEE Trans. Software Engrg. SE-9 (4) (1983) 504–512.
[4] S. Even, Graph Algorithms (Computer Science Press, Potomac, MD, 1979).
[5] A. Segall, Distributed network protocols, IEEE Trans. Inform. Theory IT-29 (1) (1983) 23–35.
[6] M.N.S. Swamy and K. Thulasiraman, Graphs, Networks and Algorithms (Wiley, New York, 1981).