A NEW OPTIMAL DISTRIBUTED ALGORITHM FOR THE SET INTERSECTION PROBLEM *

Subbiah Rajanarayanan and Sitharama S. Iyengar

Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803.
Revised (October 16, 1990)

ABSTRACT

A simple decentralized distributed algorithm for the set intersection problem is presented. This algorithm is an improvement of the previous bound for the problem and uses only $3.m(n-1)+3.(n-1)$ messages, where m is the cardinality of the smallest sized subset and n is the number of processors, compared to $5.mn+2.n-4.m$ messages as proposed in a previous solution for the same problem. Our algorithm has a time complexity of $O(mn)$.

Keywords: Asynchronous system, centralized protocol, decentralized protocol, distributed algorithm, lower bound, Set Intersection

1. Introduction

A distributed system is basically a set of autonomous processors that communicates only through a communication network with no memory shared among processors. Consider such an asynchronous point-to-point communication network. Each node is an independent processor. Each processor i in this network contains a set S_i. The problem is to find the intersection of the all sets in the network. We shall be referring to the communication network as a distributed system. In a recent paper [4] , an optimal distributed algorithm was presented to solve the set intersection problem, with a message complexity of $\Theta(mn)$, where m is the cardinality of the smallest sized subset and n is the number of processors in the system. However, the decentralized algorithm is complicated, and requires each processor to know the Universal set U, and the exact number of messages required is not minimum. In the rest of this paper we shall refer to the algorithm presented in [4] as Algorithm A. In this paper, we present a distributed decentralized algorithm which uses lesser number of messages than required by algorithm A and further more, our algorithm is simpler and does not require each processor to know the Universal set U. Our centralized and decentralized algorithms use at most $3.m(n-1)+3.(n-1)$ messages, whereas the algorithm A proposed in [4] uses $5.mn+2.n-4.m$ messages in the decentralized version. It will be evident that our algorithm is considerably simpler than the algorithm previously proposed to solve the set intersection problem.

2. The Model

A *Distributed algorithm* is a collection of automata, where one automaton is associated with each processor in the system. Given an instance I of input size n of a problem and an algorithm A which solves the problem, the *message complexity* of A on I is the total number of messages generated by all the processors during the execution of A.

* Research is partially supported by LEQSF - Board of Regents grant to Professor Iyengar.

We consider an asynchronous point-to-point connected communication network. At each node there exists an independent processor. We shall be refering to this communication network as a distributed system. Each processor is assumed to have a distinct id_i, and a private set S_i of values. Two processors can exchange information only through explicit messages on a communication medium; the communication provides a point to point communication capability (thus, at a given instance, a processor can send a message to one/more of its neighbors). The distributed algorithm is initiated at one or more nodes/processors (usually by a command given by the user or generated by a system operating at a site). A processor can send and/or receive messages to/from processors that are only adjacent to it. The communication links are bidirectional. Each processor does some local computation and sends out messages to its neighbors (some, all or none at all). We assume that it can receive messages from all its neighbors and that no message is lost in transit. All messages are guaranteed delivery within an arbitrary but finite amount of time. The order in which the messages are received by any node is immaterial for the execution of this algorithm.

The exchange of messages between two processors is *asynchronous* in that the sender always hands over the message to the communication subsystem and proceeds with its local computation if any. A similar model of distributed computing may be found in [2] and [1].

3. Motivation For The Distributed Set Intersection Problem

Scheduling meetings among personnel is one of the most common tasks in an office environment. Each person can be considered to have a collection of *available slots of time* in such a setting and the task is to find a slot in their intersection. We use the same scenario as provided in [4]. The above problem can be abstracted as that of finding an element in the intersection of the sets.

4. Description of Previous Work

We shall look into the decentralized version of algorithm A in [4]. The basic idea behind the algorithm is as follows: the algorithm is divided into several *phases* and a possibly different processor directs the processing during each phase; in the first phase, R, the designated root of the underlying spanning tree of the network, finds a processor with the smallest sized set and passes the control to that processor while simultaneously reorienting the tree to be rooted at that processor; the root of the tree now simply sends its values one at a time down that tree; each processor computes the size of the intersection of its set with the set received during that phase and sends this number to its parent (in a bottom-up fashion); thus, at the end of each phase, a new root is found which has the smallest intersection with the set received *during that phase*. The algorithm is terminated when all processors have responded with exactly the same number of values as the number of values sent by the root during that phase.

The process during each phase is as follows: each processor has a candidate set at the beginning of the first phase; the (new) root of the tree sends its values down the tree one at a time; each processor, after receiving these values, computes its intersection with this set and treats the intersection as the new candidate set. Each leaf processor sends (processor-id, size-of-candidate-set) to its parent. An internal processor receiving messages from all of its children finds the minimum of the numbers received; the root finally computes the minimum of all the processors, and if this minimum is equal to the size of its own candidate set , then it sends a "terminate" message on the tree and moves into a final state, marking the candidate set as the intersection. Each of the other processors, after receiving the terminate message, marks its candidate set as the intersection and enters a final state. If, on the other hand, the number sent by one of the processors is smaller, then a processor with the smallest value is chosen as the next root and a message is sent to that processor. This message triggers all the processors on that path to change the directions of the edges on the path (so that the tree is reoriented) and the new root is selected to start the next phase.

5. The Proposed Algorithm

We assume that there exists a spanning tree in the network. Let P be the root of this tree. The first step in the algorithm is to find that processor which has the smallest sized set. This is achieved as follows (by a standard process): P sends a "*Compute* min" message down the tree. Each processor upon receiving such a message simply forwards it to its children. A leaf processor upon receiving the message sends an ordered pair (Processor id, set-size) to its parent in the tree. An internal processor upon receiving messages from all its children computes the minimum set size among all of its children and itself and passes to its

parent (processor-id, set size) corresponding to the minimum. Thus, P can compute the minimum size of the sets and the processor with that set (a tie being arbitrarily broken). The procedure *ComputeMin* uses exactly $2*(n-1)$ messages.

Let the processor with the minimum sized set be R. We now reorient the tree so as to make R the new root of the tree. This process of reorientation of the parent child relationship in the tree takes no more than $n-1$ messages.

The root processor R now sends its entire set down the tree. The size of this message is atmost m. Each internal processor when it receives this set will compute the intersection of the set received with its own set. It will then send a message containing the elements of this intersection down the tree to its children. When a leaf node/processor receives a message it computes the intersection of its own set with the set received from its parent. The leaf processor now sends this intersection set to its parent. An internal node upon receiving the intersection sets from all its children then computes the intersection of all the sets it received. This information is then sent to its parent and this process goes on until the root note receives the intersection sets from all its children. Upon receiving the messages (intersection sets) from all its children the root node computes the intersection of all the sets in the system. This information is then sent down the tree to all the processors. For more clarity, We shall now present a step wise description of the algorithm.

5.2 Description of the Distributed Set Intersection Algorithm

Each processor in the system has a set of elements S_i which we shall refer to as its *Candidate Set* C_i.

1. Find the processor/node in the system/network which has the minimum sized set using procedure *Compute* min. (Uses exactly $2*(n-1)$ messages)
2. Reorient the tree such that the node selected in step 1. is made the root. (Uses exactly $n-1$ messages)
3. The root sends the elements of its candidate set down the tree to all its children.
   - 3.1 Each Intermediate node upon receiving the set of elements from its parent, computes the intersection of its own set with the set received and then sends this newly computed set to all its children. This newly computed set now becomes its candidate set C_i.
   - 3.2 A leaf node upon receiving the elements of the set from its parent computes the intersection of its own set with the set received and sends the newly computed candidate set C_i up the tree to its parent.
4. Upon receiving the sets from all its children, an intermediate node computes the intersection of all these sets received. It also forms a list of all the sets received along with the *id* of the processor from which it was received. It then sends this intersection set up the tree to its parent.
5. The root upon receiving the sets from all its children, computes the intersection of these sets. This set referred to as INT is the final intersection of all the sets in the system. The root also forms a list of all the sets received along with their processor *id*. This information is required to minimize the number of messages required to broadcast the final intersection set. (Steps 3, 4, 5 use at most $2.(n-1).m$ messages)
6. The root then computes the set difference D_i, which is C_i - INT for each of the sets received. If $|D_i| \leq |\text{INT}|$ then the root sends D_i to processor i, one element at a time with a flag which tells that processor to discard the element from its candidate set, otherwise it sends INT to processor i. The intermediate nodes follow the same procedure as the root. (requires at most $m(n-1)$ messages)

6. Time and Message Complexity Analysis

In the above section, the sequence of steps used in the algorithm is outlined along with the message complexity for each step. In steps 3,4, and 5, during the execution of the algorithm each node receives exactly one message from its parent (other than the root node), the size of which cannot exceed m. Each child processor sends exactly one message to its parent, which in turn sends one to its parent and so on, until the messages reach the root. The message complexity of this entire process is no more than $2.(n-1).m$. This is because, the there are exactly $n-1$ messages transmitted during that phase (there are primarily only two phases in this algorithm, the forward phase refers to messages being sent from the root

down the tree and the backward phase to messages being sent from the leaves up the tree to the root) and the size of any single message cannot exceed m. The last part of the algorithm deals with the broadcast of the result to all the processors which again requires atmost $(n-1).m$ messages. Therefore, the overall message complexity of the entire algorithm is $3.(n-1) + 3.m.(n-1)$.

The time complexity of the above algorithm is $O(mn)$. It is a common assumption in this model of computation that the transmission of a message takes one unit of time. Therefore in any tree computation which is basically top-down or bottom-up (root to leaf or leaves to root) the time complexity is $O(h)$, where h is the height of the tree. Since the value of h can be atmost n, and the size of each message can be atmost 5, the overall time complexity of our algorithm is $O(mn)$.

7. Comparison of Performance

Our algorithm is similar to the decentralized algorithm proposed in [4]. The proposed algorithm is better than the previous one [4], for the following reasons.

- Message complexity analysis of our algorithm is simple.
- Our algorithm involves only one phase, whereas algorithm A in [4] can have many phases (the notion of a phase is as described in section 4) in the execution of the algorithm with *each phase requiring a reorientation of the tree.*
- Our algorithm does not necessitate the fact that all the processors have knowledge of the universal set. *The computation of the message complexity of algorithm A also involves the fact that each of the processors has information about the universal set.*
- The complexity analysis of the proposed algorithm takes into account the number of mesages needed for the reorientation of the tree. *The messages required for reorienting the tree at each phase has not been taken into account in the calculation of the bound for the message complexity of algo A.*
- The time complexity of our algorithm is $O(mn)$. In [4], no mention is made of the time complexity for the distributed set intersection algorithm in the decentralized case. It is apparent from the nature of algorithm A, that the time complexity is of $O(m^2n)$, since there could be atmost m phases in that algorithm.

Algorithm A performs well when the number of phases is at most one or two. This is reflected by the fact that there is no need for broadcasting the results to all the processors at the completion of the algorithm. The major emphasis of this paper is in the bound of the message complexity, which is considerably lower.

The proposed decentralized algorithm is message-optimal for the set-intersection problem according to the optimality criterion provide in section 2 of [4].

9. Conclusions

We have presented a simple decentralized distributed algorithm for the set intersection problem which uses at most $3.m(n-1) + 3.(n-1)$ messages, where m is the cardinality of the smallest sized subset and n is the number of processors. This improves the result of [4] where, a centralized and a decentralized distributed algorithm using $3.m(n-1) + 3.(n-1)$ and $5.mn + 2.n - 4.m$ messages, respectively was presented. The time complexity of our algorithm is $O(mn)$. The emphasis of this paper is more on the simplicity of the proposed algorithm and the reduction in message complexity and time complexity.

It is interesting to note that there may be many cases where the intersection is empty. In these cases, we would be interested in the *most prefered* time slot [3].

10. Acknowledment

The authors would like to thank the reviewers who improved the quality of this paper. Special thanks are due to Professor Kott for his timely comments on this paper.

References

[1] A.Segall, "Distributed Network Protocols," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 23-25, 1983.

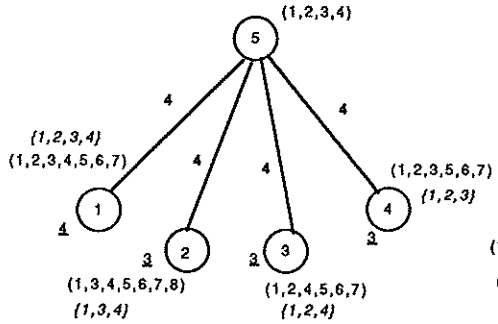
[2] Chandy, K.M. and Misra, J., "Distributed Computation on graphs: Shortest Path Algorithms," *CACM*, vol. Vol.25, 11, pp. pp. 833-837, 1982.

[3] Iyengar, S.S. and Subbiah, R., "Optimal Distributed Algorithm for determining the Mode of a distribution.," *Tech Report, Robotics Research Laboratory, LSU.*, 1990.

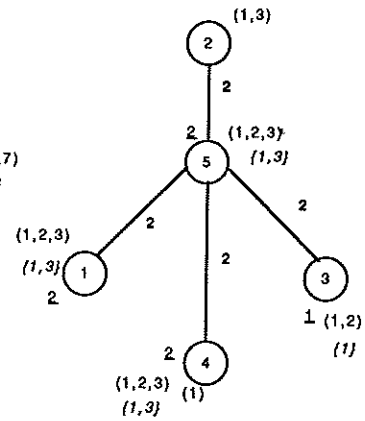
[4] Ramarao, K.V.S., Daley, Robert, and Melhem, Rami, "Message Complexity of the Set Intersection Problem," *IPL*, vol. 27, pp. 169-174, 1988.

Appendix A

Algorithm of Ramarao et al. [4]



PHASE I



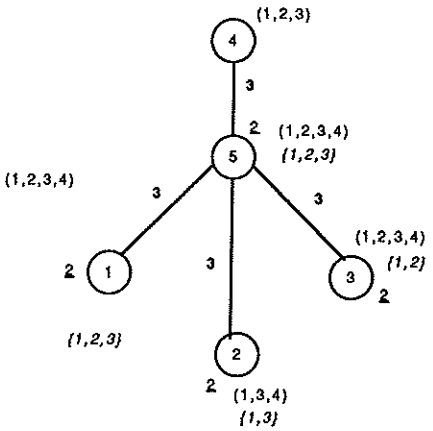
PHASE III

The total number of messages used by algorithm A.

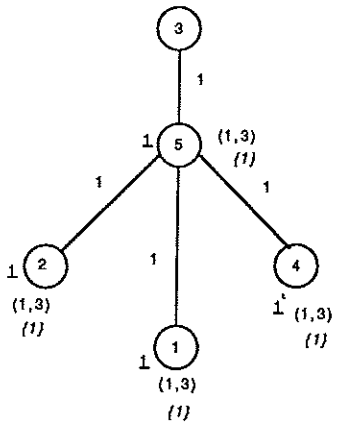
Phase I	4 + 4 + 4 + 4 + 4	= 20
Phase II	3 + 3 + 3 + 3 + 4	= 16
Phase III	2 + 2 + 2 + 2 + 4	= 12
Phase IV	1 + 1 + 1 + 1 + 4	= 8
Total		= 56

The total number of time units used by algorithm A

4 + 1 + 6 + 2 + 4 + 2 + 2 + 2 = 23 units.

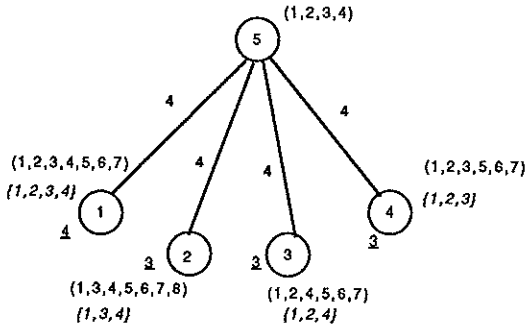


PHASE II



PHASE IV

The Proposed Algorithm



PHASE I

The total number of messages used by the proposed algorithm is

4 + 4 + 4 + 4 + 4 + 3 + 3 + 3 = 29
Broadcast = 1 + 1 + 1 + 1 = 4
Total = 33

Total time units = 4 + 4 + 1 = 9 units