A NOTE ON THE PROBLEM OF ELECTING A LEADER IN A SYNCRHONOUS RING

Eli Gafni

January 1985
CSD-850001

CSD-85001
Jan 1985

A NOTE ON THE PROBLEM OF ELECTING A LEADER IN A SYNCHRONOUS RING

(Extended Abstract)

Eli Gafni
Computer Science Department
University of California
Los Angeles, CA 90024

Abstract

Recently, Frederickson and Lynch [5], and Vitanyi [10], have proposed an election algorithm for synchronous rings, which works in $\Theta(n)$ -messages, and $\Theta(n2^{|T|})$ -time, where n is the size of the ring, and T is the set of the node identifiers (ID's). Thus the time behavior for fixed n is exponential in $|T|$. In this note we propose two other alternative algorithms, which improve on this time behavior. The first works in $\Theta(n)$ -messages, and $\Theta(n2^n+|T|^2)$ time. The second works in $O(n\alpha(n))$ -messages and $O(\alpha^{-1}\alpha(n)|T|)$ time, where $\alpha(n)$ is the functional inverse of Ackermann's function, and thus grows very slowly.

1 Introduction

The problem of electing a leader efficiently has been extensively studied [4,2,1] [6,7,8,9]. Recently, Frederickson and Lynch [5] have shown that for a synchronous ring one obtains different results, depending on whether the election algorithm uses only comparisons among the ID's of processors (comparison algorithm), or is allowed to use the numerical values of the ID's to count rounds (noncomparison algorithm). Specifically, for the case of a noncomparison algorithm they, and independently Vitanyi [10], have proposed an $\Theta(n)$ -messages, and $\Theta(n2^{\min ID})$ -time algorithm; while, on the other hand it is proved in [5] that any comparison algorithm requires $\Omega(nlogn)$ -messages. The algorithms in [5] and [10] are essentially the same and thus we will refer to them as one algorithm.

As in [5] we will assume that the ID's belong to the set { 1,2,... }. It is easy to see that this assumption includes the case that the set is countably finite or infinite, just by allowing the algorithm to associate an ID with its count. Thus, for fixed n, and a finite set of ID's T whose cardinality is viewed as a variable, the time of the

algorithm in [5] is exponential in $|T|$. Our aim in this note is to propose algorithms whose time behavior as a function of $|T|$ is significantly better, while their message complexity is independent of $|T|$ and is linear or "close" to linear in n.

We first propose an election algorithm under the assumption that the algorithm at each processor i (that is its ID=i) has access to a number $u(i)$, $u(i) \geq n$, $u(i) \geq u(j)$ if $i > j$, which is otherwise arbitrary. We show that this numbers provide delays to processors so that one processor will "capture" all other processors before they have started capturing processors. Thus, the message complexity of the algorithm is $2n$ and its time $\Theta(\ (min\ ID)u(min\ ID)\ )$. We use this algorithm as a building block in our two main algorithms.

Our main results are two election algorithms. In both we relax the assumption that $u(i) \geq n$. The idea behind the first algorithm is that each processor i will put $u(i)=i$. Using the previous algorithm as a "first phase", will automatically eliminate from the competition all ID's greater or equal to n, since those nodes used $u(i) \geq n$. We resolve the remaining contention by operating the algorithm in [5] , but now (min ID) is guaranteed to be lower than n. In the case where all ID's are greater than n, the algorithm terminates after the first phase.

The second algorithm uses the "first phase" repeatedly, until only one node remains. Each time the "first phase" algorithm is applied the processors which are still in contention for the leadership put $u(i)=\alpha^{-1}($ *the number of times the first phase algorithm was already applied*), where $\alpha^{-1}()$ is the Ackermann function, which grows extremely fast. Obviously, there is no need to apply the "first phase" algorithm more than $\alpha(n)$ times, since by then for all i, $u(i) \geq n$.

We show that the first algorithm is linear in messages while quadratic in $|T|$. The second algorithm is "almost" linear in messages and linear in $|T|$. As for the question whether we can get rid of the dependency of the time on $|T|$ all together (for any cost in n), it was answered negatively in [5], where it is shown that as long as the message complexity is $o(nlogn)$, time must strictly grow in $|T|$.

2 The Problem of Noncomparison Election for Synchronous Rings

Since all the results in this note apply with an easy modification to bidirectional rings, we restrict ourselves w.l.o.g. to unidirectional rings. A unidirectional ring consists of a collection of unidirectional processors connected in a way that forms a single ring. A unidirectional processor is a processor that possess only a single input and a single output port. Input and output ports are used by the processor to receive and send messages, respectively. A unidirectional ring is synchronous if all the processors in the ring get "ticks" from a single clock. Messages are sent at the "ticks" of the clock.

They are carried by the transmission subsystem from an output port to the input port connected to it. The transmission subsystem is fault free. A message sent at one "tick" will be received at the next "tick". The problem is to install an identical program in all processors with the following conditions and requirements:

1. A program is either activated from the outside or by its processor receiving the first message.

2. Each program has an access to a private positive integer (ID) installed in the processor in which it resides. Branching in the program is allowed to depend on the numerical value of the private integer, as well as on other private integers forwarded to its processor in messages.

3. Computation takes no time.

4. If all private variables are distinct, and an arbitrary set of programs is activated from the outside (if it is not active already, via rule 1), then in finite time the ring will reach quiescence, with a single program outputting ELECTED.

A program that satisfies these conditions and requirements is a noncomparison algorithm for synchronous rings. In the sequel we will refer to the program residing at processor i, and processor i interchangeably, and we will assume that all the ID's are distinct.

3 The AUB (Access to Upper Bound) Algorithm

In this section we present an algorithm which is the crux of this note. We assume that the program in each processor i (i=ID) has access to a positive integer variable $u(i)$, and the conditions we impose on $u(i)$ are that $u(i) \geq n$, $u(i) \geq u(j)$ if $i > j$. The first condition on $u(i)$ is relaxed in the sequel. Notice that the second condition is satisfied for $u(i)=f(i)$ by any nondecreasing function f, in particular the constant function. The analysis of this algorithm will motivate and clarify the algorithms that follow.

Processors send an AWAKE message when they are activated. An AWAKE message received after a processor was activated is ignored. Starting with the clock "tick" in which it was activated, processor i starts to count $i(2u(i)+1)$ "ticks" (or rounds). When processor i finishes counting, it sends a message STOP(i). A processor receiving a STOP() message while counting, becomes a "repeater" (it repeats this STOP() message too). A processor i receiving STOP(i) outputs ELECTED.

We will state in a quasiformal manner some properties of the algorithm which will convince the reader that the algorithm is correct.

Let $AWAKE(i)$ be the time that processor i was activated.

Fact 1:
For all i,j, $|AWAKE(i)-AWAKE(j)| \leq n$.
Fact 2:
for all $i>j$, $\{ AWAKE(i)+i(2u(i)+1) \} - \{ AWAKE(j)+j(2u(j)+1) \} \geq n+1$.

Let m be the smallest ID in the ring.
Fact 3:
STOP(m) message will reach all other nodes before they initiate their STOP() message.

It is rather trivial to see that the number of messages sent by this algorithm is $2n$ and the time spent is bounded by $2n+m(2u(m)+1)$.

At this point it is natural to ask what happens if the algorithm is run with a $u()$ that violates the first condition.
Fact 4:
Let $u(i)$ be possibly smaller than n, then all processors j initiating a STOP() message have $j<n$ or $j=m$.
We are now ready to state our main result.

4 The Main Result

Algorithm 1:

The first phase of the algorithm is a slightly modified AUB, which in addition is run with $u(i)=i$. The first modification we introduce to AUB will leave Fact 4 correct, but will prevent processor m from leaving the "competition". This is achieved by the following statement: If processor i receives STOP(j) while its count is in progress and $i<j$, then, instead of becoming a repeater, it stops counting and sends STOP(i). Secondly, we modify the rule for outputting ELECTED. Processor i outputs ELECTED upon receiving STOP(i) or upon receiving TRAVEL(i) in the second phase, as outlined below.

All processors i that initiate a STOP(i) message proceed to send a message TRAVEL(i), upon reception of STOP(j) $j \neq i$. This TRAVEL(i) message is delayed at each processor k for 2^i rounds, unless $k<i$ in which case the message is destroyed. Processor i receiving TRAVEL(i) outputs ELECTED.

The second phase of the algorithm is a restatement of the algorithm in [5], for "participating" processors, without their first phase. In order for their analysis to carry directly to our "second phase" we just need the rather easy Fact:
Fact 5:
All nonrepeaters initiate a TRAVEL(), and they do so within n rounds of each other.

It is easily verified that the number of messages incurred by this algorithm is bounded by $5n$ while, using Fact 4, its time is $O(m^2+n2^m)$.

Algorithm 2:

The idea behind Algorithm 2 is to operate the "first phase" of Algorithm 1 repeatedly on the remaining nonrepeaters, with $u(i)=\alpha^{-1}($ *the number of times the first phase was applied*). The number $u(i)$ will exceed n after few applications of the "first phase", and at this point we know the algorithm must have terminated, according to Fact 2. To accomplish this we have to define the time at which a nonrepeater processor updates $u()$ and restarts the "first phase". This times are conveniently defined as the times of reception of a STOP() message. That is, when a processor receives a STOP() message it treats it according to the rules of the "first phase", and then according to whether it became a repeater or not it updates $u()$ and restarts. The argument of $\alpha^{-1}()$ is initialized to 1 and is incremented by 1 with each restart.

We state without proof (a rigorous proof is forthcoming in the full paper) that this algorithm works, its message complexity is $\Theta(\alpha(n)\cdot n)$, and its time is $O(m\alpha^{-1}\alpha(n))$.

5 Conclusions

We have presented two algorithms for election on a synchronous ring which work with linear or "almost" linear number of messages, while their time grows quadratically or linearly in the space of the ID's, respectively. Many questions remain unresolved, like a possibility of paying with "additive" time-complexity in n in order to reduce the factors multiplying $|T|$. The more interesting question is to close the gap between the known lower bound function which grows extremely slowly, and the linear time achieved here. Also notice that the lower bound in [5], works for any message complexity which is $o(nlogn)$, thus there remains the question of the relation between the actual message complexity and the lower bound function.

Finally notice that our algorithms, as well as the algorithm in [5], apply to general bidirectional networks, by using a "shout echo" mechanism [3]. The complexity of these algorithms will stay the same, with $|E|$, the number of edges in the network, replacing n.

References

[1] J. E. Burns, "A Formal Model for Message Passing systems," TR-91, Indiana Univ., Bloomington (May 1980).

[2] Ernest Chang and Roberts R., "An improved algorithm for decentralized extrema-finding in circular configurations of processes," *Communication of the ACM* **22**(5), pp.281-283 (May 1979).

[3] Ernest Jen-Hao Chang, "Decentralized Algorithms in Distributed Systems," CSRG-103, University of Toronto, Toronto, Canada (October 1979). Ph.D. Thesis.

[4] Danny Dolev, Maria Klawe, and Michael Rodeh, "An O(n log n) Unidirectional Algorithm for Extrema Finding in a Circle," *Journal of Algorithm* **3**, pp.245-260 (1982).

[5] Greg N. Frederickson and Nancy A. Lynch, "The Impact of Synchronous Communication on the Problem of Electing a Leader in a Ring," pp. 493-503 in *Proceedings of the 16th Ann. ACM Symp. on Theory of Computing*, Washington, D.C. (1984).

[6] Robert G. Gallager, Pierre A. Humblet, and P. M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees," *ACM Trans. Program. Lang. Syst.* **5**, pp.66-77 (Jan 1983).

[7] D. S. Hirschberg and J. B. Sinclair, "Decentralized extrema finding in circular configurations of processors," *Comm. ACM* **23**, pp.627-628 (November 1980).

[8] A. Itai and M. Rodeh, "The lord of the ring or probabilistic methods for breaking symmetry in distributed networks," RJ-3110, IBM (April 1981).

[9] Gary L. Peterson, "An O(n log n) Unidirectional Algorithm for the Circular Extrema Problem," *ACM Trans. Program. Lang. Syst.* **4**(4), pp.758-762 (October 1982).

[10] P. Vitanyi, "Distributed election in an Archimedean ring of processors," pp. 542-547 in *Proceedings 16th ACM Symp. on Theory of Computing*, Washington, D.C. (1984).