Lecture Notes for CSCI 379: Distributed Computing

Set 2-Leader Election

Professor Talmage

February 1, 2023

---

1 Problem Statement

This is the problem we looked at the first day, where we want to select one, and only one, of the participating computing entities.

Problem 1. All nodes in a communication network should terminate in a finite time, each with a return value of $\langle leader \rangle$ or $\langle nonleader \rangle$. Exactly one node must return $\langle leader \rangle$.

- Alternate: Each terminating state is labeled either $\langle leader \rangle$ or $\langle nonleader \rangle$. In every admissible execution, exactly one process enters a $\langle leader \rangle$ state.
- Variation: All processes should have the ID of the process which returns $\langle leader \rangle$.

There are a couple of algorithm (solution) categories which are interesting to explore for this problem:

- *Anonymous* algorithms cannot use process IDs. While we may name processes in discussion and analysis of the algorithm, processes do not know these names, either their own or others'.
- *Uniform* algorithms cannot use n, the number of processes. That is, the exact same code must work on a system of any size. (And code which reads the number of processes is disallowed.)

The leader election problem is applicable and interesting in any graph, but we're going to restrict ourselves to a simple class of graphs for right now, as that's complex and interesting enough to get us started.

Definition 1. A *ring* is a graph which is a simple cycle: If we denote the set of nodes as $V = \{p_0, \ldots, p_{n-1}\}$, then $E = \{(p_i, p_j) \mid j \equiv i + 1 \bmod n\}$.

- In a ring, $|E| = |V| = n$, so complexity should be in terms of n.
- We will talk about nodes' *left* and *right* edges: p_i's right edge connects it to p_{i-1} and its left edge connects it to p_{i+1} (index arithmetic modulo n throughout). That is, WLOG assume that IDs increase clockwise and processes are facing the center of the ring.

2 Simple Version: Non-Anonymous

Here, assume that every process has a unique numerical (importantly, comparable) ID. We will give a uniform algorithm: processes do not need to know n to solve the problem. Of course, if we can solve the problem without knowing n, then we could also solve it with knowledge of n by running the same code.

Idea:

- All messages will travel to the left (clockwise).
- Each process sends its ID to its left neighbor.
- Each process forwards messages larger than its own ID, absorbs all smaller messages
- If a process receives its own ID, it terminates as leader.
- Send an announcement message around the ring, other processes terminate after forwarding it.

Algorithm 1 Code for each process p_i to solve non-anonymous LE in a ring

```
1: Upon event Initialization
2:     send i to left neighbor
3: Upon event Receive message containing x ≠ i
   ▷ We could more precisely say we are receiving x from the right, but in this algorithm all receives are from the right, so we omit that detail.
4:     if x > i then send x to left neighbor
5:     end if
6: Upon event Receive message containing i
7:     send ⟨leader⟩ to left
8:     return ⟨leader⟩ (terminate)
9: Upon event Receive ⟨leader⟩
10:    send ⟨leader⟩ to left
11:    return ⟨nonleader⟩ (terminate)
                    ▷ Note that the leader doesn't execute this handler, as it has already terminated.
```

Exercise: Is this algorithm synchronous? Partially synchronous? Asynchronous? That is, what timing assumptions must hold for it to be correct? Is this algorithm uniform?

Correctness:

- A node can only be leader if its ID passes every node in the ring and returns to itself, and only the largest ID will pass all other nodes. (The process with largest ID will eat every other message, unless it is already eaten.) Thus, only one leader can be elected.
- The node with the largest ID will have its ID passed all the way around the ring, so that process will terminate as leader. Thus, we have exactly one leader in any admissible execution.
- Every other node will terminate as *nonleader*, since the $\langle leader \rangle$ message is forwarded around the ring by every process. Thus all other processes will terminate and return $\langle nonleader \rangle$.

Complexity

- **Messages**: There are originally n IDs sent, and each travels no more than n times, since they can't pass their originator, so we have $O(n^2)$ messages. There are then an additional n $\langle leader \rangle$ messages, for a total of $O(n^2)$.

Exercise: Is this bound tight? Can you find an initial configuration that sends $\Theta(n^2)$ messages?

If processes are initially in clockwise decreasing order of IDs, then this will use $\Theta(n^2)$ messages.

- **Time**: At most $O(n)$ for the leader's message to go all the way around the ring, then $O(n)$ for $\langle leader \rangle$ messages, total of $O(n)$.

> **Exercise:** Is this tight? Is it possible to solve the problem faster?

This seems tight, keep it in mind as we look at other versions.

3 Anonymous Rings

We'll now remove processes' knowledge of their IDs. This means that they can't compare IDs, or *any* other information about themselves, to break ties. We'll start with the easiest version of the problem: assume the system is synchronous and non-uniform.

Idea: Not having IDs doesn't just mean that processes don't have their own numbers, it means they must start in the same state (or we could potentially distinguish them, using those differences as ID). Processes in the same state take the same step, since our algorithm is deterministic. Then they're in the same second state, take the same second step, and so on.

Theorem 1. *There is no non-uniform, anonymous LE algorithm in synchronous rings.*

Proof. We proceed by induction on the round number, k, showing that all processes are in the same state after round k, so no process can be leader unless every process is. The base case is $k = 0$, the initial state before the first round of computation. No process can be leader here, since all processes have the same initial state.

For the inductive hypothesis, assume that after some arbitrary round $k \geq 0$, all processes are in the same state. The inductive step is then to prove that all processes are in the same state after round $k + 1$. Because all processes are in the same state, the send the same message m_r to the right and the same message m_l to the left. Thus, in round $k + 1$, every process receives m_r from the left and m_l from the right. Since they are in the same state and have received the same messages, determinism implies that they make the same state transition, and end in the same state after round $k + 1$. □

Corollary 1. *There is no uniform, anonymous LE algorithm in asynchronous rings.*

This follows since such an algorithm would solve LE in a non-uniform, synchronous ring. And, while we do not state them separately here, all the possible combinations of uniformity and anonymity are similarly impossible.

4 Improving the Non-Anonymous Algorithm

Let us return to our previous non-anonymous algorithm and try to improve it. Specifically, we want to reduce our message complexity below $O(n^2)$. Recall that we are in an asynchronous system, with unique node IDs.

> **Exercise:** How might we be able to reduce the complexity? (What tool did we simply not use in the previous algorithm?) To what might we want to reduce it? Can that give us an idea what techniques might be helpful?

Idea: Try for $O(n \log n)$. To accomplish this, we need to repeatedly divide the problem.

- In 311, we might split to halves of the ring, then halves of those, and so on. We can't do that here, because the nodes don't have a global view to know which half they're in, or where the split points are. (We'd need to elect leaders to be the boundary nodes)
- Instead, we'll start from individual nodes.

  Exercise: How do we go from individual nodes to halving the problem?

  - First, every node competes with its neighbors. A node continues only if it beats both neighbors.

    Exercise: What criteria do we use for "beating" another node?

  - In the second round, compete with nodes at distance ≤ 2, only the winner continues. (A k-neighborhood for $k = 2$.)
  - Repeat this at distances $\leq 2^i$ in round i (2^i-neighborhood), until we have $i \geq log(n/2)$, at which point all (remaining) nodes are in the same competition.
- Thus, can decide a winner in $O(\log n)$ rounds. We just need to argue that each round takes $O(n)$ messages.
  - Round 1: All nodes compete, each sends 2 messages, replies to at most 2 messages, total $O(n)$.
  - Round 2: $\leq n/2$ nodes compete, each causes ≤ 8 messages (2 directions, distance 2, doubled for replies), total $O(n)$.
  - Round i: Each competitor causes $\leq 4 * 2^i$ messages (two directions, outgoing and returning, 2^i distance on each side). There must be 2^i non-competing processes between active nodes, so there are at most $\frac{n}{2^i+1}$ competitors, for a total of $(4 * 2^i)\left(\frac{n}{2^i+1}\right) \leq 4n$ messages.
  - Each round is at most $O(n)$ messages, so overall total of $O(n \log n)$ (Even if the leader sends an announcement around the ring at the end.)

  Exercise: This is an asynchronous algorithm, so what do I mean by round? What happens if a node lags behind?

Exercise: What is the time complexity of this algorithm?

Exercise: Write complete pseudocode for this algorithm. Use two types of messages, $\langle probe \rangle$ and $\langle reply \rangle$. Each message can have one of these tags and a few integers (E.g. $\langle probe \rangle$ messages should count how many times they've been forwarded to know when they hit the end of the 2^i-neighborhood.).

- What events will you need to handle?
- What information should a message convey?

Correctness A node can only become leader if it beats all nodes in up to a $2^{\log n}$-neighborhood, which is all nodes. A node cannot beat and be beaten by a single other node, so exactly one leader is elected.

Algorithm 2 Algorithm for asynchronous, non-anonymous leader election in a ring, code for each process p_i

```
1: Upon event INITIALLY
2:     send ⟨probe, i, 0, 1⟩ to left and right
3: end Upon event
4: Upon event ON RECEIVING ⟨probe, j, r, d⟩ FROM LEFT/RIGHT      ▷ j is ID, r is round, d is distance
5:     if j == i then return ⟨leader⟩
6:     end if
7:     if j > i and d < 2^r then send ⟨probe, j, r, d + 1⟩ to right/left (resp.)
8:     end if
9:     if j > i and d ≥ 2^r then send ⟨reply, j, r⟩ to left/right (resp.)
10:    end if
11: end Upon event
12: Upon event ON RECEIVING ⟨reply, j, r⟩ FROM LEFT/RIGHT
13:    if j ≠ i then send ⟨reply, j, r⟩ to right/left (resp.)
14:    else if already received ⟨reply, j, r⟩ from right/left (resp.) then send ⟨probe, i, r + 1, 1⟩ to left and
       right
15:    end if
16: end Upon event
```

Optimality We already argued that the algorithm has $O(n \log n)$ message complexity. This is actually optimal: Any LE algorithm in an asynchronous ring must send $\Omega(n \log n)$ messages.

The idea is that we can build an execution of an arbitrary LE algorithm which delays all messages on one edge in the ring. We can then argue that the algorithm will behave exactly the same way in this ring as in a ring of twice the size where all messages on *two* edges, opposite each other in the ring, are delayed. Induction can then show that, since each half of any ring could do this, we will need a total of $\Omega(n \log n)$ messages. (Details in textbook.)

5 Synchronous Rings

In an asynchronous system, not receiving a message does not tell you anything, because a message might exist, but be delayed. In a synchronous system, not receiving a message conveys information, since we can be confident that the neighbor chose not to send a message in the previous round.

Exercise: Can you use this extra information to build a LE algorithm using $O(n)$ messages?

Claim 1. *There is a LE algorithm that uses $O(n)$ messages in synchronous rings.*

Note that we still need to get away from comparison-based algorithms, and we cannot have time complexity be solely a function of n, as there are other lower bound proofs showing that $\Omega(n \log n)$ messages are still required in those domains.

Idea 1: For a non-uniform algorithm, we can use IDs to choose a time slot, instead of directly for comparison. The idea is much like time-division multiplexing or TDMA.

- Divide time into phases of n communication rounds each. That is, rounds $0, \ldots, n-1$, $n, \ldots 2n-1$, etc.
- Node i sends i to right in round $n * i$, then terminates as $\langle leader \rangle$.
- If any process receives a message with another ID, forward it and terminate as $\langle nonleader \rangle$.

Exercise: What are the time and message complexities of this algorithm? Can you improve either?

- Time complexity is $n(i+1)$, where i is the smallest ID in the system. Message complexity is n.
- By sending messages both directions, we could halve the length of each phase.

Idea 2: Similar to the first (unidirectional) algorithm for asynchronous systems, but don't blindly forward all messages. Most of them will be eaten sooner or later, so hold unlikely winners for a while to see if a "better" message overtakes it. This algorithm is uniform.

- Initially, every process sends its ID to the left.
- When receiving ID j, a process will hold it for 2^j rounds before sending it on. If the process is holding any IDs greater than j, they are discarded.

Exercise: What are the time and message complexities of this algorithm?

Note that the winner will always be the **lowest ID**, which we will call i.

- Time complexity is the time for the winner to get all the way around, which is $n2^i$.
- Any other message is at least $i+1$, so in $n2^i$ rounds, it will travel at most halfway around the ring ($n2^i/2^{i+1}$.) The next farthest a message could go is a quarter of the way around the ring, since its ID is at least $i+2$, etc. Total number of messages is thus $\leq n + n/2 + n/4 + \cdots \leq 2n$. (This is why we need to have the exponential waiting function–to ensure that non-leader messages cannot get too far around, leading to a high message complexity.)

Exercise: When are these algorithms good or bad? Are those scenarios realistic?