

Leader Election

Goal is to elect a unique leader of the entire network. Initially all processes start in the same state. At the end, exactly one process knows that it is the leader; all others know they are not.

Assumptions:

1. The system is synchronous. Can be generalized to asynchronous.
2. Processes have unique identifiers.
3. Comparison-based algorithms.

LE in a Ring

Random-NNT algorithm (Khan, Pandurangan, and Kumar, Theoretical Computer Science, 2007).

Random-NNT: 1. Each node chooses a unique rank as follows:

1. v chooses $p(v)$, a uniform random number $\in [0, 1]$.
2. $rank(w) > rank(v)$ if $p(w) > p(v)$ or if $p(w) = p(v)$ and $id(w) > id(v)$.

2. Each node initiates a token that contains its rank. The token is sent along the ring in one direction (say, in a clockwise direction).

3. When a node v receives a token with a smaller rank than itself the token is killed. Else it is allowed to progress further and v knows that it is not a leader.

4. If a node receives its own token, then it becomes the leader.

Analysis

Theorem 1. *Random NNT algorithm takes expected $O(n \log n)$ messages and $O(n)$ time.*

Let x_0 be the random number generated by node v and x_i denotes the random number generated by the i th nearest neighbor of v .

The probability that v 's token gets killed by the i th nearest neighbor is equal to the probability that x_0 and x_1 are the largest and second largest, respectively, among $(i + 1)$ random numbers: x_0, x_1, \dots, x_i .

This probability is $\frac{1}{i(i+1)}$.

The expected number of messages for a node is

$$= \sum_{i=1}^n i \cdot \frac{1}{i(i+1)} \leq H_{n+1} = \Theta(\log n).$$

By linearity of expectation, total message complexity is at most $\Theta(n \log n)$.

Worst-case message complexity is $O(n^2)$.

Peterson/Dolev-Klawe-Rodeh Algorithm (For Rings)

Worst case complexity is $O(n \log n)$.

Basic idea:

0. Initially all nodes (identities) are active.
1. Each active identity compares itself with two neighboring active identities in the clockwise and counterclockwise direction.
2. If the identity is not a local minima, it becomes passive. Otherwise it remains active in the next round.

In each round, at least half the identities become passive. After $\log n$ rounds only one remains.

LE in Complete Networks

Consider the following implementation of Random NNT (Khan, Pandurangan, Kumar, Theoretical Computer Science, 2007):

In round i :

Each node probes 2^{i-1} neighbors if it didn't succeed in the previous round.

If a node finds a neighbor of higher rank it stops. Otherwise it proceeds to the next round.

Analysis

Theorem 2. *The above protocol takes expected $O(n \log n)$ messages and $O(\log n)$ time.*

Proof: $O(\log n)$ time because at most $O(\log n)$ rounds.

The expected number of messages for a node is bounded by

$$1 + \sum_{i=1}^{\log n} \text{No. messages exchanged in round } i \Pr(\text{not successful in } i \text{ rounds})$$
$$\leq 1 + \sum_{i=1}^{\log n} 2^{i+1} (1/2^{i-1}) = O(\log n)$$

Linearity of expectation gives the expected total number of messages as $O(n \log n)$.

Peleg's LE Algorithm for Arbitrary Networks

Time lower bound: $\Omega(D)$.

Message lower bound: $\Omega(|E| + n \log n)$.

Peleg gave an algorithm that takes $O(D)$ time and $O(D|E|)$ messages.

Synchronous LE Algorithm

Each node i :

1. Keeps variable x and d .

x contains the id of the (current) leader candidate.

d contains the longest distance of any node from the node x , according to i 's knowledge.

2. In every step, set x to be the maximum id value among its current value and all the messages received during the previous timestep (pulse).

If it updates x then it adds 1 to the maximum d value it receives from the neighbor that sent it x .

If it already has x , then it simply updates its d value to that it receives from the neighbor that sent it x .

broadcast the current values of x and d to all neighbors.

3. If a node receives its own value in three consecutive pulses, it elects itself to be the leader. It then floods a completion message to terminate the LE algorithm.

Complexity Analysis

Assume that all initiators wake up at the same time and start the algorithm.

Let a be the highest ranked initiator. Let d_a be the depth of the BFS tree rooted at a .

a will receive an echo message from successively farther neighbors at every second pulse.

After $2d_a + 2$ pulses a will elect itself as leader.

$$d_a \leq D \leq 2d_a.$$

Time complexity is $O(D)$ and message complexity is $O(D|E|)$.

How to convert to an asynchronous algorithm

Any synchronous algorithm can be converted to an asynchronous algorithm by imposing the following condition:

Every node generates its own pulse.

A node generates pulse p only after receiving the p th message from **all** its neighbors.

Same time and message complexity as synchronous algorithm.