The University of Melbourne
School of Computing and Information Systems
COMP90020 Distributed Algorithms

Tutorial Week 6: Leader Election

Notes

Exercises

25. It is necessary so far as to maintain the invariant that the highest ranked processor should win.

In practical terms it increases message complexity as instead the recovering process could first attempt to get the status of other processes before initializing an election if there were no current coordinator.

The book suggests that it is also useful in the case where the processor rank is tied to the resources available and you wish to maintain the coordinator as the most resourceful processor.

26. Consider the following algorithm (invitation algorithm)

1. Each node initially considers itself to be the coordinator
2. Each node will then periodically check (union find) to see what other nodes exist within its connected component.
3. Upon finding some other connected component, the coordinators must first wait an inversely proportional amount of time before attempting to merge the groups.
4. The coordinator for some group (first to finish waiting) will send an invitation to all other coordinators in detected neighbors.

If after some timeout there are no signals from the coordinator, it attempts to failure detect (with bounds). In which case it can re-start the steps.

Likewise for a node rejoining, it follows steps 1 to 4

27. When there are no identifiers, this essentially becomes a task of symmetry breaking, this is commonly conducted via randomization.

Assume such an algorithm exists. let G^0 = set of all processors local state initially $(P_0, \cdots, P_N)$, Each processor must execute the same algorithm $A_1 = \cdots = A_N$.

As all processes execute the same local determinstic algorithm A_i, there is a synchronous execution during which all processes execute the same step of their local deterministic algorithm A_i, and each process pi proceeds consequently from its initial state to state 1.

Since they all enter the same state, G^0 progresses to another symmetric state G^1 (where all processes are in the same local state).

As all Ai are deterministic and identical, the previous synchronous execution can be continued and the set of processors progress from G^1 to $G^2 \cdots G^{\inf}$.

Therefore the algorithm will never terminate, as it must enter an asymmetric state to do so, yet by the symmetry of each process and each algorithm being executed being identical this can never occur.

28. Construct a virtual grid of size $(\sqrt(n) \cdot \sqrt(n))$. Where the rows, columns are numbered $1, \cdots, \sqrt{n}$

To be a leader, each node will send its own id to each node that matches its id and column number and receives an ack from each member.

A node however, will only send only one ack to the initiator whose id is the largest among all initiators.

All other contenders will receive a failed message.

Only one node can receive all the acks, and it becomes the leader. Every other node will receive at least one failed message.

The message complexity is $2 \cdot \sqrt{n}$

In the case of distributing to all nodes who the leader is. This is more complex, instead we can try and amortize that cost over subsequent communication.

For all nodes (other than the leader) to set $elected_i$, it must require at least n messages.

29. let $n = 2^m$. If two (m-1) cubes choose their leaders, then these leaders will communicate to choose the final leader of the m-cube. if each leader can pass its itd to the other leader through an entry node in the next dimension, then the leader with the smaller id opts out and the leader with the larger id is elected to the leader of the m cube. Each node is initially a leader of its 0 cube. Let L_i be the message complexity for choose the leader of an i-cube. Then:
$L_i = 2 \cdot L_i - 1+$ (messages required to pass each leader id to other leader)
For the () part, each node will exchange the id of its leader with its peer in the other (i-1)-cube, then select the largest among the two leader ids as the id of the leader of the i-cube. Its message complexity equals to the number of nodes in the (i-1)-cube, which is 2^{i-1}.

$$
\begin{aligned}
L_1 &= 2 \cdot L_0 + 2 \cdot 2^0 \\
L_2 &= 2 \cdot L_1 + 2 \cdot 2^1 \\
L_3 &= 2 \cdot L_2 + 2 \cdot 2^2 \\
L_m &= 2 \cdot L_{m-1} + 2 \cdot 2^{m-1} \cdots \\
L_m &= O(nlogn)
\end{aligned}
$$