CPSC 5520—Distributed Systems

Bully Algorithm Dance

Solution

Analysis

Is this algorithm correct?

1. Define correct. Think about what would constitute an incorrect algorithm, for instance.
   We're trying to accomplish establishing a coordinator amongst a well-connected group of peers. A correct algorithm will accomplish that with the following properties:
   - **Safety**: something bad will never happen. In this case, bad would be if processes didn't agree on who is the coordinator.
   - **Liveness**: progress will be made. In other words, we would avoid a situation where the peers never reach agreement.

2. Could you prove that the Bully Algorithm is correct (or not)? Outline how you'd do that.

   To prove safety, use contradiction. Under the algorithm, a leader is known to a node only if that node itself declares victory or has gotten a COORDINATOR message (the leader donning the crown is a *COORDINATOR* message to all the other peers). Once an election is over, the only way for two nodes to have a different idea of who's the leader is that there were two nodes, call them A and B, that had declared victory. Therefore, according to the algorithm they each had sent the other one a COORDINATOR message. But this cannot happen, since before sending the victory message, VOTE messages would have been exchanged between the two, and the node with a lower ID would not have sent out COORDINATOR messages.

   To prove liveness, we can look at the plausibility of nobody declaring victory. Consider the would-be leader failing after sending an OK message to a VOTE but before sending all the COORDINATOR messages. If it does not recover before the set timeout on lower ID processes, one of them will become leader. If the failed process recovers in time, it simply sends a COORDINATOR message once it is back up.

3. What assumptions did you make about the setup and execution in your proof? For example, you may be assuming that the VOTE card arrives in one piece.

   For our proofs and algorithm, we made the assumptions:
   - There is always at least one member of the group alive
   - Roundtable overlay (everyone knows everyone else)
   - Everyone has a unique process id and there is an ordering on them
   - Messages are reliable and in order for each connection
   - Failures can be detected
   - Fail-recover failure mode

What is the cost and latency of this algorithm (where n is the number of nodes and we are measuring one election sequence)? Remember to think about worst, average, and best cases, if they are different.

4. Number of messages?

   Cost: For the worst case, the lowest ID node starts an election. Then she sends n-1 VOTE messages, the next higher ID node sends n-2, etc., then there are the OK replies, and also n-1 COORDINATOR messages.
   - VOTE messages: (n-1) + (n-2) + (n-3) …..+ 1 = (n-1)n/2
   - OK replies: similarly, (n-1)n/2
   - COORDINATOR Messages: n-1
   - Total messages: (n-1)n/2 + (n-1)n/2 + (n-1) = (n-1)(n+1)

   So (n-1)(n+1), which is in O(n^2). The average case will be similar with O(n^2) messages. Best case is if the new leader is the one noticing the old leader's absence in which case there is one VOTE message (no OK) and n - 1 COORDINATOR and messages, n, so O(n).

5. Elapsed time? The way to think about this is to assume maximum concurrency and measure time as the number of sequential (dependent) messages that have to be sent in the system, i.e., think as though there were a global clock ticking away and each message was sent out on one click as soon as it could be and then received on the next.

   Latency: We'll assume that all the outgoing messages can be sent concurrently. The timing is that the soon-to-be new leader will:
   - get a VOTE message in one timestep from whoever notices that there is no current leader,
   - then if not the highest id, she will have to wait the requisite timeout, (the reply of OK to the VOTE is subsumed into this timeout—we are assuming timeout is greater than time for a message)
   - then declare herself leader and send out COORDINATOR messages to all the others concurrently.

   So worst case latency starting from someone noticing the absence of a leader is *mtime + timeout + mtime, O(1)*

and best case is

*2*mtime, also O(1),*

so we can assume average case is also O(1).