CALIFORNIA INSTITUTE OF TECHNOLOGY
Computing and Mathematical Sciences

CS 142

R. Murray, M. Chandy
Fall 2019

Homework Set #8

Issued: 27 Nov 2019
Due: 6 Dec 2019

Note: Please keep track of the number of hours that you spent on this homework set (including reading) and enter this when you turn in your homework on Moodle.

1. In the "Paxos made simple" paper by Lamport, why is maintaining the invariance of $P2^c$ enough to satisfy $P2^b$? Detailed proofs are not required, but you should provide enough information to capture the key ideas behind your answer.

2. The Paxos algorithm uses the concept of a set of a majority of acceptors. If a system has 7 acceptors, then a majority set is any set with 4 or more acceptors. Suppose that during the execution of Paxos, 2 of the acceptors fail—i.e., halt execution and no longer send messages. After 2 acceptors fail, the system has only 5 working acceptors and so the actual working majority is only 3. The algorithm determines the majority set in terms of the *initial* number of acceptors, *not in terms of the currently working acceptors.* So, if the system starts with 7 acceptors, the majority set is always a set with 4 or more acceptors, regardless of the number of acceptors that may have ceased execution.

   Does the Paxos algorithm work regardless of the number of acceptors that fail for the case in which each proposer sends messages (i.e. prepare and accept messages) to all acceptors. So, if the system has 7 acceptors and 2 of them fail, each proposer sends messages to all 7 acceptors (even though the 2 failed acceptors won't receive the messages.

   Give a rationale for your answer.

   Note this question asks only about safety, and not progress.

3. Suppose that we modify the Paxos algorithm for proposer P as follows:

   State: (P.t, P.value) Initially (-1, x)

   Start timer

   While not timed_out:
   choose t greater than P.t and set P.t = t

   Phase 1:

   1. send prepare(P.t) to all acceptors
   2. wait for promise(t, value) replies from (at least) a majority of acceptors where t == P.t.
   3. If value is not None (or '_') for one or more of these promise messages then set P.value to the majority value in these promise messages.

   Phase 2:

   4. send request(P.t, P.value) to all acceptors.

5. Wait for accepted(t, value) replies from majority of acceptors where (t, value) == (P.t, P.value)

And for acceptor A:

State: (A.t, A.value) Initially (None, None)

Start timer

While not timed_out:

upon receiving prepare(t)

if t >= A.t:

A.t = t

reply with promise(t, A.value)

upon receiving request(t, value)

if t >= A.t:

A.t = t

accepted(t, value)

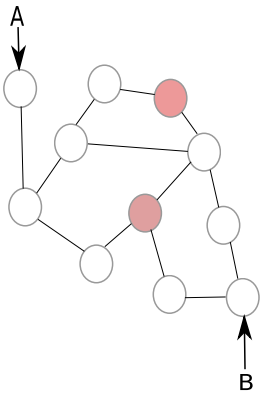
This modification allows us to use a simpler algorithm. Instead of keeping both prepare_t and accept_t, the acceptor just uses a single variable t.

Is the modified algorithm correct? Sketch a proof or give a clear counter example.

4. [Bitcoin] Double-spend attacks try to spend the same input currency more than once. The idea is that the attacked broadcasts multiple transactions to the network and by the time the victim realizes that his transaction is invalid, he would have already delivered the service.

   (a) Consider the network below. Minnie tries to launch a double-spend attack by broadcasting two transactions to the network at the two indicated points A and B. Assume there is negligible latency in message passing, compared to the time taken for mining and that each link in the network induces the same delay in the transfer of messages. If each node has the same computation power and that no new blocks are mined till the transactions are fully propagated, what is the probability for each of the transactions to be accepted?

   (b) Assuming the nodes in red have twice as much computational power as the uncolored nodes, what is the probability for the transactions to be confirmed?


A
B