DAIRON ANDRÉS BENITES ALDAZ
daba2@kth.se

DD2443 Pardis22
Exercises for lecture 13

Mads Dam, Karl Palmskog

Exercise 1 Three phase commit requires multiple processes to simultaneously commit or to simultaneously abort a transaction. We assume that processes can crash at any time, and there are no Byzantine processes. The network is asynchronous, and messages are never lost or reordered in transmission. A process may crash in the middle of a broadcast. This may cause some messages to be correctly delivered at their destination, and some messages to be lost. A process broadcasting requests can receive replies while other processes have yet to receive the request.

Three phase commit involves the following steps:

1. TM/coordinator sends *ready* to all participants
2. Cohorts answer *yes* or *no*
3. TM sends *prepare* or *abort*
4. Cohorts answer *ack* and aborts, if applicable
5. TM aborts or sends *commit*
6. Cohorts commit and answer *ackCommit*
7. TM commits

a) In all steps but one, processes wait for a message before the step canbe executed. Write down in which steps which processes have to wait for what messages.

b) If a message does not arrive because sender has crashed, the waitingprocess times out. For each of the steps where processes wait, how should be processes react to a timeout? Note that if there is any uncertainty about the outcome, no process can decide to commit. Hint: Can the processes safely abort, or are they forced to commit? Must they elect a new coordinator?

DAIRON ANDRÉS BENITES ALDAZ
daba2@kth.se

```
function choose(Node N1, Node N2, Timeout t, Value x, ReqNumber n)
 Send prepare(x,n) to nodes N1, N2
 Wait t seconds
 If within these t seconds, either N1 or N2 has not replied then
   choose(N1, N2, t, x, n + 2)
 Let (y, m) be the received proposal with the largest request number
 if m == 0 then
   u := x
 else
   u := y
 Send propose(u, n) to N1, N2
 Wait t seconds
 if within these t seconds, either N1 or N2 has not replied with
 ack(u, n) then
   choose(N1, N2, t, x, n + 2)
 Print("value is chosen" + u)
```

Figure 1: Proposer implementation for exercise 2

Exercise 2 We run Paxos on a set of 3 nodes A, B, C, that act as acceptors. We assume there are two nodes Q, R that act as proposers. The implementation of the acceptors is as in the slides. The two proposers use the implementation given in figure 1. Draw a timeline containing all transmitted messages if a user invokes choose(A,B,1,22,1) on Q at time T_0 and choose(B,C,2,33,2) on R at time $T_0 + 0.5s$. Assume that node processing time is zero and that all messages arrive within $0.5s$.

Exercise 3

a) Assume in Paxos that the register n_{max} is faulty such that n_{max} may return a value less than the most recently assigned. Can this pose a problem to the Paxos algorithm? Explain what happens in a worst case scenario.

A chosen value must be accepted by a majority of the servers, and acceptors reject older proposals. With two invariants, we know that the latest accepted proposal returned to the proposal in phase one either has already been chosen, or no value has been chosen at the moment. Either way, it's safe to propose that value. Therefore after broadcasting *n* it may not be chosen.

DAIRON ANDRÉS BENITES ALDAZ
daba2@kth.se

b) Paxos assumes there are no Byzantine failures. That is, memory failures as in a) are not supposed to happen. Under these original assumptions, can you explain what the purpose of the Paxos *prepare* step is?

In this phase, proposer tries to get a *promise* from acceptor to not accept any proposals older than his. At the same time, acceptors will inform the proposers what values have already been accepted so far.Each acceptor that receives the PREPARE message looks at the ID in the message and decide.

If the proposer receives a PROMISE response from a majority of acceptors, it now knows that a majority of them are willing to participate in this proposal. The proposer can now proceed with getting consensus. Each of these acceptors made a promise that no other proposal with a smaller number can make it to consensus.