ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

HS 2010 M. Kuhn / S. Welten / Prof. R. Wattenhofer

Distributed Systems Part II

Solution to Exercise Sheet 5

1 Three Phase Commit

a) The five steps are:

i) Step 2: Participants wait for VOTE request
ii) Step 3: Coordinator waits for votes
iii) Step 4: Participants wait for coordinators decision
iv) Step 5: Coordinator waits for acknowledgments
v) Step 6: Participants wait for COMMIT

b) The reactions are:

i) no one has yet decided on commit, so abort
ii) no one has yet decided on commit, so abort
iii) some processes already know the coordinators decision, some do not. Some might already aborted, or have sent ACK. The processes have to elect a new coordinator, the new coordinator has to find out what the decision was and resend the decision to all the processes which do not already know it.
iv) a crashed processes did not send an ACK. But the transaction can continue because the correct processes are prepared to commit.
v) the next message must be COMMIT, but committing would violate the non-blocking property as some processes may not yet have received PREPARE. So the processes first have to elect an new coordinator, which has to find out about the decision and inform uninformed processes about it.

c) Assume there are three processes p_1, p_2 and p_3. Process p_1 is the coordinator. All the processes vote `YES` on the transaction. p_1 receives and processes the votes, but p_2 and p_3 detach from p_1 before receiving the `PREPARE` message sent by p_1.

p_2 is elected as new coordinator. It sees both p_2 and p_3 are undecided. Not knowing their state p_2 decides to abort, sending the `ABORT` message. p_3 receives the message, then detaches from p_2. If now p_1 and p_3 become connected they cannot progress: p_1 already decided to commit, p_3 already decided to abort.

2 Paxos Timeline

The timeline consists of two concurrent processes, one on the client Q and one on the client R. In Figure 1 you can see how both clients prepare and propose their values at first, but only the value of client Q gets accepted:

- $T_0 + 0.0$: Q sends a prepare(22,1). As A and B have never accepted a value they reply with acc(ø,0).
- $T_0 + 0.5$: R sends a prepare(33,2). As B and C have never accepted a value they reply with acc(ø,0).
- $T_0 + 1.0$: Q sends a propose(22,1). This is acknowledged by A with ack(22,1) because its $n_{max} = 0$. B does not reply as its value $n_{max} = 2$.
- $T_0 + 2.0$: Q sends a prepare(22,3). As B has never accepted a value it replies with acc(ø,0). A returns the latest accepted value: acc(22,1).
- $T_0 + 2.5$: R sends a propose(33,2). This is acknowledged by C with ack(33,2). B does not reply as its value n_{max} is 3.
- $T_0 + 3.0$: Q sends a propose(22,3). This is acknowledged by A and B with ack(22,3).
- $T_0 + 4.5$: R sends a prepare(33,4). C sends back its latest accepted value acc(33,2). B also sends back its latest accepted value acc(22,3)
- $T_0 + 6.5$: R sends a propose(22,4) (It took the newest value from the prepare phase). Both clients B and C reply with an ack(22,4). All clients have accepted the same value. This means we have achieved consensus.

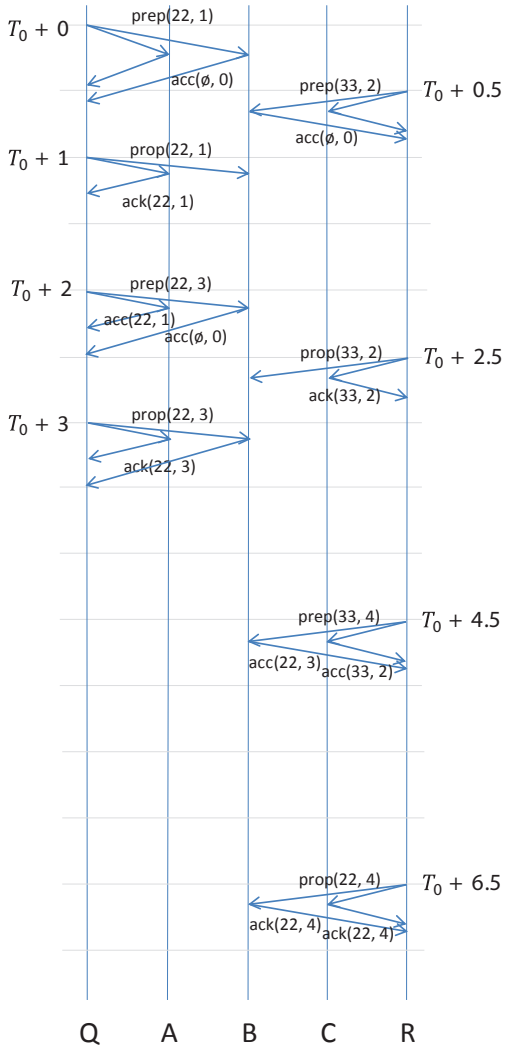

Figure 1: The timeline of the two clients running the given paxos-proposer-program with different timeout values

3 Paxos Acceptors

a) Figure 2 shows an example of how a byzantine client can lead to a failure of the Paxos protocol (i.e. why Paxos is not resilient against byzatine failures):

1. The red proposer sends a prepare with value 1.
2. The red acceptors (incl. the byzantine) send an ack(ø,0) back.
3. The blue proposer sends a prepare with its value.
4. The blue acceptors (incl. the byzantine) send an ack(ø,0) back. We assume that a read on the faulty register of the byzantine node returned $n_{max} = 0$.
5. The red proposer sends a propose with value 1.
6. The red acceptors (incl. the byzantine) send an ack(1,3) back. We assume that a read on the faulty register of the byzantine node returned $n_{max} = 3$.
7. The blue proposer sends a propose with value 1.

8. The blue acceptors (incl. the byzantine) send an ack(2,4) back. We assume that a read on the faulty register of the byzantine node returned $n_{max} = 4$.

At the end of these 8 steps the red proposer thinks that a majority has accepted the value 1 and the blue proposer thinks that a majority has accepted value 2. Both proposers will start to disseminate their value as each of them thinks that they have achieved consensus.

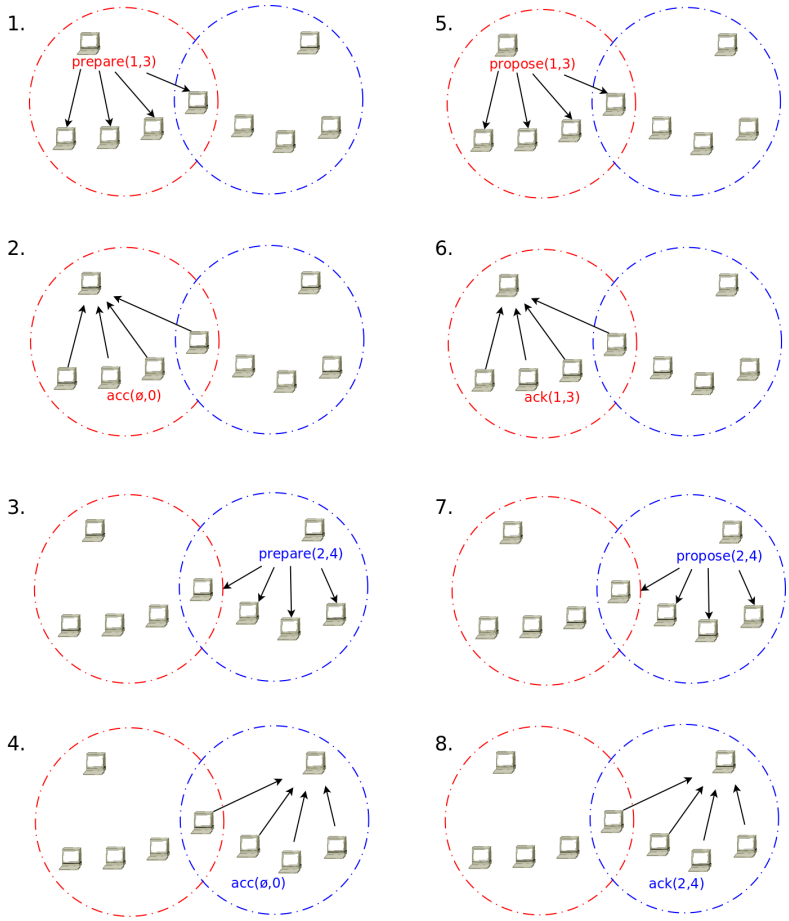

Figure 2: How a byzantine client can lead to different values that are accepted by a majority.

b) The prepare step allows the proposer and the acceptor to agree on a lower bound of the proposal number that will be accepted. By sending an ack(x,y) message, the acceptor guarantees the proposer that it will never accept a proposed value that has a smaller timestamp than the one in the prepare message of the proposer.