Universität Freiburg
Institut für Informatik
Prof. Dr. P. Fischer/Prof. Dr. F. Kuhn

Georges-Köhler Allee, Geb. 51
D-79110 Freiburg
peter.fischer@informatik.uni-freiburg.de

Exercises
Distributed Systemes: Part 2
Summer Term 2015
14.7.2015
Solution Proposal

5. Exercise sheet: Paxos and Concurrency Control

Exercise 1
You are running a set of three numbered processes $p_i, i \in \{1;2;3\}$. They use Paxos to exchange values consistently over a network. Each of the processes can fail for some time, but will return eventually, never more than one process will fail at the same time. Provide an execution of the following setups and count the number of messages and round-trip times.

a) p1 wants to publish a value and no failures occur.

b) p1 uses and optimized version , where it only communicates with a quorum, again without failures!

c) Another iteration of b. optimized by using Multi-Paxos (System is already in a steady state).

d) The leader crashes in Multi-Paxos. What will happen?

e) A second proposer shows up in Multi-Paxos. What will happen?

f) p3 wants to publish a value and a failure occurs at the worst possible time. Maximise the number of messages!

g) An acceptor loses its memory after having made a promise. What could happen?

Solution:

a) The protocols goes through the usual two rounds of proposing/promising and accepting, requiring 4 exchanges and 8 messages (see Figure 1). P_1 also acts as acceptor, so that the majority is 2 acceptors overall.

b) Since a quorum of 2 acceptors is sufficient, communicating successfully with only another node (in our example P_2) is sufficient. As a results, only 4 messages are needed, but the number of exchanges stays the same(see Figure 3).

c) In Multi-Paxos, a proposer which has achieved promises from the acceptors has become the "leader" and can retain this role. As a result, any further proposal do not have to go through the first phase, and only the second phase needs to be run (see Figure **??**). Therefore, only 2 exchanges are needed and, as a consequence, only 2 messages.

d) After some waiting time, another node will start the first stage of normal/not-multi Paxos. If the proposal number of this new proposer is bigger than the proposal number of the "dead" proposer, it will succeed in getting promises and become the leader. If the proposal number is too low, it will have later attempts (with higher proposal numbers) until it eventually succeeds.

e) The argument is similar to the failure cases, only now we have to active proposers. Their respective proposal numbers are compared, and depending on the values the new one may take over, the old one may keep its role or the promises/accept rounds may change between both of them.

f) The worst cases would trigger a race by two proposers to complete their proposals. In the absence of any mitigating strategies (like randomized wait times), the protocol may be in livelock, producing an infinite number of messages.

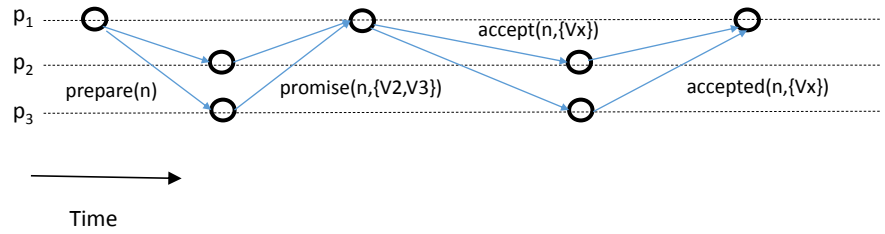


Figure 1: Protocol Flow for exercise a)

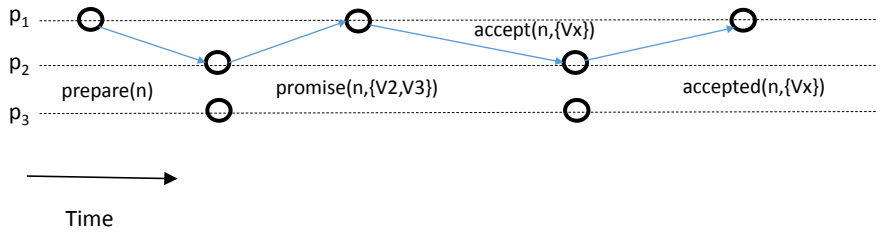


Figure 2: Protocol Flow for exercise b)

g) The memory loss will eradicate the information on the proposals it has accepted and the promises it made. As a result, it may promise and accept proposals with lower proposal numbers, possibly leading to the acceptance of proposals that would not have gained the quorum.

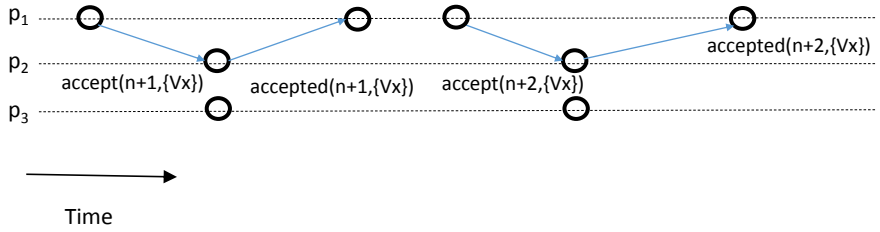


Figure 3: Protocol Flow for exercise c)

Exercise 2

Consider the following schedules.

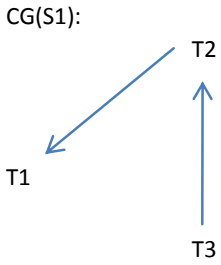
S_1: $R_3X\ R_2Y\ W_2Y\ R_1Y\ W_1Y\ R_2X\ W_2X\ R_1X\ W_1X\ W_3Z$.

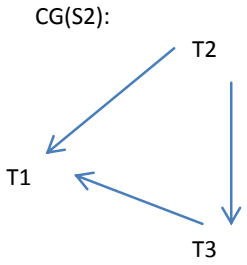
S_2: $R_3X\ R_2Y\ W_2Y\ R_1Y\ W_1Y\ R_2X\ W_2X\ R_1X\ W_1X\ W_3Y$.

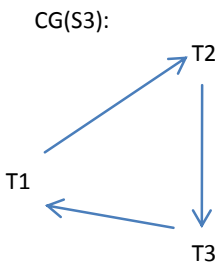
S_3: $R_1Y\ W_1Y\ R_2Y\ W_2Y\ R_2X\ W_2X\ R_3Z\ W_3X\ R_1X\ W_1X$.

For each schedule give its conflict graph. Which schedules are serializable, which are not?

Solution:







S_1 is serializable, S_2, S_3 are not serializable.

Exercise 3
Assume on a database three transactions are being executed.

a) The transactions are of the form:
$$\begin{array}{lllll} T_1: & RA & WA \\ T_2: & RA & WA \\ T_3: & RA & WA \end{array}$$

(i) How many serial schedules do exist for T_1, T_2, T_3? Give the reasons!

(ii) How many serializable schedules do exist for T_1, T_2, T_3, which are not serial ones? Give the reasons!

b) The transactions are of the form:
$$\begin{array}{lllll} T_1: & RA & WC \\ T_2: & RB & WA \\ T_3: & RC & WD \end{array}$$

(i) How many schedules do exist for T_1, T_2, T_3, which are not serializable? Give the reasons!

(ii) Applying 2-phase-locking, is it possible that all serializable schedules may occur? Give the reasons!

Solution:

(ai) 6 - all possible permutations

(aii) none - every non-serial schedule either contains $R_i...R_j...W_i...W_j...$ or $R_i...R_j...W_j...W_i...$ where $i \neq j$. Both groups of actions lead to cycles in the conflict graph

bi) None - There only a conflict between T_1 and T_2 as well T_1 and T_3, respectively, but no conflict between T_2 and T_3. As a result, no schedule could ever generate a cycle in the conflict graph.

(bii) No - the following schedule provides a counterexample:

$$S = R_1A\ R_2B\ W_2A\ R_3C\ W_3D\ W_1C$$

In order to generate the prefix $R_1A\ R_2B\ W_2A$ of the schedule S within the constraints of 2PL, U_1A has to occur before W_2A. Because of W_1C and the constraints of 2P L_1C has to precede U_1A. Furthermore U_1C has to follow W_1C. This yields (without restricting generality) the following order of Lock- and Unlock operations in the schedule

$$S = L_1A\ R_1A\ L_2B\ R_2B\ L_1C\ U_1A\ W_2A\ R_3C\ W_3D\ W_1C\ U_1C$$

With this order under 2PL, T_3 cannot be executed.