Universität Freiburg
Institut für Informatik
Prof. Dr. P. Fischer

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

Exercises
Distributed Systemes: Part 2
Summer Term 2014
17.7.2014
Solution Proposal

5. Exercise sheet: Distributed Concurrency Control and Recovery

Exercise 1
Consider the following local schedules:

- $S_1: \quad R_1A \quad W_1A \quad R_2A \quad W_2A$
  $S_2: \quad R_2B \quad W_2B \quad R_1B \quad W_1B$
- $S_1: \quad R_1A \quad W_2A$
  $S_2: \quad R_3B \quad W_1B \quad R_2C \quad W_3C$
- $S_1: \quad R_1A \quad R_3A \quad R_3B \quad W_3A \quad W_3B \quad R_2B$
  $S_2: \quad R_4D \quad W_4D \quad R_1D \quad R_2C \quad R_4C \quad W_4C$
- $S_1: \quad W_1A \quad c_1 \quad R_3A \quad R_3B \quad c_3 \quad W_2B \quad c_2$
  $S_2: \quad W_2C \quad c_2 \quad R_4C \quad R_4D \quad c_4 \quad W_1D \quad c_1$

(1) Verify whether or not the schedules are serializable.
(2) Demonstrate that by applying Distributed 2PL (Timestamp Protocol) the non-serializable schedules could not have occured.
(3) Check whether or not the schedules are rigourous and commit-deferred.
(4) Demonstrate that by applying a Ticket-based concurrency control the not serializable schedules could not have occured.

Solution:

(1)
- locally yes, globally no, S_1: $T_1 \to T_2$, S_2: $T_2 \to T_1$
- locally yes, globally no, S_1: $T_1 \to T_2$, S_2: $T_2 \to T_3 \to T_1$
- locally yes, globally no, S_1: $T_1 \to T_3 \to T_2$, S_2: $T_2 \to T_4 \to T_1$
- locally yes, globally no, S_1: $T_1 \to T_3 \to T_2$, S_2: $T_2 \to T_4 \to T_1$

(2) Distributed 2PL: If a local transaction reaches the point when it would start unlocking, it would ask the other sites running the same transaction if they also reached this point. If not, it would have to wait
- not possible, since either transaction cannot make progress after the first two steps
- not possible, since $R_1 1$ and $R_3 2$ cannot make progress after the first step
- not possible, since $R_1 1$ and $R_4 2$ cannot make progress after the first and second step, respectively
- local commit violate global 2 PL protocols if they went through. At c_1, the lock on A cannot be released since $T_1 2$ has not yet claimed all its locks

Timestamp Protocol: Abort transactions, if a conflicting access is performed with a later timestamp. Without restricting generality, we always assume that S_1 start is transactions earlier than S_2
- $TS_1 < TS_2$, so R_1B performs a read on B which has been written to by a "later" transaction before (W_2B)
- $TS_1 < TS_3$, so W_1B performs a write on B which has been read to by a "later" transaction before (R_3B)

- $TS_1 < TS_4 < TS_3$, so R_1D performs a read on D which has been written to by a "later" transaction before (W_4D)
- $TS_1 < TS_2 < TS_3 < TS_4$, so W_2B performs a write on B which has been read by a "later" transaction before (R_3B). The same problem occurs for W_1D and R_4D.

(3) The last case is easiest: The schedules are rigorous since all a commit happens before any conflicting operation. It is not commit-deferred since e.g. $T_1 1$ commits before $T_1 2$.

In first three cases, no commit is specified. We therefore have the options to either perform the commit at (i) the global end of a transaction or (ii) as soon as possible after the local end. (i) would make the schedules commit-deferred (by definition), but not rigorous, since conflict pairs exist before abort or commit. (ii) would make some of the schedules rigorous, but not all of them.

(4) Tickets are expressed by adding a Ticket access into the local schedules of global transactions. When "locking" the ticket (which we denote as I_j for server j) we add an explicit Read/Write Operation.

- $S_1: \; R_1I_1 \; W_1I_1 \; R_1A \; W_1A \; R_2I_1 \; W_2I_1 \; R_2A \; W_2A$
  $S_2: \; R_2I_2 \; W_2I_2 \; R_2B \; W_2B \; R_1I_2 \; W_1I_2 \; R_1B \; W_1B$
  In this case, no local detection is possible, but the access to I_1 and I_2 happens in different order. Using dependency graph on the tickets we can detect the cycle.
- $S_1: \; R_1I_1 \; W_1I_1 \; R_1A \; R_2I_1 \; W_2I_1 \; W_2A$
  $S_2: \; R_3B \; R_1I_2 \; W_1I_2 \; W_1B \; R_2I_2 \; W_2I_2 \; R_2C \; W_3C$
  Tickets introduce T_1T_2 order on S_2 which makes the conflict explicit and locally detectable at S_2, since the execution without ticket yields the order $T_2 \to T_3 \to T_1$
- Like in the previous case, we ticket introduce T_1T_2 order on S_2, making the conflict locally detectable.
- Like in the first case, no local detection is possible, but a depedency graph on the the tickets detects the conflict.

Exercise 2

Think about a distributed database management system that runs 2PC and how it deals with failures

(a) What happens when a participant votes abort in phase 1? Use the state transition graphs shown in the slides to explain your answer.

(b) What happens when a participant fails in phase 2 without sending anything to the coordinator (e.g. a kernel freeze)? Again use the state transition graphs to further explain your answer.

(c) How would using 3PC change the situation of b)

Solution:

(a) In phase 1, no global decision has been made; the coordinator is requesting the local decisions. An abort decision of a participant has two consequences: 1) Since at least one participant will not commit, the global outcome will be an abort. 2) Since no global commit can happen, the participant that voted abort can directly go to abort, sidestepping the prepare phase.

(b) Since the participant has voted commit and received a global commit, it is obliged to actually perform a commit. Given that it is hanging, no more progress can be made. A timeout at the coordinator will detect the problem at this participant, but there is no way to resolve it without manual intervention - in particular the whole commit protocol is blocked.

(c) 3PC splits the commit part into two stages in order to reduce the uncertainty period. If the participant fails at PRECOMMIT, it has already been informed that the commit will happen, but does not know if the others have been informed. At this stage, it can inquire any other participant if it knows about the outcome. The coordinator does not have to worry about the failed node, since it already was informed about the outcome.

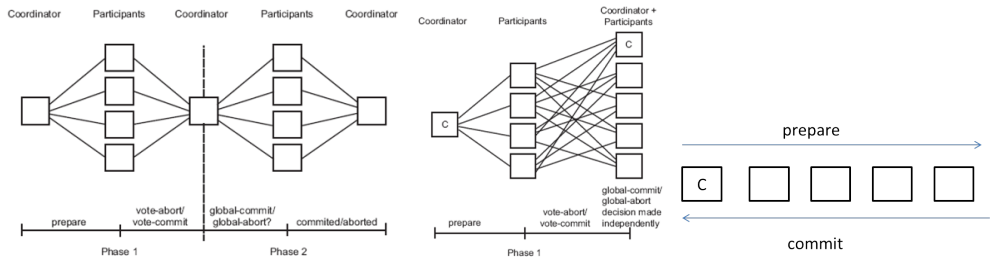
Exercise 3

(a) Describe the communication topology of centralized, decentralized and linear 2PC.

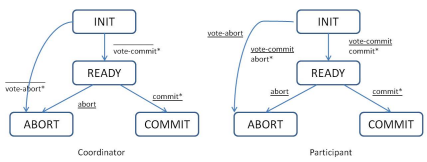
(b) Give the state diagrams of decentralized 2PC, in analogy to the state diagramm of centralized 2PC.

Solution:

(a)



(b)



Exercise 4

Characterize centralized 2PC and linear 2PC with respect to

(1) message and time complexity,

(2) possibilities of processes to become uncertain.

Solution:

(a) Centralized 2PC requires 3n messages (if ACKs are not considered), and takes 3 rounds. Linear 2PC requires 2n messages, but needs also 2n rounds.

(b) For centralized 2PC, all participants are uncertain (in the same way) from the moment they cast their vote until they receive the decision from the coordinator. In linear 2PC, period of uncertaintly shrinks with the distance from the coordinator, the rightmost participant is actually never uncertain, whereas the leftmost participant is uncertain from 2n-2 rounds.