Week 9 Transaction processing

1. Are there any serializable but not conflict-serializable schedules that strict 2PL can process without aborting transactions? If yes, give an example and briefly explain how the algorithm processes such a schedule. If no, explain why.

   Answer: no, by definition. 2PL accepts exactly the conflict-serializable schedules.

2. Some of the CC protocols 2PL and MVCC can suffer from cascading aborts. Which? Give examples. How to prevent cascading aborts?

   Answer: Both. Examples for 2PL in textbook and for MVCC in the slides.

   To prevent cascading aborts, use strict 2PL, i.e., all locks are released while transaction completes. For MVCC, cascading aborts can be avoided by making the read operation of transaction Ti wait until the commit of transaction Tj, which wrote the version Ti is reading. Note: This version of MVCC is pretty much same as strict 2PL with silly bells and whistles.

3. Consider this schedule.

   T1 W(B) W(A)

   T2 R(A) W(A) R(B) W(B)

   T3 R(A) W(A)

   T4 R(A) R(B) W(C)

   a) Execute this schedule under (serialisable) MVCC. Are all these transactions able to commit? If any transactions need to abort, when is the earliest time the system can know?

   b) Construct the dependency graph, write the serialized schedule in 2PL.

   Answer:

   a) Run of MVCC:

   Initially, we have object versions A@0(RTS=0, WTS=0), B@0(RTS=0, WTS=0), C@0(RTS=0, WTS=0)

   On op T1:W(B), create new version B@1(RTS=1, WTS=1)

   On op T1: W(A) , create new version A@1(RTS=1, WTS=1)

   On op T2: R(A), reads version A@1 and sets its RTS:=2

   On op T2: W(A), create new version A@2(RTS=2, WTS=2)

   On op T3: R(A), reads version A@2 and sets its RTS:=3

   On op T3: W(A), create new version A@3(RTS=3, WTS=3)

   On op T2: R(B), reads version B@1 and sets is RTS:=2

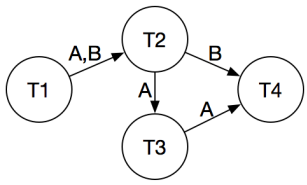
   On op T2: W(B), create new version B@2(RTS=2, WTS=2)

   On op T4: R(A), reads version A@3 and sets its RTS:=4

On op T4: R(B): reads version B@2 and sets is RTS:=4

On op T4: W(C), create new version C@4(RTS=4, WTS=4)

b) all Xacts can commit



T1 before T2 before T3 before T4.

4. Execute this schedule under Multiversion CC:

   T1: R(A) R(A) W(A)

   T2: W(A) W(A)

   Answer:

   Initially, we have version A@0(RTS=0, WTS=0).

   On op T1:R(A), read A@0 and set its RTS=1

   On op T2:W(A), create A@2(RTS=2, WTS=2)

   On third op in schedule, T1:R(A), read A@0

   On op T2:W(A), write to A@2

   On op T1:W(A), create A@1(RTS=1, WTS=1)

5. Execute this schedule under MVCC:

   T1: W(A)

   T2: R(A)

   Answer:

   Initially, we have version A@0(RTS=0, WTS=0).

   On op T2:R(A), we read A@0 and set its RTS to 2

   On op T1:W(A), we have a problem. We cannot write because whatever we write now should have been read in the previous operation (in T2), but we didn't do that. We abort T1 and restart it as T3. Now the schedule looks like

   T2: R(A)

   T3: W(A)

   Now, on op T3:W(A), we create new version A@3(RTS=3, WTS=3).

6. Snapshot isolation: A hospital implements a database system to manage the doctor responsibility sheet. It is known that at least one doctor has to be on duty for a particular date.

   A appointment algorithm is implemented as, application will

   - Receive user' s request to reserve doctor (X) on a date D.
   - Determine if the number of doctor status == on duty is larger than 2 on D, if yes, proceed, if not, reject user' s request.
   - Make the status to reserve and commit the changes.

   We know the backend database adopt the snapshot isolation, and given the current database as in table.

Doctor	**Shift**	**Status**
House	12 June	On duty
Grey	12 June	On duty

   Please answer the following questions, with explanation of transaction schedule.

   E.g. you can use the abstraction X = Status where Doctor = House and Shift = 12 June.

   a) If there are two persons making the appointment of Dr. House on 12 June, what will happen?

Let X = Status where Doctor = House and Shift = 12 June

Let M = Count(Status == on duty WHERE Shift = 12 June)

T1: R(X) -> R(M) -> W(X) -> C

T2: R(X) -> R(M) -> W(X).-> C (aborted due to X is modified by T1)

Assuming T1 executes slightly earlier, and the program will be successfully executed and committed. However, during commit T2, it will be aborted due to X is modified by T1.

   b) What if person A want to book Dr. House, and person B want Dr. Grey on June 12?

Similarly, Let X = Status where Doctor = House and Shift = 12 June

Let Y = Status where Doctor = Grey and Shift = 12 June

T1: R(X) -> R(M) -> W(X) -> C

T2: R(Y) -> R(M) -> W(Y) -> C

T1 T2 will both executed because of the write skew anomaly. Since snapshot isolation have the read version only. R(M) checking the doctor' s status will pass without problem. And since W(X) and W(Y) goes to different place, both T1 and T2 will be committed, however, it does violate the rule, "at least one doctor should be on duty" .

7. Which of the following schedules are conflict-serializable? For Conflict-serializable schedules, provide a serialization order.

   a) T1:R(X), T2:R(X), T1:W(X), T2:W(X)

   b) T1:W(X), T2:R(Y), T1:R(Y), T2:R(X)

   c) T1:R(X), T2:R(Y), T3:W(X), T2:R(X), T1:R(Y)

d) T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y)

Answer:

a) not conflict-serializable.

b) conflict-serializable, T1 before T2

c) conflict-serializable, T1 before T3 before T2

d) not conflict-serializable

8. Deadlocks. Consider the following two sequences

S1: T1:R(x), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:C, T2:C, T3:C

S2: T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:C, T2:C, T3:C

Assume that the timestamp of transaction Ti is i. For lock-based concurrency control mechanisms, add lock and unlock requests to the previous sequence of actions as per the locking protocol. The DBMS processes actions in the order shown. If a transaction is blocked, assume that all its actions are queued until it is resumed; the DBMS continues with the next action (according to the listed sequence) of an unblocked transaction.

a) Strict 2PL with deadlock detection, (show the waits-for graph)

b) Multiversion concurrency control

Answer:

a) *In deadlock detection, transactions are allowed to wait, they are not aborted until a deadlock has been detected.*

S1. T1 gets a shared-lock S(X), T2 blocks waiting for an exclusive lock X(X); T3 got X(Y); T1 waits for X(Y);

T3 finishes with release of all locks, (which give T1 the chance to obtain X(Y).

T1 wakes up, get X(Y), and finish all and release S(X)

T2 obtain X(Y), X(X) and finishes.

S2: Deadlock with T1 waits for T2, T2 waits for T1.

b) *Multiversion CC (Note: there are never deadlocks in MVCC, but livelocks where transactions repeatedly cause each other to abort are possible)*

S1:

On op T1:R(X), read X@0 and set its RTS to 1
On op T2:W(X), RTS(X@0) = 1 < 2, so create X@2(RTS=2,WTS=2)
On op T2:W(Y), RTS(Y@0) = 0 < 2, so create Y@2(RTS=2,WTS=2)
On op T3:W(Y), RTS(Y@2) = 2 < 3, so create Y@3(RTS=3,WTS=3)
On op T1:W(Y), RTS(Y@0) = 0 < 1, so create Y@1(RTS=1,WTS=1)

S2: same as above with second and third lines swapped.