CS143: Homework 5

Problem A: T1 is the oldest transaction, and T3 is the youngest.

T1	T2	T3
write D		
	write C	
read C		
		read C
		write A
	read A	
write A		

1. Is this first schedule conflict-serializable[1]?

Answer: We have that T2 $\rightarrow$ T3 on C and T3 $\rightarrow$ T2 on A. The directed cycle T3 $\leftrightarrows$T2 tells us that the schedule is not confict-serializable.

Now assume that the transaction manager uses a 2PL protocol where each exclusive/shared lock is set just before it is needed for the write/read action.

2. If we do not use any deadlock prevention strategy, will the resulting transactions (i) complete, or (ii) deadlock[1]? If your answer is (i), show a completed schedule;

Answer: if it is (ii) show the schedule up to the deadlock. So T2 sets a lock-X C just before write C, and because of 2PL it must keep until it is done with its read A. Thus when T1 does lock-S C it stops with a wait-for arc T1 $\rightarrow$ T2. Ditto for T3 that sets the wait-for arc T3 $\rightarrow$ T2. There is no cycle: thus no deadlock. T2 completes. Then T3 and T1 will complete also.

3. If we use a wait-die deadlock prevention strategy will the resulting transactions (i) complete, or (ii) deadlock[1]? If your answer is (i), show a completed schedule; if it is (ii) show the schedule up to the deadlock.

Answer: Obviously there is going to be no deadlock since Wait-Die prevents deadlocks. Here, T1 will wait for the older transaction to release C. But T3 is younger than T2; so when it requests C it will actually die. Again a correct completion sequence is T2, T1, T3.

Now consider a second schedule, as follows:

T1	T2	T3
write D		
	write C	
read C		
		write A
		read C
	read A	
write A		

[1] Justify your answer using the applicable graph

4. Is this second schedule conflict-serializable[1]?

Answer: Here too, we have that T2 → T3 on C and T3 → T2 on A. This cycle tells us that the schedule is not confict-serializable.

Now assume that the transaction manager uses a 2PL protocol where each exclusive/shared lock is set just before it is needed for the write/read action, and answer the following questions for this second schedule.

5. If we do not use any deadlock prevention strategy, will the resulting transactions (i) complete, or (ii) deadlock[1]? If your answer is (i), show a completed schedule; if it is (ii) show the schedule up to the deadlock.

Answer: So T2 sets a lock-X C just before write C, and because of 2PL it must keep until it is done with its read A. Thus when T1 does lock-S C it stops with a wait-for arc T1 → T2. Next, T3 does lock-X A, and then by requesting lock-s C it sets a wait-for arc T3 → T2. TSo fat no cycles and no deadlock. But then T2 does a lock-S A and sets a waitfor arc: T2 → T3. Thus we now have deadlock as per the directed cycle T3 $\underset{\rightarrow}{\leftarrow}$ T2 in the wait-for graph.

6. If we use a wound-wait deadlock prevention strategy will the resulting transactions (i) complete, or (ii) deadlock[1]? If your answer is (i), show a completed schedule; if it is (ii) show the schedule up to the deadlock.

Answer: Obviously there is going to be no deadlock since Wound-Wait prevents deadlocks. Here, T1 will only wait for the older transaction to release C. But T2 is younger than T1, so T1 wounds T2 (forces T2 to roll back). T3 acquired A's lock before write A, but it waits for C's lock before read C (as C's lock is currently held by T1 and T1 is older than T3). When T1 tries to acquire the A's lock before write A, T1 wounds T3 (forces T3 to roll back) as T1 is older than T3. As a result, T1 will finish first and both T2 and T3 need to restart. **(The completed schedule is omit here as there may be many possibilities, but you need to write it out.)**

CS143: Fall 2017.
Homework 5, Problem B

1. Consider the following schedule:

$$w_3(A)r_1(A)c_3w_1(B)c_1r_2(B)w_2(C)r_4(B)c_2c_4$$

   (a) Is it a serial schedule?

   (b) Is the schedule conflict serializable? If so, what are all the equivalent serial schedules?

   (c) Is the schedule recoverable? If not, can we make it recoverable by moving a single commit operation to a different position?

   (d) Is the schedule cascadeless? If not, can we make it cascadeless by moving a single commit operation to a different position?

	r1(A)		w1(B)	c1					
					r2(B)	w2(C)		c2	
w3(A)		c3							
							r4(B)		c4

ANSWER:

(a) No! It's NOT a serial schedule because there's a interleaving between transactions.

(b) Yes! It's conflict serializable.

i. switch: $r_1(A), c_3$

ii. switch: $r_4(B), c_2$

iii. results: $w_3(A)c_3r_1(A)w_1(B)c_1r_2(B)w_2(C)c_2r_4(B)c_4$

(c) Yes! It's recoverable since:

i. A - $w_3(A)$ is in front of $r_1(A)$, and c_3 is in front of c_1.

ii. B - $w_1(B)$ is in front of $r_2(B)$, and c_1 is in front of c_2.
B - $w_1(B)$ is in front of $r_4(B)$, and c_1 is in front of c_4.

(d) No. We need to move c3 to the second position to make it cascadeless.