# CENG352 Written Assignment 3

Mustafa Ozan Alpay

26/06/2021

## 1 Question 1

1. recoverable
2. conflict serializable
3. conflict serializable, recoverable
4. conflict serializable, recoverable, cascadeless, strict
5. conflict serializable, recoverable

## 2 Question 2

This schedule is allowed by 2PL.

| **T1** | **T2** |
|---|---|
| L1 (A);<br>R(A)<br>W(A);<br>U1 (A); | |
| | L2 (A); L2 (B);<br>R(A)<br>W(A);<br>U2 (A);<br>R(B)<br>W(B);<br>U2 (B);<br>COMMIT |
| ABORT | |

# 3 Question 3

This schedule is allowed by strict 2PL.

| T1 | T2 |
|---|---|
| L1 (A);<br>R(A)<br>W(A); | |
| | L2 (A); DENIED |
| U1 (A);<br>ROLLBACK | |
| | GRANTED<br>R(A)<br>W(A);<br>L2 (B);<br>R(B)<br>W(B);<br>U2 (A);<br>U2 (B);<br>COMMIT |

# 4 Question 4

a. When we look at the schedules, it it clearly marked that the locks are unlocked after the commit. That gives us the hint of these schedules being strict 2PL. Since strict 2PL guarantees conflict serializability, it is conflict serializable.

b. Deadlock is possible. Let's assume the following thread schedule on a single-core CPU: (// indicates context switch)

$$X_1(A)R_1(A)//X_2(B)//W_1(A)//R_2(B)//X_1(B) \text{ DENIED } //X_2(A) \text{ DENIED } \rightarrow \text{ DEADLOCK}$$

In the case of deadlock where T1 is the older thread, the younger thread (T2) will be aborted to prevent the deadlock.

c. Since strict 2PL avoids cascading aborts, it is not possible.

d. When we look at the schedules, there is no signs of commit information. In addition to that, no unlock happens until all locks are acquired. That indicates 2PL scheduling. Since 2PL ensures conflict serializability, it is guaranteed.

e. Since the schedules do not wait until the commit or abort to release the locks, and since 2PL starts releasing the locks after having all necessary locks, deadlock cannot happen. There is a possibility of waiting the other thread to complete its job to the fullest, but deadlock will not happen. To exemplify: (// means context switch)

$$X_1(A)R_1(A)//X_2(A) \text{ DENIED } \text{ WAIT } //W_1(A)X_1(B)// \text{ WAIT } //U_1(A)//X_2(A)//...$$

f. It is possible. Consider having the schedule (T2 starting first, // means context switch)

$$X(A)X(B)R(B)R(A)W(A)U(A)//X(A)R(A)W(A)//W(B) \text{ ABORT}$$

In that case, a ROLLBACK with the following order should happen: W2(B), W1(A), W2(A), which is a cascading rollback.

# 5 Question 5

a.i.

| Operation | A | | | B | | | C | | |
|---|---|---|---|---|---|---|---|---|---|
| | RTS | WTS | C | RTS | WTS | C | RTS | WTS | C |
| R1(A) | 1 | 0 | True | 0 | 0 | True | 0 | 0 | True |
| R2(B) | 1 | 0 | True | 2 | 0 | True | 0 | 0 | True |
| R3(A) | 3 | 0 | True | 2 | 0 | True | 0 | 0 | True |
| W1(A) - ROLLBACK | 3 | 0 | True | 2 | 0 | True | 0 | 0 | True |
| R2(C) | 3 | 0 | True | 2 | 0 | True | 2 | 0 | True |
| W3(B) | 3 | 0 | True | 2 | 3 | False | 2 | 0 | True |
| W2(C) | 3 | 0 | True | 2 | 3 | False | 2 | 2 | False |
| C1 - NOTHING TO COMMIT | 3 | 0 | True | 2 | 3 | False | 2 | 2 | False |
| R2(A) | 3 | 0 | True | 2 | 3 | False | 2 | 2 | False |
| W3(C) | 3 | 0 | True | 2 | 3 | False | 2 | 3 | False |
| C3 | 3 | 0 | True | 2 | 3 | True | 2 | 3 | True |
| W2(B) - IGNORED | 3 | 0 | True | 2 | 3 | True | 2 | 3 | True |
| C2 - NOTHING TO COMMIT | 3 | 0 | True | 2 | 3 | True | 2 | 3 | True |

Table 1: TS(T1) = 1, TS(T2) = 2, TS(T3) = 3

a.ii.

| Operation | A | | | B | | | C | | |
|---|---|---|---|---|---|---|---|---|---|
| | RTS | WTS | C | RTS | WTS | C | RTS | WTS | C |
| R1(A) | 2 | 0 | True | 0 | 0 | True | 0 | 0 | True |
| R2(B) | 2 | 0 | True | 3 | 0 | True | 0 | 0 | True |
| R3(A) | 2 | 0 | True | 3 | 0 | True | 0 | 0 | True |
| W1(A) | 2 | 2 | False | 3 | 0 | True | 0 | 0 | True |
| R2(C) | 2 | 2 | False | 3 | 0 | True | 3 | 0 | True |
| W3(B) - ROLLBACK | 2 | 2 | False | 3 | 0 | True | 3 | 0 | True |
| W2(C) | 2 | 2 | False | 3 | 0 | True | 3 | 3 | False |
| C1 | 2 | 2 | True | 3 | 0 | True | 3 | 3 | False |
| R2(A) | 3 | 2 | True | 3 | 0 | True | 3 | 3 | False |
| W3(C) - ROLLBACK | 3 | 2 | True | 3 | 0 | True | 3 | 3 | False |
| C3 - NOTHING TO COMMIT | 3 | 2 | True | 3 | 0 | True | 3 | 3 | False |
| W2(B) | 3 | 2 | True | 3 | 3 | False | 3 | 3 | False |
| C2 | 3 | 2 | True | 3 | 3 | True | 3 | 3 | True |

Table 2: TS(T1) = 2, TS(T2) = 3, TS(T3) = 1

b. The commit bit tells us that if it is True, then the transaction with the highest timestamp that wrote X committed. That allows us to keep track of the writes and allows recoverable schedules. Without the commit bit, we might never know if the write is committed or not, therefore we might read some garbage value that can be rolled back (dirty bit).