Theory Assignment 2

Xuanlang Zhao(kxs806), Runzhuo Li(qmc887), Yaokun Li(xnf483)

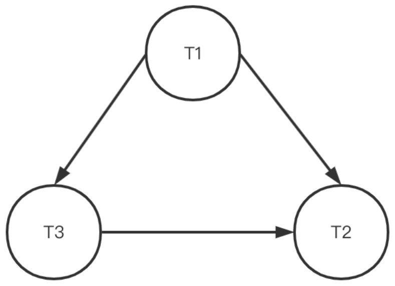
December 22, 2022

Contents

1 Question 1: Concurrency Control Concepts

1.1

T1	T2	T3
W(A)		
		W(A)
	W(A)	
C		
	C	
		C



As T1, T3 unlock A after operations, it reveals that the schedule follows 2PL instead of S2PL. As S2PL cannot release the exclusive lock during the execution.

1.2

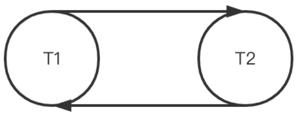
T1	T2
W(A)	
	W(B)
	C
W(B)	
C	

In terms of C2PL, it needs locks on objects from the start and in T1 transaction, B won't be released when T2 needs it. For S2PL, B can be unlocked in T2 after execution.

1.3

T1	T2
W(A)	
	W(A)
	C
W(A)	
C	

T1	T2
	W(A)
	C
W(A)	
W(A)	
C	



According to the definition, this schedule's outcome is equal to the other's, therefore, it's serializable. The graph is not acyclic, so the schedule is not conflict-serializable.

1.4

T1	T2
W(A)	
	R(A)
	W(A)
	C
C	

It's not view-serializable due to the execution is not changeable in T2 and T1. As all schedules act like in a sequence and the write operations must be operated ealier in the serialization order.

2 Question 2: More Concurrency Control

2.1

No, because in transaction Tb X get exclusively locked and for Tc, it waits for the release. It cannot release X during the execution.

2.2

No, X in Tb and Y in Tc will be locked exclusively and they won't be released before the transaction Tb and Tc get completed.

2.3

No, as C2PL requires all exclusive locks from the beginning. Tc will lock on X and Y, which results in that Ta and Tb conflict with Tc.

2.4

No, in terms of 2PL, the exclusive lock can only be unlocked after the execution of transaction. X is locked by Tc when Ta acquire a lock on it, thus, it's not 2PL.

3 Question 3: Recovery Concepts

3.1

As steal is used, we need to implement undo because if some data being written to the disk because of steal of RAM and there's a crush, we will need to undo it. As force is used, every update are written into the disk instead of RAM so we don't need to Redo.

3.2

Volatile storage loses data once it's powered off but its access time is faster and non-volatile storage can retain the data after being powered off. Stable storage can persist its data and will never be lost regarding any crush.

Volatile storage will fail when there's a crash. Non-volatile storage will survive crashes but fail to media failures, and stable storage can survive both.

3.3

Situations:1. Before saving the updated data page to disk. 2. Before commit

For situation 1, before saving the data it needs to save the log first to make sure it can be redone if a crush occurs. For situation 2, after saving all log records we can know if there are some changes being done after a crash and restore them.

These measures are taken to ensure that we can undo or redo modifications and that all committed transactions can be stored properly after a crash occurs.

4 Question 4: ARIES

4.1

Transaction table:

ID	STATUS	LASTLSN
T1	in progress	9
T3	committed	7

ID	RECLSN
P2	3
P1	4

Dirty page:

4.2

There are T1 and T3 in the transaction table and they are active in the time system crush, so

Winner set:{T2,T3}

Loser set:{T1}

4.3

Redo phase starts at the smallest recLSN in the dirty page so it is LSN3

Undo phase follow redo and ends all loser set transaction, go through LSN 9 and LSN 4, and ends at LSN3

4.4

All the logs related to pages in the dirty page need to be redone, so:

set is {3,4,5,6,9}

4.5

Undo phase will find the lastLSN of the transactions in the loser set and undo all its work, so:

Set is {9,4,3}

4.6

The following is the new log appended:

LSN	PREV_LSN	XACT_ID	TYPE	PAGE_ID
11	-	T1	abort	-
12	9	T1	CLR:Undo T1 LSN 9 (undonextLSN 4)	-
13	12	T1	CLR:Undo T1 LSN 4 (undonextLSN 3)	-
14	13	T1	CLR:Undo T1 LSN 3	-
15	14	T1	end	-

5 Question 5: More ARIES

5.1

A is 5 as we are undoing log 7 and its PREV_LSN is 5
B is 4 as we are undoing log 8 and its PREV_LSN is 4
C is NULL as we are undoing log 5 and its PREV_LSN is NULL
D is 3 as we are undoing log 4 and its PREV_LSN is 3

5.2

We start from LSN3, as T3 is ended it's erased from the transaction table, so the results are as follows:
transaction table:

ID	STATUS	LASTLSN
T1	aborted	16
T2	aborted	15

Dirty table:

PAGE	RECLSN
P3	3
P1	4
P2	6
P4	8

5.3

As T3 had ended so it doesn't need additional work. So the following is the new log added:

LSN	PREV_LSN	XACT_ID	TYPE	PAGE_ID	UNDONEXTLSN
17	15	T2	end	-	-
18	16	T1	CLR	P3	-
19	18	T1	end	-	-

5.4

5.4.1 a

Commit log records should trigger writes to stable storage. If update triggers then it would increase latency, if commit triggers it will be more efficient and it's very easy to redo.

5.4.2 b

We think all information are necessary. Without any of them may not be able to do a proper recovery so we need them to achieve crush recovery operations like undo and redo.

"Q1(1): Incorrect assumption. 2PL strategies releases locks after the last op, but before commit
Q1(2): Fine
Q1(3): Fine
Q1(4): The schedule you've presented is serial, and thus view-serial

Q2(1): Good
Q2(2): Good
Q2(3): Good, as a general comment for the whole task: another way of answering the question would have been to show that the schedules are not conflict serializable
Q2(4): Yes thats the correct conflict, but in general the exclusive lock can be released at any point if the transaction doesn't need to acquire more locks.

Q3(1): Good. Please write ""crash"" and not ""crush"".
Q3(2): Good. Please write ""crash"" and not ""crush"".
Q3(3): Good. Please write ""crash"" and not ""crush"".

Q4(1): Good.
Q4(2): Good.
Q4(3): Good.
Q4(4): Good.
Q4(5): Good.
Q4(6): Nice, but you missed T3 end after the crash. What about the page IDs?

Q5(1): okay
Q5(2): okay
Q5(3): okay
Q5(4a): okay
Q5(4b): what about undo information?

Grade: 88/100"