Ludwig Huang

Author: Ludwig Huang
Date: Oct 12, 2022

CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (FALL 2021)
PROF. LIN MA AND ANDREW CROTTY

Homework #4 (by Sophie Qiu) – Solutions

Due: **Wednesday Nov 10, 2021 @ 11:59pm**

IMPORTANT:

- **Upload this PDF** with your answers to **Gradescope by 11:59pm on Wednesday Nov 10, 2021**.
- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.
- **You have to use this PDF for all of your answers.**

For your information:

- Graded out of **100** points; **4** questions total
- Rough time estimate: ≈ 1 - 2 hours (0.5 - 1 hours for each question)

Revision : 2021/11/24 03:46

Question	Points	Score
Serializability and 2PL	18	18
Deadlock Detection and Prevention	42	35
Hierarchical Locking	20	16
Optimistic Concurrency Control	20	8
Total:	100	77

Question 1: Serializability and 2PL............................[18 points]

(a) Yes/No questions:

i. **[2 points]** A conflict serializable schedule need not always be view serializable.
☐ Yes ■ **No**

ii. **[2 points]** There could be schedules under 2PL (not rigorous) that are not serializable.
☐ Yes ■ **No**

iii. **[2 points]** A view serializable schedule may contain a cycle in its precedence graph.
■ **Yes** ☐ No

iv. **[2 points]** It is not possible to have a deadlock in rigorous 2PL.
☐ Yes ■ **No**

v. **[2 points]** You will never have unrepeatable reads in rigorous 2PL.
■ **Yes** ☐ No

Grading info: -2 for each incorrect answer

(b) Serializability:
Consider the schedule given below in Table 1. R(·) and W(·) stand for 'Read' and 'Write', respectively.

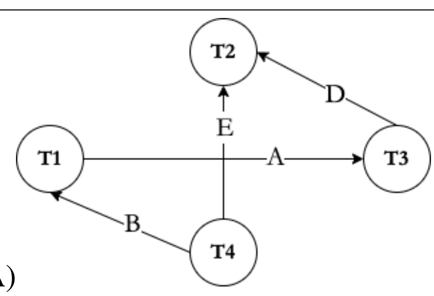
time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
T_1	R(A)					R(C)	R(B)		W(C)		
T_2				R(C)						W(D)	W(E)
T_3					W(A)			R(D)			
T_4		R(E)	W(B)								

Table 1: A schedule with 4 transactions

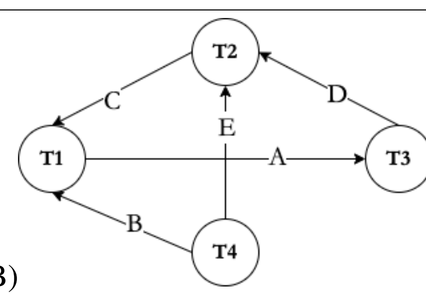
i. **[1 point]** Is this schedule serial?
☐ Yes ■ **No**

Grading info: -1 for incorrect answer

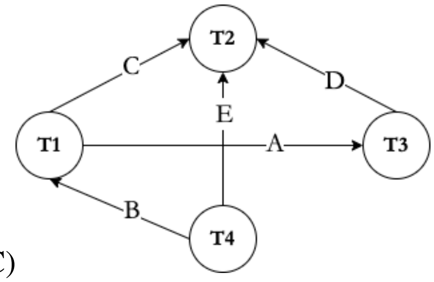
ii. **[2 points]** Choose the correct dependency graph of the schedule given above. Each edge in the dependency graph looks like this: '$T_x \rightarrow T_y$ with Z on the arrow indicating that there is a conflict on Z where T_x read/wrote on Z before T_y'.



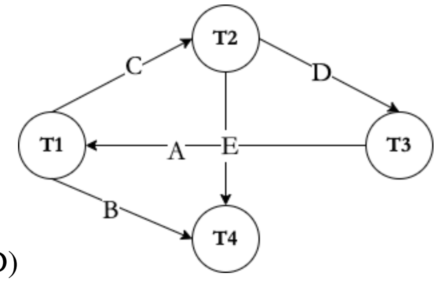
☐ A)



☐ Ans B)



☐ C)



☐ D)

Grading info: -3 for incorrect answer.

iii. **[1 point]** Is this schedule conflict serializable?
☐ Yes ■ **No**
Grading info: -1 for incorrect answer

iv. **[3 points]** Mark all the transactions that can be removed from the schedule that can make it serializable.
■ **`T1`** ■ **`T2`** ■ **`T3`** ☐ `T4` ☐ `Original schedule is also serializable`
Grading info: -1 for any option missed

v. **[1 point]** Is this schedule possible under 2PL?
☐ Yes ■ **No**
Grading info: -1 for incorrect answer

Question 2: Deadlock Detection and Prevention................[42 points]

(a) **Deadlock Detection:**
Consider the following two transactions and note that

- S(·) and X(·) stand for 'shared lock' and 'exclusive lock', respectively.
- T_1 and T_2 represent two transactions.
- LM stands for 'lock manager'.
- Transactions will never release a granted lock.

T_1: (a) read(A); (b) read(B); (c) write(B);

T_2: (d) write(A); (e) read(B); (f) read(A);

i. For each position, what lock should be requested: Only one way to avoid deadlock: Serial

α) **[1 point]** At (a): ■ **S(A)** ☐ S(B) ☐ X(A) ☐ X(B) ☐ No lock needs to be requested

−6

β) **[1 point]** At (b): ☐ S(A) ■ **S(B)** ☐ X(A) ☐ X(B) ☐ No lock needs to be requested

γ) **[1 point]** At (c): ☐ S(A) ☐ S(B) ☐ X(A) ■ **X(B)** ☐ No lock needs to be requested

δ) **[1 point]** At (d): ☐ S(A) ☐ S(B) ■ **X(A)** ☐ X(B) ☐ No lock needs to be requested

ϵ) **[1 point]** At (e): ☐ S(A) ■ **S(B)** ☐ X(A) ☐ X(B) ☐ No lock needs to be requested

ζ) **[1 point]** At (f): ☐ S(A) ☐ S(B) ☐ X(A) ☐ X(B) ■ **No lock needs to be requested**

ii. **[4 points]** Which of the following schedule can cause a deadlock?

T_1	S(A)			read(A)	S(B)	
T_2		S(B)	read(B)			X(A)

☐

T_1	S(A)			read(A)	S(B)	
T_2		S(B)	read(B)			S(A)

☐

T_1	X(A)			read(A)	S(B)	
T_2		X(B)	read(B)			S(A)

■

T_1	S(A)			read(A)	X(B)	
T_2		S(B)	read(B)			X(A)

■

(b) Consider the following lock requests in Table 2. And note that

- S(·) and X(·) stand for 'shared lock' and 'exclusive lock', respectively.
- T_1, T_2, and T_3 represent three transactions.

- LM stands for 'lock manager'.
- Transactions will never release a granted lock.

time	t_1	t_2	t_3	t_4	t_5	t_6	t_7
T_1	S(C)				S(B)	X(B)	S(A)
T_2		S(B)		X(C)			
T_3			X(A)				
LM	g						

Table 2: Lock requests of three transactions

i. For the lock requests in Table 2, determine which lock will be granted or blocked by the lock manager. Please write 'g' in the LM row to indicate the lock is granted and 'b' to indicate the lock is blocked or the transaction has already been blocked by a former lock request. For example, in the table, the first lock (S(A) at time t_1) is marked as granted.

α) **[1 point]** At t_2: ■ **g** ☐ b

Solution:
Grading info: Full points if they got it right

β) **[1 point]** At t_3: ■ **g** ☐ b

Solution:
Grading info: Full points if they got it right

γ) **[1 point]** At t_4: ☐ g ■ **b**

Solution: There is already a shared lock on C by T1

δ) **[1 point]** At t_5: ■ **g** ☐ b

Solution:

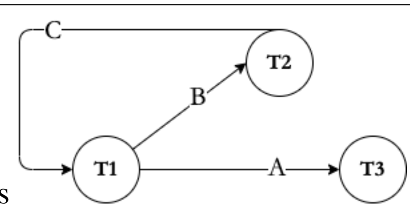
ϵ) **[1 point]** At t_6: ☐ g ■ **b**

Solution: There is a shared lock on B by T2

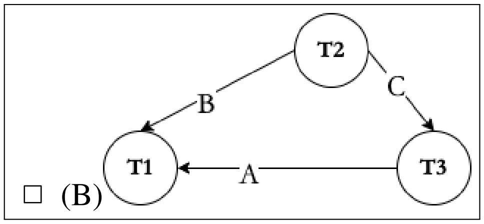
ζ) **[1 point]** At t_7: ☐ g ■ **b**

Solution: There is an exlusive lock on A by T3

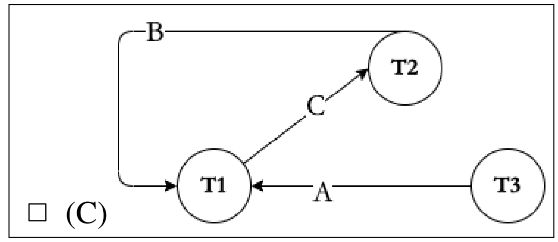
ii. **[2 points]** Mark the correct wait-for graph for the lock requests in Table 2. Each edge in the wait-for graph looks like this: $T_x \rightarrow T_y$ because of Z. Z is denoted in the arrow in the figure. (i.e., T_x is waiting for T_y to release its lock on resource Z).



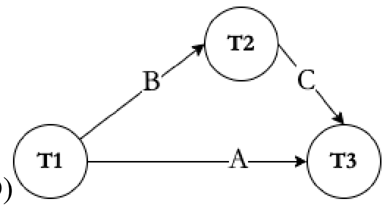
☐ (A) Ans



☐ (B)



☐ (C)



☐ (D)

Grading info: -4 for incorrect answer.

iii. **[2 points]** Determine whether there exists a deadlock in the lock requests in Table 2. Mark all that apply.

☐ There is no deadlock
☐ Cycle ($T_3 \rightarrow T_1 \rightarrow T_3$) exists and schedule deadlocks
☐ There is no cycle
■ **Cycle ($T_1 \rightarrow T_2 \rightarrow T_1$) exists and schedule deadlocks**

Solution:
Grading info: Full points if they got it right

(c) **Deadlock Prevention:**
Consider the following lock requests in Table 3.
Like before,

- S(·) and X(·) stand for 'shared lock' and 'exclusive lock', respectively.
- T_1, T_2, T_3, T_4, and T_5 represent five transactions.
- LM represents a 'lock manager'.
- Transactions will never release a granted lock.

time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
T_1	S(B)						S(A)	
T_2		S(D)			X(B)			
T_3			X(A)	X(D)				S(C)
T_4						X(C)		
LM	g	g						

Table 3: Lock requests of four transactions

i. For the lock requests in Table 3, determine which lock request will be granted, blocked or aborted by the lock manager (LM), if it has no deadlock prevention policy. *Please mark 'g' for grant, 'b' for block (or the transaction is already blocked), 'a' for abort, and '–' if the transaction has already died.*

α) **[1 point]** At t_3: ■ **g** □ b □ a □ –

Solution:
Grading info: Full points if they got it right

β) **[1 point]** At t_4: □ g ■ **b** □ a □ –

Solution:
Grading info: Full points if they got it right

γ) **[1 point]** At t_5: □ g ■ **b** □ a □ –

Solution:
Grading info: Full points if they got it right

δ) **[1 point]** At t_6: ■ **g** □ b □ a □ –

Solution:
Grading info: Full points if they got it right

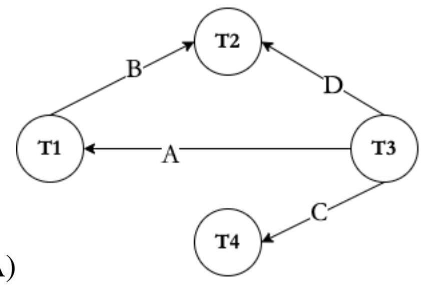
ϵ) **[1 point]** At t_7: □ g ■ **b** □ a □ –

Solution:
Grading info: Full points if they got it right

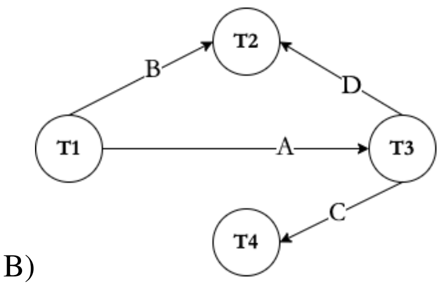
ζ) **[1 point]** At t_8: □ g ■ **b** □ a □ –

Solution:
Grading info: Full points if they got it right

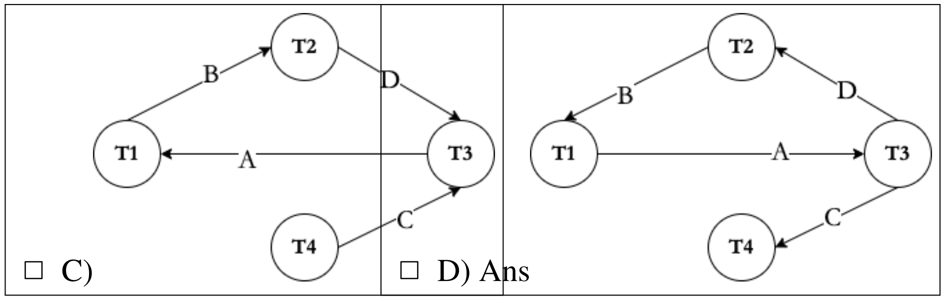
ii. **[2 points]** Mark the correct wait-for graph for the lock requests in Table 3. Each edge in the wait-for graph looks like this: $T_x \rightarrow T_y$ because of Z. Z is denoted in the arrow in the figure. (i.e., T_x is waiting for T_y to release its lock on resource Z).



☐ A)



☐ B)



☐ C) ☐ D) Ans

Grading info: -4 for incorrect answer.

iii. **[2 points]** Determine whether there exists a deadlock in the lock requests in Table 3. Mark all that apply.

☐ There is no deadlock
☐ Cycle ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$) exists and schedule deadlocks
☐ There is no cycle
■ **Cycle ($T_3 \rightarrow T_2 \rightarrow T_1 \rightarrow T_3$) exists and schedule deadlocks**

Solution:
Grading info: Full points if they got it right

iv. To prevent deadlock, we use the lock manager (LM) that adopts the Wait-Die policy. We assume that in terms of priority: $T_1 > T_2 > T_3 > T_4$. Here, $T_1 > T_2$ because T_1 is older than T_2 (i.e., older transactions have higher priority). *Determine whether the lock request is granted ('g'), blocked ('b'), aborted ('a'), or already dead('–').* Follow the same format as the previous question.

α) **[1 point]** At t_3: ■ **g** ☐ b ☐ a ☐ –

Solution:
Grading info: Full points if they got it right

β) **[1 point]** At t_4: ☐ g ☐ b ■ **a** ☐ –

Solution:
Grading info: Full points if they got it right

γ) **[1 point]** At t_5: ☐ g ☐ b ■ **a** ☐ –

Solution:
Grading info: Full points if they got it right

δ) **[1 point]** At t_6: ■ **g** ☐ b ☐ a ☐ –

Solution:
Grading info: Full points if they got it right

ϵ) **[1 point]** At t_7: ■ **g** ☐ b ☐ a ☐ –

Solution: T_3 was aborted.

ζ) **[1 point]** At t_8: ☐ g ☐ b ☐ a ■ **–**

Solution: T_3 already aborted

v. Now we use the lock manager (LM) that adopts the Wound-Wait policy. We assume that in terms of priority: $T_1 > T_2 > T_3 > T_4$. Here, $T_1 > T_2$ because T_1 is older than T_2 (i.e., older transactions have higher priority). *Determine whether the lock request is granted ('g'), blocked ('b'), granted by aborting another transaction ('a'), or the requester is already dead('–').* Follow the same format as the previous question.

α) **[1 point]** At t_3: ■ **g** ☐ b ☐ a ☐ –

Solution:
Grading info: Full points if they got it right

β) **[1 point]** At t_4: ☐ g ■ **b** ☐ a ☐ –

Solution: T_3 has lower priority so it waits

γ) **[1 point]** At t_5: ☐ g ■ **b** ☐ a ☐ –

Solution: T_2 has lower priority so it waits

δ) **[1 point]** At t_6: ■ **g** ☐ b ☐ a ☐ –

Solution:
Grading info: Full points if they got it right

ϵ) **[1 point]** At t_7: ☐ g ☐ b ■ **a** ☐ –

Solution: T_1 has higher priority so T_3 aborts

ζ) **[1 point]** At t_8: ☐ g ☐ b ☐ a ■ **–**

Solution:
Grading info: Full points if they got it right

Young waits for old:
Old > Young
If requesting txn > prev txn, kill prev txn.
Else young waits for old

Question 3: Hierarchical Locking .... [20 points]

Consider a database (D) consisting of two tables, Release (R) and Artists (A). Specifically,

- `Release(rid, name, artist_credit, language, status, genre, year, number_sold)`, spans 1000 pages, namely R_1 to R_{1000}
- `Artists(id, name, type, area, gender, begin_date_year)`, spans 50 pages, namely A_1 to A_{50}

Further, **each page contains 100 records**, and we use the notation $R_3 : 20$ to represent the 20^{th} record on the third page of the Release table. Similarly, $A_5 : 10$ represents the 10^{th} record on the fifth page of the Artists table.

We use Multiple-granularity locking, with **S, X, IS, IX** and **SIX** locks, and **four levels of granularity**: (1) *database-level (D)*, (2) *table-level (R, A)*, (3) *page-level ($R_1 - R_{1000}$, $A_1 - A_{50}$)*, (4) *record-level ($R_1 : 1 - R_{1000} : 100$, $A_1 : 1 - A_{50} : 100$)*.

For each of the following operations on the database, check all the sequence of lock requests based on intention locks that should be generated by a transaction that wants to efficiently carry out these operations by maximizing concurrency. Please take care of efficiency for e.g., share vs. exclusive lock and granularity.

Please follow the format of the examples listed below:

- mark **"IS(D)"** for a request of **database-level IS lock**
- mark **"X(A_2 : 30)"** for a request of **record-level X lock for the 30^{th} record on the second page of the Artists table**
- mark **"S(A_2 : 30 – A_3 : 100)"** for a request of **record-level S lock from the 30^{th} record on the second page of the Artists table to the 100^{th} record on the third page of the Artists table**.

(a) **[4 points]** Fetch the 70^{th} record on page R_{450}.

☐ S(R_{450} : 70)
☐ IS(D), IS(R_{450}), S(R_{450} : 70)
■ **IS(D), IS(R), IS(R_{450}), S(R_{450} : 70)**
☐ SIX(D), SIX(R), SIX(R_{450}), X(R_{450} : 70)

Solution:
Grading info: -2 for each incorrect answer

(b) **[4 points]** Scan all the records on pages R_1 through R_{10}, and modify the record R_{10} : 33.

☐ IX(D), IX(R), IX($R_1 - R_{10}$), X(R_{10} : 33)
■ **IX(D), IX(R), S($R_1 - R_9$), SIX(R_{10}), X(R_{10} : 33)**
■ **IX(D), SIX(R), IX(R_{10}), X(R_{10} : 33)**
☐ IX(D), SIX(R), IS(R_{10}), X(R_{10} : 33)

-2

Solution:

Grading info: -1 *for each incorrect answer,* $+2$ *for each correct answer*

(c) **[4 points]** Count the number of releases with 'year' > 2011.

- ■ **`IS(D), S(R)`**
- □ `S(D), S(R)`
- □ `X(R)`
- □ `IS(D), X(R)`

Solution:

Grading info: -2 *for each incorrect option*

(d) **[4 points]** Increase the number_sold of all release by 2021.

- □ `IX(D), IS(R), X(`R_{100}`)`
- □ `IX(R)`
- □ `X(R), S(A)`
- ■ **`IX(D), X(R)`**

Solution:

Grading info: -4 *for each incorrect option*

(e) **[4 points]** Increase the artist_credit in release and id in artist by 1 for all the tuples in the respective tables.

- ■ **`X(D)`**
- □ `S(D), IS(R), X(A)`
- ■ **`IX(D), X(R), X(A)`**
- □ `IX(D), X(R), S(A)`

−2

Solution:

Grading info: Full points for selecting either or both of the correct answers. -2 for each incorrect option

Question 4: Optimistic Concurrency Control .... [20 points]

Consider the following set of transactions accessing a database with object *A, B, C, D*. The questions below assume that the transaction manager is using **optimistic concurrency control** (OCC). Assume that a transaction switches from the READ phase immediately into the VALIDATION phase after its last operation executes.

Note: VALIDATION may or may not succeed for each transaction. If validation fails, the transaction will get immediately aborted.

You can assume that the DBMS is using the serial validation protocol discussed in class where only one transaction can be in the validation phase at a time, and each transaction is doing forward validation (i.e. Each transaction, when validating, checks whether it intersects its read/write sets with any active transactions that have not yet committed.)

time	T_1	T_2	T_3
1	READ(A)		
2	READ(C)		
3		READ(B)	
4	WRITE(A)		
5			READ(B)
6	WRITE(C)		
7	VALIDATE?		
8		READ(D)	
9	WRITE?		
10		WRITE(D)	
11		WRITE(B)	
12		VALIDATE?	
13			READ(A)
14		WRITE?	
15			WRITE(A)
16			WRITE(B)
17			VALIDATE?
18			WRITE?

Figure 1: An execution schedule

(a) **[6 points]** Will T1 abort?
- ☐ Yes
- ■ **No**

Solution: T1 does not need to abort because it does not read or write B.

Grading info: Full points if they got it right

(b) **[6 points]** Will T2 abort? Only kill txn at validation phase, no priority here.
- ■ **Yes**
- ☐ No

-6

Solution: T2's write-set intersects with T3's read-set (B), so it will fail the VALIDATION phase.

Grading info: Full points if they got it right

(c) **[6 points]** Will T3 abort?
☐ Yes
■ **No**

−6

Solution: Although T3's read-set intersects with T2's write-set, T2 will get aborted, so T3 does not need to abort.

Grading info: Full points if they got it right

(d) **[2 points]** OCC is good to use when there are few conflicts.
■ **True**
☐ False

Solution: From the slides: If the database is large and the workload is not skewed, then there is a low probability of conflict, so locking is wasteful.

Grading info: Full points if they got it right.