CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (FALL 2019)
PROF. ANDY PAVLO

Homework #4 (by Weichen Ke) – Solutions

Due: **Wednesday Nov 13, 2019 @ 11:59pm**

IMPORTANT:

- **Upload this PDF** with your answers to **Gradescope by 11:59pm on Wednesday Nov 13, 2019**.
- **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: $\approx$ 1 - 2 hours (0.5 - 1 hours for each question)

Revision : 2019/11/18 20:45

Question	Points	Score
Serializability and 2PL	20	
Deadlock Detection and Prevention	30	
Hierarchical Locking	30	
Optimistic Concurrency Control	20	
Total:	100	

Number of Days this Assignment is Late:

Number of Late Day You Have Left:

Question 1: Serializability and 2PL........................[20 points]

(a) Yes/No questions:

i. **[2 points]** Schedules under rigorous 2PL will not have dirty reads.
[✓ **Yes**] □ No

ii. **[2 points]** A schedule generated by rigorous 2PL will never cause a deadlock.
□ Yes [✓ **No**]

iii. **[2 points]** A schedule generated by 2PL is always view serializable.
[✓ **Yes**] □ No

iv. **[2 points]** A conflict serializable schedule will never contain a cycle in its precedence graph.
[✓ **Yes**] □ No

v. **[2 points]** Every view serializable schedule is conflict serializable.
□ 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)		W(A)						R(B)		W(B)
T_2				R(C)	R(A)		W(A)			W(C)	
T_3		R(B)				W(B)		R(A)			

Table 1: A schedule with 3 transactions

i. **[1 point]** Is this schedule serial?
□ Yes [✓ **No**]

Grading info: -1 for incorrect answer

ii. **[3 points]** Give the dependency graph of this schedule. List each edge in the dependency graph like this: '$T_x \rightarrow T_y$ because of Z'. This notation signifies that T_x precedes T_y because Z was last read/written by T_x before it was read/written by T_y. Order the edges in ascending order with respect to x.

Solution:

- $T_1 \rightarrow T_2$ because of A
- $T_1 \rightarrow T_3$ because of A
- $T_2 \rightarrow T_3$ because of A
- $T_3 \rightarrow T_1$ because of B

Grading info: -1 for each missing/incorrect edge.

iii. **[1 point]** Is this schedule conflict serializable?
□ Yes [✓ **No**]

Question 1 continues...

Grading info: -1 for incorrect answer

iv. **[3 points]** If you answer "yes" to (iii), provide the equivalent serial schedule. If you answer "no", briefly explain why.

Solution: The schedule is not serializable because there are two cycles in the dependency graph ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$ and $T_1 \rightarrow T_3 \rightarrow T_1$).

Grading info: -3 for a justification that does not agree with previous part

v. **[2 points]** Is this schedule possible under 2PL?

☐ Yes [✓ **No**]

Grading info: -2 for incorrect answer

Question 2: Deadlock Detection and Prevention................[30 points]

(a) **Deadlock Detection:**
Consider the following lock requests in Table 2. And note that

- S($\cdot$) and X($\cdot$) 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(A)	S(B)			S(C)
T_2	S(B)				X(A)		
T_3		X(C)				X(B)	
LM	g						

Table 2: Lock requests of three transactions

i. **[3 points]** 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(D) at time t_1) is marked as granted.

Solution:

- X(C) at t_2: g
- S(A) at t_3: g
- S(B) at t_4: g
- X(A) at t_5: b
- X(B) at t_6: b
- S(C) at t_7: b

Grading info: Half points for one mistake in the schedule, no points for > 1 mistake.

ii. **[4 points]** Give the wait-for graph for the lock requests in Table 2. List each edge in the graph like this: $T_x \rightarrow T_y$ because of Z (i.e., T_x is waiting for T_y to release its lock on resource Z). Order the edges in ascending order with respect to x.

Solution:

- $T_1 \rightarrow T_3$ because of C
- $T_2 \rightarrow T_1$ because of A

Question 2 continues...

- $T_3 \rightarrow T_1$ because of B
- $T_3 \rightarrow T_2$ because of B

Grading info: Half points for 1 missing directed edge, no points if missing > 1.

iii. **[3 points]** Determine whether there exists a deadlock in the lock requests in Table 2. If there is a deadlock, point out one cycle.

Solution: Deadlock exists because there is a cycle ($T_1 \rightarrow T_3 \rightarrow T_1$) in the dependency graph.
OR: Deadlock exists because there is a cycle ($T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$) in the dependency graph.

Grading info: -2 points for not explaining why there is a deadlock

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

- S($\cdot$) and X($\cdot$) 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	X(B)			S(A)				
T_2					X(D)	X(C)		
T_3			S(C)				X(B)	
T_4		X(A)						S(D)
LM	g							

Table 3: Lock requests of four transactions

i. **[3 points]** 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 write 'g' for grant, 'b' for block (or the transaction is already blocked), 'a' for abort, and '$-$'if the transaction has already died.* Again, example is given in the first column.

Solution:

- X(A) at t_2: g
- S(C) at t_3: g
- S(A) at t_4: b

- X(D) at t_5: g
- X(C) at t_6: b
- X(B) at t_7: b
- S(D) at t_8: b

Grading info: Half points for one mistake in the schedule, no points for > 1 mistake.

ii. **[4 points]** Give the wait-for graph for the lock requests in Table 3. List each edge in the graph like this: $T_x \rightarrow T_y$ because of Z (i.e., T_x is waiting for T_y to release its lock on resource Z). Order the edges in ascending order with respect to x. If a lock request is not proposed because the transaction is blocked before making that lock request, you can ignore that lock request, as if it does not exist.

Solution:

- $T_1 \rightarrow T_4$ because of A
- $T_2 \rightarrow T_3$ because of C
- $T_3 \rightarrow T_1$ because of B
- $T_4 \rightarrow T_2$ because of D

Grading info: Half points for 1 missing directed edge, no points if missing > 1.

iii. **[3 points]** Determine whether there exists a deadlock in the lock requests in Table 3. If there is a deadlock, point out one cycle.

Solution: Deadlock exists because there is a cycle ($T_1 \rightarrow T_4 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$) in the dependency graph.

Grading info: -2 points for not explaining why there is a deadlock

iv. **[5 points]** 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.

Solution:

- X(A) at t_2: g
- S(C) at t_3: g
- S(A) at t_4: b (T_1 has higher priority)
- X(D) at t_5: g

Question 2 continues…

- X(C) at t_6: b (T_2 has higher priority)
- X(B) at t_7: a (T_3 aborts because T_1 is holding B)
- S(D) at t_8: a (T_4 aborts because T_2 is waiting for the lock)

Grading info: -2 *points for one mistake in the schedule, no points* > 1 *mistake.*

v. **[5 points]** 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.

Solution:

- X(A) at t_2: g
- S(C) at t_3: g
- S(A) at t_4: a (T_1 wounds T_4)
- X(D) at t_5: g
- X(C) at t_6: a (T_2 wounds T_3)
- X(B) at t_7: $-$ (T_3 is already dead)
- S(D) at t_8: $-$ (T_4 is already dead)

Grading info: -2 *points for one mistake in the schedule, no points* > 1 *mistake.*

Question 3: Hierarchical Locking .... [30 points]

Consider a database (D) consisting of two tables, Tablets (T) and CPUs (C). Specifically,

- `Tablets(tid, brand, CPU_id, OS, year, sales)`, spans 1000 pages, namely T_1 to T_{1000}
- `CPUs(CPU_id, brand, frequency, cache)`, spans 50 pages, namely C_1 to C_{50}

Further, **each page contains 100 records**, and we use the notation $T_3 : 20$ to represent the 20^{th} record on the third page of the Tablets table. Similarly, $C_5 : 10$ represents the 10^{th} record on the fifth page of the CPUs 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 (T, C)*, (3) *page-level ($T_1 - T_{1000}$, $C_1 - C_{50}$)*, (4) *record-level ($T_1 : 1 - T_{1000} : 100$, $C_1 : 1 - C_{50} : 100$)*.

For each of the following operations on the database, please determine the sequence of lock requests that should be generated by a transaction that wants to efficiently carry out these operations by maximizing concurrency. **Please use the fewest number of locks in your answer.** If you use much more locks than necessary (more than 10x), you will get **half** points.

Please follow the format of the examples listed below:

- write **"IS(D)"** for a request of **database-level IS lock**
- write **"X($C_2 : 30$)"** for a request of **record-level X lock for the 30^{th} record on the second page of the CPUs table**
- write **"S($C_2 : 30 - C_3 : 100$)"** for a request of **record-level S lock from the 30^{th} record on the second page of the CPUs table to the 100^{th} record on the third page of the CPUs table**.

(a) **[6 points]** Fetch the 25^{th} record on page T_{233}.

Solution: IS(D), IS(A), IS(T_{233}), S($T_{233} : 25$)

Grading info: -2 for each missing/incorrect mistake, -3 for correct but too many locks

(b) **[6 points]** Scan all the records on pages T_1 through T_{10}, and modify the record $T_2 : 33$.

Solution: IX(D), SIX(T), IX(T_2), X($T_2 : 33$);
also acceptable: IX(D), IX(T), S(T_1), S($T_3 - T_{10}$), SIX(T_2), X($T_2 : 33$)

Grading info: -2 for each missing/incorrect mistake, -3 for correct but too many locks

(c) **[6 points]** Count the number of tablets with 'year' > 2014.

Solution: IS(D), S(T);

Grading info: -2 for each missing/incorrect mistake, -3 for correct but too many locks

(d) **[6 points]** Increase the sales of all tablets by 114514.

Question 3 continues...

Solution: IX(D), X(T)

Grading info: -2 *for each missing/incorrect mistake,* -3 *for correct but too many locks*

(e) **[6 points]** Capitalize the 'brand' of ALL tablets and ALL CPUs.

Solution: X(D)
Also acceptable: IX(D), X(T), X(C)

Grading info: -2 *for each missing/incorrect mistake,* -3 *for correct but too many locks*

Homework #4 continues…

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(A)	
3	READ(B)		
4	WRITE(B)		
5	WRITE(C)		
6	VALIDATE?		
7			READ(D)
8		READ(B)	
9		WRITE(D)	
10		WRITE(B)	
11		VALIDATE?	
12	WRITE?		
13		WRITE?	
14			WRITE(D)
15			VALIDATE?
16			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 write A.

Grading info: Full points if they got it right

(b) **[6 points]** Will T2 abort?
[✓ **Yes**]
□ No

Solution: T2's read-set intersects with T1's write-set (B), and its write-set intersects with T3's read-set (D), so it will fail the VALIDATION phase.

Question 4 continues. . .

Grading info: Full points if they got it right

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

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.