CARNEGIE MELLON UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
15-445/645 – DATABASE SYSTEMS (FALL 2017)
PROF. ANDY PAVLO

Homework 5 (by Joy Arulraj) – Solutions

Due: **Monday Nov 13, 2017 @ 11:59pm**

IMPORTANT:

- **Upload this PDF** with your answers to **Gradescope by 11:59pm on Monday Nov 13, 2017**.
- **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

Revision : 2017/12/10 13:51

Question	Points	Score
Serializability and 2PL	20	
Deadlock Detection and Prevention	30	
Hierarchical Locking - A Blogging Website	30	
B+ tree Locking	20	
Total:	100	

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

(a) Yes/No questions:

i. **[2 points]** Schedules under Strict 2PL could have cascading aborts.
☐ Yes ■ **No**

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

iii. **[2 points]** Schedules under Strict 2PL could have deadlocks.
■ **Yes** ☐ No

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

v. **[2 points]** If a precedence graph associated with a schedule has cycles, then it is not 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_1	R(A)	W(A)	W(E)	R(B)		W(B)	R(C)	W(C)	R(D)	W(D)
T_2			R(A)		W(A)	R(D)	W(D)	W(E)		W(A)
T_3				R(C)	W(C)			R(F)	W(F)	

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 (i.e., Transaction T_y reads/writes Z which was last written by T_x.). Order the edges in ascending order with respect to x.

Solution:

- $T_1 \rightarrow T_2$ because of A, E
- $T_2 \rightarrow T_1$ because of D
- $T_3 \rightarrow T_1$ because of C

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

iii. **[1 point]** Is this schedule conflict serializable?

☐ Yes ■ **No**

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: This schedule is not conflict serializable because there exists a cycle ($T_1 \rightarrow T_2 \rightarrow T_1$) in the dependency graph.

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: -1 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(·) 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(A)				S(B)	S(C)	
T_2		S(B)		X(C)			X(B)
T_3			X(A)				
LM	g						

Table 2: Lock requests of 3 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. For example, in the table, the first lock (S(D) at time t_1) is marked as granted.

Solution:

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

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

- $T_1 \rightarrow T_2$ because of C

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

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, and briefly explain why.

Solution: Deadlock exists because there is a cycle ($T_1 \rightarrow T_2 \rightarrow T_1$) 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. Again,

- S($\cdot$) and X($\cdot$) stand for 'shared lock' and 'exclusive lock', respectively.
- T_1, T_2, T_3, and T_4 represent four 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(A)		S(B)					
T_2		X(B)						X(D)
T_3				S(C)	X(D)		X(A)	
T_4						X(C)		
LM	g							

Table 3: Lock requests of 4 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 and 'a' for abort.* Again, example is given in the first column.

Solution:

- X(B) at t_2: g
- S(B) at t_3: b
- S(C) at t_4: g
- X(D) at t_5: g

- X(C) at t_6: b
- X(A) at t_7: b
- X(D) at t_8: b

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

Solution:

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

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, and briefly explain why.

Solution: Deadlock exists because there is a cycle ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$) 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 which lock request will be granted ('g'), blocked ('b') or aborted ('a').* Follow the same format as the previous question.

Solution:

- X(B) at t_2: g
- S(B) at t_3: b (T_1 waits for T_2.)
- S(C) at t_4: g
- X(D) at t_5: g
- X(C) at t_6: a (T_4 dies for T_3.)
- X(A) at t_7: a (T_3 dies for T_1.)

- X(D) at t_8: g (T_2 is granted the lock, since T_3 is aborted.)

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 which lock request will be granted ('g'), blocked ('b') or aborted ('a').* Follow the same format as the previous question.

Solution:

- X(B) at t_2: g
- S(B) at t_3: g (T_1 wounds T_2.)
- S(C) at t_4: g
- X(D) at t_5: g
- X(C) at t_6: b (T_4 waits for T_3.)
- X(A) at t_7: b (T_3 waits for T_1.)
- X(D) at t_8: – (T_2 is dead.)

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

Question 3: Hierarchical Locking - A Blogging Website . . . . [30 points]

Consider a Database (D) consisting of two tables, Users (U) and Posts (P). Specifically,

- Users(uid, first_name, last_name), spans 300 pages, namely U_1 to U_{300}
- Posts(pid, uid, title, body), spans 600 pages, namely P_1 to P_{600}

Further, **each page contains 100 records**, and we use the notation $U_3 : 20$ to represent the 20^{th} record on the third page of the Users table. Similarly, $P_5 : 10$ represents the 10^{th} record on the fifth page of the Posts 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 (U, P)*, (3) *page-level ($U_1 - U_{300}$, $P_1 - P_{600}$)*, (4) *record-level ($U_1 : 1 - U_{300} : 100$, $P_1 : 1 - P_{600} : 100$)*.
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 follow the format of the examples listed below:

- write **"IS(D)"** for a request of **database-level IS lock**
- write **"X(P_2 : 30)"** for a request of **record-level X lock for the 30^{th} record on the second page of the Posts table**
- write **"S($P_2 : 30 - P_3 : 100$)"** for a request of **record-level S lock from the 30^{th} record on the second page of the Posts table to the 100^{th} record on the third page of the Posts table**.

(a) **[6 points]** Fetch the 10^{th} record on page P_{100}.

> **Solution:** IS(D), IS(P), IS(P_{100}), S(P_{100} : 10)
> *Grading info:* -2 *for each missing/incorrect mistake*

(b) **[6 points]** In remembrance of our TA (RIP Christopher "Inf" Wallace), set the last_name of all the users to be "Inf".

> **Solution:** IX(D), X(U)
> also acceptable: IX(D), IX(U), IX($U_1 - U_{300}$), X($U_1 : 1 - U_{300} : 100$)
> *Grading info:* -2 *for each missing/incorrect mistake*

(c) **[6 points]** Scan all the records on pages P_1 through P_{20}, and modify the record $P_5 : 10$.

> **Solution:** IX(D), SIX(P), IX(P_5), X(P_5 : 10);
> also acceptable: IX(D), IX(P), S($P_1 - P_4$), S($P_6 - P_{20}$), SIX(P_5), X(P_5 : 10)
> *Grading info:* -2 *for each missing/incorrect mistake*

(d) **[6 points]** Order all the posts by their title.

Question 3 continues. . .

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

Grading info: -2 *for each missing/incorrect mistake*

(e) **[6 points]** Delete ALL the records from ALL tables.

Solution: X(D)

Grading info: -2 *for each missing/incorrect mistake*

Question 4: B+ tree Locking .... [20 points]

Consider the following B+ tree:

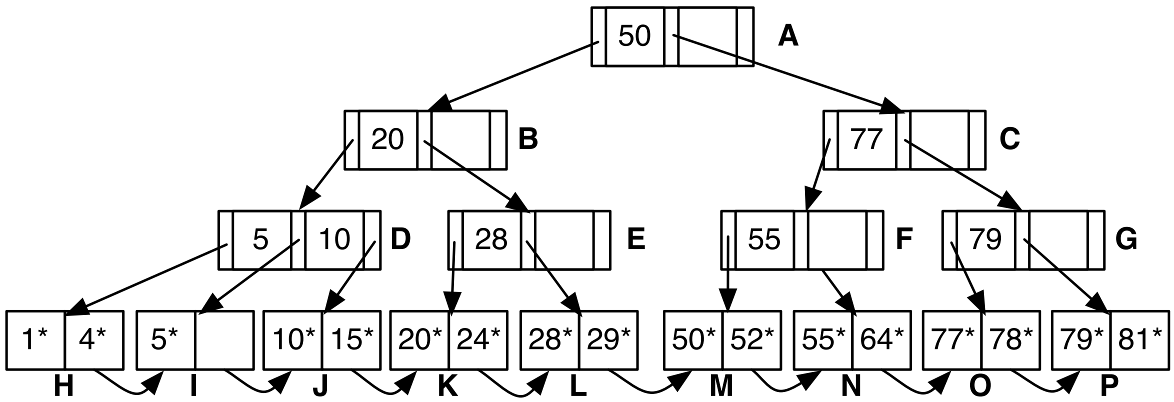

Figure 1: B+ tree locking

To lock this B+ tree, we would like to use the **Bayer-Schkolnick** algorithm. **Important**: we use the version as presented in the lecture, which **does not** use lock upgrade.
For each of the following transactions, give the sequence of lock/unlock requests. For example, please write $S(A)$ for a request of shared lock on node A, $X(B)$ for a request of exclusive lock on node B and $U(C)$ for a request of unlock node C.
Important notes:

- Each of the following transactions is applied on the *original tree*, i.e., ignore any tree changes described in other problems.
- For simplicity, *ignore* the changes on the pointers between leaves.

Solution:

Grading info: -2 *for each missing/incorrect mistake*

(a) **[5 points]** Search for data entry "15*"

Solution: S(A), S(B), U(A), S(D), U(B), S(J), U(D), U(J)

(b) **[5 points]** Delete data entry "28*"

Solution: S(A), S(B), U(A), S(E), U(B), X(L), U(E), U(L)

(c) **[5 points]** Insert data entry "9*"

Solution: S(A), S(B), U(A), S(D), U(B), X(I), U(D), U(I)
Also acceptable: S(A), S(B), U(A), S(D), X(I), U(B), U(D), U(I)

(d) **[5 points]** Insert data entry “45*”

Solution: S(A), S(B), U(A), S(E), U(B), X(L), *This leaf is not safe because we need to split. We must restart*, U(E), U(L)
X(A), X(B), U(A), X(E), U(B), X(L), U(E), U(L)
Final answer: S(A), S(B), U(A), S(E), U(B), X(L), U(E), U(L), X(A), X(B), U(A), X(E), U(B), X(L), U(E), U(L)

Grading info: No points deducted for swapping U(E) and U(L). We cannot release X(E) and X(L) until after the insertion (since we need to split) so we end up releasing both locks at roughly the same time.