CS4224/CS5424 Quiz 5: Concurrency Control Wk 6, Sem 1, 2023/24

In this quiz, questions 1 to 4 are review questions covering serializability and 2PL protocol. We will discuss only questions 5 to 7 in class.

1. Answer the following questions for each of the schedules (a) to (e):

   (i) Is the schedule conflict serializable?

   (ii) Is the schedule a 2PL schedule? Assume that each executed read/write operation is immediately preceded by an appropriate lock request (i.e., S-lock, X-lock, or lock upgrade).

   (a) $W_3(a)$, $R_1(a)$, $W_1(b)$, $R_2(b)$, $W_3(c)$, $R_3(c)$

   (b) $R_1(a)$, $R_2(a)$, $R_1(b)$, $R_2(b)$, $R_3(a)$, $R_4(b)$, $W_1(a)$, $W_2(b)$

   (c) $R_1(a)$, $R_2(a)$, $R_3(b)$, $W_1(a)$, $R_2(c)$, $R_2(b)$, $W_2(b)$, $W_1(c)$

   (d) $R_1(a)$, $W_1(b)$, $R_2(b)$, $W_2(c)$, $R_3(c)$, $W_3(a)$

   (e) $R_1(a)$, $R_2(b)$, $R_3(c)$, $R_1(b)$, $R_2(c)$, $R_3(d)$, $W_1(a)$, $W_2(b)$, $W_3(c)$

Solution:

(a) $W_3(a)$, $R_1(a)$, $W_1(b)$, $R_2(b)$, $W_3(c)$, $R_3(c)$

  (i) Conflict serializable. The schedule is conflict equivalent to (T_3, T_1, T_2).

  (ii) Schedule is not 2PL since T_3 would need to release its lock on a to enable $R_1(a)$ which implies that T_3 would not be allowed to subsequently request a lock for $W_3(c)$.

(b) $R_1(a)$, $R_2(a)$, $R_1(b)$, $R_2(b)$, $R_3(a)$ $R_4(b)$, $W_1(a)$, $W_2(b)$

  (i) Not conflict serializable. Since $R_1(b)$ precedes $W_2(b)$, T_1 must precede T_2 in a conflict-equivalent serial schedule. However, $R_2(a)$ precedes $W_1(a)$ implies that T_2 must precede T_1 in a conflict-equivalent serial schedule. Thus, there does not exist a conflict-equivalent serial schedule.

  (ii) Not a 2PL schedule since it is not conflict serializable.

(c) $R_1(a)$, $R_2(a)$, $R_3(b)$, $W_1(a)$, $R_2(c)$, $R_2(b)$, $W_2(b)$, $W_1(c)$

  (i) Conflict serializable. The schedule is conflict equivalent to (T_3, T_2, T_1).

  (ii) Schedule is not 2PL since T_2 would need to release its lock on a to enable $W_1(a)$ which implies that T_2 would not be allowed to subsequently request a lock for $R_2(c)$.

(d) $R_1(a)$, $W_1(b)$, $R_2(b)$, $W_2(c)$, $R_3(c)$, $W_3(a)$

  (i) Conflict serializable. The schedule is conflict equivalent to (T_1, T_2, T_3).

  (ii) Schedule is a 2PL schedule.

(e) $R_1(a)$, $R_2(b)$, $R_3(c)$, $R_1(b)$, $R_2(c)$, $R_3(d)$, $W_1(a)$, $W_2(b)$, $W_3(c)$

  (i) Conflict serializable. The schedule is conflict equivalent to (T_1, T_2, T_3).

  (ii) Schedule is a 2PL schedule.

2. Prove that a conflict serializable schedule is necessarily also a view serializable schedule.

Solution: Let S be a conflict serializable schedule. Thus, there exists a serial schedule S' that is conflict equivalent to S. We establish that S and S' must be view equivalent by contradiction. Suppose that S and S' are not view equivalent. This implies that S and S' differ in some (1) final write or (2) read-from relationship.

Consider case (1). Suppose that $W_i(A)$ is the final write on A in S and $W_j(A)$ is the final write on A in S', $i \neq j$; i.e.,

$$S : \cdots W_j(A) \cdots W_i(A) \cdots$$
$$S' : \cdots W_i(A) \cdots W_j(A) \cdots$$

However, this contradicts the fact that S and S' are conflict equivalent since the pair of conflicting actions, $W_i(A)$ and $W_j(A)$, are ordered differently in S and S'. Hence, S and S' must agree on the same final writes.

Consider case (2). Suppose that T_k reads A from T_i in S, and T_k reads A from T_j in S', $i \neq j$; i.e.,

$$S : \cdots W_i(A) \cdots R_k(A) \cdots$$
$$S' : \cdots W_j(A) \cdots R_k(A) \cdots$$

Since $W_i(A)$ precedes $R_k(A)$ in S, and S and S' are conflict equivalent, we must have $W_i(A)$ precedes $R_k(A)$ in S'. By a similar argument, we must have $W_j(A)$ precedes $R_k(A)$ in S. Therefore, we have the following:

$$S : \cdots W_j(A) \cdots W_i(A) \cdots R_k(A) \cdots$$
$$S' : \cdots W_i(A) \cdots W_j(A) \cdots R_k(A) \cdots$$

However, this contradicts the fact that S and S' are conflict equivalent since the pair of conflicting actions, $W_i(A)$ and $W_j(A)$, are ordered differently in S and S'. Hence, S and S' must agree on the same read-from relationships.

Therefore, S and S' must be view serializable if they are conflict serializable.

3. For each transaction T_i in a 2PL schedule S, let $lastlocktime(T_i)$ denote the time that T_i acquires its last lock in S. For example, in the following 2PL schedule S:

$$W_2(\text{X}),\ R_1(\text{Y}),\ R_2(\text{Y}),\ R_1(\text{X}),\ Commit_1,\ Commit_2$$

we have $lastlocktime(T_2) < lastlocktime(T_1)$ since T_2 can only release its locks after $R_2(Y)$ and T_1 can only acquire its lock on X after T_2's release of locks.

Given a 2PL schedule S, let S' denote the serial schedule obtained from S where the transactions in S' are serialized in increasing order of their $lastlocktime()$. Prove that S and S' are conflict equivalent.

Solution: Consider a pair of conflicting actions $(A_i(O), A_j(O))$ in S on object O, $i \neq j$, where $A_i(O)$ precedes $A_j(O)$ in S. Since the actions are conflicting, T_i must release its lock on O before $A_j(O)$ in S. Furthermore, T_i does not acquire any other lock in S after releasing its lock on O before $A_j(O)$. Therefore, $lastlocktime(T_i) < lastlocktime(T_j)$, and T_i precedes T_j in S'. Hence, $A_i(O)$ must also precede $A_j(O)$ in S'. Therefore, S and S' are conflict equivalent.

4. Prove that a S2PL schedule is necessarily a recoverable schedule.

Solution: Consider a non-recoverable schedule S. This implies that there exists two transactions T_1 and T_2 in S where T_2 reads from T_1 and T_2 commits before the completion of T_1; i.e.,

$$S : \quad \cdots, W_1(x), \cdots, R_2(x), \cdots, Commit_2, \cdots, Commit_1/Abort_1$$

However, since $W_1(x)$ and $R_2(x)$ are conflicting actions, and S2PL does not allow a transaction to release locks before its completion, it is impossible for $R_2(x)$ to be executed after $W_1(x)$ before the completion of T_1 if S is a S2PL schedule. Hence, a S2PL schedule is necessarily a recoverable schedule.

5. For each of the following schedules, state whether it is a Snapshot Isolation schedule under the First-Committer-Wins rule.
   (a) R_1(X), R_2(X), W_1(X), W_2(X), $Commit_1$, $Commit_2$
   (b) W_1(X), R_2(Y), R_1(Y), R_2(X), $Commit_1$, $Commit_2$
   (c) R_1(X), R_2(Y), W_3(X), R_2(X), R_1(Y), $Commit_1$, $Commit_2$, $Commit_3$
   (d) R_1(X), R_1(Y), W_1(X), R_2(Y), W_3(Y), W_1(X), R_2(Y), $Commit_1$, $Commit_2$, $Commit_3$
   (e) R_1(X), W_2(X), W_1(X), $Commit_2$, $Commit_1$
   (f) W_1(X), R_2(X), W_1(X), $Commit_2$, $Commit_1$
   (g) R_2(X), W_3(X), $Commit_3$, W_1(Y), $Commit_1$, R_2(Y), W_2(Z), $Commit_2$
   (h) R_1(X), W_2(X), $Commit_2$, W_1(X), $Commit_1$, R_3(X), $Commit_3$
   (i) R_1(X), W_2(X), W_1(X), R_3(X), $Commit_1$, $Commit_2$, $Commit_3$

Solution: (b), (c), (d), (f), (g): Yes.
(a) No. T_2 must abort due to $W_2(X)$.
(e) No. T_1 must abort due to $W_1(X)$.
(h) No. T_1 must abort due to $W_1(X)$.
(i) No. T_2 must abort due to $W_2(X)$.

6. Consider the following schedule involving transactions T_1, T_2, T_3, and T_4 which are executed in a database system using the Snapshot Isolation protocol with the First Committer Wins Rule.

   Assume that the start timestamp of each transaction is given by the timestamp of its first operation (e.g., start(T_1) = 7).

   Assume the following values for the state of the database before the start of the schedule:

$$a_0 = 10, \quad b_0 = 20, \quad c_0 = 30.$$

Timestamp	T_3	T_1	T_2	T_4	**Comments**
1	$R_3(c)$				
2			$R_2(b)$		
3			$W_2(b)$		T_2 updates b to 45.
4			$Commit_2$		
5	$R_3(b)$				
6	$W_3(b)$				T_3 updates b to 60.
7		$R_1(a)$			
8		$R_1(b)$			
9		$W_1(a)$			T_1 updates a to 100.
10				$R_4(b)$	
11				$R_4(a)$	
12				$W_4(b)$	T_4 updates b to 16.
13	$R_3(a)$				
14	$W_3(c)$				T_3 updates c to 200.
15		$W_1(c)$			T_1 updates c to 40.
16		$Commit_1$			
17				$R_4(b)$	
18				$R_4(c)$	

   (a) Write down the value read by $R_3(b)$ at timestamp 5.
   (b) Write down the value read by $R_1(b)$ at timestamp 8.
   (c) Write down the value read by $R_4(b)$ at timestamp 10.

(d) Write down the value read by $R_4(a)$ at timestamp 11.

(e) Write down the value read by $R_3(a)$ at timestamp 13.

(f) Write down the value read by $R_4(b)$ at timestamp 17.

(g) Write down the value read by $R_4(c)$ at timestamp 18.

(h) If T_3 were to attempt to commit at timestamp 19, would this commit be successful?

(i) If T_4 were to attempt to commit at timestamp 20, would this commit be successful?

(j) Assume that T_3 and T_4 attempted to commit at timestamps 19 and 20, respectively. Let S denote the subset of successfully committed transactions (i.e., $S \subseteq \{T_1, T_2, T_3, T_4\}$). State whether the Snapshot Isolation schedule for S is MVSS.

Solution:

(a) Write down the value read by $R_3(b)$ at timestamp 5. (20)

(b) Write down the value read by $R_1(b)$ at timestamp 8. (45)

(c) Write down the value read by $R_4(b)$ at timestamp 10. (45)

(d) Write down the value read by $R_4(a)$ at timestamp 11. (10)

(e) Write down the value read by $R_3(a)$ at timestamp 13. (10)

(f) Write down the value read by $R_4(b)$ at timestamp 17. (16)

(g) Write down the value read by $R_4(c)$ at timestamp 18. (30)

(h) T_3's commit attempt would fail since it has updated some object that was already updated by a committed concurrent transaction (specifically, b updated also by T_2 and c updated also by T_1).

(i) T_4's commit attempt would succeed since it is concurrent with only one committed transaction T_1 which did not update b.

(j) $S = \{T_1, T_2, T_4\}$). Since both T_1 & T_4 reads b from T_2, T_2 must precede both T_1 & T_4 in a serial execution. Since T_4 did not read c from T_1, T_4 must precede T_1 in a serial execution; however, this would imply that T_1 must read b from T_4 which is not the case here. Hence, S is not MVSS.

7. Consider the execution of a set of distributed transactions $T = \{T_1, T_2, T_3\}$ over sites A, B, and C, where Site A $= \{a, b, c\}$, Site B $= \{d, e\}$, and Site C $= \{f\}$ such that

$$\begin{array}{ll} T_1\text{:} & R_1(a), R_1(d), R_1(f), W_1(d), W_1(f) \\ T_2\text{:} & R_2(c), R_2(e), R_2(d), W_2(e) \\ T_3\text{:} & R_3(a), R_3(b), R_3(c), R_3(f), W_3(a),\ W_3(c) \end{array}$$

Answer the following questions for each of the scenarios (a) and (b):

(i) State whether a serializable global schedule exists for T and $\{S_A, S_B, S_C\}$.

(ii) State whether the local schedules could be produced by S2PL protocol.

(iii) State whether the local schedules could be produced by SI protocol.

(a) Local schedules:

$$\begin{array}{ll} S_A\text{:} & R_1(a), R_3(a), R_3(b), R_3(c), W_3(a), W_3(c), R_2(c) \\ S_B\text{:} & R_2(e), R_1(d), R_2(d), W_1(d), W_2(e) \\ S_C\text{:} & R_1(f), W_1(f),\ R_3(f) \end{array}$$

(b) Local schedules:

$$\begin{array}{ll} S_A\text{:} & R_1(a), R_3(a), R_3(b), R_2(c), R_3(c), W_3(a), W_3(c) \\ S_B\text{:} & R_2(e), R_1(d), R_2(d), W_1(d), W_2(e) \\ S_C\text{:} & R_1(f), W_1(f),\ R_3(f) \end{array}$$

Solution:

(a) (i) No serializable global schedule exists: the only serial order for S_A is (T_1, T_3, T_2); the only serial order for S_B is (T_2, T_1); and the only serial order for S_C is (T_1, T_3). The local serial orders (T_1, T_3, T_2) and (T_2, T_1) are not compatible.

(ii) The local schedules can't be S2PL since no serializable global schedule exists.

(iii) Since each object is updated by at most a single transaction (i.e., the transactions' write-sets are disjoint), the local schedules could be produced by SI protocol.

(b) (i) A serializable global schedule exists. Global serial order is (T_2, T_1, T_3); a serial order for S_A is (T_2, T_1, T_3); the only serial order for S_B is (T_2, T_1); and the only serial order for S_C is (T_1, T_3).

(ii) In S_B, for $W_1(d)$ to be executed, T_2 would need to release its shared lock on d, but this would imply that T_2 can't later request for an exclusive lock for $W_2(d)$. Hence, the local schedules are not generated by S2PL protocol.

(iii) Same answer as (a)(iii).