Fall 2007 CS186 Discussion Section: Week 14, 11/26 - 11/30

Your friendly TAs

November 25, 2007

1 Transactions and Concurrency Control

1. **In general, is it possible to have a deadlock when the regular two-phase-locking (i.e., non-strict) protocol is obeyed? If yes, give an example; if not, explain briefly. What happens with strict 2PL and conservative 2PL?**

   Yes. For example, consider the following schedule that deadlocks under 2PL:

   ```
   T1: X-Lock(A)W(A)              X-Lock(B) ...
   T2:              X-Lock(B)W(B)          X-Lock(A) ...
   ```

   This schedule is allowable under strict 2PL as well, so strict 2PL also has the deadlock problem. Conservative 2PL requires that all locks to be used during the transaction be acquired before the transaction starts. If any one lock cannot be acquired, none of the locks are acquired. So, conservative 2PL does avoid the deadlock problem. On the other hand, DBMSs do not generally use conservative 2PL for avoiding deadlock for performance reasons: all data that MAY be accessed by the transaction must be locked, which greatly reduces the concurrency possible. Instead one of the other many solutions for avoiding deadlocks (e.g., looking for cycles in wait-for graphs) is used.

2. **For each of the following schedules:**

   (a) **Sa = r1(A); w1(B); r2(B); w2(C); r3(C); w3(A);**

   (b) **Sb = r1(A); r2(A); r1(B); r2(B); r3(A); r4(B); w1(A); w2(B);**

   Answer the following questions:

   (a) **What is the precedence graph for the schedule?**
   For the first schedule, the precedence graph: T1→T2, T2→T3, T1→T3. For the second: T2→T1, T3→T1, T1→T2, T4→T2.

   (b) **Is the schedule conflict-serializable? If so, what are all the equivalent serial schedules?**
   For the first graph, yes; equivalent schedules: T1→T2→T3. For the second, the answer is no; there are cycles in the precedence graph (T2→T1, T1→T2).

3. **Consider the following two transactions: T1 = w1(C) r1(A) w1(A) r1(B) w1(B); T2 = r2(B) w2(B) r2(A) w2(A). Say our scheduler performs exclusive locking only (i.e., no shared locks). For each of the following three instances of transactions T1 and T2 annotated with lock and unlock actions, say whether the annotated transactions:**

   (a) **obey two-phase locking,**

   (b) **will necessarily result in a conflict serializable schedule (if no deadlock occurs),**

   (c) **will necessarily result in a recoverable schedule (if no deadlock occurs),**

   (d) **will necessarily result in a schedule that avoids cascading rollback (if no deadlock occurs),**

   (e) **will necessarily result in a strict schedule (if no deadlock occurs),**

   (f) **will necessarily result in a serial schedule (if no deadlock occurs), and**

   (g) **may result in a deadlock.**

- **T1 = L1(C) w1(C) L1(A) r1(A) w1(A) L1(B) r1(B) w1(B) *Commit* U1(A) U1(C) U1(B)**
  T2 = L2(B) r2(B) w2(B) L2(A) r2(A) w2(A) *Commit* U2(A) U2(B)
- **T1 = L1(B) L1(C) w1(C) L1(A) r1(A) w1(A) r1(B) w1(B) *Commit* U1(A) U1(C) U1(B)**
  T2 = L2(B) r2(B) w2(B) L2(A) r2(A) w2(A) *Commit* U2(A) U2(B)
- **T1 = L1(C) L1(A) w1(C) r1(A) w1(A) L1(B) r1(B) w1(B) U1(A) U1(C) U1(B) *Commit***
  T2 = L2(B) r2(B) w2(B) L2(A) r2(A) w2(A) *Commit* U2(A) U2(B)

	2PL	Necessarily Conflict Serializable	Necessarily Recoverable	Necessarily Avoid Cascading Abort	Necessarily Strict Schedule	Necessarily Serial Schedule	May Result in Deadlock
(a)	Y	Y	Y	Y	Y	N	Y
(b)	Y	Y	Y	Y	Y	Y	N
(c)	Y	Y	N	N	N	Y*	Y

Figure 1: Answer of question 3.

Format your answer in a table with Yes/No entries.

(*) In terms of this question, a serial schedule is one where the reads and writes of each transaction (not the locks) are run serially. If all the reads/writes of this transaction are not run serially, then there will be a deadlock (Can you see why?) Look at the first locks requested by each transaction.

As a general comment to help interpret the above table, we know from our theory that every schedule that obeys the string 2PL protocol will be conflict serializable, recoverable and cascading-abort free.

For the third instance of T1 and T2, here's an example of their interleaving that exposes why there might be schedules involving them that might be non recoverable and suffer from cascading aborts. Consider that T1 executes up to (but excluding) the *Commit* statement, then T2 executes and commits, releasing all the locks and then, for some reason, the Transaction Manager decides to abort T1. In that case, T2, although committed, has to be "undone" (non-recoverable schedule). For the cascading aborts problem, consider T1 reaching the *Commit* statement as before, T2 executing up to, but excluding its own *Commit*, and then T1 aborts. T2 has to abort also.

4. **Examine the schedule given below. There are four transactions, T1, T2, T3, and T4.**

T1	T2	T3	T4
			R(tax)
R(salary)			
			W(tax)
	R(tax)		
	W(tax)		
R(tax)			
W(salary)			
		R(salary)	
W(tax)			
		W(salary)	
			R(salary)
			W(salary)

Figure 2: Interleaved transactions of question 4.

(a) **Draw the precedence graph for this schedule.**

(b) **What is the equivalent serialization order for this schedule? If no order is possible, then state 'none'.**

(c) **Assume that transaction T4 did not run at all. What is the precedence graph in this case?**

(d) **What is the equivalent serialization order for this second schedule? If no order is possible, then state 'none'.**

(a) See figure **??**.

(b) None, the conflict graph has a cycle.

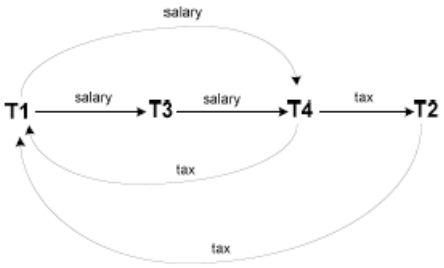

Figure 3: Answer of question 4.

(c) Same as above with T4 removed.

(d) T2;T1;T3