CMPT506 – Advanced Database System
Fall 2016 – Homework 3

Due Thursday 29/12/2016 at 11pm – Submit your softcopy to blackboard.

Exercise 1 [6 points]
Consider the following 2 transactions T1 and T2:

T_1	T_2
read(X);	read(X);
X:=X-10;	X:= X*1.02;
write(X);	read(Y);
read(Y);	Y:= Y*1.02;
Y:=Y+10;	write(X);
write(Y);	write(Y);

a. [2 pts] Give a *serial schedule* for these transactions

T_1	T_2

read(X);
X:=X-10;
write(X);
read(Y);
Y:=Y+10;
write(Y);
	read(X);
	X:=X*1.02;
	read(Y);
	Y:=Y*1.02;
	write(X);
	write(Y);

b. [4 pts] Give an *equivalent conflict-serializable schedule* for these transactions.

T1	T2
read(X) X:=X-10; write(X)	
	read(X); X:=X*1.02
read(Y); Y:=Y+10 write(Y)	

	read(Y);
	Y:=Y*1.02;
	write(X);
	write(Y);

Exercise 2 [12 points]

Consider the following 2 transactions T_1 and T_2:

T_1	T_2

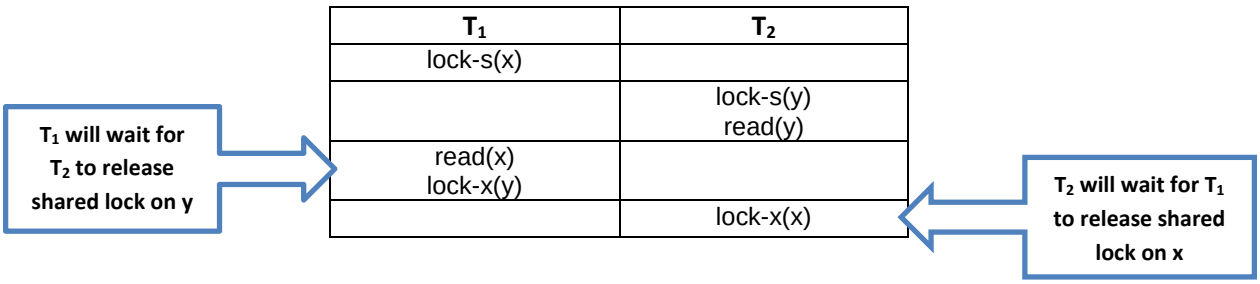
read(X);
read(Y);
if X > 0 then Y := Y + 10;
write(Y);
	read(Y);
	read(X);
	if Y > 0 then X := X + 20;
	write(X);

a. [6pts] Add lock and unlock instructions to transactions T_1 and T_2 such that they conform to the two-phase locking protocol.

T_1	**T_2**
lock-s(x) read(x) lock-s(y) read(y) lock-x(y) if (x>0) y = y+10 write (y) unlock(x) unlock(y)	lock-s(y) read(y) lock-s(x) read(x) lock-x(x) if (y>0) x = x+20 write (x) unlock(x) unlock(y)

b. [6pts] Present a schedule where the execution of these 2 transactions can result in deadlock.

The following schedule will produce deadlock:

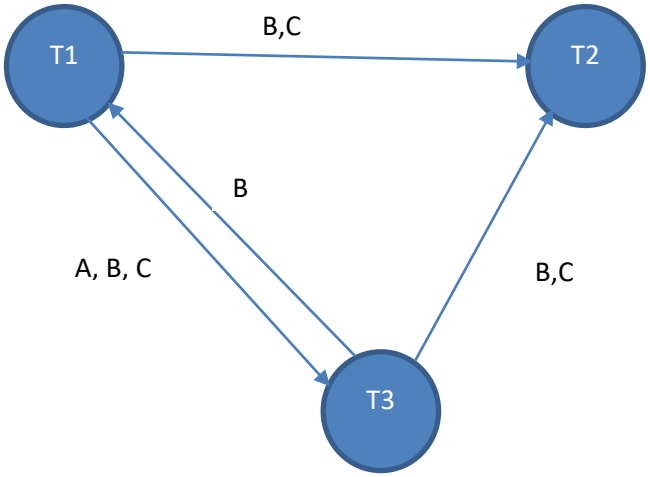


T_1	**T_2**
lock-s(x)	
	lock-s(y) read(y)
read(x) lock-x(y)	
	lock-x(x)

The transactions are now deadlocked.

Exercise 3 [12 points]
Draw the precedence graph for the following schedule and test whether it is conflict serializable or not.

T1	T2	T3
W(A)		
R(B)		
R(C)		
		W(B)
W(B)		
		W(C)
	R(C)	
	W(B)	
	W(C)	
		R(A)



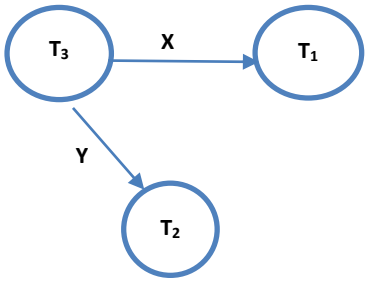
The schedule not conflict serializable because there is a cycle between T1 and T3

Exercise 4 [20 points]
Consider the three transactions T1, T2, and T3, and the schedules S1 and S2 given below. Draw the serializibility (precedence) graphs for S1 and S2 and state whether each schedule is serializable or not. If a schedule is serializable, give the equivalent serial schedule (i.e., just write the order of the transactions).

T1: r1(x); r1(z); w1(x)
T2: r2(z); r2(y); w2(z); w2(y)
T3: r3(x); r3(y); w3(y)
S1: r1(x); r2(z); r1(x); r3(x); r3(y); w1(x); w3(y); r2(y); w2(z); w2(y)
S2: r1(x); r2(z); r3(x); r1(z); r2(y); r3(y); w1(x); w2(z); w3(y); w2(y)

(*r* means read operation and *w* means write operation)

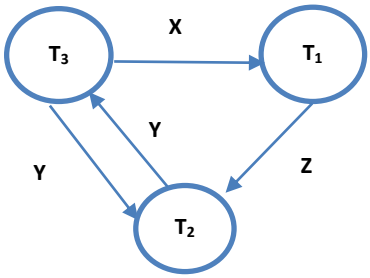
Precedence graph for schedule S_1



S_2 is serializable as it does not have a cycle. There are two possible equivalent serial schedules:

$T_3 \rightarrow T_1 \rightarrow T_2$ or $T_3 \rightarrow T_2 \rightarrow T_1$

Precedence graph for schedule S_2



Schedule S_2 is not serializable as it has a cycle between T_2 and T_3 nodes.

Exercise 5 [20 points]

Consider the following transaction log from the start of the execution of a database system that is capable of running undo/redo logging with checkpointing:
(1) < START T1 >
(2) <T1, A, 55, 20>
(3) <T1, B, 255, 20>
(4) <START T2>
(5) <T1, A, 89, 45>
(6) <T2, C, 40, 20>
(7) <COMMIT T1>
(8) <START T3>
(9) <T3, E, 50, 20>
(10) <T2, D, 50, 20>
(11) <START CKPT (T2,T3)>
(12) <T2, C, 70, 30>
(13) <COMMIT T2>
(14) <START T4>
(15) <T4, F, 105, 20>
(16) <COMMIT T3>
(17) <END CKPT>
(18) <T4, F, 155, 95>
(19) <COMMIT T4>
Suppose the log entries are in the format < T_{id}, Variable, Newvalue, Oldvalue >. What is the value of the data items A, B, C, D, E, and F on disk after recovery:
(1) if the system crashes just before line 10 is written to disk?
(2) if the system crashes just before line 13 is written to disk?
(3) if the system crashes just before line 14 is written to disk?
(4) if the system crashes just before line 19 is written to disk?
(5) if the system crashes just after line 19 is written to disk?

Solution

(1) (A,B,C,D,E,F)=(89,255,20,20,20,20). When the system crashes just before line 10 is written to disk, T1 is committed, and T2 and T3 are incomplete. Thus we undo T2 and T3 and redo T1.

(2) (A,B,C,D,E,F)=(89,255,20,20,20,20). When the system crashes just before line 13 is written to disk, T1 is committed, and T2 and T3 are incomplete. Thus we undo T2 and T3 and redo T1.

(3) (A,B,C,D,E,F)=(89,255,70,50,20,20). When the system crashes just before line 14 is written to disk, T1 and T2 are committed, and T3 is incomplete. Thus we undo T3 and redo T1 and T2.

(4) (A,B,C,D,E,F)=(89,255,70,50,50,20). When the system crashes just before line 19 is written to disk, T1 is committed before START CKPT, thus changes made by T1 must have appear on disk, we can ignore T1. T2 and T3 are committed, and T4 is incomplete, thus we undo T4 and redo T2 and T3 (only for actions after START CKPT, since changes before START CKPT must appear on disk).

(5) (A,B,C,D,E,F)=(89,255,70,50,50,155). When the system crashes just after line 19 is written to disk, T1 can be ignored, and T2, T3, T4 are committed, we redo them (only for actions after START CKPT, since changes before START CKPT must appear on disk).

Question 6 [18 Points]

The table below shows the log corresponding to a particular schedule at the point of a system crash for the four transactions T1, T2, T3, and T4. Suppose that we use the immediate update protocol with check pointing. Describe the recovery process at the database restart after a system crash. Specify which transactions are rolled back, which operations in the log are redone and which (if any) are undone, and whether any cascading rollback takes place. Assume strict 2PL was followed as the concurrency control technique.

[Start_transaction, T1]
[read_item, T1, A]
[read_item, T1,D]
[write_item, T1, D, 20,25]
[Commit, T1]
[checkpoint]
[start_transaction, T2]
[read_item, T2, B]
[write_item, T2, B, 12, 18]
[start_transaction, T4]
[read_item, T4, D]
[write_item, T4, D, 25, 15]
[start_transaction, T3]
[write_item, T3, C, 30, 40]
[read_item, T4, A]
[write_item, T4, A, 30, 20]
[read_item,T2, B]
[write_item, T2, B, 15, 35]
[commit, T3]

← System Crash

Solution:

- Recovery process:
  - Backward pass: rollback T_2 and T_4 as they were incomplete.
  - Forward pass: redo T_3 operations since it committed after the checkpoint.
    (T_1 will be redone since its data changes has been flushed to disk by the checkpoint).
- Backward pass:
  - Undo T_2 operations
    - [write_item, T_2, B, 15, 35] => B will be set to 15
    - [write_item, T_2, B, 12, 18] => **B will be set to 12**
  - Undo T_4 operations
    - [write_item, T_4, A, 30, 20] => A will be set to 30
    - [write_item, T_4, D, 25, 15] => D will be set to 25
- Forward pass:
  - Redo T_3 operations
    - [write_item, T3, C, 30, 40] => C will be set to 40
- There are no cascading rollbacks because **strict 2PL produces cascadeless schedules**.

Another valid answer: There are no cascading rollbacks because T_1 committed its changes before the start of the aborted T_2 and T_4 transactions. Also, T_3 did not read any of data changed by the aborted transactions.

Exercise 7 [12 points]

Consider a distributed database transaction processing system that manages data replicated across two or more servers. A transaction that updates a data item must change all replicas to ensure that after a transaction is committed all replicas has exactly the same data.

Present and explain scenarios of violation of the ACID properties on such a system.

Solution:

Scenarios of violation of the ACID properties

- Atomicity: A transaction updates one replica and then crashes before it can update the others, but the update to the first replica is not rolled back.
- Consistency: A transaction updates one replica but failed to update others, thus violating the consistency constraint that all replicas have the same value.
- Isolation: One transaction updates two different items, each of which is replicated; Another transaction sees one replica before change, the other after change.
- Durability: Two replicas are updated; one site crashes and lost the update.