

COMP90050 Advanced Database Systems: Tutorial
Winter term, 2023 (Week 4)

Exercises

Part 1

1. Given the operations for a transaction T1 below, please list the lines that this transaction is executing that cannot happen with two-phase locking. Briefly explain.

1	Slock(A)
2	Read(A)
3	Unlock(A)
4	Slock(B)
5	Read(B)
6	Unlock(B)
7	Xlock(C)
8	Write(C)
9	Unlock(C)
10	Xlock(A)
11	Write(A)
12	Unlock(A)

2. Discuss why two-phase locking guarantees serializability.

Solution:

Review lecture slides. You will see that with two-phase we want to prevent transactions from getting in-between each other and causing cycles in the dependencies basically.

3. The following transactions are issued in a system at the same time. Answer for both scenarios.

(a) Scenario 1: When the value of A is 3, which of the following transactions can run concurrently from the beginning till commit (that is, all operations and locks are compatible to run concurrently with another one) and which ones need to be delayed? Please give explanation for the delayed transactions.

(b) Scenario 2: When the value of A is 2, which of the following transactions can run concurrently from the beginning till commit (that is, all operations and locks are compatible to run concurrently with another one) and which ones need to be delayed? Please give explanation for the delayed transactions.

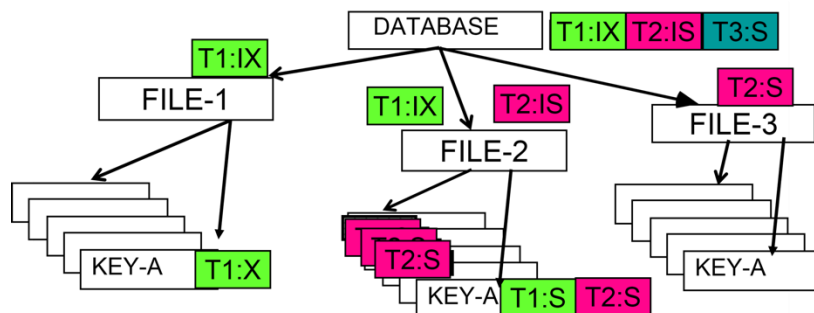
	T2	T3
	Lock (U,A)	Lock (IX,A)
T1	Read A	Read A
Lock (S,A)	if(A ==3) {	if(A ==3){
Read A	Lock(X,A)	Lock(X,A)
Unlock A	Write A	Write A
	}	}
	Unlock A	Unlock A

Solution

Scenario 1: None of the transactions can run concurrently. T2's update lock and T3's IX conflict with each other, and the subsequent X locks for A=3 will conflict. T3's IX lock conflicts with T1's S lock. Although T1's shared lock and T2's update lock are compatible if T1 gets the lock first, for A=3, T2's X lock request will conflict with T1's shared lock. So only one can run at a time while the other transactions must be deDlaged.

Scenario 2: When A is 2, T2 and T3 will not request exclusive lock. Hence, T1 and T2 can run concurrently as T2's update lock can be granted if T1 gets the shared lock on A first. T1 and T3 cannot run concurrently - one of them must be delayed as the locks are not compatible. T2 and T3 cannot run concurrently - one of them must be delayed as the locks are not compatible.

4. Review the concepts of granular locks then answer the following question. Given the hierarchy of database objects and the corresponding granular locks in the following picture, which transactions can run if the transactions arrive in the order T1-T2-T3? What if the order is T3-T2-T1? Note that locks from the same transaction are in the same colour. We assume that the transactions need to take the locks when they start to run.



Solution:

If the order of the arrival of transactions is T1-T2-T3, then T1 and T2 can run in parallel while T3 waits. This is because

T1's IX lock at the root node is not compatible with T3's S lock at the same node.

If the order is T3-T2-T1, then T3 and T2 can run while T1 waits. This is due to the similar reason as above. This example shows that the order of transactions can be a deciding factor of the set of parallel-running transactions. We should also note that granular locks can lead to the delay of transactions at any level in the hierarchy below the root node, e.g., a transaction may need to wait for a lock at the FILE-3 node or a KEY-A node due to lock compatibility issues.

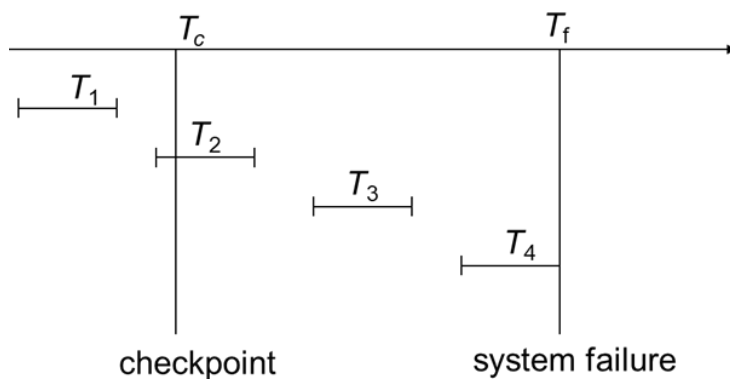
5. With two-phase locking we have already seen a successful strategy that will solve concurrency problems for DBMSs. Then discuss why someone may want to invent something like Optimistic Concurrency control in addition to that locking mechanism.

Solution:

Two-phase locking or in general locking assumes the worst, i.e. there are many updates in the system and most of them will lead to conflicts in access to objects. This means there is good rationale to pay the overhead of locking and stop problems from occurring in the first place. But what if the DBMS is one such that people tend to work on different parts of data, or most of the operations are read operations, and as such there aren't many conflicts at all. Then there is no need to pay the overhead of lock management but rather it may be better to allow transactions run freely and have a simple check when they finish whether there was any conflict with concurrent transactions. Most of the time there will not be so one will get increased concurrency with less overheads. Obviously, the reverse is also true, i.e., if there were really many conflicting writes then doing optimistic concurrency control means many problems would be observed only after running the transactions and a lot of work will need to be wasted to preserve consistency of the data. So there is no clear answer but depending on the situation a strategy may be good or bad.

Part 2

1. In the following figure the first vertical line T_c denotes the point where checkpointing was done and the second on the right, T_f , is where a system crash occurs. Please discuss what would change if the checkpointing was done right at the beginning of each transaction instead of the following case in the figure.



Solution:

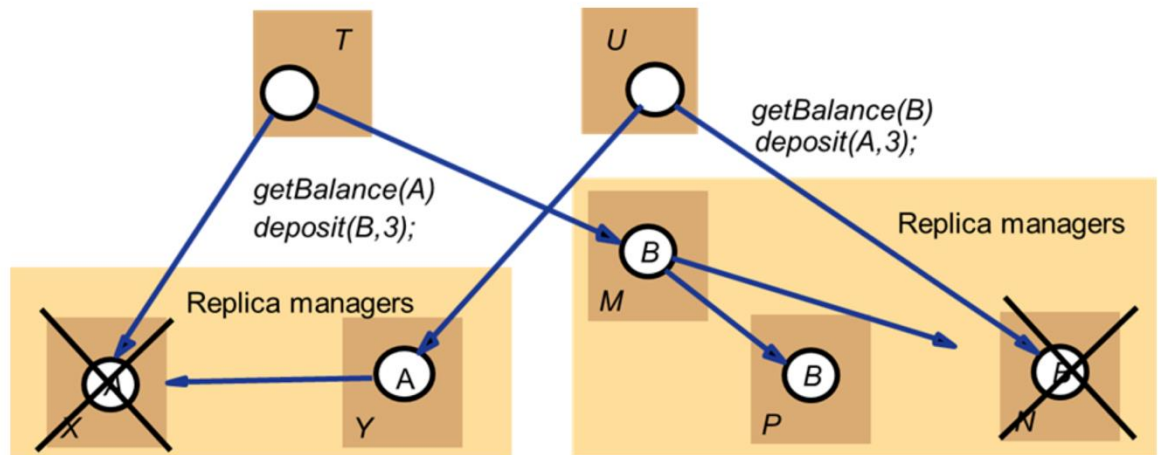
If we were to do checkpointing at the beginning of each transaction, then after a system failure there would be much less to do during recovery time. This is because unlike the case above there would be much less transactions to consider for redo/undo. T3 for example would not be considered if at the beginning of T4 we did a checkpoint. Thus, with more checkpointing recovery becomes fast. Having said this, this does not come for free. There would be many output operations to the disk for each checkpoint as well as entries to the log regarding checkpointing. Thus, the cost is during transaction processing there would be more overheads. One needs to consider the frequency of checkpointing carefully. There is no one good frequency and it is deployment dependent. For systems where immediate recovery is utmost importance then frequency can be increased. For systems where transaction processing should be done fast, but recovery can take a long time, less checkpointing should be considered.

2. Assume a two-phase commit involves a coordinator and three participants, P1, P2 and P3. What would happen in the following scenarios?

- Scenario 1: P1 and P2 voted yes and P3 voted no.
- Scenario 2: P2 crashed when it was about to send a vote message to the coordinator.

Solution:

- a. Scenario 1: coordinator asks all participants to rollback. Abort logs are forced to disk at coordinator and all participants. (Because, to achieve atomicity, all the participants should commit or none will commit.)
 - b. Scenario 2: The coordinator will try to get a response within the timeout period. If the coordinator cannot receive the vote from P2 within the timeout period, it will abort the transaction and ask all participants to rollback. Abort logs will be forced to disk at coordinator and all participants.
3. Given the two following transactions T and U that run on replica managers X, Y, M, P, and N, review the problem that would occur if X and N were to crash during execution. Then state the solution that we have seen in the lecture. Discuss what would happen, if rather than X and N becoming unavailable we have the following scenario: If Y were to become unavailable during the execution and right after U accessed A at Y, but X and N do not fail, rest of the assumptions of this scenario is the same as we discussed in class.



Solution:

For the first discussion part, please refer to lecture slides. Please review them and see that if there's any changes in the set of available and unavailable servers during the execution of a transaction, that transaction should terminate. Now for the case where Y fails instead of others. The scenario is different. We wish that: U can lock Y and is delayed at X as A is locked by T there, similar to the previous scenario. On M, P, and N either U gets the lock first and T waits, in which case there is a deadlock which will be resolved by a timer, or T locks on N as well and U also waits there and T finishes first and N later. In any case, as you see there is no problem as T and U should have been able to run without causing any harm. Nevertheless, the executions above will not occur, as seeing Y is gone U will self-terminate based on the rule. As you can see, the implementation of the strategy that we have seen, and in fact many strategies in DBMSs, are only approximations and in general pessimistic ones so that they guarantee proper executions but sometimes terminate transactions that would not really cause harm.

ARIES Examples (from <https://inst.eecs.berkeley.edu/~cs186/sp08/aries.html>)

Example 1:

Log:

After a crash, we find the following log:

0	BEGIN CHECKPOINT
5	END CHECKPOINT (EMPTY XACTTABLE AND DPT)
10	T1: UPDATE P1 (OLD: YYY NEW: ZZZ)
15	T1: UPDATE P2 (OLD: WWW NEW: XXX)

20	T1: COMMIT
----	------------

Analysis phase:

Scan forward through the log starting at LSN 0.

LSN 5: Initialize XACT table and DPT to empty.

LSN 10: Add (T1, LSN 10) to XACT table. Add (P1, LSN 10) to DPT.

LSN 15: Set LastLSN=15 for T1 in XACT table. Add (P2, LSN 15) to DPT.

LSN 20: Change T1 status to "Commit" in XACT table

Redo phase:

Scan forward through the log starting at LSN 10.

LSN 10: Read page P1, check PageLSN stored in the page. If PageLSN<10, redo LSN 10 (set value to ZZZ) and set the page's PageLSN=10.

LSN 15: Read page P2, check PageLSN stored in the page. If PageLSN<15, redo LSN 15 (set value to XXX) and set the page's PageLSN=15.

Undo phase:

Do nothing; no transactions to undo.

Example 2:

Log:

After a crash, we find the following log:

0	BEGIN CHECKPOINT
5	END CHECKPOINT (EMPTY XACT TABLE AND DPT)
10	T1: UPDATE P1 (OLD: YYY NEW: ZZZ)
15	T1: UPDATE P2 (OLD: WWW NEW: XXX)
20	T2: UPDATE P3 (OLD: UUU NEW: VVV)
25	T1: COMMIT
30	T2: UPDATE P1 (OLD: ZZZ NEW: TTT)

Analysis phase:

Scan forward through the log starting at LSN 0.

LSN 5: Initialize XACT table and DPT to empty.

LSN 10: Add (T1, LSN 10) to XACT table. Add (P1, LSN 10) to DPT.

LSN 15: Set LastLSN=15 for T1 in XACT table. Add (P2, LSN 15) to DPT.

LSN 20: Add (T2, LSN 20) to XACT table. Add (P3, LSN 20) to DPT.

LSN 25: Change T1 status to "Commit" in XACT table

LSN 30: Set LastLSN=30 for T2 in XACT table.

Redo phase:

Scan forward through the log starting at LSN 10.

LSN 10: Read page P1, check PageLSN stored in the page. If PageLSN<10, redo LSN 10 (set value to ZZZ) and set the page's PageLSN=10.

LSN 15: Read page P2, check PageLSN stored in the page. If PageLSN<15, redo LSN 15 (set value to XXX) and set the page's PageLSN=15.

LSN 20: Read page P3, check PageLSN stored in the page. If PageLSN<20, redo LSN 20 (set value to VVV) and set the page's PageLSN=20.

LSN 30: Read page P1 if it has been flushed, check PageLSN stored in the page. It will be 10. Redo LSN 30 (set value to TTT) and set the page's PageLSN=30.

Undo phase:

T2 must be undone. Put LSN 30 in ToUndo.

Write Abort record to log for T2

LSN 30: Undo LSN 30 - write a CLR for P1 with "set P1=ZZZ" and undonextLSN=20. Write ZZZ into P1. Put LSN 20 in ToUndo.

LSN 20: Undo LSN 20 - write a CLR for P3 with "set P3=UUU" and undonextLSN=NULL. Write UUU into P3.

Example 3:

Log:

After a crash, we find the following log:

10	T1: UPDATE P1 (OLD: YYY NEW: ZZZ)
15	T2: UPDATE P3 (OLD: UUU NEW: VVV)
20	BEGIN CHECKPOINT
25	END CHECKPOINT (XACT TABLE=[[T1,10],[T2,20]]; DPT=[[P1,10],[P3,15]])
30	T1: UPDATE P2 (OLD: WWW NEW: XXX)
35	T1: COMMIT
40	T2: UPDATE P1 (OLD: ZZZ NEW: TTT)
45	T2: ABORT
50	T2: CLR P1(ZZZ), undonextLSN=15

Analysis phase:

Scan forward through the log starting at LSN 20.

LSN 25: Initialize XACT table with T1 (LastLSN 10) and T2 (LastLSN 20). Initialize DPT to P1 (RecLSN 10) and P3 (RecLSN 15).

LSN 30: Set LastLSN=30 for T1 in XACT table. Add (P2, LSN 30) to DPT.

LSN 35: Change T1 status to "Commit" in XACT table

LSN 40: Set LastLSN=40 for T2 in XACT table.

LSN 45: Change T2 status to "Abort" in XACT table

LSN 50: Set LastLSN=50 for T2 in XACT table.

Redo phase:

Scan forward through the log starting at LSN 10.

LSN 10: Read page P1, check PageLSN stored in the page. If PageLSN<10, redo LSN 10 (set value to ZZZ) and set the page's PageLSN=10.

LSN 15: Read page P3, check PageLSN stored in the page. If PageLSN<15, redo LSN 15 (set value to VVV) and set the page's PageLSN=15.

LSN 30: Read page P2, check PageLSN stored in the page. If PageLSN<30, redo LSN 30 (set value to XXX) and set the page's PageLSN=30.

LSN 40: Read page P1 if it has been flushed, check PageLSN stored in the page. It will be 10. Redo LSN 40 (set value to TTT) and set the page's PageLSN=40.

LSN 50: Read page P1 if it has been flushed, check PageLSN stored in the page. It will be 40. Redo LSN 50 (set value to ZZZ) and set the page's PageLSN=50.

Undo phase:

T2 must be undone. Put LSN 50 in ToUndo.

LSN 50: Put LSN 15 in ToUndo

LSN 15: Undo LSN 15 - write a CLR for P3 with "set P3=UUU" and undonextLSN=NULL. Write UUU into P3.

----- Discussion, Q/A on any topics of the subject -----