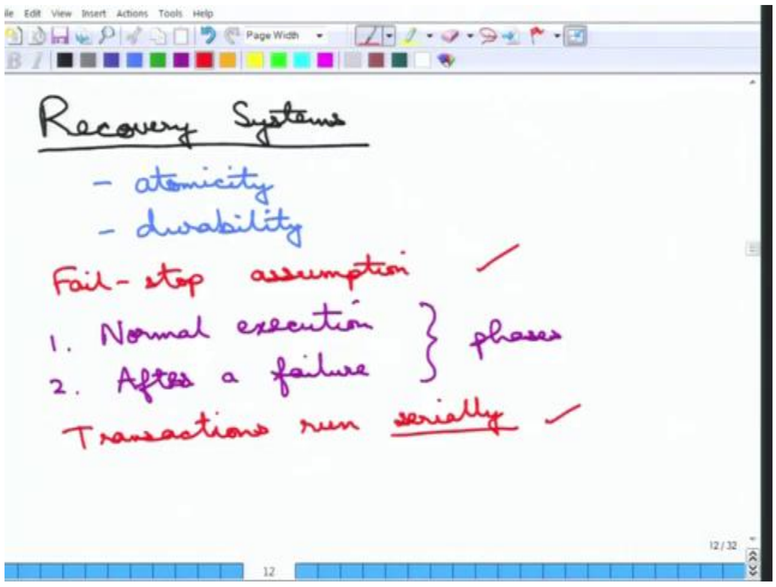
Fundamentals of Database Systems

Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 30
Recovery Systems: Deferred Database Modification

So, welcome, we will continue with the Database Transactions. Last time, we saw an introduction to what the database transactions are, what are their properties and what they are used for. Today, we will go over one area of the database transactions in much more detail, which is the recovery system.

(Refer Slide Time: 00:29)



Recovery Systems

- atomicity
- durability

Fail-stop assumption ✓

1. Normal execution
2. After a failure

} phases

Transactions run serially ✓

So, today's topic is on recovery systems. So, essentially, if a transaction fails or something ((Refer Time: 00:37)) happens, how does the database recover. So, the recovery management system essentially the two properties that it tries to maintain is the atomicity and the durability. So, these are the two important things that the recovery management system targets and there is an atomicity and durability as we will be covered, but there is an assumption, there is an important assumption, which is called a Fail-stop assumption. This is, the recovery management systems make this thing, this is called a Fail-stop assumption, because what it is being assume in this Fail-stop assumption is the data; that is on a non-volatile storage is never lost.

So, what is the non-volatile storage? The magnetic disks, etcetera, it is never lost. Now, one may argue that how can it be, that it is never lost, because the magnetic disk may be corrupted, may be broken, may be harm, lost, etcetera, what happens. The assumption for recovery management system is that, there will be adequate number of backups of the non-volatile storage, data in the non-volatile storage. So, that can be always retrieved when necessary.

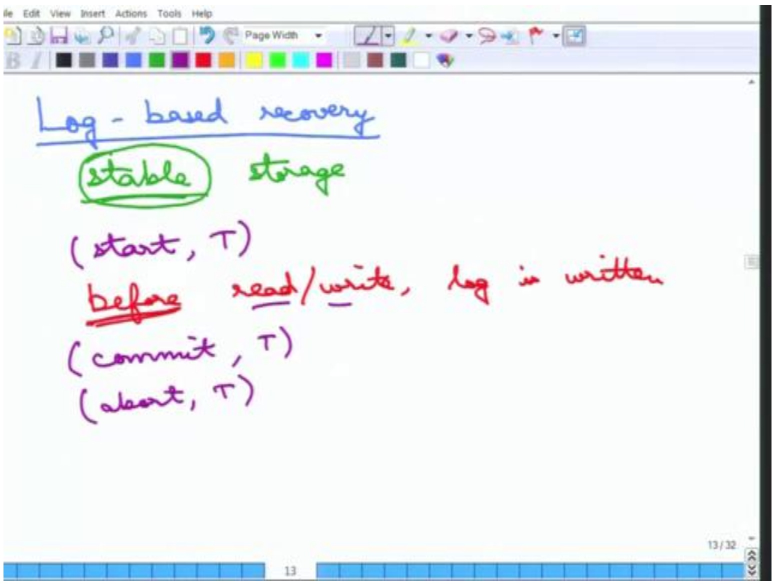
So, that is the Fail-stop assumption, is that the data on the non-volatile storage is not lost due to any problems, system crash, power failure, etcetera, nothing can be lost there. So, that is the thing and the recovery management systems have two main parts. The first one is action taken during the normal execution, when a transaction is running normally; that is during the thing and then, action taken after a failure.

Now, note that, why is the second part important is that recovery systems, so that word recovery means that, something has gone wrong. So, that is why you need to recover from that fault. So, that is why, there is a failure, failure is assumed and then, that needs to be recover from that. If there is no failure, then nothing needs to be done, every transaction runs successfully and then, the recovery management system is not invoked at all.

But, the point to study is that, when there is a failure that happens. So, these are the two important phases of the recovery management systems. So, these can be consider the two phases of a recovery management system and one other assumption that we do when we study this recovery management system is that the transactions run serially, that is there are no concurrent transactions.

Again, this may sound a little non-ideal that how can this happen, but as we will see that the argument about the correctness or the algorithmic views of the recovery systems, do not depend on how the transactions are concurrent, etcetera. So, for simplicity, we just assume that the transactions run serially. So, these two are the assumptions of a recovery management system and this is essentially a recovery management system will try to argue about a single transaction and what happens, we need to fails.

(Refer Slide Time: 03:40)


Log - based recovery

stable storage

(start, T)

before read/write, log is written

(commit, T)

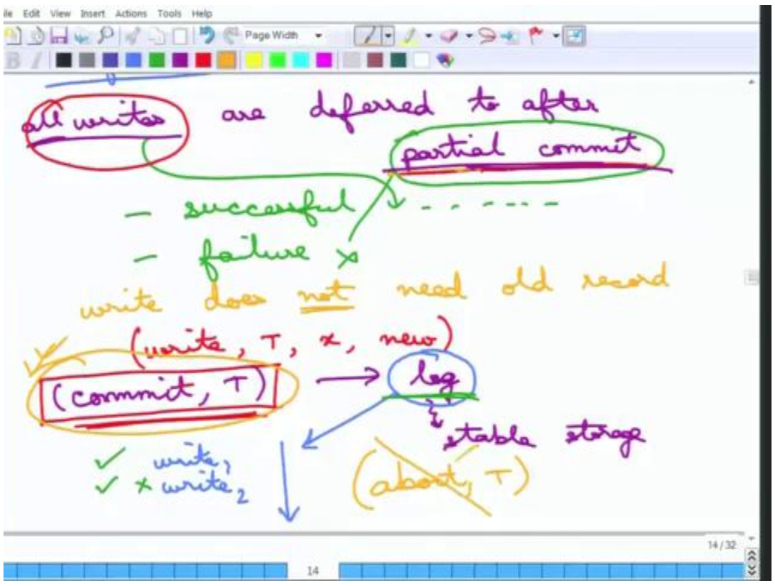
(abort, T)

So, we do this Log based recovery first, the logs or the records that are return and as we saw last time, there are five types of entries that are made for a log, for a transaction and so the log is maintained on a stable storage. So, log is maintained on a stable storage, this is… What is the stable storage? This is essentially the secondary storage or the hard drive, etcetera and again, the stable storage the idea is that, the log is never lost.

So, the stable storage meaning one something is stored into that stable storage, because of it is log stable; that means, it is never lost. So, lost can be always retrieved; that is the idea, so that is the Fail-stop assumption idea. And then, there are log records as we showed last time. So, whenever, just to recover, whenever a transaction starts, there is a start T; that log record entries made before a read or write is done, the corresponding entries made in the log.

So, this is very importantly before the actual read or write, before the read or write is done, log is updated. So, log is written before an actual read or write on the data item is ((Refer Time: 04:51)). So, this is before that is very important and then, if the transaction commits, then successfully, then there is a commit T, otherwise there is an abort T, so these are the five operations that is read. So, now, there are two types of approaches based on these log based.

(Refer Slide Time: 05:17)



all writes are deferred to after partial commit

– successful ↓ - - - - - -

– failure ×

write does not need old record

(write, T, x, new)

(commit, T) → log

stable storage

✓ write$_1$

✓ × write$_2$

~~(abort, T)~~

So, there are two ways of a recovery, the first one is called a deferred database modification, so this is called a deferred database modification, this is the recovery management system based on logs. So, what is the deferred database modification is that, important point here is that, all writes are deferred to after the partial commits. So, writes, all writes, I should say, all writes are deferred till the point of partial commits.

So, no write is actually made to the database. So, the transaction thus notes, what the writes are needed to be done, but these are not actually written to the database and only when the transaction partially commits, then these writes happened. Now, you may remember what a partial commit is, a partial commit for a transaction happens, when the transaction has finished all the operations in it is successfully.

So, now, suppose what may happen either two cases may happen is that, all the things are successful, everything is being successful in the transaction. Then, these writes may then go through one after another, first ((Refer Time: 06:31)) all those things may go through or there is some problem, there is a failure. Now, if there is a failure, then this partial commits stage is not reached at all, but now you see the state of the database if there is a failure.

Because, it has not reach the partial commit, none of the writes are actually gone to the database. So, the database is in the state that before the transaction was started. So, nothing has been changed into the database. So, nothing needs to be done actually, because the transaction just aborts none of the database… So, all the writes are on to some temporary storage, may be in the main memory or somewhere else, nothing in the databases actually being updated, because the transaction has not reach the partial commit, because there may be some failures, so nothing needs to be done.

So, essentially what all these means is the something very interesting is that, the write record does not need the old value, write does not need old value. Why is it? Because, it will never need to undo, because only the new values need to be recorded, because when there is a transaction write, when it is committed partially; that means, all the transactions in all the operations you need are successful.

Otherwise, there is no need to go back to the old value, because the database already contains the old value. So, it is not needed, so the entries for this deferred database modification, the write entries are of this one, it is simply write the transaction, this says what is the data item and the new value, that is it. So, the old value is not needed. Now, what happens is that, after the transaction partially commits, there commit T entry is return to the log.

So, commit T is return to the log and this log is then stored to stable storage, this log is then flushed to stable storage. So, every record in the log is then stored to the stable storage and then, there are two things. So, now, what may happen is again, there are two cases. So, what will, a thing is that, so once the committee has been written; that means that the transaction is supposed to finish correctly.

So, that means, now what is happening now, if the transaction actually finishes correctly, the database should have the effect of all the writes that the transaction deep. And now how does one get all the effects from the log? So, the database actually collects all those new write values from the log and just keeps on doing the writes. So, this is the write, the first write operation there, then the second write operation there and so on and so forth, it just keeps doing it.

And again, so if all these writes are successful, then that is it, nothing needs to be done; the databases make sure that every write has gone through. Otherwise, what may happen is that, during this write may be there is a problem, there is a problem, because there is a system crash, there is a failure, etcetera, etcetera. Now, what needs to be done then?

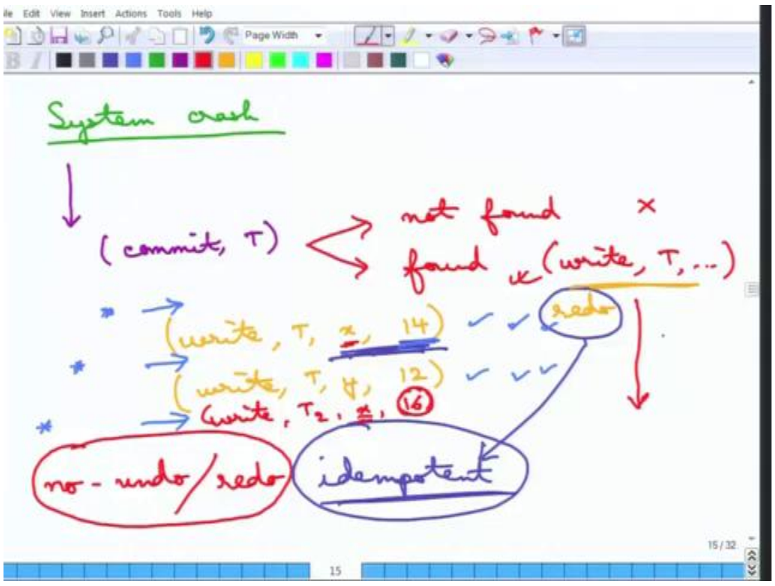
Nothing really, because the next time the system comes up, the system will check this log and will attempt these writes again.

So, it will keep attempting this log and till, it is essentially successful. Now, one important thing here is that, the couple of things. First of all, this commit is return to the log after the partial commit is being done successfully. Now, the abort T entry is again not needed, just like the old value is not needed in a deferred database modification, the abort entry for the transaction T is also not needed.

Why is that? The abort is not needed, because when does an abort happened, when a transaction fails, there is some failure. Now, when the transaction fails, it simply stops doing anything else, it simply restarts; that is it. Why is that correct? Because, the transaction has not yet changed anything into the database and when does the transaction actually starts changing anything to the database, only after the commit T entry has been made.

Now, this means that, the everything in the transaction has gone through successfully. So, the transaction does not need to abort and therefore, the abort T entry, so this is the point, the abort T entry is never used, it is because, it never return.

(Refer Slide Time: 10:51)



Now, whenever suppose there is a system crash that is happened, so if everything goes on successfully, then there fine. So, suppose there is the system crash happens, what does the database do, then the database of recovery management system, it reads to the log and finds out all the log records and finds out all the transactions for which there is a commit entry, commit T entry.

Now, so these commit T entry, if this is not found, so there are two cases, if this is not found; that means, none of the writes into this transaction have started. So, nothing needs to be done essentially. So, there is nothing to be done. On the other end, if it is found that means, if the transaction has said that, it is committed that means everything has to be successful.

Then, all the writes pertaining to the transaction needs to be done, these needs to be applied and these needs to be applied again and again, so this is called a redo operation, so those writes are needed to be done on the database. Now, the question is the following is that, suppose as part of this transaction there are two write entries, write T x to some values a 14 and then, there is a write T y some other values a 12.

The issue is the transaction has return commit T and there is a system crash. Now, the system crash may happen at this stage or this stage or this stage, they can were assuming these write entries are atomic. So, other this complete write as happens for this. Now, let us take all of these cases one by one, suppose a system crash happened at this stage, now we are doing this redo operation or whatever we are doing, so we are now writing all these two things together, so then that is correct.

Now, the second case is, this has happened at this stage, so that was the first write is gone through and the second write has in gone through. Now, the transaction or the log or the database or any system has no way of knowing, whether it is gone through the x as we need successfully or not. So, it does not clear, if there it is successfully or not, it still goes ahead and writes the new value of x to it.

Now, what may happen is that, x may have been the old value or it may have the new value 14. Now, it does not matter because the new value 14 will be again return to x, which is doing it once more, which is not probably efficient in the sense of doing that operation again and again, but it is correct. So, that is more important, so this is correct and then of course, T is return.

And finally, the third cases both have been return, but the database did not know this, again it does not matter, it is just return again and again. The important point here is the following is that for all of these this redo operation may be done multiple times, even if x has been return, it may be return again and again and again. So, this redo must follow, what is called a, the redo must be, what is called an idempotent, this is an important term.

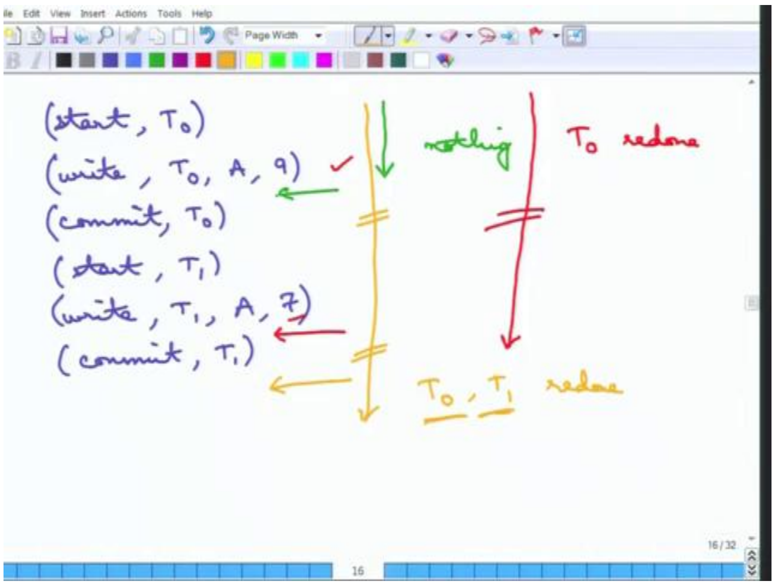
The idempotent means the following is that, the multiple operations has the same effect as a single operation. So, redoing and operation redoing and write operation multiple times as the same effect is doing it once. So, in other words, if the value of 14 is return to x in the example, multiple times it has the same effect of writing 14 to x once and it does not a matter.

So, redo must be idempotent, it must be idempotent, because it may happen that the value 14 is return again and again and again that is all right, so this is called a redo. Now, this redo is needed, but undo is not needed. So, this deferred database modification scheme is sometimes called a no-undo, but redo operation. So, this called a no-undo, but redo recovery scheme. So, this is the turn for the deferred database things.

So, no-undo because undoing is not needed, but read doing is needed. So, this is called no undo or redo recovery scheme and just to complete this, redo must be done in the order that they appear in the lock; otherwise there may be wrong things. And very small example to do that is that, suppose there is another write by some other transaction T 2, which writes x 2 16.

Now, it must happen that the final value is noted, because the x is a common. So, it must be done in this manner. So, 16 should be the final value; that is mean, so redoing must be done in the operation that they appear in the log. Before moving on to the next scheme, let us see an example.

(Refer Slide Time: 15:31)



(start, T_0)
(write, T_0, A, 9) ✓
(commit, T_0)
(start, T_1)
(write, T_1, A, 7)
(commit, T_1)

So, here is a complete example of these things. So, there is a start T 0; that is the first operation, then a write of T 0 is writing a value to some data item A, which is 9, then there is a commit. So, then T 0 commits, then T 1 starts, we are assuming this is the serial, then there is a write of T 1 to the same A with value 7 and then, there is a commit suppose this first what was intended to follow.

And now, suppose there is a crash after the write T 0 statement, so there is a crash at this point. Now, what happens is that the following things, so when the log is searched and there is no transaction that is found with the commits statement. So, nothing needs to be done, because there is nothing committed. So, nothing has been return by any of this transaction to the database and nothing needs to be change.

So, that is done, the next thing is that, suppose there is a crash after this write T 1 statement, then what needs to be done is that, again the log is searched and here this commit T 0 is being found. So, the operations of T 0 need to be redone, so T 0 is redone. So, essentially this write T 0 A to end this operation is being redone, but nothing on T 1 needs to be done, because T 1 has not committed.

So, that means, the value of A should not be 7 and it is not 7, because it has not committed. So, that A value the new value 7 has not gone through A. So, that is fine and finally, may happen that the crash takes after commit T 1 is then. Then, again this entire thing searched and both commit T 0 and commit T 1 is found. So, both T 0 and T 1 are redone and very importantly in the order of T 0 first and then T 1 on the order of the log records. So, redo of T 0 must be done in the first and then, T 1. So, that is the example for this deferred database scheme.