COS221 - L34 - Database Recovery Techniques

Linda Marshall

25 May 2023

Recovery Overview

- Recovery means that the database is restored to its most recent consistent state before time of failure.
- Information about changes to the database state therefore needs to be kept.
- These changes are captured in the System Log.
- Recovery techniques are often intertwined with Concurrency Control. That is certain recovery techniques are best used with certain concurrency control methods.
- Reasons why a transaction fails:
  - Transaction - incorrect I/O, deadlock
  - System - OS fault, RAM failure
  - Media - Disk head crash, power problem

Recovery Strategy

- Catastrophic Failure:- extensive damage to large portion of the database (e.g. disk failure)
  - Restore backed-up copy of the database
  - Recover to just before the failure from the backed up system log (redo)
- Non-catastrophic Failure:- no physical damage to disk (e.g. system crash, transaction/system error, transaction exceptions, violation of serialisability detected by concurrency control method)
  - Identify changes that cause inconsistency
  - Undo write operations
  - Redo operations from online system log to bring the database into a consistent state

Recovery Strategy - Non-catastrophic failure policies

Two main policies are defined for non-catastrophic failure:

- Deferred update: - the database on disk is not updated until after a transaction commits
  - Until the transaction commits, all updates are written to a copy held in main memory buffers and recorded in a log on disk.
  - If a transaction fails before commit, no undo is necessary, only a redo from the log on the current state of the database on the disk.
  - Also known as a NO-UNDO/REDO algorithm
- Immediate update: - the database on disk is updated during transaction execution and before the transaction reaches its commit point
  - Changes to the database items are written to the disk-based log before they are written to the database.
  - If the transaction fails before a commit, the effect of the writes to the disk must be rolled back and may require a redo.
  - Also known as UNDO/REDO or UNDO/NO-REDO algorithms

Recovery Strategy - Non-catastrophic failure policies

- UNDO and REDO are idempotent. That is executing an operation multiple times is equivalent to executing it just once. It is possible for a `write_item` to be undone and redone multiple times
- The recovery of the database state is therefore idempotent. That is:
  "the result of recovery from a system crash during recovery should be the same as the result of recovering when there is no crash during recovery!"

Recovery Process and the OS

The recovery process is often closely linked to operating system functionality. Especially buffering of the database disk pages in the DBMS main memory cache.

- Cache is finite and therefore values in cache may be replaced (flushed).
- Each buffer in the cache has a *dirty bit* associated with it indicating whether the buffer has been modified or not. If it has been modified the contents of the cache must be written back to the relevant disk page.
- Each page in cache is either pinned or unpinned (`pin-unpin` bit). This bit is used to determine whether the page may be written back to disk or not.
- Two strategies exist for flushing a modified buffer back to disk:
  - In-place updating - write the buffer back to the original location on disk. Single copy of the data is kept.
  - shadowing - write the buffer back at a different location on disk. Versions of the data is kept.

Recovery Process - In-place updating

- In-place recovery makes use of the log.
- The log file must be flushed to disk before the Before Image (BFIM) version of the data item is overwritten by the After Image (AFIM) version of the data item on the disk.
- This process is referred to as *write-ahead logging* and is required so that the operation can be UNDOne if required.
- Distinction is made between two types of information required in the log for a write:
  - REDO-type log entry - New value (AFIM) is included
  - UNDO-type log entry - Old value (BFIM) is included
- The UNDO/REDO Algorithm writes both AFIM and BFIM to a single log entry.
- If a cascading rollback is done, read entries in the log are considered to be UNDO-type entries.

Recovery Process - When can a page from the database cache be written to disk?

The rules that specify when the database cache can be written to disk are referred to as:

- steal/no-steal
  - steal approach - cache page updated by a transaction can be written to disk before the transaction commits.
  - no-steal approach - cache page updated by a transaction cannot be written to disk before the transaction commits. That is, UNDO's are never required
- force/no-force
  - force approach - when a transaction commits, all pages updated by the transaction are immediately written to disk. That is, REDO's are never required.
  - no-force approach - when a transaction commits, all pages updated by the transaction are not immediately written to disk.

Recovery Process

- Deferred update (NO-UNDO) follows a no-steal approach
- A typical database applies a steal/no-force (UNDO/REDO) strategy (Immediate update).
  - steal saves on buffer space to store updated pages in memory
  - no-force saves on I/O as pages may still be in memory
  - The Write-Ahead Logging (WAL) protocol saves writes in the log before they are applied to the database.

The Checkpoint Log Entry

- Writes a list of all active transactions into the log
- All transactions that have reached their commit (written to disk) before the checkpoint do not have to have their write operations redone in the event of system failure.
- The recovery manager manages the intervals at which checkpoints are written to the log.
- When taking a checkpoint, the following occurs:
  1. Suspend execution of transactions
  2. Force-write main memory buffers to disk
  3. Write `[checkpoint]` to log and force-write log to disk
  4. Resume transaction execution

Transaction Rollback and Cascading Rollback

- If a transaction fails before it commits it must be rolled back.
- If a transaction is rolled back, then any transaction that has read a data item that was written by the transaction must be rolled back and so on. This is referred to as a cascading rollback.

Transaction Rollback and Cascading Rollback - Example

(a)

T_1	T_2	T_3
read_item(A)	read_item(B)	read_item(C)
read_item(D)	write_item(B)	write_item(B)
write_item(D)	read_item(D)	read_item(A)
	write_item(D)	write_item(A)

(b)

		A	B	C	D
		30	15	40	20
	[start_transaction,T_3]				
	[read_item,T_3,C]				
*	[write_item,T_3,B,15,12]		12		
	[start_transaction,T_2]				
	[read_item,T_2,B]				
**	[write_item,T_2,B,12,18]		18		
	[start_transaction,T_1]				
	[read_item,T_1,A]				
	[read_item,T_1,D]				
	[write_item,T_1,D,20,25]				25
	[read_item,T_2,D]				
**	[write_item,T_2,D,25,26]				26
	[read_item,T_3,A]				

System crash

* T_3 is rolled back because it did not reach its commit point.

** T_2 is rolled back because it reads the value of item B written by T_3.

(c)

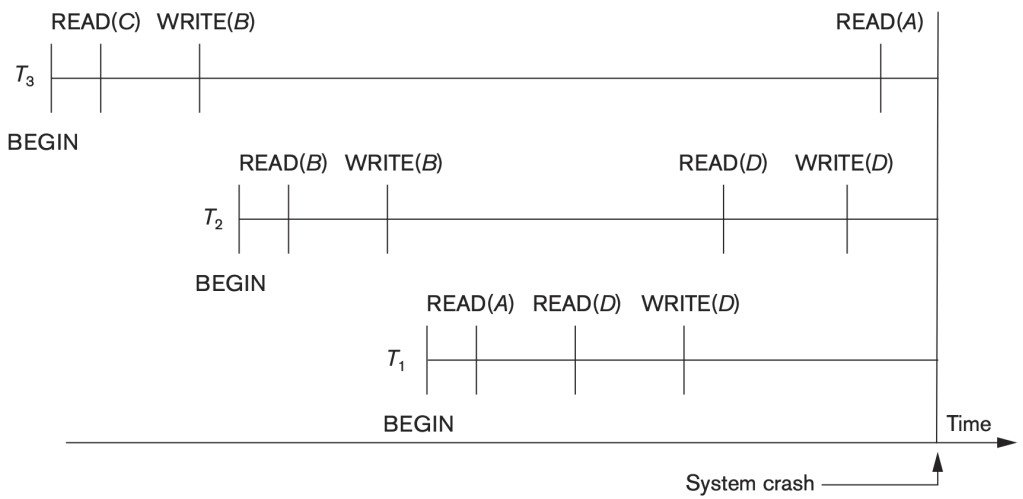


Figure 22.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.

Recovery Based on Deferred Update - NO-UNDO/REDO

- Updates are recorded in the log and cache buffers during transaction execution
- When a transaction reaches a commit point, the logs are force-written to disk and the changes written to the database.
- If a transaction fails before reaching the commit point, there is no need to make changes to the disk version. Only redoing the writes from the log version are required.
- This strategy works well for short transactions.

Recovery Based on Deferred Update - NO-UNDO/REDO

How do checkpoints affect this strategy?

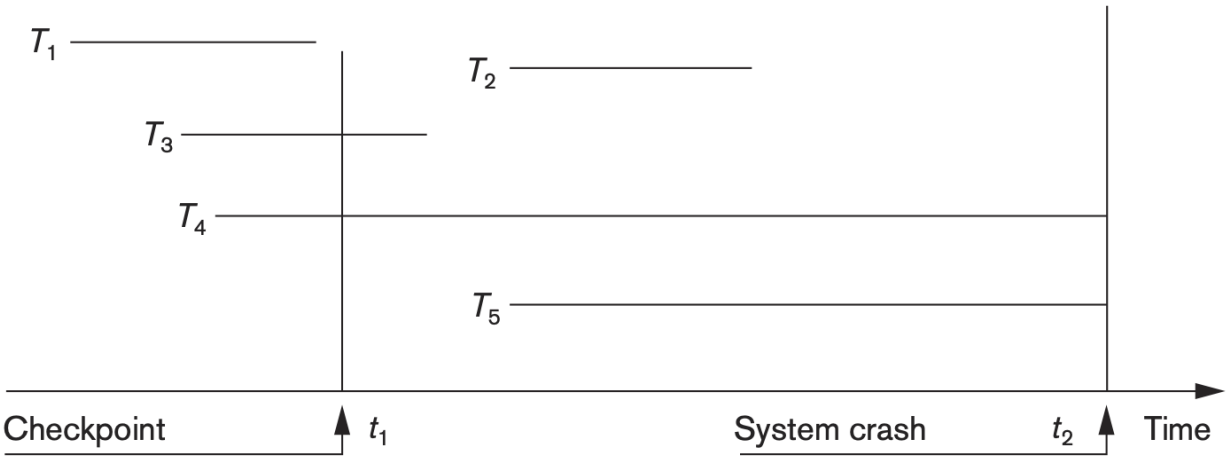


Figure 22.2
An example of a recovery timeline to illustrate the effect of checkpointing.

- T_1 had committed before the checkpoint, T_3 and T_4 had not.
- At system crash T_2 and T_3 had committed and T_4 and T_5 had not.
- No need to redo writes for T_1, writes for T_2 and T_3 must be redone, and T_4 and T_5 are ignored.

Recovery Based on Immediate Update

Remember, there are two possibilities:

1. UNDO/NO-REDO recovery algorithm utilising the steal/force strategy - all updates by a transaction must be recorded on disk before the transaction commits, so that REDO is never needed.
2. UNDO/REDO recovery algorithm utilising the steal/no-force strategy - the transaction is allowed to commit before all its changes are written to the database. This is also the most general case and the most used in practice.

Recovery Based on Immediate Update - UNDO/REDO

- Two lists are kept, one of committed transactions and one of active transactions
- Transactions in the active transaction list are undone in reverse order of how they were written in the log
- Redo all write operations in the order they appear in the log for the committed transactions

Note, this approach when using 2PL is prone to deadlocks.

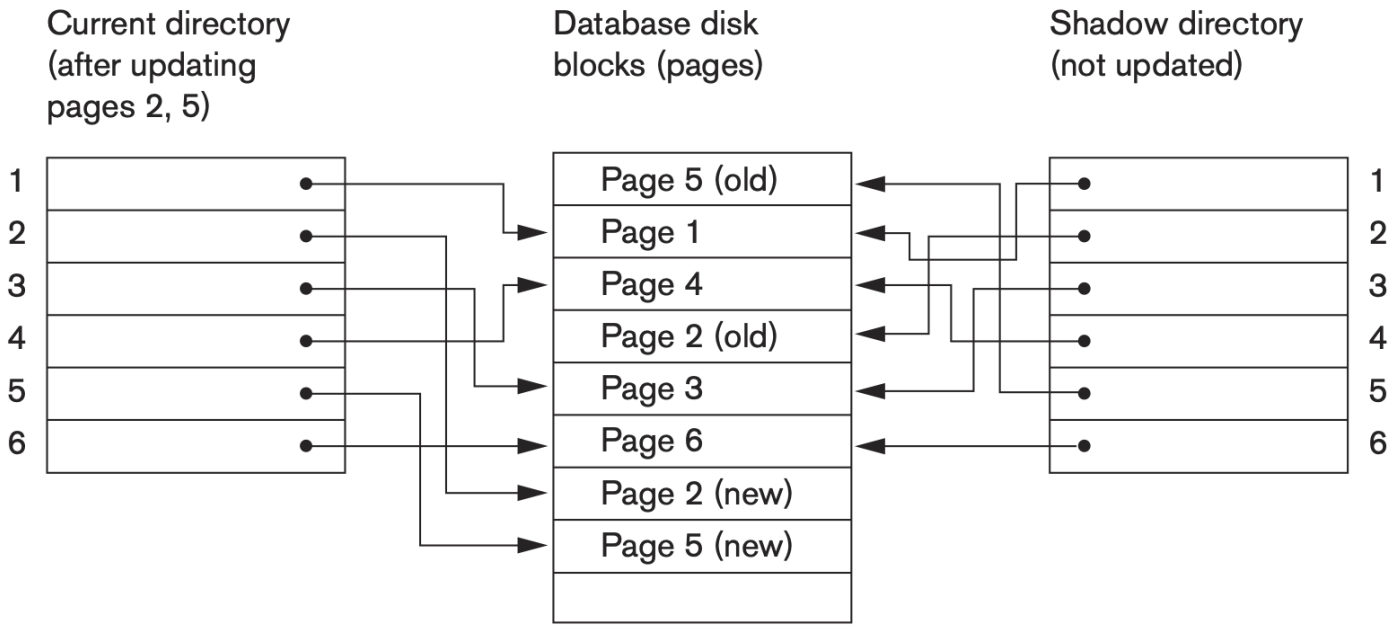
Shadow paging - NO-UNDO/NO-REDO

- A directory of n entries is constructed
- On transaction startup, the current directory points to the most recent data item values from disk and is copied to the shadow directory
- The shadow directory is never modified, only the current directory.
- Modifying an item in the current directory results in a copy of the current directory being made and the change made to this copy - the new current directory.

Note, in a single-user environment a log is not required, in a multiuser environment a log is required for concurrency control.

Shadow paging - NO-UNDO/NO-REDO

Figure 22.4
An example of shadow paging.



The ARIES Recovery Algorithm

- An example of an algorithm used in amongst others, IBM relational database products,
- Uses a steal/no-force approach for writing
- It is based on three concepts
  - write-ahead logging
  - repeating history during redo. That is, all actions of the database system prior to the crash are retraced to reconstruct the database state when the crash occurred. All uncommitted transactions are undone.
  - logging changes during undo. This prevents repeating completed undo operations if failure occurs during recovery.

The ARIES Recovery Algorithm

The ARIES algorithm's recovery consists of 3 steps:

1. Analysis - Identifies the dirty (updated) pages in the buffer and the set of transactions active at the time of the crash, and determines the appropriate point in the log for the REDO step to start.
2. REDO - Reapplies updates from the log to the database. This will include committed transactions from the REDO point identified and then all transactions until the end of the log is reached.
3. UNDO - Scans the log backward and undo the actions of the active transactions in the reverse order.

The ARIES Recovery Algorithm - Analysis step

(a)

Lsn	Last_lsn	Tran_id	Type	Page_id	Other_information
1	0	T_1	update	C	...
2	0	T_2	update	B	...
3	1	T_1	commit		...
4	begin checkpoint				
5	end checkpoint				
6	0	T_3	update	A	...
7	2	T_2	update	C	...
8	7	T_2	commit		...

(b)

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	commit
T_2	2	in progress

DIRTY PAGE TABLE

Page_id	Lsn
C	1
B	2

(c)

TRANSACTION TABLE

Transaction_id	Last_lsn	Status
T_1	3	commit
T_2	8	commit
T_3	6	in progress

DIRTY PAGE TABLE

Page_id	Lsn
C	7
B	2
A	6

Figure 22.5
An example of recovery in ARIES. (a) The log at point of crash. (b) The Transaction and Dirty Page Tables at time of checkpoint. (c) The Transaction and Dirty Page Tables after the analysis phase.

(lsn - log sequence number)

Recovery Techniques - Summary

- *Deferred update*:- NO-UNDO/REDO algorithms
  - Data on the disk is only updated when the transaction commits.
- *Immediate update*:- UNDO/REDO and UNDO/NO-REDO algorithms
  - Data on the disk is updated during transaction execution.
- *Shadowing or shadow paging*:- NO-UNDO/NO-REDO algorithm
  - Keeps history of writes.
- *ARIES algorithm*
  - Includes analysis before REDOing and UNDOing