Chapter 21
Database Recovery Techniques

Recovery Concepts

- **Recovery Outline and Categorization of Recovery Algorithms**
- **Recovery from transaction failures:**
  - the database is **restored** to the most recent consistent state before the time of failure.
  - The system keeps information about the changes that were applied to data items by the various transactions in the **system log**.
  - In case of failure, Just recover the committed transactions as uncommitted transactions doesn’t modify database. So, they will be rolled back

Recovery Concepts

- **Recovery Outline and Categorization of Recovery Algorithms**
- **CATASTROPHIC FAILURE:**
- Extensive damage to database because of disk crash,
- **Solution:**
  - **restores a past copy** of the db. from some archival storage
  - and **redo the operations of committed transactions** from the log file up to failure point to ensure Database consistency.

Recovery Concepts

- **Recovery Outline and Categorization of Recovery Algorithms**
- **NON-CATASTROPHIC FAILURE:**
- When the database on disk is not physically damaged,
  - the recovery strategy is **to identify any changes** that may cause an inconsistency in the database.
- For example,
  - A transaction has not yet committed and is failed,
  - Then it is possible that some of its write operations are done and some are left.
  - So all the operations needs to be undone
  - And redo for the whole transaction is required to ensure atomicity & data consistency

Recovery Concepts

- **Recovery Outline and Categorization of Recovery Algorithms**
- **Recovery from non-Catastrophic failure:**
- Conceptually, we can define 2 policies for recovery from non-catastrophic failures:
  - Deferred update
  - Immediate update

Recovery Concepts

- **Recovery Outline and Categorization of Recovery Algorithms**
- **Deferred update techniques:**
- Do not update the database on disk before COMMIT operation
- Before COMMIT,
  - all changes done by the transactions are saved in local workspace or the Main memory buffers.
  - all the updates operations are recorded in log file, once COMMIT is done, all the updates are written to database

Recovery Concepts

- **Recovery Outline and Categorization of Recovery Algorithms**
- **Deferred update techniques:**
- In case of transaction failure:
  - No need to UNDO the changes done by the transaction as the changes were done locally
  - Just REDO of the operations is required as the changes were not written to database yet.
  - That's why Deferred update is also known as the

NO-UNDO/REDO algorithm.

Recovery Concepts

- **Recovery Outline and Categorization of Recovery Algorithms**
- **Deferred update techniques:**
- **Example: (Log For Deferred Update)**
  - <start_Transaction,T1>
  - <read_item,T1,A>
  - <write_item,T1,A,10>
  - <commit, T1>
- Note: here, we don’t need to store outdated value of data item because in this case we never undo any operation

Recovery Concepts

- **Recovery Outline and Categorization of Recovery Algorithms**
- **Immediate update technique:**
- The database is updated by some operations of a transaction before reaching the COMMIT point
  - i.e., Writing in main memory and disk side by side without waiting for COMMIT operation
  - All operations are forcefully written in log file before COMMIT to make recovery possible.
- If a transaction fails after some operations but before COMMIT
  - the effect of its operations on the database must be undone.
- This technique, known as the **UNDO/REDO** algorithm, requires both operations during recovery

Recovery Concepts

- **Recovery Outline and Categorization of Recovery Algorithms:**
- **Immediate update technique**
- Example: (Log For ImmediateUpdate)
  - <start_Transaction,T1>
  - <read_item,T1,A>
  - <write_item,T1,A,10,11>
  - <commit, T1>
  - Note: here, we store old and new values of data item

Recovery Concepts

- **Caching (Buffering) of Disk Blocks**
- For recovery purposes, buffering of disk blocks is done in DBMS cache
- DBMS cache:
  - Collection of in-memory buffers.
- A **directory** for the cache is maintained to track which **database items are in the buffers**.
- Directory is a table that saves entries as:

  <Disk_page_address,Buffer_location, … >

Recovery Concepts

- **Caching (Buffering) of Disk Blocks**
- Whenever, DBMS requests some item:
  - cache directory is checked whether the requested item is in DBMS cache or not.
  - If it is not, the item must be located on disk, So the appropriate disk pages are copied into the cache.
- **If DBMS cache is already filled?**
  - cache buffers needed to be replaced to make space available for the new item

Recovery Concepts

- **Caching (Buffering) of Disk Blocks**
- **The entries in DBMS cache Dictionary:**
- Holds additional info. For buffer management
  - Dirty bit
  - pin-unpin bit

Recovery Concepts

- **Caching (Buffering) of Disk Blocks**
- **The entries in DBMS cache Dictionary:**
- **Dirty bit:**
  - Associated with each buffer in the cache
  - included in the directory entry
  - to indicate **whether or not the buffer has been modified**.
- **Dirty bit = 0 :**
  - page is first read from the database disk into a cache buffer,
  - a new entry is inserted in the cache directory with the new disk page address.
- **Dirty bit = 1 :**
  - As soon as the buffer is modified.
  - Data needs to be written to disk when the buffer contents are replaced from the cache

Recovery Concepts

- **Caching (Buffering) of Disk Blocks**
- **Pin-unpin bit**:
  - page in the cache is pinned (bit value 1) if it cannot be written back to disk as yet.
  - Why?
    - Because the recovery protocol may restrict certain buffer pages from being written back to the disk until the transactions that changed this buffer have committed.

Recovery Concepts

- **Caching (Buffering) of Disk Blocks**
- **Strategies for replacing cache buffers**
  - **In-place updating**
    - Writes the buffer to the same original disk location
    - Overwrites old values of any changed data items
  - **Shadowing**
    - Writes an updated buffer at a different disk location, to maintain multiple versions of data items
    - Not typically used in practice

Recovery Concepts

- **Caching (Buffering) of Disk Blocks**

- **Before-image(BFIM):**
  - old value of data item before updating
- **After-image(AFIM):**
  - new value of data item after updating

Recovery Concepts

- **Write-Ahead Logging, Steal/No-Steal, and Force/No-Force**
- **REDO-type log entry includes the new value** (AFIM) of the item written by the operation since this is needed to redo the effect of the operation from the log.
- **UNDO-type log entries include the old value** (BFIM) of the item since this is needed to undo the effect of the operation from the log.
- In an **UNDO/REDO algorithm, both BFIM and AFIM are recorded into a single log entry**.

Recovery Concepts

- **Write-Ahead Logging, Steal/No-Steal, and Force/No-Force**
- DBMS cache holds the cached database disk blocks in main memory buffers.
- When an update to a data block—stored in the DBMS cache—is made, an associated log record is written to the log buffer in the DBMS cache.
- **Write-ahead logging approach:** the log buffers (blocks) that contain the associated log records for a particular data block update must first be written to disk before the data block itself can be written back to disk from its main memory buffer.

Recovery Concepts

- **Write-Ahead Logging, Steal/No-Steal, and Force/No-Force**
- **No-steal approach**: a cache buffer page updated by a transaction cannot be written to disk before the transaction commits.
  - UNDO will never be needed during recovery, since a committed transaction will not have any of its updates on disk before it commits.
- **pin-unpin bit will be set to 1 (pin)** to indicate that a cache buffer cannot be written back to disk.
- **Steal:** the recovery protocol allows writing an updated buffer before the transaction commits.

Recovery Concepts

- **Write-Ahead Logging, Steal/No-Steal, and Force/No-Force**
- **Force approach:** If all pages updated by a transaction are immediately written to disk before the transaction commits.
  - REDO will never be needed during recovery, since any committed transaction will have all its updates on disk before it is committed.
- Otherwise, it is called no-force.
- For frequently changed objects, a no-force policy reduces the number of write operations to the on-disk database object.

Recovery Concepts

- **Write-Ahead Logging, Steal/No-Steal, and Force/No-Force**
- The deferred update (NO-UNDO) recovery scheme follows a **no-steal approach**.
- Typical database systems employ a **steal/no-force (UNDO/REDO)** strategy.
- The advantage of steal is that it **avoids the need for a very large buffer space to store all updated pages in memory**.
- The advantage of no-force is that an updated page of a committed transaction may still be in the buffer when another transaction needs to update it.

Recovery Concepts

- **Write-Ahead Logging, Steal/No-Steal, and Force/No-Force**
- Write-ahead logging (WAL) protocol for a recovery algorithm that requires both UNDO and REDO:
- The before image of an item cannot be overwritten by its after image in the database on disk until all UNDO-type log entries for the updating transaction have been force-written to disk.
- The commit operation of a transaction cannot be completed until all the REDO-type and UNDO-type log records for that transaction have been force written to disk.

Recovery Concepts

- **Checkpoints in the System Log and Fuzzy Checkpointing**
- Another type of entry in the system log is called a checkpoint.
- A checkpoint is written in log file as :
  - **[checkpoint, list of active transactions]**
- When system writes the modified DBMS buffers on disk, then this checkpoint entry is inserted into system log
- All transactions that have their [commit, T] entries in the log before a [checkpoint] entry
  - do not need to have their WRITE operations redone in case of a system crash,
  - since all their updates will be recorded in the database on disk during checkpointing.
- List of active transactions from checkpoint entry will help in identifying transactions for recovery.

Recovery Concepts

- **Checkpoints in the System Log and Fuzzy Checkpointing**
- The DB recovery manager will decide at what time the checkpoint is taken.
  - The interval for checkpoints can be decided in terms of :
    - time (i.e., after every 4 minutes) or
    - after the number of committed transactions since the last checkpoint (i.e., after every 4 transactions' commit)

Recovery Concepts

- **Checkpoints in the System Log and Fuzzy Checkpointing**
- Taking a check point involves following actions:
  - **Suspend execution of transactions temporarily.**
  - **Force-write all main memory buffers that have been modified to disk.**
  - **Write a [checkpoint] record to the log, and force-write the log to disk.**
  - **Resume executing transactions.**

Recovery Concepts

- **Checkpoints in the System Log and Fuzzy Checkpointing**
- The cost of writing modified buffers back to disk (i.e., step 2) will induce delay in currently executing transactions as they are at halt state.
- That's why fuzzy checkpointing is used.
- **Fuzzy checkpointing**.
- The system can resume transaction processing after a [begin_checkpoint] record is written to the log without having to wait for step 2 to finish.
- When step 2 is completed, an [end_checkpoint, …] record is written in the log with the relevant information collected during checkpointing

Recovery Concepts

- **Transaction Rollback and Cascading Rollback**
- If a **transaction fails for some reason after updating the database, but before the transaction commits**,
  - it may be necessary to **roll back the transaction.**
- If any **data item values have been changed by the transaction** and written to the database on disk,
  - they must be **restored to their previous values (BFIMs).**
- The **undo-type log entries** are used to restore the old values of data items that must be rolled back.
  - **Because they contains both old and new values**

Recovery Concepts

- **Transaction Rollback and Cascading Rollback**
- **cascading rollback:**
- If a transaction T is rolled back, any transaction S that has read the value of some data item X written by T ,must also be rolled back.
- Similarly, once S is rolled back, any transaction R that has read the value of some data item Y written by S must also be rolled back; and so on.
- it can occur when the recovery protocol ensures recoverable schedules but does not ensure strict or cascadeless schedules.

Recovery Concepts

- **Transaction Rollback & Cascading Rollback**

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)

Recovery Concepts

T_1
read_item(A)
read_item(D)
write_item(D)

T_2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

T_3
read_item(C)
write_item(B)
read_item(A)
write_item(A)

		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]				

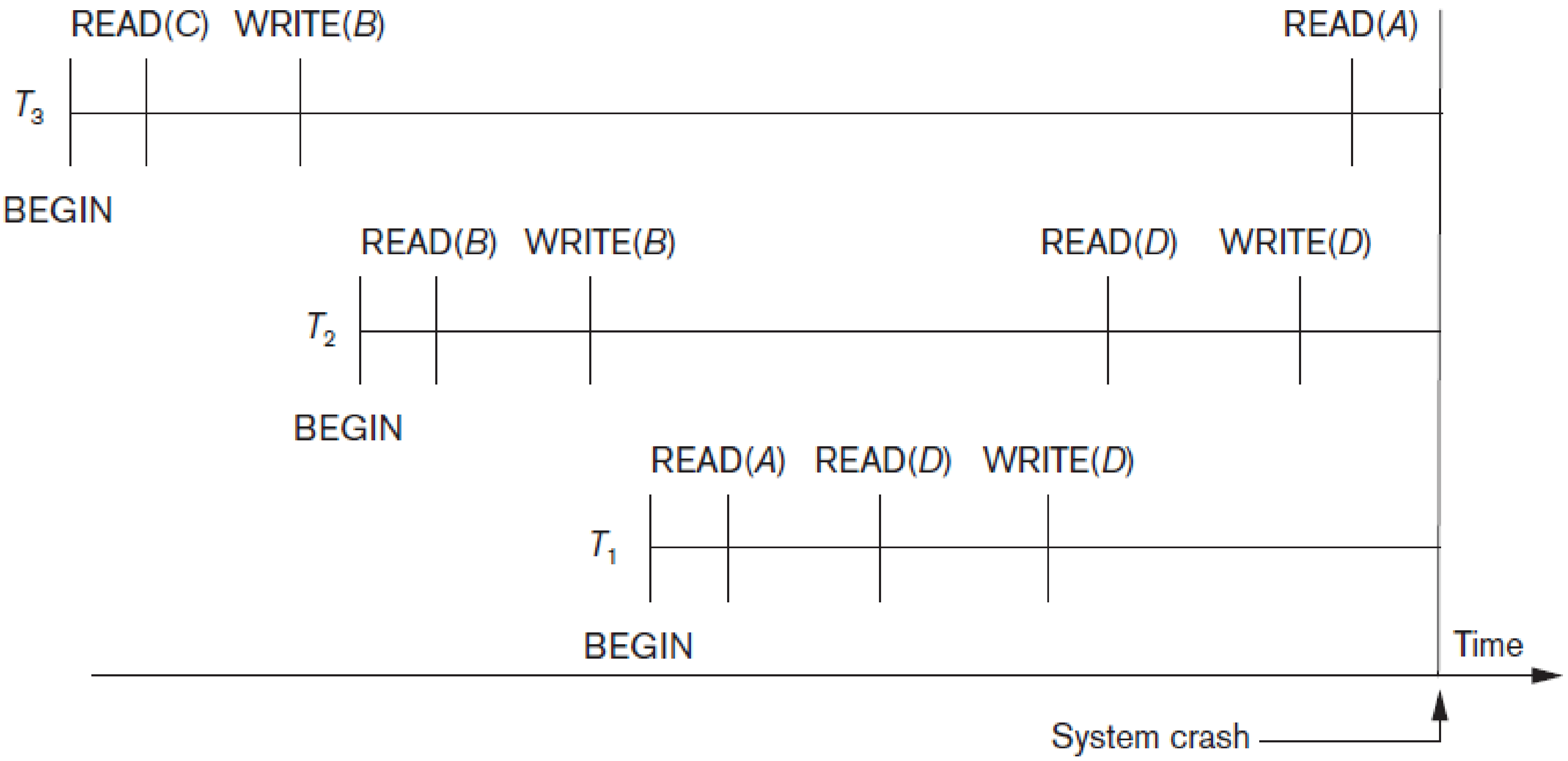
← System crash

* 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.

Recovery Concepts

- **Transaction Rollback and Cascading Rollback**



Recovery Concepts

Transaction Rollback and Cascading Rollback

We must now check for cascading rollback. T2 reads the value of item B that was written by transaction T3.
Because T3 is rolled back, T2 must now be rolled back, too.
The WRITE operations of T2, marked by ** in the log, are the one that are undone.

Note that only **write_item** operations need to be undone during transaction rollback; **read_item** operations are recorded in the log only to determine whether cascading rollback of additional transactions is necessary.

(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.

point of crash. (c) Op
before the crash.