Recovery SubSytem

Mirunalini.P

SSNCE

April 3, 2020

Table of Contents

Session Outcome

At the end of this session, participants will be able to

- Understand Recovery Concepts
- Understand WAL
- Understand Checkpointing
- Understand Deferred Update
- Understand Immediate Update

Recovery Concepts

- Recovery process restores database to the most **recent consistent state** before the time of failure.
- The system must keep information about the changes that were applied to data items by the various transactions which were kept in **system log**
- Typical recovery strategies are:
  - Restore backed-up copy of database from **archival storage** when the damage due to catastrophic failure
  - Identify any changes that may cause inconsistency in a noncatastrophic failure, reverse the changes by applying either of the operations like **undoing or redo**

Log Files

Holds the information that is necessary for the recovery process
Records all relevant operations in the order in which they occur
Is an append-only file.

- **[start_transaction,T]**: Records that transaction T has started execution.
- **[write_item,T,X,old_value,new_value]**: T has changed the value of item X from old_value to new_value.
- **[read_item,T,X]**: T has read the value of item X (not needed in many cases).
- **[end_transaction,T]**: T has ended execution
- **[commit,T]**: T has completed successfully, and committed.
- **[abort,T]**: T has been aborted.

Log Files

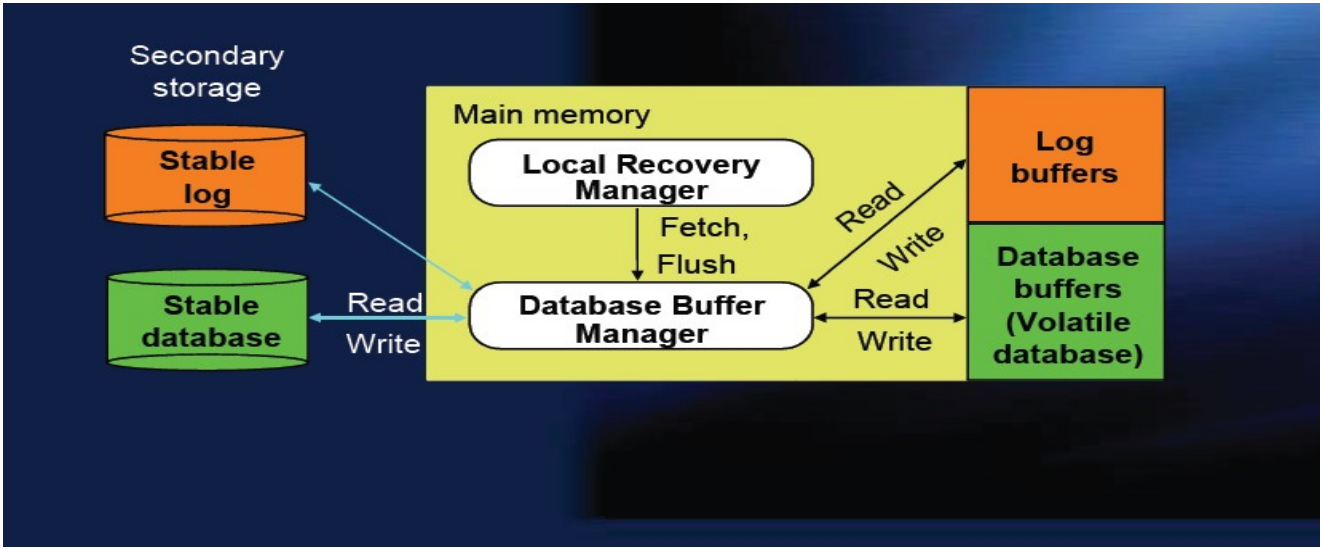
For write_item log entry, old value of item before modification (BFIM - BeFore Image) and the new value after modification (AFIM AFter Image) are stored.

BFIM needed for UNDO, AFIM needed for REDO.

A sample log is given below:

T ID	Back P	Next P	Operation	Data item	BFIM	AFIM
T1	0	1	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	W	Y	Y = 50	Y = 100
T1	4	7	R	M	M = 200	M = 200
T3	0	9	R	N	N = 400	N = 400
T1	5	nil	End			

Transaction Recovery



Caching and DBMS cache

- The DBMS keeps directory of the buffers (disk blocks) called the **DBMS cache**.
- The data items being read and written by the database transactions are stored in the the disk blocks.
- A directory for the cache is used to keep track of which database items are in the buffers.
- The buffer directory contains a table of < buffer addr, disk block addr, modified bit, pin/unpin bit, ...>
- It is often necessary to replace (flush) some of the cache buffers to make space for new items.

Caching and DBMS cache

- When DBMS requests action on some items , if it can be checked in the cache if not present then it is copied from disk blocks.
- Each buffer has several bits or flags associated with it to inform about the buffer modification
- **The dirty bit:** Set to 0 if the buffer has not been modified and set to 1 if not.
- **The pin-unpin bit**: Set to 1 if it cannot be written back to disk yet (usually waiting for a commit)

Flushing the buffers

Two main strategies are used when a buffer is written to disk

- **In-place updating**:
  - Writes the buffer back to the same original disk location
  - It overwrites the old values of any changed data items
  - Recovery requires a log to undo and/or redo
- **Shadowing**
  - Write an updated buffer to a different disk location
  - Multiple versions of data items can be maintained
  - The old value of the data item before updating is called before image (BFIM).
  - The new value after updating is called the after image (AFIM)

Write Ahead Log [WAL]

- Inplace updating log recovery is used.
- **Undo-Type log Entries**: Includes the old value (BFIM) of the item since this is needed to undo the effect of the operation from the log.
- **Redo-Type Log Enteries**: 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.
- The log enteries are stored in the current log buffer in the DBMS cache.
- When data item is updated enteries are made in appropriate log entry.
- With the help of WAL, the information needed for recovery must be **first written to the log file on disk before changes are made to the database on disk**.

Write Ahead Log [WAL] Rule:

Write-Ahead Logging (WAL) protocol consists of two rules for Undo and Redo operations:

- The **BFIM cannot be overwritten by its AFIM** in the database on disk **until all UNDO - type log records** 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 **(STABLE STORE)**.

Techniques used to move a page from cache to memory

Standard DBMS recovery terminology includes the terms steal/no-steal and force/no-force which specify some rules when a page from database cache can be written to disk:

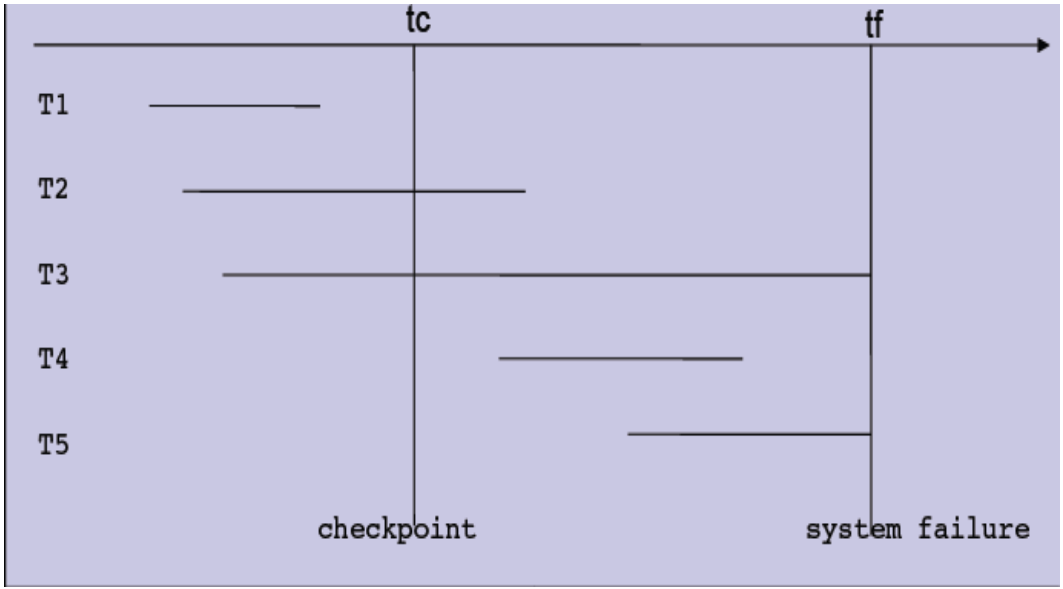
- **No-Steal Approach:** if a cache page updated by a transaction cannot be written to dist before the transaction commits.
- **Steal Approach:** if the recovery protocol allows writing an updated before the transaction commits.
- **Force Approach:** If all pages updated by a transaction are immediately written to disk before the transaction commits.
- **No-Force Approach:** if all pages are updated after the commit of transactions

Typical database systems employ a steal/no-force (UNDO/REDO) strategy and deferred update (NO-UNDO) recovery scheme follows a no-steal approach.

Check Pointing

- Checkpoint : Type of entry in the log
- For any system failure, the contents of the main memory was lost.
- Recovery system used to maintain a list of **active transactions,committed and aborted** transactions since the last check point.
- When system restarts:
  1. All transactions have their [**commit,T**] enteries in the log before checkpointing will not require any operations, updates done before.
  2. Any transactions that have commited after checkpointing should be **redone**.
  3. Any transactions that have started before or after checkpoint but not committed should be **undone.**

Checkpointing



T1 - Commited before Checkpoint
T2,T4 - Commited before system failure
T3, T5 - Not commited still system failure

Checkpointing

1. Start with two lists of transactions, the UNDO list and the REDO list.

2. Set UNDO = list of all transactions given in the most recent checkpoint record; REDO = empty.

3. Search forward through log, starting from checkpoint record.

4. For a transaction T, if a BEGIN TRANSACTION log record is found, add T to the UNDO list.

5. For a transcation T, if a COMMIT log record is found, move T from UNDO to REDO list.

6. At the end of the log, the UNDO and REDO lists identify, transactions T3 and T5 has to be UNDONE and transactions T2 and T4 has to be REDONE.

Commit Point

- A transaction T reaches its commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database have been recorded in the log.
- Beyond the commit point, the transaction is said to be committed, and its effect must be permanently recorded in the database.
- The transaction then writes a **commit record** [**commit , T**] into the log.

Non-catastrophic Transaction Failures

Two main policies for recovery from non-catastrophic transaction failures: **deferred update and immediate update**.
Deferred Update:

- Do not **physically update the database until after transaction commits**.
- Before reaching the commit point transactions updates are recorded in the log file.
- After commit updates are written to the database from the main memory buffer.
- If a transaction fails before reaching its commit point, no changes in the database so no UNDO.
- After commit REDO the effect of the opeations of the committed transaction from the log.
- Deffered update also known as the **NO-UNDO/REDO** algorithm

Deferred Update

Deferred update protocol are as follows:

- A transaction **cannot change the database** on disk until it reaches its **commit point**
- A transaction does not **reach its commit point** until all its **REDO -type** log entries are recorded in the log and the log buffer is force-written to disk.

Deferred Update Concurrent User

- Concurrency control uses strict two-phase locking, so the locks on items remain in effect until the transaction reaches its commit point.
- After that locks are released.
- Use two lists of transactions maintained by the system: **the committed transactions** T since the last checkpoint (commit list), and the **active transactions** T (active list).
- **REDO** all the WRITE operations of the **committed transactions** from the log, in the order in which they were written into the log.
- The transactions that are **active** and did not commit are effectively **canceled** and must be resubmitted.

Deferred update -REDO

Redoing a write_item operation consists of examining its log entry [**write_item, T , X , new_value**] and setting the value of item X in the database to new_value , which is the after image **(AFIM)**.

Example

(a)

T_1	T_2	T_3	T_4
read_item(A)	read_item(B)	read_item(A)	read_item(B)
read_item(D)	write_item(B)	write_item(A)	write_item(B)
write_item(D)	read_item(D)	read_item(C)	read_item(A)
	write_item(D)	write_item(C)	write_item(A)

(b)

[start_transaction, T_1]
[write_item, T_1, D,20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4, B,15]
[write_item, T_4, A,20]
[commit, T_4]
[start_transaction, T_2]
[write_item, T_2, B,12]
[start_transaction, T_3]
[write_item, T_3, A,30]
[write_item, T_2, D,25] ←system crash

T_2 and T_3 are ignored because they did not reach their commit points.
T_4 is redone because its commit point is after the last system checkpoint.

Non-catastrophic Transaction Failures

Immediate Update When a transaction issues an update command, the database can be updated immediately, without any need to wait for transaction to reach its commit point

- Provisions is needed for undoing the effect of update operations that have been applied to the database by a failed transaction.
- Accomplished by rolling back the transaction and undoing the effect of the transactions write_item operations.
- **UNDO** is required because **AFIM** is flushed to the disk before a transaction commits.
- **REDO** is required in case the system fails after a transaction commits but before all its changes are recorded in the database on disk.

Immediate update- UNDO/REDO wth checkpoints

- Use two lists of transactions maintained by the system: the **committed transactions** since the last checkpoint and the **active transactions**.
- **Undo** all the write_item operations of the **active (uncommitted) transactions**, using the UNDO procedure. The operations should be undone in the reverse of the order in which they were written into the log.
- **Redo** all the write_item operations of the**committed transactions** from the log, in the order in which they were written into the log, using the REDO procedure defined earlier.

Procedure UNDO

Undoing a write_item operation consists of examining its log entry [**write_item, T , X , old_value, new_value**] and setting the value of item X in the database to old_value , which is the before image **(BFIM)**.

Undoing from the log must proceed in the reverse order in which the operations were written in the log.

Reference

 Fundamentals of Database systems 7^{th} Edition by Ramez Elmasri.