Chapter 11 – Database Recovery Techniques

Recovery Concepts

- Recovery from transaction failures usually means that the database is *restored* to the most recent consistent state just before the time of failure.
- A typical strategy for recovery may be summarized informally as follows:
  1. If there is extensive damage to a wide portion of the database due to catastrophic failure, such as a disk crash, the recovery method restores a past copy of the database that was *backed up* to archival storage (typically tape) and reconstructs a more current state by reapplying or *redoing* the operations of committed transactions from the *backed up* log, up to the time of failure.
  2. When the database is not physically damaged but has become inconsistent due to noncatastrophic failures of types 1 through 4 of Section 19.1.4, the strategy is to reverse any changes that caused the inconsistency by *undoing* some operations
- We can distinguish two main techniques for recovery from non-catastrophic transaction failures: (1) deferred update and (2) immediate update.
- The deferred update techniques do not physically update the database on disk until after a transaction reaches its commit point; then the updates are recorded in the database. Before reaching commit, all transaction updates are recorded in the local transaction workspace (or buffers). During commit, the updates are first recorded persistently in the log and then written to the database. If a transaction fails before reaching its commit point, it will not have changed the database in any way, so UNDO is not needed. It may be necessary to REDO the effect of the operations of a committed transaction from the log, because their effect may not yet have been recorded in the database. Hence, deferred update is also known as the **NO-UNDO/REDO algorithm.**
- In the immediate update techniques, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations are typically recorded in the log on disk by force writing before they are applied to the database, making recovery still possible. If a transaction fails after recording some changes in the database but before reaching its commit point, the effect of its operations on the database must be undone; that is, the transaction must be rolled back. In the general case of immediate update, both undo and redo may be required during recovery. This technique, known as the UNDO/REDO algorithm, requires both operations, and is used most often in practice. A variation of the algorithm where all updates are recorded in the database before a transaction commits requires undo only, so it is known as the UNDO/NO-REDO algorithm

Transaction Log

Log is a sequence of records, which maintains the records of actions performed by a transaction. For recovery from any type of failure, data values prior to modification (BFIM - BeFore Image) and the new value after modification (AFIM – AFter Image) are required. This, it is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows −

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.

$<T_n, \text{Start}>$

- When the transaction modifies an item X, it write logs as follows −

$<T_n, X, V_1, V_2>$

It reads T_n has changed the value of X, from V_1 to V_2.

- When the transaction finishes, it logs −

$<T_n, \text{commit}>$

Data Update

- **Immediate Update**: As soon as a data item is modified in cache, the disk copy is updated.
- **Deferred Update:** All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.
- **Shadow update:** The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.
- **In-place update:** The disk version of the data item is overwritten by the cache version.

Data Caching

- Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk. The flushing is controlled by **Modified** and **Pin-Unpin** bits.
- **Pin-Unpin**: Instructs the operating system not to flush the data item.
- **Modified**: Indicates the AFIM of the data item.

Write-Ahead Logging, Steal/No-Steal, and Force/No-Force

Write-Ahead Logging

- When in-place update (immediate or deferred) is used then log is necessary for recovery and it must be available to recovery manager. This is achieved by Write-Ahead Logging (WAL) protocol. WAL states that
    - For Undo: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).
    - For Redo: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

Steal/No-Steal and Force/No-Force

- Possible ways for flushing database cache to database disk:
    - Steal: Cache can be flushed before transaction commits.
    - No-Steal: Cache cannot be flushed before transaction commit.
    - Force: Cache is immediately flushed (forced) to disk.
    - No-Force: Cache is deferred until transaction commits.
- These give rise to four different ways for handling recovery:
- Steal/No-Force (Undo/Redo), Steal/Force (Undo/No-redo), No-Steal/No-Force (Redo/No-undo) and No-Steal/Force (No-undo/No-redo).

Checkpointing

Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:

1. Suspend execution of transactions temporarily.
2. Force write modified buffer data to disk.
3. Write a [checkpoint] record to the log, save the log to disk.
4. Resume normal transaction execution.

During recovery **redo** or **undo** is required to transactions appearing after [checkpoint] record.

Transaction Roll-back (Undo) and Roll-Forward (Redo)

- To maintain atomicity, a transaction's operations are **redone** or **undone**.
    - **Undo**: Restore all BFIMs on to disk (Remove all AFIMs).
    - **Redo**: Restore all AFIMs on to disk.
- Database recovery is achieved either by performing only Undos or only Redos or by a combination of the two. These operations are recorded in the log as they happen.

Recovery Techniques Based on Deferred Update

Deferred Update (No Undo/Redo)

The data update goes as follows:

1. A set of transactions records their updates in the log.
2. At commit point under WAL scheme these updates are saved on database disk.

After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

Deferred Update in a single-user system

There is no concurrent data sharing in a single user system. The data update goes as follows:

1. A set of transactions records their updates in the log.
2. At commit point under WAL scheme these updates are saved on database disk.

After reboot from a failure the log is used to redo all the transactions affected by this failure. No undo is required because no AFIM is flushed to the disk before a transaction commits.

Deferred Update with concurrent users

Two tables are required for implementing this protocol:

Active table: All active transactions are entered in this table.

Commit table: Transactions to be committed are entered in this table.

During recovery, all transactions of the **commit** table are redone and all transactions of **active** tables are ignored since none of their AFIMs reached the database. It is possible that a **commit** table transaction may be **redone** twice but this does not create any inconsistency because of a redone is "**idempotent**", that is, one redone for an AFIM is equivalent to multiple redone for the same AFIM.

Recovery Techniques Based on Immediate Update

Undo/No-redo Algorithm

- In this algorithm AFIMs of a transaction are flushed to the database disk under WAL before it commits.
- For this reason the recovery manager **undoes** all transactions during recovery.
- No transaction is **redone**.

- It is possible that a transaction might have completed execution and ready to commit but this transaction is also **undone**.

Undo/Redo Algorithm (**Single-user** environment)

- Recovery schemes of this category apply undo and also redo for recovery.
    - In a single-user environment no concurrency control is required but a log is maintained under WAL.
    - Note that at any time there will be one transaction in the system and it will be either in the commit table or in the active table.
    - The recovery manager performs:
        - Undo of a transaction if it is in the active table.
        - Redo of a transaction if it is in the commit table.

Undo/Redo Algorithm (**Concurrent** execution)

- Recovery schemes of this category applies undo and also redo to recover the database from failure.
- In concurrent execution environment a concurrency control is required and log is maintained under WAL.
- Commit table records transactions to be committed and active table records active transactions. To minimize the work of the recovery manager checkpointing is used.
- The recovery performs:
    - Undo of a transaction if it is in the active table.
    - Redo of a transaction if it is in the commit table.

Shadow Paging (NO-UNDO / NO-REDO)

- It is a recovery scheme
    - In a single-user environment, doesn't require the use of log.
    - In multi-user environment, the log may be needed for concurrency control method.
- The DB is made up of n fixed-size disk pages -blocks-
- A directory with n entries where the ith entry points to the ith DB page on disk.
    - All references –reads or writes- to the DB pages on disk go through the directory.
    - The directory is kept in main memory if not too large.
- When a transaction begins executing, the current directory is copied into a shadow directory and the shadow directory is saved on disk
    - The current directory entries point to the most recent or current DB pages on disk
- During transaction execution, all updates are performed using the current directory and the shadow directory is never modified.
- When a write_item operation is performed

    - A new copy of the modified DB page is created and the old copy is not overwritten.
    - Two version, of the pages updated by the transaction, are kept.
        - The new page is written elsewhere on some unused disk block.
        - The current directory entry is modified to point to the new disk block.
            - The shadow directory is not modified.
- To recover from a failure
    - Free the modified database pages and discard the current directory.
    - Reinstate the shadow directory to recover the state of the DB before transaction execution.
    - Return to the state prior to the transaction that was executing when the crash occurred.
- To commit a transaction
    - Discard the previous shadow directory.
- NO-UNDO/NO-REDO technique since neither undoing or redoing of data items