Database Recovery Techniques

Introduction

- The purpose of database recovery is to bring the database into the last consistent state, which existed prior to the failure.
- Database recovery also preserves transaction properties (atomicity, consistency, isolation and durability).
- For example, if the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts.

Types of Failures

- **Transaction failure** :
  - **Logical errors**: transaction cannot complete due to some internal error condition such as bad input, data not found.
  - **System errors**: the database system must terminate an active transaction due to an error condition (e.g., deadlock).
- **System crash**: a hardware malfunction, a bug in the database software or operating system, or a power failure causes the system to crash. The content of nonvolatile storage remains intact, and is not corrupted.
- **Disk failure**: a head crash or similar disk failure destroys all or part of disk storage. Copies of data on other disks and tapes are used to recover from failure.

Transaction Log

- 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. These values and other information is stored in a sequential file called **Transaction log**. **Back P** and **Next P** point to the previous and next log records of the same transaction.

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			

Database Recovery Concepts

- **Recovery Outline and Categorization of Recovery Algorithms:** A typical strategy for recovery may be summarized informally as follows:
  - **Manual Reprocessing:**
    - 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.
    - **Limitations of Manual Reprocessing:**
      - Time required to reapply transactions
      - Transactions might have other (physical) potential failures
      - Reapplying concurrent transactions is not straight forward

Database Recovery Concepts

- **Automated Recovery:**
  - When the database is not physically damaged but has become inconsistent due to non-catastrophic failure, the strategy is to reverse any changes that caused the inconsistency by ***undoing*** some operations.
  - It may also be necessary to ***redo*** some operations in order to restore a consistent state of the database.
  - In this case we do not need a complete archival copy of the database. Rather, the entries kept in the *online system log* are consulted during recovery.
  - Under this strategy, we can distinguish two main techniques **deferred update** and **immediate update**.
  - **Deferred Update**
    - This technique do not physically update the database on disk until after a transaction reaches its commit point; then the updates are recorded in the database.

Database Recovery Concepts

- 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 transactions 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**.

Immediate Update

- The database may be updated by some operations of a transaction before the transaction reaches its commit point.

Database Recovery Concepts

- 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 effects of its operations on the database must be undone; that is, the transaction must be rolled back.
- In general case of immediate update, both undo and redo may be required during recovery and also called **UNDO/REDO algorithm**.
- A variation of this algorithm where all updates are recorded in the database before a transaction commits requires undo only is called **UNDO/NO-REDO algorithm**.

Database Recovery Concepts

- **Caching (Buffering) of Disk Blocks**:
  - Here, one or more disk pages that include the data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk.
  - The caching of disk pages is traditionally an operating system function, but because of its importance to the efficient recovery procedures, it is handled by the DBMS by calling low-level operating system routines.
  - A **directory** for cache is used to keep track of which database items are in the buffers.
  - When the DBMS requests action on some item, it first checks the cache directory to determine whatever the disk page containing the item is in the cache.
  - It may be necessary to **replace** (or **flush**) some of the cache buffers to make space available for the new item.

Database Recovery Concepts

- 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 using **dirty bit** and **pin-unpin** bits.
- Associated with each buffer in the cache is a dirty bit, which can be included in the directory entry to indicate whether or not buffer has been modified.
- When a page is first read from the database disk into a cache buffer, the cache directory is updated with the new disk page address, and the dirty bit is set to 0 (zero).
- As soon as the buffer is modified, the dirty bit for the corresponding directory entry is set to 1 (one). When the buffer contents are replaced (flushed) from the cache, the contents must first be written back to the corresponding disk page only if its dirty bit is 1.

Database Recovery Concepts

- Another bit, called the pin-unpin bit is also needed – a page in the cache is pinned (bit value 1) if it cannot be written back to disk yet.
- Two main strategies can be employed when flushing a modified buffer back to disk: **shadow update** and **in place update**.
- **Shadow Update**:
  - The modified version of a data item does not overwrite its disk copy but is written at a separate disk location. So, multiple versions of data items can be maintained. In shadowing, both AFIM and BFIM can be kept on disk.
- **In-place Update**:
  - The disk version of the data item is overwritten by cache version after modification. This technique writes the buffer back to same original disk location, thus overwriting the old value of any changed data items on disk. Hence, a single copy of each database block is maintained.

Database Recovery Concepts

- **Write-Ahead Logging**
  - When in-place update is used, it is necessary to use a log for recovery.
  - In this case, the recovery mechanism must ensure that the BFIM of the data item is recorded in the appropriate log entry and that the log entry is flushed to the disk before the BFIM is overwritten with the AFIM in the database on disk.
  - This process is generally know as **write-ahead logging (WAL)**.
  - This process requires two types of log entry information: **UDO-type log entry** and **REDO-type log entry**.
  - The **UNDO-type log entries** include the old value (BFIM) of the item since this is needed to undo the effect of the operation form the log.
  - The **REDO-type log entries** include 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.

Database Recovery Concepts

- **Steal/No-Steal and Force/No-Force**
  - Possible ways for flushing database cache to database disk.
  - **No-Steal** – A page updated by a transaction cannot be written to disk before the transaction commits.
  - **Steal** – If the protocol allows writing an updated buffer before the transaction commits, it is called steal. Steal is used when the DBMS cache (buffer) manager needs a buffer frame for another transaction and the buffer manager replaces an existing page that has been updated but whose transaction has not committed.
  - **Force** – All pages updated by a transaction are immediately written to disk when the transaction commits.
  - **No-Force** – The updated pages are not immediately flushed when the transaction commits.
  - The deferred update approach follows a no-steal approach. However, most database systems employ a steal/no-force approach.

Database Recovery Concepts

- **Checkpoint**
  - Another type of entry in the log is called **checkpoint**. A checkpoint record is written into the log periodically at the point when the system writes out to the database on disk all DBMS buffers that have been modified.
  - 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 define a checkpoint operation:
    - 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
  - In many environments, it is possible to take checkpoints each 15 minutes or half hour, etc. Recovery must then only be done from the time of the last checkpoint.

Database Recovery Concepts

- **Transaction Rollback**
  - If a transaction fails for whatever reason after updating the database, it may be necessary to roll back the transaction.
  - If any data values have been changed by the transaction and written to the database, they must be restored to their previous values (BFIMs).
  - The undo type log entries are used to restore the old version of data items that must be rolled back.
  - Cascading rollback can occur when the recovery protocol ensures recoverable schedule but does not ensure cascadeless schedules.
  - Since cascading rollback is time-consuming, almost all recovery mechanisms are designed such that cascading rollback is never required.

Recovery Techniques

- **Recovery Techniques Based on Deferred Update**
  - The idea behind deferred update techniques is to defer or postpone any actual updates to the database until the transaction completes its execution successfully and reaches its commit point.
  - During transaction execution, the updates are recorded only in the log and in the cache buffers.
  - After the transaction reaches its commit point and the log is force written to disk, the updates are recorded in the database.
  - If a transaction fails before reaching its commit point, there is no need to undo any operations, because the transaction has not affected the database on disk any way.

Recovery Techniques

- We can state typical deferred update protocol 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 update operations are recorded in the log and the log is force-written to disk.
- Since, the database is never updated on disk until after the transaction commits, there is never a need to UNDO any operations. Hence, this is known as the **NO-UNDO/REDO recovery algorithm**.
- REDO is needed in case the system fails after a transaction commits but before all its changes are recorded in the database on disk.
- This technique cannot be used in practice for long transactions because there is the potential for running out of buffer space since transaction changes must be held in the cache buffers until the commit point.

Recovery Techniques

- **Recovery using deferred update in a single user environment**
  - Since, there is no concurrent data sharing in a single user system, the recovery algorithm is simple.
  - A set of transactions records their updates in the log.
  - At commit point, 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.
  - Since, the database is never updated on disk until after the transaction commits, there is never a need to UNDO any operations.

Recovery Techniques

- **Recovery using deferred update with concurrent execution in a multi user environment**
  - This environment requires some concurrency control mechanism to guarantee **isolation** property of transactions.
  - Here, transactions which were recorded in the log after the last checkpoint were **redone** or **ignored**.
  - Two tables are required for implementing this protocol: **active table** and **commit table**.
  - All active transactions are entered in active table.
  - All transactions to be committed are entered in commit table.
  - During recovery, all transactions of the commit table are redone in the order in which they were written into the log and all transactions of active tables are ignored.

Recovery Techniques

- 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**
  - Here, when a transaction issues an update command, the database can be updated immediately without any need to wait for the transaction to reach its commit point.
  - However, an update operation must still be recorded in the log before it is applied to the database using write-ahead logging protocol so that we can recover in case of failure.

Recovery Techniques

- If the recovery technique ensures that all updates of a transaction are recorded in the database on disk before the transaction commits, there is never need to REDO any operations of committed transactions. This is called the **UNDO/NO-REDO recovery algorithm**.
- On the other hand, if the transaction is allowed to commit before all its changes are written to the database, we have the most general case, known as the **UNDO/REDO recovery algorithm**.
- **UNDO/REDO recovery based on immediate update in a single-user environment**
  - In a single-user environment no concurrency control is required.
  - In a single user system, if a failure occurs, the executing (active) transaction at the time of failure may have recorded some changes in the database.

Recovery Techniques

- So the effect of all such operations must be undone and redone.
- In this algorithm, two lists are maintained by the system: the committed transactions since the last checkpoint and the active transactions.
- Undo all the write operations of the active transaction.
- Redo the write operations of committed transactions from the log

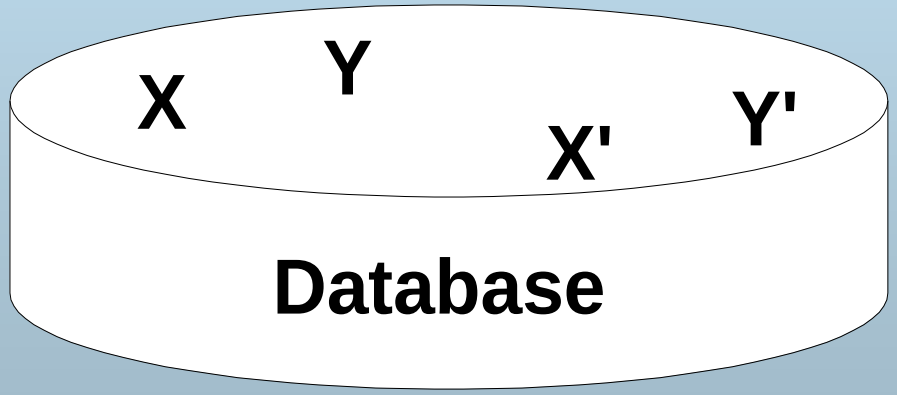
- **UNDO/REDO recovery based on immediate update with concurrent execution**
  - In concurrent execution, the recovery process depends on the protocols used for concurrency control.
  - Recovery schemes of this category applies undo and also redo to recover the database from failure.

Recovery Techniques

- In this algorithm, two lists are maintained by the system: the committed transactions since the last checkpoint and the active transactions.
- Undo all the write operations of the active transaction. The operations should be undone in the reverse of the order in which they were written into the log.
- Redo the write operations of committed transactions from the log in the order in which they were written into the log.

Shadow Paging

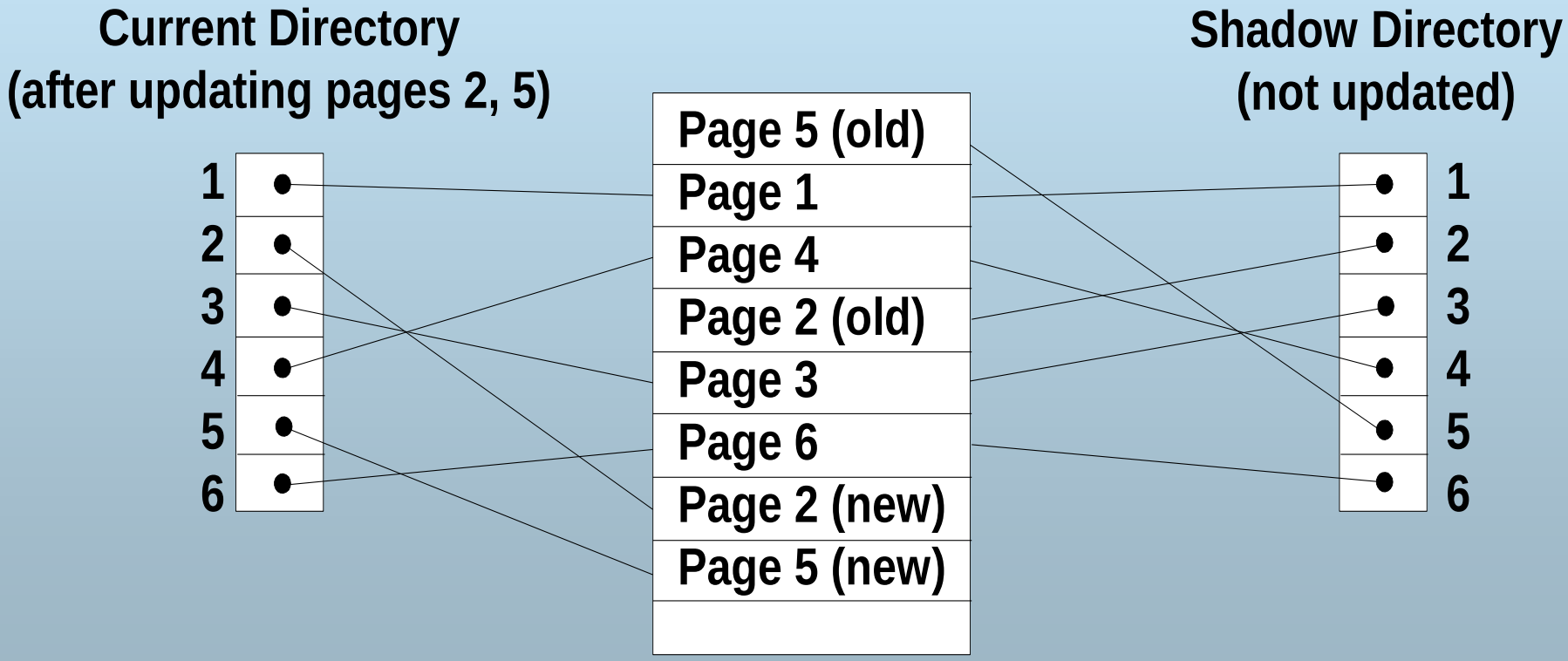
- In this technique, the AFIM (After Image) of a data item does not overwrite its BFIM (Before Image) but recorded at another place on the disk.
- Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.



X and Y: Shadow copies of data items
X` and Y`: Current copies of data items

Shadow Paging

- To manage access of data items by concurrent transactions two directories (current and shadow) are used. The directory arrangement is illustrated below. Here a page is a data item.



Shadow Paging

- To recover, it is sufficient to free the modified pages and discard the current directory. The state of the database before transaction execution is available through the shadow directory. Database can be returned to its previous state.
- Committing a transaction corresponds to discarding the previous shadow directory.
- Can be categorized as a **NO-UNDO/NO-REDO** technique for recovery.
- Logs and checkpoints must be incorporated into the shadow paging technique with multiuser environment.
- Disadvantages: complex storage management strategies, the overhead of writing shadow directories to disk, garbage collection overhead (old pages referenced by the shadow directory).

Database backup and Recovery from Catastrophic Failures

- The most of recovery techniques use the entries in the system log or shadow directory to recover from failure by bringing the database back to a consistent state.
- Both system log and shadow directory are stored on disk to allow recovery from non catastrophic failures.
- The recovery manager of a DBMS must also be equipped to handle more catastrophic failures such as disk crashes.
- The main technique used to handle such crashes is that of database backup.
- The whole database and the log are periodically copied onto a cheap storage medium such as magnetic tapes.

Database backup and Recovery from Catastrophic Failures

- In case of a catastrophic failure, the latest backup copy can be reloaded from the tape to the disk, and the system can be restarted.
- To avoid losing all effects of transactions that have been executed since the last backup, it is customary to back up the system log at more frequent intervals than full database backup by periodically coping it to magnetic tape.
- The system log is usually substantially smaller than the database itself and hence can be backed up more frequently.
- Thus users do not lose all transactions they have performed since the last database backup.

Database backup and Recovery from Catastrophic Failures

- All committed transactions recorded in the portion of the system log that has been backed up to tape can have their effect on the database recovery.
- A new log is started after each database backup.
- Hence, to recover from disk failure, the database is first reloaded on disk from its latest backup copy on tape. And the effects of all the committed transactions whose operations have been recorded in the backed-up copies of system log are reconstructed.