Chapter 9: Recovery System

Crash recovery is the process by which the database is moved back to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred. In other words, Recovery System in an integral part of a database system which is responsible for the restoration of the database to a consistent state that existed prior to the occurrence of the failure.

Types of Failure/Failure Classification

The major types of failures are

1. Transaction failure: The transaction failure occurs when a transaction is aborted. There are two types of error that can make a transaction to fail.
    - Logical errors: transaction cannot complete due to some internal error condition such as bad input data, resource limit exceeds etc.
    - System errors: the database system must terminate an active transaction due to an error condition (e.g., deadlock). The transaction however can be re-executed at later time.
2. System crash: a power failure or other hardware or software failure causes the system to crash. This generally occurs when there is hardware malfunctioning, there is bug in the database software or OS resulting loss of content of volatile storage.
3. Disk failure: a head crash or similar disk failure destroys all or part of disk storage. Copies of data on another disk or ternary media such as tapes are used to recover from the failure.
4. Natural Disaster: fire, flood, earthquake etc. may cause physical loss of all information on all media.

Storage Structure

To ensure how to recover data from any kind of failure, we must know about types of storage media and their access methods. The storage media can be classified as:

1. Volatile storage:

- Information here does not survive system crashes
- examples: main memory, cache memory

2. Nonvolatile storage:

- Information normally does survives system crashes
- Examples: disk, tape, flash memory, but may still fail in case of head crash.

3. Stable storage:

- A mythical form of storage that survives all failures i.e. system designed not to loss data.
- approximated by maintaining multiple copies on distinct nonvolatile media

Stable Storage Implementation/Transaction Failure and Recovery

- RAID systems guarantee that the failure of a single disk will not result in the loss of data.
- Failure during data transfer can still result in inconsistent copies: transfer between memory and disk storage result in
  - → Successful completion: Transferred information arrived safely at its destination.
  - → Partial failure: A failure occurred in the midst of a transfer and the destination block has incorrect information.
  - → Total failure: The failure occurred sufficiently early during the transfer the destination block was never updated.
- In database system, the recovery manager (RM) is responsible for maintaining the database consistency in the case of failure.

Log Based Recovery

- A log is kept on stable storage for the log to be useful for recovery from system and disk failure. The log is a sequence of log records, and maintains a record of update activities on the database.
- When transaction Ti starts, it registers itself by writing a
  <Ti start> log record
- Before Ti executes write(X), a log record
  <Ti, X, V1, V2> is written, where V1 is the value of X before the write (the old value), and V2 is the value to be written to X (the new value). When Ti finishes it last statement, the log record <Ti commit> is written.

- The problem with this mechanism is that the volume of data stored in the log may become unreasonable large.
- There are two approached to log
    i. Deferred Database Modification
    ii. Immediate Database Modification

Explanation:

i. Deferred database Modification

- The deferred database modification scheme doesn't physically update the database on disk until the transaction reaches its commit point. Before reaching commit, all transaction updates are recorded in the transaction local workspace (buffer or main memory).
- During commit, the updates are first recorded in log and then written to the database. If the transaction fails before reaching its commit point, it will have made no changes to database in any way so no UNDO operation is necessary.
- It may be necessary to REDO effect of the operations of commit transactions from the log because their effect may not have been recorded in the database. Therefore this technique is also called as NO UNDO/REDO algorithm.
- Transaction starts by writing < Ti start> record to log. A write(X) operations result in a log record <Ti, X, V>, where V is the new value for X. No old value in needed in this scheme. The write operation is not performed on X, but is deferred. When Ti partially commits, <Ti commit> is written to the log. Finally the log records are read and used to actually execute the previously deferred writes.
- In case of any failure, no UNDO is necessary for recovery scheme however a transaction having <Ti start> and < Ti Commit> record in the log must be redone (REDO) to set the value of all data items updated by transaction Ti to the new value found in log.

Example:

Log	Write
<To start>	
<To A45>	
<To B56>	
<To commit>	
	A=45
	B=56

ii. **Immediate Database Modification (Uncommitted Modification)**

- In this technique, the database may be updated by some operation of transaction before the transaction reaches to its commit point.
- However these operations are typically recorded in the log on the disk before they are applied to database, making recovery still possible.
- If the transaction fails after recording some changes in the database but before reaching its commit point, the effect of its operation must be undone (UNDO) i.e. the transaction must be Rolled Back. So immediate update requires both UNDO and REDO during recovery. Therefore it is also known as UNDO / REDO algorithm.
- Transaction starts by writing <Ti start> record to log. A write(X) operation result in a log record <Ti X, U, V> where U is the old value of X and V is the new value of X. Since undoing may be needed, update logs must have both old value and new value. The write operation on X is recorded in the log on disk and is output directly to stable storage without concerning the transaction commits or not.
- In case of failure, recovery scheme has two operations: UNDO and REDO. UNDO restores the value of all data items updated by Ti to their old values, going backward from the last log record for Ti. REDO sets the value of all data items updated by Ti to the new values, going forward from the first log record for Ti. A transaction Ti must be undone if the log contains the record < Ti, start> but doesn't contain < Ti, commit> and Ti must be redone if the log contains both < Ti, start> and < Ti, commit> record.

Example:

Log	Write
<T_0 **start**>	
<T_0, A, 1000, 950>	
<T_0, B, 2000, 2050	
	$A = 950$
	$B = 2050$
<T_0 **commit**>	
<T_1 **start**>	
<T_1, C, 700, 600>	
	$C = 600$
<T_1 **commit**>	

2. **Checkpoints:**

The main problems faced in recovery procedure discussed earlier are

1. Searching the entire log to determine which transaction need to be redone and those that need to be undone is time consuming.
2. We might unnecessarily redo transactions which have already output their updates to the database.

A scheme called check point is used to limit the volume of log information that has to be handled and processed in the events of system failure. A check point performs periodic copy of log information onto the stable storage. A check point requires the following sequence of actions to take place.

1. Output all the log records currently residing in main memory onto stable storage.
2. Output to the disk all modified buffer block.
3. Write a log record <check point> onto stable storage.

During recovery, we need to consider only the most recent transaction Ti that started just before the check point and the transactions that started after it. For this the following task is done.

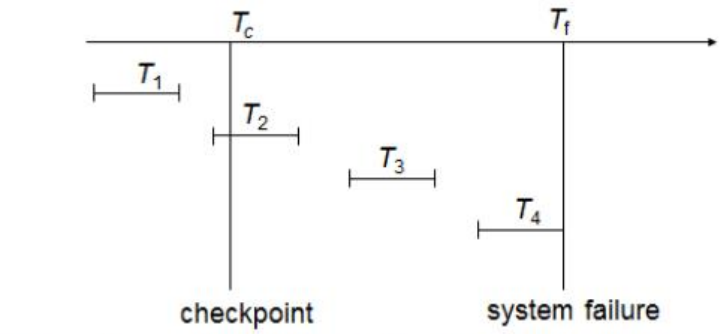
1. Scan backwards from end of log to the most recent <check point> record.

2. Continue scanning backward till a record <Ti start> is identified.

3. So consider only those log records following <Ti start>. Earlier part of log records can be ignored during recovery.

The UNDO and REDO operation on all transaction Tj that started execution after identified Ti is as below.

- If no <Tj commit> record in log and Tj has done immediate modification then execute UNDO for all Tj.
- If <Tj commit> record in log then execute REDO for all Tj.



- T_1 can be ignored (updates already output to disk due to checkpoint)
- T_2 and T_3 redone.
- T_4 undone

3. Shadow Paging

Shadow paging is an alternative to log based recovery which is useful if transaction executes serially. Here, when a page of storage is modified by a transaction, a new page is allocated for the modified data and the old page remains as a shadow copy of the data.

In this technique, database is partitioned into some number of fixed length block called as pages. The key idea behind shadow paging technique is to maintain two page tables during the lifetime of a transaction. These tables are called as current page table and shadow page table. Each entry in the page tables contain pointer to a page on a disk. When the transaction starts, both tables are identical. The shadow page table is never changed during the life of the transaction the current page table is updated with each write operation.

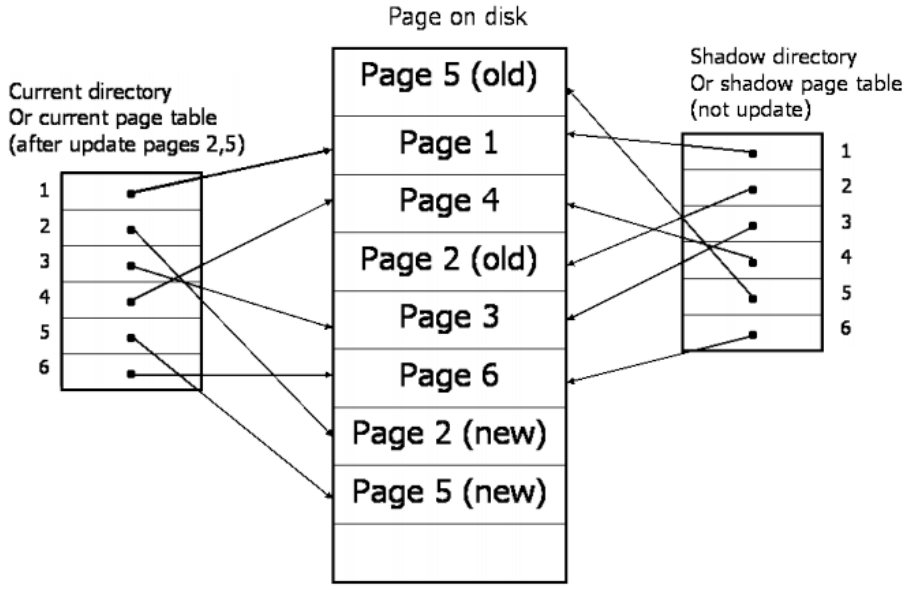


Fig. Shadow Paging

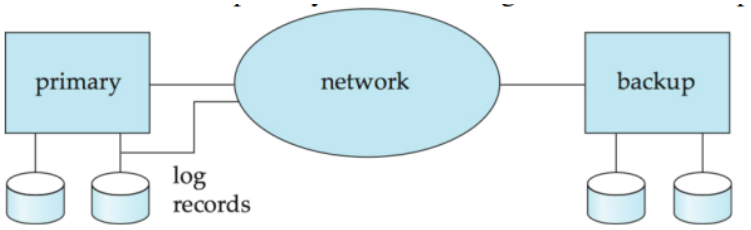
Advantage

1. No overhead of writing log records.
2. Recovery is simple.
3. No Undo / No Redo algorithm.

Disadvantage

1. Copying the entire page table is very expensive
2. Commit overhead: The commit of a single transaction requires multiple blocks to be output: the current page table, the actual data i.e. updated pages. Log based scheme need to output only the log records.
3. Data fragmentation: It causes database pages to change location making no longer contiguous.
4. Garbage collection: Each time a transaction commits, the database pages containing the old version of data changed by the transaction must become inaccessible. Such pages do not contain usable information hence considered as garbage. Periodically it is necessary to find all the garbage pages and add them to the list of free pages. The process is called garbage collection and requires additional overhead and complexity on the system.
5. Hard to extend algorithm to allow transaction to run concurrently.

Remote Backup System



Architecture of remote backup system.

Traditional transaction-processing systems are centralized or client–server systems. Such systems are vulnerable to environmental disaster such as fire, flooding, or earthquakes. Increasingly, there is a need for transaction-processing systems that can function in spite of system failures or environmental disasters. Such systems must provide high availability; that is, the time for which the system is unusable must be extremely small.

We can achieve high availability by performing transaction processing at one site, called the primary site, and having a remote backup site where all the data from the primary site are replicated. The remote backup site is sometimes also called the secondary site. The remote site must be kept synchronized with the primary site, as updates are performed at the primary.

We achieve synchronization by sending all log records from primary site to the remote backup site. The remote backup site must be physically separated from the primary—for example; we can locate it in a different state- so that a disaster at the primary does not damage the remote backup site.

When the primary site fails, the remote backup site takes over processing.

- → First, however, it performs recovery, using its (perhaps outdated) copy of the data from the primary, and the log records received from the primary.
- → In effect, the remote backup site is performing recovery actions that would have been performed at the primary site when the latter recovered.
- → Once recovery has been performed, the remote backup site starts processing transactions.
- → When the backup site takes over processing it becomes the new primary and the old primary site become new backup site.

QUESTIONS TO FOCUS

1. What are the three types of failure? Explain each with example.
2. Justify the statement "Remote backup system is a must in country like Nepal'
3. Explain two types of recovery techniques.
4. Compare the shadow paging recovery scheme with the log based recovery schema.
5. What do you mean by deferred modification and immediate modification?