CHAPTER - 9

Crash Recovery

9.1 Introduction

A major responsibility of the database administrator is to prepare for the possibility of hardware, software, network, process, or system failure. If such a failure affects the operation of a database system, we must usually recover the database and return to normal operation as quickly as possible. Recovery should protect the database and associated users from unnecessary problems and avoid or reduce the possibility of having to duplicate work manually. Recovery processes vary depending on the type of failure that occurred, the structures affected, and the type of recovery performed.

Failure Classification

There are various types of failure that may occur in a system, each of which needs to be dealt with in a different manner. In this chapter, we shall consider only the following types of failure:

- **Transaction failure:** When a transaction is failed to execute or it reaches a point after which it cannot be completed successfully it has to abort. This is called transaction failure. Where only few transaction or process are hurt. Reason for transaction failure could be:
  - **Logical errors:** where a transaction cannot complete because of it has some code error or any internal error condition
  - **System errors:** where the database system itself terminates an active transaction because DBMS is not able to execute it or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability systems aborts an active transaction.

- **System crash:** There are problems, which are external to the system, which may cause the system to stop abruptly and cause the system to crash. For example interruption in power supply, failure of underlying hardware or software failure. Examples may include operating system errors.

- **Disk failure:** In early days of technology evolution, it was a common problem where hard disk drives or storage drives used to fail frequently. Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or part of disk storage. Copies of the data on other disks, or archival backups on tertiary media, such as DVD or tapes, are used to recover from the failure.

9.2 Recovery and Atomicity

When a system crashes, it many have several transactions being executed and various files opened for them to modifying data items. As we know that transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained that is, either all operations are executed or none. When DBMS recovers from a crash it should maintain the following:

- It should check the states of all transactions, which were being executed.
- A transaction may be in the middle of some operation; DBMS must ensure the atomicity of transaction in this case.
- It should check whether the transaction can be completed now or needs to be rolled back.
- No transactions would be allowed to leave DBMS in inconsistent state.

There are two types of techniques, which can help DBMS in recovering as well as maintaining the atomicity of transaction:

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory and later the actual database is updated.

A. Log-based recovery

The log is a sequence of **log records**, recording all the update activities in the database. There are several types of log records. An **update log record** describes a single database write. It has these fields:

- **Transaction identifier**, which is the unique identifier of the transaction that performed the write operation.
- **Data-item identifier**, which is the unique identifier of the data item written. Typically, it is the location on disk of the data item, consisting of the block identifier of the block on which the data item resides, and an offset within the block.
- **Old value**, which is the value of the data item prior to the write.
- **New value**, which is the value that the data item will have after the write.

Log based recovery works as follows:

- The log file is kept on stable storage media
- When a transaction enters the system and starts execution, it writes a log about it: <Tn, Start>
- When the transaction modifies an item X, it write logs as follows: <Tn, X, V1, V2>; It reads Tn has changed the value of X, from V1 to V2.
- When transaction finishes, it logs: <Tn, commit>

Whenever a transaction performs a write, it is essential that the log record for that write be created and added to the log, before the database is modified. Once a log record exists, we can output the modification to the database if that is desirable. Also, we have the ability to *undo* a modification that has already been output to the database. We undo it by using the old-value field in log records. For log records to be useful for recovery from system and disk failures, the log must reside in stable storage.

Checkpoints

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. At time passes log file may be too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in storage disk. Checkpoint declares a point before which the DBMS was in consistent state and all the transactions were committed. There are two major difficulties with this approach:

1. The search process is time-consuming.
2. Most of the transactions that, according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will nevertheless cause recovery to take longer.

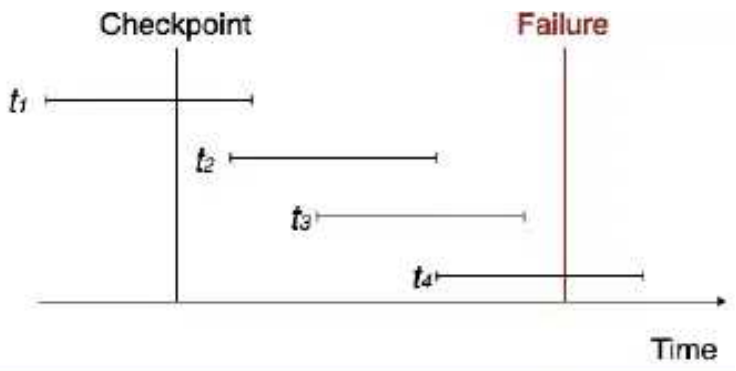
A checkpoint is performed as follows:

1. Output onto stable storage all log records currently residing in main memory.
2. Output to the disk all modified buffer blocks.
3. Output onto stable storage a log record of the form <checkpoint L>, where L is a list of transactions active at the time of the checkpoint.

Recovery

When system with concurrent transaction crashes and recovers, it does behave in the following manner:

- The recovery system reads the logs backwards from the end to the last Checkpoint.
- It maintains two lists, undo-list and redo-list.



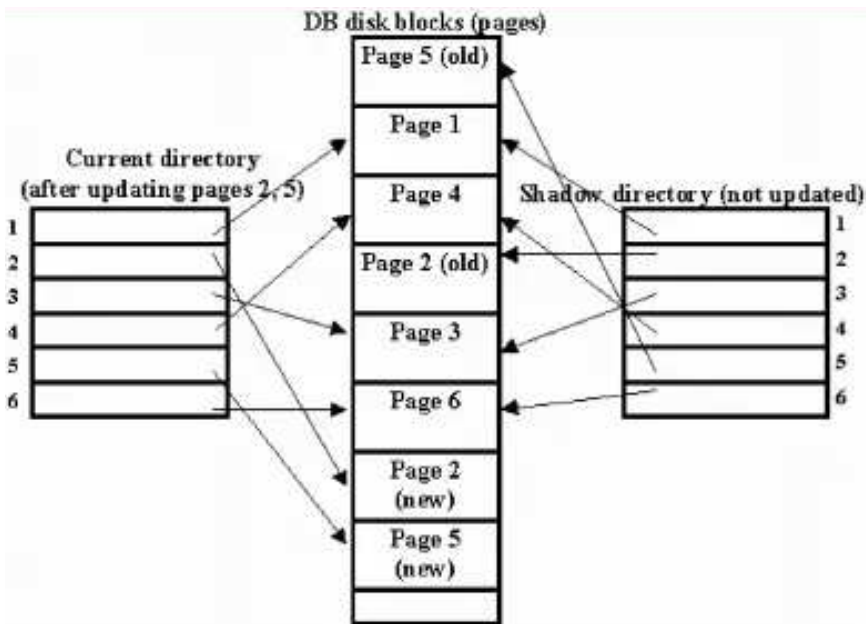
- If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in redo-list.
- If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in undo-list.

All transactions in undo-list are then undone and their logs are removed. All transaction in redo-list, their previous logs are removed and then redone again and log saved.

B. Shadow-paging

Shadow paging is a technique for providing atomicity and durability (two of the ACID properties) in database systems. A *page* in this context refers to a unit of physical storage (probably on a hard disk).

It is inconvenient to maintain logs of all transactions for the purposes of recovery. An alternative is to use a system of shadow paging. This is where the database is divided into pages that may be stored in any order on the disk. In order to identify the location of any given page, we use something called a page table. During the life of a transaction two page tables are maintained, one called a shadow page table and current page table. To start with, both the page tables are identical. Only current page table is used for data item accesses during execution of the transaction. Whenever any page is about to be written for the first time, a copy of this page is made onto an unused page. The current page table is then made to point to the copy, and the update is performed on the copy.



- Advantages of shadow-paging over log-based schemes – no overhead of writing log records; recovery is trivial

Disadvantages:

- Commit overhead is high (many pages need to be flushed)
- Data gets fragmented (related pages get separated)
- After every transaction completion, the database pages containing old versions of modified data need to be garbage collected and put into the list of unused pages
- Hard to extend algorithm to allow transactions to run concurrently

9.3 Data backup and Recovery

A **backup**, or the process of backing up, refers to the copying and archiving of computer data so it may be used to *restore* the original after a data loss event. The verb form is to **back up** in two words, whereas the noun is *backup*.

Backups are needed in case a file or a group of files is lost. The reasons for losing files include hardware failure like disk breaking, accidentally deleting wrong file and computer being stolen. Backups help in all the above situations. In addition, it may be good to have access to older versions of files, for example a configuration file worked a week ago.

A database backup consists of backups of the physical files (all data files and a control file) that constitute a database. Replacing a current, possibly damaged, copy of a data file, table space, or database with a backup copy is called *restoring* that portion of the database. Backups have two distinct purposes. The primary purpose is to recover data after its loss, be it by data deletion or corruption. The secondary purpose of backups is to recover data from an earlier time, according to a user-defined data retention policy, typically configured within a backup application for how long copies of data are required. Since a backup system contains at least one copy of all data worth saving, the data storage requirements can be significant. A data repository model can be used to provide structure to the storage. Before data are sent to their storage locations, they are selected, extracted, and manipulated. Many different techniques have been developed to optimize the backup procedure. These include optimizations for dealing with open files and live data sources as well as compression, encryption, and de-duplication, among others.

The basic types of Backup

There are many techniques for backing up files. The techniques you use will depend on the type of data you're backing up, how convenient you want the recovery process to be, and more. If you view the properties of a file or directory in Windows Explorer, you'll note an attribute called Archive. This attribute often is used to determine whether a file or directory should be backed up. If the attribute is on, the file or directory may need to be backed up. The basic types of backups you can perform include

- **Normal/full backups:** All files that have been selected are backed up, regardless of the setting of the archive attribute. When a file is backed up, the archive attribute is cleared. If the file is later modified, this attribute is set, which indicates that the file needs to be backed up.
- **Copy backups:** All files that have been selected are backed up, regardless of the setting of the archive attribute. Unlike a normal backup, the archive attribute on files isn't modified. This allows you to perform other types of backups on the files at a later date.
- **Differential backups:** Designed to create backup copies of files that have changed since the last normal backup. The presence of the archive attribute indicates that the file has been modified and only files with this attribute are backed up. However, the archive attribute on files isn't modified. This allows you to perform other types of backups on the files at a later date.
- **Incremental backups:** Designed to create backups of files that have changed since the most recent normal or incremental backup. The presence of the archive attribute indicates that the file has been modified and only files with this attribute are backed up. When a file is backed up, the archive attribute is cleared. If the file is later modified, this attribute is set, which indicates that the file needs to be backed up.
- **Daily backups** Designed to back up files using the modification date on the file itself. If a file has been modified on the same day as the backup, the file will be backed up. This technique doesn't change the archive attributes of files.

Recovery: Recovery of the database is *restored* to the most recent consistent state just before the time of failure. Usually the system log or trail or journal, keeps the information about the changes that were applied to the data items by the various transactions. Main recovery techniques are of two type:

A. Deffered update techniques

This technique do not physically update the database on disk until after a transaction reaches its commit point. Before reaching the commit point, 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 DB. If a transaction fails before reaching its commit point, no UNDO is needed because it will not have changed the database anyway. If there is a crash, it may be necessary to REDO the effects of committed transactions from the Log because their effect may not have been recorded in the database. Deferred update also known as NO-UNDO/REDO algorithm.

B. Immediate update techniques

In this technique, the DB may be updated by some operations of a transaction before the transaction reaches its commit point. To make recovery possible, force write the changes on the log before to apply them to the DB. If a transaction fails before reaching commit point, it must be rolled back by undoing the effect of its operations on the DB. It is also required to redo the effect of the committed transactions. Immediate update also known as UNDO/REDO algorithm. A variation of the algorithm where all updates are recorded in the database before a transaction commits requires only redo –UNDO/NO-REDO algorithm

9.4 Remote Backup Systems

Traditional transaction-processing systems are centralized or client–server systems. Such systems are vulnerable to environmental disasters 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**.

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. Figure shows the architecture of a remote backup system.

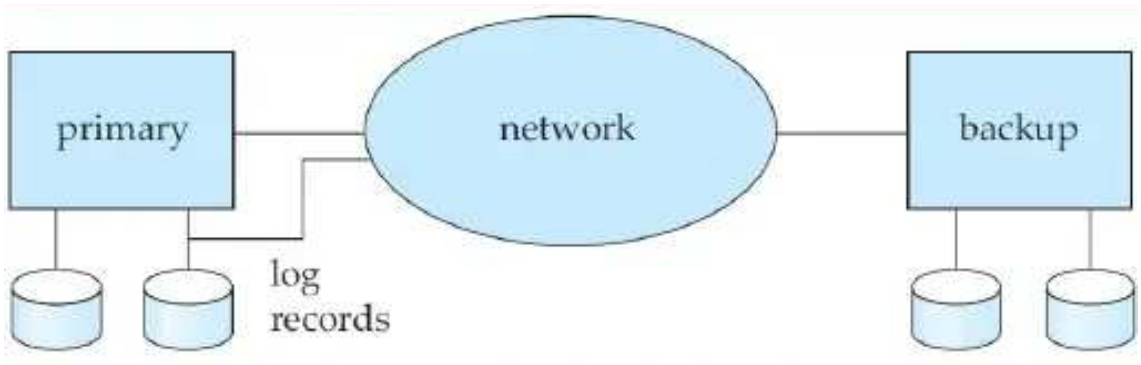


Figure Architecture of remote backup system.

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. Standard recovery algorithms, with minor modifications, can be used for recovery at the remote backup site. Once recovery has been performed, the remote backup site starts processing transactions.