CHAPTER - 9

CRASH RECOVERY

9.1 Failure Classification

To find where the problem has occurred, we generalize a failure into following categories:

i. Transaction Failure
ii. System Crash
iii. Disk Failure

Transaction Failure

The transaction failure occurs when it fails to execute or when it reaches a point it cannot go any further if few transactions or process is hurt, then this is called transaction failure.

Reasons for transaction failure could be:

a. Logical Errors

If a transaction cannot complete due to some code error or an internal error or an internal error condition then the logical error occurs.

b. System Error

System error occurs where the DBMS itself terminates an active transaction because the database system is not able to execute it.

Example:

The system aborts an active transaction in case of deadlock or resource unavailability.

System Crash

System crash can occur due to power failure or other hardware or software failure.

Example:

Operating System Error.

Database systems have numerous integrity checks to prevent corruption of disks data.

Disk Failure

Disk Failure occurs where hard disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution. Disk failure occurs due to the formation of bad sectors, disk head crash and unreachability to the disk or any other failure, which destroy all or part of the disk storage.

9.2 Concept of Log Based Recovery and Shadow Paging

9.2.1 Log Based Recovery

The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there. If any operation is performed on the database, then it will be recorded in the log. But the process of storing the logs should be done before the actual transaction is applied in 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 and V2 is the value to be written to X. Log record notes that Ti has performed a write on data item X. X had value V1 before the write and will have value V2 after the write. When Ti finishes its last statement, the log record <Ti, Commit> is written. We assume that log records are written directly to stable storage (that is they are not buffered)

Let us assume that, there is transaction to modify the city of a student. The following logs are written for this transaction.

When the transaction is initiated, then it writes 'start' log.

<Tn, Start>

When the transaction modifies the city from 'ktm' to 'Pokhara' then another log is written to the file.

<Tn, city, 'ktm', 'Pokhara'>

When the transaction is finished then it writes another log to indicate the end of the transaction.

<Tn, commit>

There are two approaches to use logs:

i. Deferred database modification.

ii. Immediate database Modification

Deferred Database Modification

The deferred modification technique occurs if the transaction does not modify the database until it has committed. In this method all the logs are created and stored in the stable storage and the database is updated when the transaction commits. The deferred database modification scheme records all modifications to the log but defers all the writes to partial commit.

If the system crashes before the transaction completes its execution or if the transaction aborts, then the information on the log is simply ignored. Deferred database modification technique ensures transaction atomicity by recording all database modification in the log. Assume that transactions execute serially Transaction starts by writing <Ti, Start> record to the log. A Write (X) operation results in a log record <Ti, X, V> being written where V is the new value of X.

Note: Old value is not needed for this scheme.

The write is not performed on X at this time 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. Using the logs, the system handles any failure that results in information loss. The recovery schemas use the following recovery procedures.

Redo (Ti) sets the value of all data items updated by transaction Ti to the new values. New value can be found in the log. The redo operation must be idempotent i.e executing it several times must be equivalent to executing it once. This is required if we are to generate correct behavior even if a failure occurs during the recovery process. During the recovery after a crash, a transaction needs to be redone if and only if both <Ti, Start> and <Ti, Commit> are there in the log. Redoing a transaction Ti (Redo Ti) sets the value of all data items updated by the transaction to the new value. Crashes can occur while the transaction is executing the original updates or while recovery action is being taken.

<To, start>	<To, start>	<To, start>
<T0, A,950>	<T0, A,950>	<T0, A,950>
<T0, B,2050>	<T0, B,2050>	<T0, B,2050>
	<To, Commit>	<To, Commit>
	<T1, Start>	<T1, Start>
	<T1, C,600>	<T1, C,600>
		<T1 Commit>
A	B	C

If log on stable storage at the time of crash is given as above (i.e., a, b, c). No redo actions need to be taken for case 'a'. Redo (T0) must be performed for case 'b' since <T0, Commit> is present. Redo (T0) must be performed followed by redo (T1) since <T0, Commit> and <T1, Commit> are present for case 'c'.

Immediate Database Modification

The immediate update technique allows database modification to output the database while the transaction is still in the active state. Data modification written by active transaction are called uncommitted modification. Since undoing may be needed. Update logs must have both old and new value. Recovery procedure has two operations instead of one.

i. **Undo (Ti)** restores the value of all data items updated by Ti to their old values, going backwards from the last record of Ti.

ii. **Redo (Ti)** sets the value of all data items updated by Ti to the new values going forward from the first log record of Ti.

Both operations must be idempotent i.e even if the operation is executed multiple times the effect is the same as if it has executed once. When recovering after failure :

i. Transaction Ti needs to be undone if the log contains the record <Ti, Start> but does not contain the record <Ti, Commit>

ii. Transaction Ti needs to be redone if the log contains both the record <Ti, Start> and the record <Ti, Commit>

Undo operations are performed first then redo operations.

<T0 start>
<T0,A,1000,950>
<T0,B,2000,2050>

Undo(To):
B is restored to 2000 and A to 1000

<T0, Start>
<T0,A,1000,950>
<T0,B,2000,2050>
<T0, Commit>

Undo(T1) and redo(T0):
C is restored to 700 and, then A and B are set to 950 and 2050 respectively

<T1, Start>
<T1,C,700,600>

<T0, Start>
<T0,A,1000,950>
<T0,B,2000,2050>
<T0, Commit>

Redo (T0) and Redo (T1): A and B are set to 950 and 2050 respectively. Then C is set to 600

<T1 start>
<T1,c,700,600>
<T1, Commit>

Check Pointing

Check pointing is the mechanism to mark the entries in the log file that those changes are permanently updated into database and if there is any failure log files need not traversed beyond that point. Only those entries after checkpoint are not written to DB and have to be redone/undone. This is done at periodic intervals or as per the schedule.

Checkpointing is done as follows:

i. Output all log records currently residing in main memory onto stable storage.

ii. Output all modified buffer blocks to the disk.

iii. Output a log record of the form <checkpoint L>, where L is a list of transactions active at the time of the checkpoint onto stable storage.

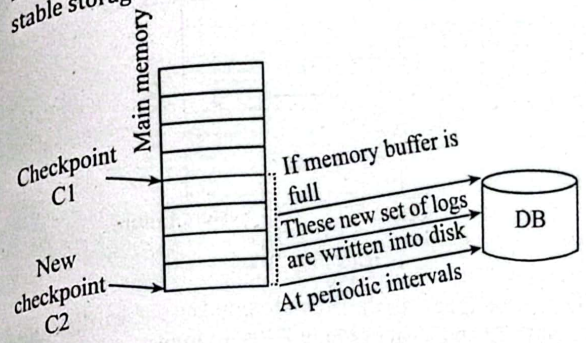

Figure: Checkpointing

Recovery using Checkpoints

Following problems occur in recovery procedure as discussed earlier:

i. Searching the entire log is time consuming

ii. We might unnecessarily redo transactions which have already output their updates to the database.

During recovery we need to consider only the most recent transaction T_i that started before the checkpoint

i. Scan backward from the end of the log to find the most recent <Checkpoint> record

ii. Continue scanning backwards till a record $<T_i$, Start> is found.

iii. Need only to consider the part of the log following above start record. Earlier part of the log is can be ignored during the recovery and can be erased whenever desired.

After identified T_i, the redo and undo operations to be applied to the T_i and all T_j that started execution after Transaction T_i. For all transactions (starting from T_i or later) with no <T_i, Commit> execute undo (T_i)(Done only in case of immediate modification. Scanning forward in the log for all transactions start from T_i or later with a <T_i, Commit> execute redo (T_i)

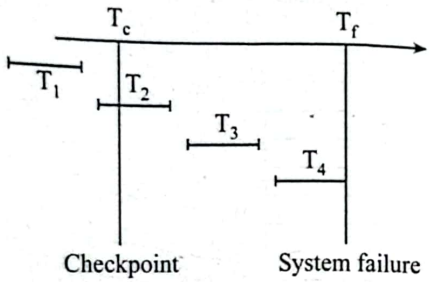

Figure: Checkpoint

T1 can be ignored (Updates already output to disk due to checkpoint). T2 and T3 are redone. T4 is undone

Fuzzy checkpointing

Fuzzy checkpointing, in contrast to normal check pointing allows transaction to execute while logs are being copied to disks. During fuzzy checkpointing it follows below steps:

- Temporarily stops the transactions to make note of blocks to be copied
- Marks the new checkpoint L in the log
- Creates a list M for all the logs between the last checkpoint to new checkpoint i.e M is the list of log records which are yet to be written to disk

  Once all M is listed, it allows the transaction to execute. Now the transaction will start entering the logs after the new checkpoint l. It shouldn't enter the logs into the blocks that are in M or old checkpoints
- The buffer blocks in list M are written to the disk or stable storage. No transactions should update these blocks. In addition all the records in these blocks in list M are written to the disk first and then the block is updated to the disk

- Disk should have pointer to the last checkpoint. Last checkpoint in the main memory at fixed location. This will help to read the blocks for the next update and maintain new list M.

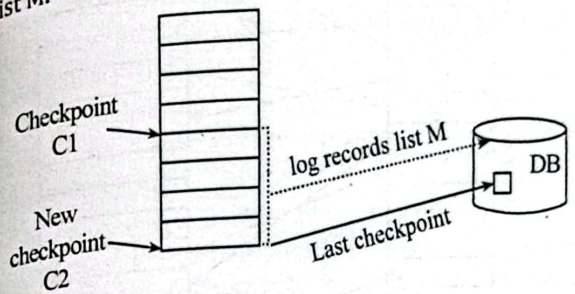

Figure: Fuzzy checkpointing

- Whenever there is a recovery from failure then logs are read from the last checkpoint stored in DB. This is because logs before last checkpoint are already updated to DB and those after these points have to be written. These logs are recovered as described in above methods

9.2.2 Shadow Paging

Shadow paging is an alternative to log-based recovery; this scheme is useful if transactions execute serially. The Idea of shadow paging is to maintain two page tables during the lifetime of a transaction –the current page table, and the shadow page table.

Store the shadow page table in nonvolatile storage, such that state of the database prior to transaction execution may recovered. Shadow page table is never modified during execut

To start with, both the page tables are identical. Only curr page table is used for data item accesses during execution of the transaction. When the transaction completes, the current page table becomes the shadow page table. Whenever any page is about to be written for the first time

i. A copy of this page is made onto an unused page.
ii. The current page table is then made to point to the copy

iii. The update is performed on the copy

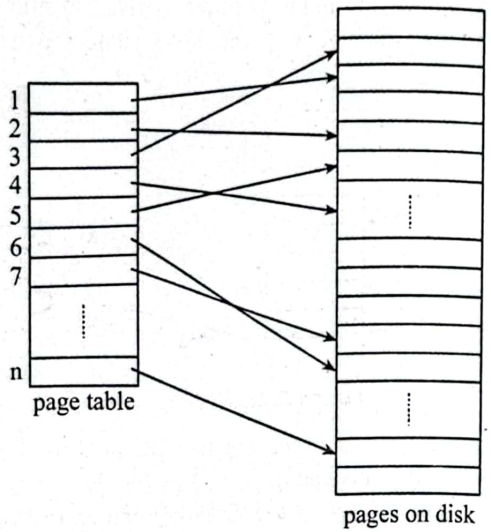

Figure: Paging

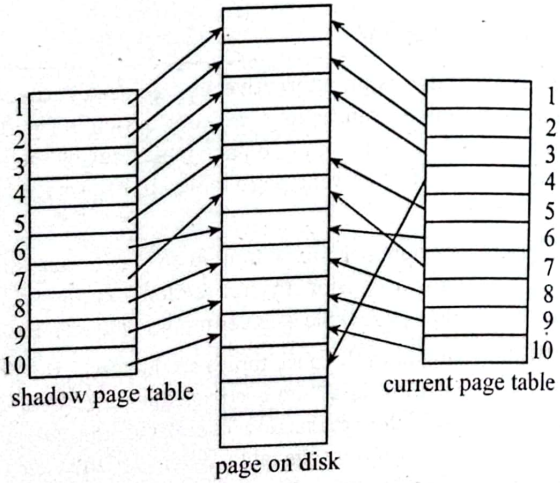

Figure: Shadow paging

To commit a transaction we must do the following:

i. Ensure that all buffer pages in main memory that have been changed by the transactions are output to the disk.

ii. Output the current page table to the disk. Note that we must not overwrite the shadow page table, since we may need it for recovery from a crash.

iii. Output the disk address of the current page table to the fixed location in stable storage containing the address of the shadow page table.

This action overwrites the address of the old shadow page table. Therefore, the current page table has become the shadow page table and the transaction is committed. If a crash occurs prior to the completion of step iii, the effects of the transaction will be preserved; no redo operations need to be invoked.

Following are the advantages of Shadow paging over log-based schemas:

i. No overhead of writing log to the records.

ii. Recovery from crashes is significantly faster (since no undo or redo operations are needed)

Following are the disadvantages:

i. **Commit overhead**

Requires multiple blocks to be output-the actual data blocks, the current page table and the disk address of the current page table (log-based schemas need to output only the log records) can be reduced by using a page table structured like a B+ tree. No need to copy entire tree, only to copy paths in the tree that lead to updated leaf nodes. Commit overhead is high even with above extension. Need to flush every updated page and page table

ii. **Data fragmentation**

Shadow paging cause database pages to change location when they are updated. Data gets fragmented (related pages get separated on disk)

iii. **Garbage collection**

After every transaction completion the database pages containing old versions of modified data need to be garbage collected. Hard to extend algorithm to allow transactions to run concurrently. Easier to extend log based schemes.

9.3 Data Backup/Recovery

9.3.1 Data Backup

When we back up our data, copies of our data are made to ensure availability and protect against data loss. Important files, databases, or entire systems are duplicated and stored in separate locations or storage media. Data backup is essential for following purpose:

i. **Data Protection**

Backups are designed to protect against accidental deletion, hardware failure, software corruption, cyberattacks, natural disasters, or other unforeseen events that can lead to data loss.

ii. **Business Continuity**

In the event of data loss or system failure, backups enable organizations to recover critical data and resume operations quickly, minimizing downtime and reducing impact on business continuity.

iii. **Data Recovery**

Backup provides the ability to recover lost or corrupted data. By restoring data from backups, organizations can restore previous versions of files or restore systems to a known good state.

iv. **Disaster Recovery**

In the event of a catastrophic incident such as a fire, flood, or cyberattack, offsite backups ensure that our data is safe and can be restored even if the primary location is inaccessible or compromised.

v. **Compliance and Legal Requirements**

Many industries and jurisdictions have specific data retention and compliance regulations. Regular backups help organizations meet these requirements and provide proof of data integrity.

Following aspects should be considered while implementing data backups:

i. **Backup Strategy**

Right backup strategy should be determined based on the data size, frequency of changes, recovery time objectives (RTOs), and recovery point objectives (RPOs) of the organization. Common backup strategies include full backup, incremental backup, differential backup, and continuous data protection.

ii. **Backup Frequency**

Backup schedule should be set up that suits business needs and the speed at which data changes. Important data may require more frequent backups than less important data.

iii. **Storage Media**

Appropriate storage medium should be chosen for backup, such as an external hard drive, network attached storage (NAS), tape drive, or cloud storage. Multiple backups can be created and stored on different media for redundancy.

iv. **Off-Site Storage**

Store backups in a physical location should be separated from primary data to protect against disasters that could impact the primary location.

v. **Automation and Monitoring**

Backup software or tools that automate the backup process and provide monitoring to ensure successful backups and detect errors or problems should be used.

vi. **Testing and Verification**

Regular test of the integrity and recoverability of backups should be done by performing restoration tests. This ensures

that the backup data is valid and can be successfully restored when needed.

Other aspects such as data security, access controls, and data retention policies should be considered to ensure the overall protection and effective management of data.

Types of Backup

There are several types of backups that organizations can use to protect their data. The choice of backup type depends on factors such as data importance, recovery time objective (RTO), storage capacity, and available resources. It is important to consider the specific requirements by the organization while choosing the appropriate backup types. Typically, businesses use a combination of backup types to balance data protection needs, recovery time, and storage efficiency. Here are some common types of backups:

i. **Full Backup**

A full backup involves making a copy of all selected data at a specific point in time. It includes all files, databases or system components. Full backups capture all data but can be time consuming and require significant storage space.

ii. **Incremental Backup**

Incremental backups only record changes made since the last backup, whether it's a full backup or an incremental backup. It backs up new or changed files since the last backup, reducing backup time and storage requirements. To restore data, the last full backup and all subsequent incremental backups must be applied.

iii. **Differential Backup**

Differential backup only records the changes made since the last full backup. Unlike incremental backups, they do not take into account previous differential backups. When restoring data, just the latest full backup and the latest differential backup are required

iv. **Synthetic Full Backup**

Composite full backups are created by combining a previous full backup with subsequent incremental backups. This approach eliminates the need to perform regular full backups while maintaining a full backup set. This can help reduce backup time and storage space requirements.

v. **Mirror Backup**

Mirror backup creates an exact copy or mirror image of the source data in real time or with minimal latency. Changes to the source data are immediately reflected in the mirror backup. Cloned backups offer high availability and fast recovery, but may require significant storage space.

vi. **Snapshot Backup**

Snapshot backup captures the state of data at a specific point in time, allowing for quick backup and recovery. They use copy-on-write or redirect-on-write techniques to create a virtual copy of point in time. Snapshots can be used as the basis for other types of backups.

vii. **Cloud Backup**

Cloud backup involves storing backup copies of data in offsite cloud storage. It provides scalability, ease of management, and protection against physical damage or loss. Cloud backup can use different backup types, such as full, incremental, or differential backup.

viii. **Continuous Data Protection (CDP)**

CDP captures every change made to data in real-time or near real-time, creating a continuous stream of backups. It enables granular recovery at any point in time, providing minimal data loss in case of failures or data corruption.

9.3.2 Data Recovery

Data recovery is the process of retrieving and restoring lost, deleted, corrupted or inaccessible data from storage media or the system. It involves the use of specialized techniques, tools and

expertise to recover data that is unavailable due to various reasons, such as accidental deletion, hardware or software failure, file system corruption, virus attack or malware, physical damage to storage media or other causes unforeseen events.

The data recovery process normally involves the following steps:

1. **Evaluation and Assessment**

   The first step is to assess the nature and extent of data loss. Data recovery specialists or software tools analyze the affected storage media or system to determine the cause of data loss and the likelihood of successful recovery.

2. **Data Recovery Method Selection**

   Based on the evaluation, suitable data recovery methods and techniques are selected. This may include file-level recovery, partition recovery, disk image creation and recovery, RAID data recovery, or other specialized methods depending on the specific situation.

3. **Data Recovery Execution**

   The selected data recovery methods are performed to recover lost or inaccessible data. This may involve the use of specialized hardware and software tools, or the involvement of professional data recovery services.

4. **Data Reconstruction and Repair**

   In the event of data corruption or damaged file system, data recovery may include repairing or rebuilding damaged components to restore the integrity and accessibility of the data.

5. **Data Verification and Validation**

   Once the data recovery is complete, the recovered data will be validated and verified to ensure its integrity and accuracy. This may involve verifying file and directory structures, performing checksums or validating hashes, and comparing the recovered data with known good copies or backups.

6. **Data Restoration**

   After successful data recovery and validation, the recovered data will be restored to its original location or other storage medium. This may involve transferring data to a new disk or system, rebuilding the RAID array, or restoring files and folders to their original directories.

9.4 Remote Backup Systems

Remote backup provides a sense of security in case the primary location where the database is located gets destroyed. Remote backup can be offline or real-time or online. In case it is offline, it is maintained manually.

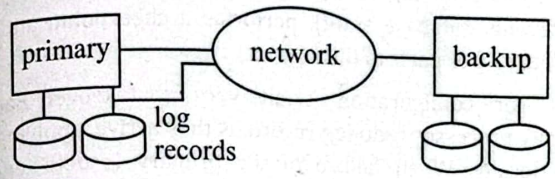

Figure: Remote backup systems

Online backup systems are more real-time and lifesavers for database administrators and investors. An online backup system is a mechanism where every bit of the real-time data is backed up simultaneously at two distant places. One of them is directly connected to the system and the other one is kept at a remote place as backup.

As soon as the primary database storage fails, the backup system senses the failure and switches the user system to the remote storage. Sometimes this is so instant that the users can't even realize a failure.

Detection of Failure

Backup site must detect when primary site has failed to distinguish primary site failure from link failure maintain several communication links between the primary and the remote backup.

Transfer of Control

To take over control backup site first perform recovery using its copy of the database and all the long records it has received from the primary. Thus, completed transactions are redone and incomplete transactions are rolled back.

When the backup site takes over processing it becomes the new primary To transfer control back to old primary when it recovers, old primary must receive redo logs from the old backup and apply all updates locally.

Time to Recover

To reduce delay in takeover, backup site periodically processes the redo log records (in effect, performing recovery from previous database state), performs a checkpoint, and can then delete earlier parts of the log.

Hot-Spare configuration permits very fast takeover: Backup continually processes redo log record as they arrive, applying the updates locally. When failure of the primary is detected the backup rolls back incomplete transactions, and is ready to process new transactions. Alternative to remote backup: distributed database with replicated data. Remote backup is faster and cheaper, but less tolerant to failure. Ensure durability of updates by delaying transaction commit until update is logged at backup; avoid this delay by permitting lower degrees of durability.

Time to Commit

To ensure that the updates of a committed transaction are durable, a transaction must not be declared committed until its log records have reached the backup site. This delay can result in a longer wait to commit a transaction, and some systems therefore permit lower degrees of durability. The degrees of durability can be classified as follows.

i. **One-safe**

One-safe configuration commits as soon as transaction's commit log record is written at primary. Problem in one safe is that the updates may not arrive at backup before it takes over.

ii. **Two-very-safe**

Two very safe commits when transaction's commit log record is written at primary and backup. It reduces availability since transactions cannot commit if either site fails.

iii. **Two-safe**

Two safe proceed as in two-very-safe if both primary and backup are active. If only the primary is active, the transaction commits as soon as is commit log record is written at the primary. Better availability than two-very-safe; avoids problem of lost transactions in one-safe.

SOLUTION TO EXAMS' AND OTHER IMPORTANT QUESTIONS

1. What happens for a log based recovery
<T0 Start>
<T0,A,1000,950>
<T0,B,2000,2050>
<T0,Commit>
<T1 Start>
<T1,C,700,600> [2021 Fall]

ANS: In log-based recovery, the transaction logs are used to recover the database to a consistent state after a failure. Here, we assume that the system has crashed after T1 started and before T1 committed. Therefore, T1 must be rolled back and T0 must be redone.

Here is the step-by-step recovery process using the transaction logs:

First, we identify the most recent checkpoint, which records the state of the database and the transaction table at a certain point in time. Since there is no checkpoint in the sequence given, we assume that the checkpoint is at the beginning of the log.

i. **Redo Pass**

In this pass, we redo all the committed transactions after the checkpoint in the log. In this case, T0 is the only committed transaction after the checkpoint, and it updates records A and B. Therefore, we apply the updates of T0 to the database, and the values of A and B are set to 950 and 2050, respectively.

ii. **Undo Pass**

In this pass, we undo the incomplete transactions that started after the checkpoint in the log. Here, we need to undo the changes made by T1, which updated record C. Since T1 has not committed, we simply discard the changes made by T1 and leave the value of C at 700.

After the recovery process is completed, the database is in a consistent state, and the changes made by the committed transaction T0 have been saved, while the changes made by the incomplete transaction T1 have been discarded.

2. Discuss several issues that must be addressed while designing remote backup system. [2017 Spring]

ANS: To maintain the integrity, dependability, and security of data, numerous elements must be carefully taken into account while designing a remote backup system. The following are a few critical challenges that must be handled during the design process:

i. **Data Transfer Efficiency:** Transferring enormous amounts of data across the network with as little impact as possible on network capacity and latency is one of the main issues in remote backup. The amount of data transported can be optimized and decreased via effective data compression, incremental backup methods, and intelligent data deduplication.

ii. **Data Encryption and Security:** Since sensitive information is carried through networks that may not be trusted, encryption is essential to prevent unauthorized access to or interception of the data. To secure data during transit, robust encryption techniques like SSL/TLS should be deployed. To ensure that only authorized people may access and restore the backed-up data, access controls and authentication measures should also be in place.

iii. **Redundancy and Reliability:** In order to guarantee data availability and resilience against hardware or network failures, remote backup systems should use redundancy methods. Utilizing redundant network lines or RAID setups can assist maintain data integrity and reduce downtime by implementing redundancy at both the storage and network levels.

iv. **Data Integrity and Verification:** Mechanisms for checking the reliability of backed-up data should be

included in backup systems. Checksums, digital signatures, and hash algorithms may be used in this procedure to identify and fix any data corruption or manipulation that occurred while the backup was being made. To ensure that data can be recovered when needed, routine integrity checks and validation procedures should be carried out.

v. **Scalability and Performance:** The backup system must to be built to manage growing data volumes and support future expansion. This necessitates scaling factors like load balancing, distributed storage structures, and parallel data transfer. Speeds for backup and restore can also be increased through performance optimizations like efficient indexing and metadata management.

vi. **Disaster Recovery Planning**

vii. **Compliance and Regulatory Requirements**

viii. **Monitoring and Alerting**

ix. **Data Retention and Archiving**

x. **Cost Considerations**

3. Compare the Shadow Paging recovery scheme and Log based recovery scheme. [2016 Fall,2015 Fall]

ANS: The difficulties with log-based approach are as follows:

- The search process is time consuming.
- Most of the transactions according to log-based 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.

In shadow paging system, the overhead of log-record output is eliminated, and recovery from crashes is significantly faster since no undo or redo operations are needed.

The commit of a single transaction using shadow paging requires multiple blocks to be output:

- The actual data blocks
- The current page tables
- The disk address of the current page table.

Log-based schemes need to output only the log records, which, for typical small transactions, fit within one block.

Shadow paging causes database pages to change location when they are updated. Hence, the locality property of the pages is lost. This results in the usage of complex, higher overhead schemes for physical storage management in shadow paging as compared to log-based recovery.

Also, garbage collection becomes necessary for shadow paging recovery system. Because, each time a transaction commits, the database pages containing the old version of data changed by the transaction become inaccessible. Such pages are considered garbage, since they are not part of free space and do not contain usable information. Garbage may be created also as a side effect of crashes. Periodically, it is necessary to find all the garbage pages, and to add them to the list of free pages. Thus, garbage collection in shadow paging recovery system results in additional overhead and complexity on the system.

4. In a log based recovery, how does deferred modifications scheme differ with immediate modification scheme? [2015 Spring]

ANS: Deferred modifications and immediate modifications are two methods for handling database updates and their corresponding logging schemes in a log-based recovery scheme. These two schemes differ in the following ways:

Deferred Modifications Scheme:

The database updates are initially made directly to the database in the deferred modifications method without immediately generating corresponding log records. This implies that during the actual update operation, the modifications are not instantly logged. Instead, the system waits until a later checkpoint or commit point before logging these modifications.

The database is kept updated and consistent under the deferred modifications method, but the corresponding log

records are created after the actual updates have been made. Since it avoids the overhead of writing log records for each individual modification, this technique offers better efficiency during the update process.

The deferred modifications technique introduces the danger that the system could lose the changes made since the last checkpoint or commit point if a failure happens before the updates are recorded. A checkpoint operation is periodically carried out to force the writing of all pending alterations to the log, minimizing this danger and assuring their persistence.

Immediate Modifications Scheme:

The immediate modifications approach, on the other hand, makes sure that each database update is promptly followed by a corresponding log entry. The system first creates a log record that contains information about the modification (such as the kind of operation, the affected data, and the old and new values) whenever a modification operation takes place. Before the actual database alteration is made, this log record is written to the log.

In comparison to the deferred modifications scheme, the immediate modifications scheme provides higher durability and recovery guarantees. The log records capture all the information required to restore the database to a consistent state in the event of a failure because they are generated before the changes are made. The recovery procedure uses the log data to undo or repeat the modifications as necessary in the event of a failure.

The necessity to produce log records for each update operation, however, adds extra overhead to the immediate modifications scheme. The update process' performance may be slightly impacted by this, particularly in high-volume transactional systems.

◆ ◆ ◆