Database Recovery Techniques

Outline

1. Recovery Basics
   1.1 Purpose of Database Recovery
   1.2 Types of Failure
   1.3 Transaction Log
   1.4 Types of Data Updates
   1.5 Transaction Roll-back (Undo) and Roll-Forward
   1.6 Checkpointing
2. Recovery Schemes
   2.1 Differed Update Recovery
   2.2 Immediate Update Recovery
   2.3 Shadow Paging
   2.4 ARIES Recovery Scheme
3. Recovery in Multi-database System

1.1 Purpose of Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure.
- To preserve transaction properties (Atomicity, Consistency, Isolation and Durability).

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.

1.2 Types of Failure

The database may become unavailable for use due to

- Transaction failure: Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
- System failure: System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
- Media failure: Disk head crash, power disruption, etc.

1.3 Transaction Log

3 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. A sample log is given below. **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			

1.4 Types of 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.

1.5 Transaction Rollback

To maintain consistency, a transaction's operations are **redone** or **undone**.

Undo (Roll-back): Restore all BFIMs on to disk (Remove all AFIMs).

Redo (Roll-forward): 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.

1.5 Transaction Rollback

Roll-back

Consider the following three transactions T1, and T2 and T3.

T1	T2	T3
read_item (A)	read_item (B)	read_item (C)
read_item (D)	write_item (B)	write_item (B)
write_item (D)	read_item (D)	read_item (A)
	write_item (A)	write_item (A)

1.5 Transaction Rollback

Roll-back: **One execution of T1, T2 and T3 as recorded in the log.**

A	B	C	D
30	15	40	20

	[start_transaction, T3]	
	[read_item, T3, C]	
*	[write_item, T3, B, 15, 12]	12
	[start_transaction,T2]	
	[read_item, T2, B]	
**	[write_item, T2, B, 12, 18]	18
	[start_transaction,T1]	
	[read_item, T1, A]	
	[read_item, T1, D]	
	[write_item, T1, D, 20, 25]	25
	[read_item, T2, D]	
**	[write_item, T2, D, 25, 26]	26
	[read_item, T3, A]	

---- system crash ----

* T3 is rolled back because it did not reach its commit point.

** T2 is rolled back because it reads the value of item B written by T3.
Since T1 reached its commit point, it need not be rolled back.

This is an example of a cascaded rollback; does not happen in strict schedule.

1.6 Checkpointing

Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery, and checkpointed.

The following steps defines a checkpoint operation:

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

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

2. Database Recovery

Database recovery depends on the type of update – deferred, immediate

Depending of the type of update, recovery can be No Undo/ No Redo, No Undo/ Redo, Undo/ No Redo or Undo/ Redo for committed and uncomited transactions.

2.1 Deferred Update Recovery

Deferred Update is No Undo/Redo.

The data update goes as follows:

1. A set of transactions records their updates in the log.
2. AFTER the 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.

(*WAL: Write-ahead Logging – BFIM is recorded in the log entry and then stored in the hard disk before BFIM is overwritten with AFIM)

2.1 Deferred Update Recovery

Deferred Update in a single-user system

There is no concurrent data sharing in a single user system. The recovery is strait-forward and follows the basic protocol.

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.

2.1 Deferred Update Recovery

Deferred Update in a single-user system

(a)

T1	T2
read_item (A)	read_item (B)
read_item (D)	write_item (B)
write_item (D)	read_item (D)
write_item (A)	

(b)

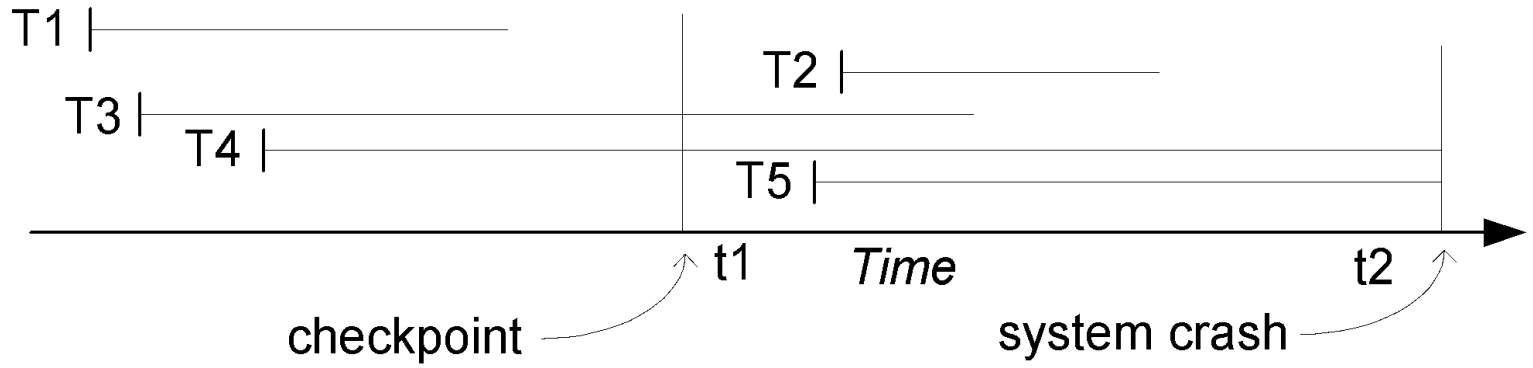
[start_transaction, T1]
[write_item, T1, D, 20]
[commit T1]
[start_transaction, T2]
[write_item, T2, B, 10]
[write_item, T2, D, 25] ← system crash

The [write_item, …] operations of T1 are redone.
T2 log entries are ignored by the recovery manager.

2.1 Deferred Update Recovery

Deferred Update with concurrent users

This environment requires some concurrency control mechanism to guarantee **isolation** property of transactions. In a system recovery transactions which were recorded in the log after the last checkpoint were **redone**. The recovery manager may scan some of the transactions recorded before the checkpoint to get the AFIMs.



Recovery in a concurrent users environment.

2.1 Deferred Update Recovery

Deferred Update with concurrent users

(a)

T1	T2	T3	T4
read_item (A)	read_item (B)	read_item (A)	read_item (B)
read_item (D)	write_item (B)	write_item (A)	write_item (B)
write_item (D)	read_item (D)	read_item (C)	read_item (A)
	write_item (D)	write_item (C)	write_item (A)

(b)

[start_transaction, T1]
[write_item, T1, D, 20]
[commit, T1]
[checkpoint]
[start_transaction, T4]
[write_item, T4, B, 15]
[write_item, T4, A, 20]
[commit, T4]
[start_transaction T2]
[write_item, T2, B, 12]
[start_transaction, T3]
[write_item, T3, A, 30]
[write_item, T2, D, 25] ← system crash

T2 and T3 are ignored because they did not reach their commit points.
T4 is redone because its commit point is after the last checkpoint. (Draw the time line to verify)

2.1 Deferred Update Recovery

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.

2.2 Immediate Update Recovery

Immediate update recovery can Undo/No-redo or Undo/ Redo

In this algorithm, AFIMs of a transaction are flushed to the database disk under WAL BEFORE it commits. (Hence the word immediate)

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**.

If the transactions are allowed to commit before all the changes are written to the disk, we have Undo/ Redo.

2.2 Immediate Update Recovery

Immediate update recovery algorithm

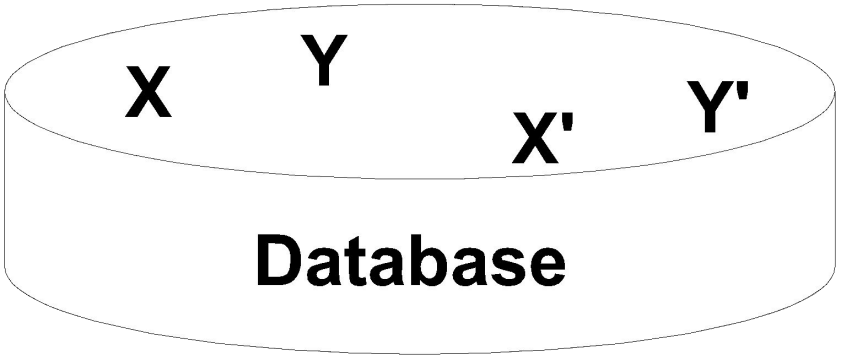
Note that at any time there will be one transaction in the system and it will be either in the commit table or in the active table. The recovery manager performs:

1. **Undo** of a transaction if it is in the **active** table.
2. **Redo** of a transaction if it is in the **commit** table.

In a single-user environment no concurrency control is required but in multi-user environment it is necessary.

2.3 Shadow Paging Recovery

The AFIM does not overwrite its BFIM 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. (in blocks of data)

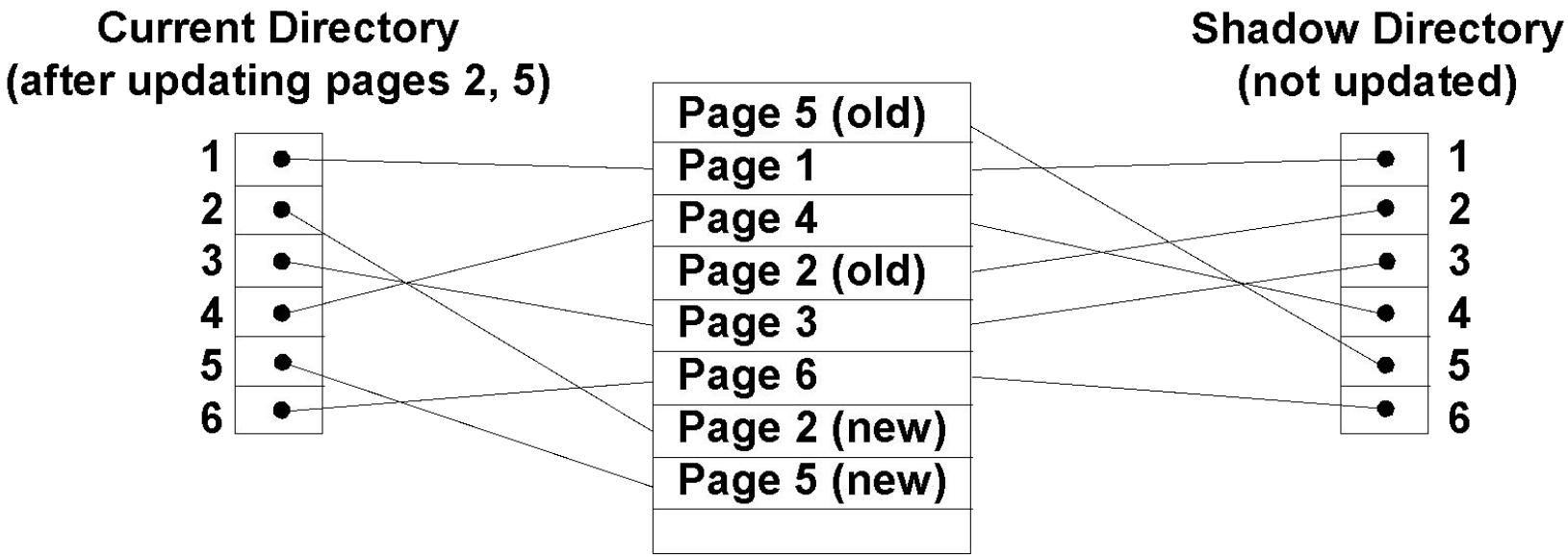


X and Y: Shadow copies of data items

X` and Y`: Current copies of data items

2.3 Shadow Paging Recovery

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.



2.3 Shadow Paging Recovery

During execution, shadow directory is not modified but a modified copy is saved some where else.

Committing – discarding the previous shadow directory

Recovering from a failure – discard the current directory

This is No redo/ No undo type

2.4 ARIES Recovery Algorithm

ARIES Recovery Algorithm is Redo/ Undo

- Uncommitted transactions (active transactions) at the time of transaction and committed transactions before a checkpoint are undone.

- Committed transactions after a check point are redone.

Analysis - identifies the dirty (updated) pages in the buffer and the set of transactions active at the time of crash. The appropriate point in the log where redo is to start is also determined.

2.4 ARIES Recovery Algorithm

The Log and Log Sequence Number (LSN)

A log record is written for all transactions. (update, commit etc.)

A unique LSN is associated with every log record. LSN increases monotonically and indicates the disk address of the log record it is associated with. In addition, each data page stores the LSN of the latest log record corresponding to a change for that page.

A log record stores (a) the previous LSN of that transaction, (b) the transaction ID, and (c) the type of log record.

2.4 ARIES Recovery Algorithm

The Transaction table and the Dirty Page table

For recovery following tables are also stored in the log during checkpointing:

Transaction table: Contains an entry for each active transaction, with information such as transaction ID, transaction status and the LSN of the most recent log record for the transaction.

Dirty Page table: Contains an entry for each dirty page in the buffer, which includes the page ID and the LSN corresponding to the earliest update to that page.

2.4 ARIES Recovery Algorithm

The following steps are performed for recovery

1. **Analysis phase**: Start at the begin_checkpoint record and proceed to the end_checkpoint record. Access transaction table and dirty page table are appended to the end of the log. Note that during this phase some other log records may be written to the log and transaction table may be modified. The analysis phase compiles the set of redo and undo to be performed and ends.

2. **Redo phase**: Starts from the point in the log up to where all dirty pages have been flushed, and move forward to the end of the log. Any change that appears in the dirty page table is redone.

3. **Undo phase**: Starts from the end of the log and proceeds backward while performing appropriate undo. For each undo it writes a compensating record in the log.

The recovery completes at the end of undo phase.

2.4 ARIES Recovery Algorithm

An example

(a)

LSN	LAST-LSN	TRAN-ID	TYPE	PAGE-ID	Other Info.
1	0	T1	update	C	-----
2	0	T2	update	B	-----
3	1	T1	commit		-----
4	begin checkpoint				
5	end checkpoint				
6	0	T3	update	A	-----
7	2	T2	update	C	-----
8	7	T2	commit		-----

(b)

TRANSACTION TABLE			DIRTY PAGE TABLE	
TRANSACTION ID	LAST LSN	STATUS	PAGE ID	LSN
T1	3	commit	C	1
T2	2	in progress	B	2

(c)

TRANSACTION TABLE			DIRTY PAGE TABLE	
TRANSACTION ID	LAST LSN	STATUS	PAGE ID	LSN
T1	3	commit	C	1
T2	8	commit	B	2
T3	6	in progress	A	6

2.4 ARIES Recovery Algorithm

Smallest LSN of dirty page table is 1. So, 1, 2, 6, 7 redone.

Undo is applied to active transaction T3.

3. Recovery in Multi-database System

– A multidatabase system is a special distributed database system where one node may be running relational database system under Unix, another may be running object-oriented system under window and so on.

3. Recovery in Multi-database System

– A transaction may run in a distributed fashion at multiple nodes. In this execution scenario the transaction commits only when all these multiple nodes agree to commit individually the part of the transaction they were executing.

3. Recovery in Multi-database System

– This commit scheme is referred to as "*two-phase commit*" (2PC). If any one of these nodes fails or cannot commit the part of the transaction, then the transaction is aborted. Each node recovers the transaction under its own recovery protocol.