Chapter 23

Database Recovery Techniques

Chapter 23 Outline

1 Purpose of Database Recovery
2 Types of Failure
3 The Log File
4 Data Cache and Buffers
5 Data Updating
6 Roll-back (Undo) and Roll-Forward (Redo)
7 Checkpointing
8 Recovery Techniques
9 Example: ARIES Recovery Scheme
10 Recovery in Multidatabase Transactions

Purpose of Database Recovery

- To bring the database into a consistent state after a failure occurs.
- To ensure the transaction properties of **Atomicity** (a transaction must be done in its entirety; otherwise, it has to be rolled back) and **Durability** (a committed transaction cannot be canceled and all its updates must be applied permanently to the database).
- After a failure, the **DBMS recovery manager** is responsible for bringing the system into a consistent state before transactions can resume.

Types of Failure

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

The Log File

- Holds the information that is necessary for the recovery process
- Records all relevant operations in the order in which they occur (looks like a *schedule of transactions*, see Chapter 21)
- Is an append-only file.
- Holds various types of log records (or log entries).

Log File Entries (from Chapter 21)

Types of records (entries) in log file:

- [start_transaction,T]: Records that transaction T has started execution.
- [write_item,T,X,old_value,new_value]: T has changed the value of item X from old_value to new_value.
- [read_item,T,X]: T has read the value of item X (not needed in many cases).
- [end_transaction,T]: T has ended execution
- [commit,T]: T has completed successfully, and committed.
- [abort,T]: T has been aborted.

The Log File (cont.)

For **write_item** log entry, *old value* of item before modification (**BFIM** - BeFore Image) and the *new value* after modification (**AFIM** – AFter Image) are stored. BFIM needed for UNDO, AFIM needed for REDO. 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			

Database Cache

Database Cache: A set of *main memory* buffers; each buffer typically holds contents of one disk block. Stores the disk blocks that contain the data items being read and written by the database transactions.

Data Item Address: (disk block address, offset, size in bytes).

Cache Table: Table of entries of the form (buffer addr, disk block addr, modified bit, pin/unpin bit, ...) to indicate which disk blocks are currently in the cache buffers.

Database Cache (cont.)

Data items to be modified are first copied into database cache by the Cache Manager (CM) and after modification they are flushed (written) back to the disk. The flushing is controlled by **Modified** and **Pin-Unpin** bits.

Pin-Unpin: If a buffer is pinned, it cannot be written back to disk until it is unpinned.

Modified: Indicates that one or more data items in the buffer have been changed.

Data Update

- **Immediate Update**: A data item modified in cache can be written back to disk *before the transaction commits*.
- **Deferred Update**: A modified data item in the cache cannot be written back to disk till *after the transaction commits* (buffer is **pinned**).

- **Shadow update**: The modified version of a data item does not overwrite its disk copy but is written at a separate disk location (new version).
- **In-place update**: The disk version of the data item is overwritten by the cache version.

UNDO and REDO Recovery Actions

To maintain atomicity and durability, some transaction's may have their operations **redone** or **undone** during recovery. UNDO (roll-back) is needed for transactions that are not committed yet. REDO (roll-forward) is needed for committed transactions whose writes may have not yet been flushed from cache to disk.

Undo: Restore all BFIMs from log to database on disk. UNDO proceeds backward in log (from most recent to oldest UNDO).

Redo: Restore all AFIMs from log to database on disk. REDO proceeds forward in log (from oldest to most recent REDO).

Write-ahead Logging Protocol

The information needed for recovery must be written to the log file on disk before changes are made to the database on disk. **Write-Ahead Logging** (WAL) protocol consists of two rules:

For Undo: Before a data item's AFIM is flushed to the database on disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved to disk.

For Redo: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

Checkpointing

To minimize the **REDO** operations during recovery. The following steps define a checkpoint operation:

- Suspend execution of transactions temporarily.
- Force write modified buffers from cache to disk.
- Write a [checkpoint] record to the log, save the log to disk. This record also includes other info., such as the *list of active transactions* at the time of checkpoint.
- Resume normal transaction execution.

During recovery **redo** is required only for transactions that have committed *after the last [checkpoint] record* in the log.

Checkpointing (cont.)

Steps 1 and 4 in previous slide are not realistic.

A variation of checkpointing called **fuzzy checkpointing** allows transactions to continue execution during the checkpointing process.

We discuss fuzzy checkpointing in the ARIES protocol later.

Other Database Recovery Concepts

Steal/No-Steal and Force/No-Force

Specify how to flush database cache buffers to database on disk:

Steal: Cache buffers updated by a transaction may be flushed to disk before the transaction commits (recovery may require UNDO).

No-Steal: Cache buffers cannot be flushed until after transaction commit (NO-UNDO). (Buffers are *pinned* till transactions commit).

Force: Cache is flushed (forced) to disk before transaction commits (NO-REDO).

No-Force: Some cache flushing may be deferred till after transaction commits (recovery may require REDO).

These give rise to four different ways for handling recovery:

Steal/No-Force (Undo/Redo), Steal/Force (Undo/No-redo), No-Steal/No-Force (Redo/No-undo), No-Steal/Force (No-undo/No-redo).

Deferred Update (NO-UNDO/REDO) Recovery Protocol

System must impose **NO-STEAL** rule. Recovery subsystem analyzes the log, and creates two lists:

Active Transaction list: All active (uncommitted) transaction ids are entered in this list.
Committed Transaction list: Transactions committed after the last checkpoint are entered in this table.

During recovery, transactions in **commit** list are **redone**; transactions in **active** list are *ignored* (because of NO-STEAL rule, none of their writes have been applied to the database on disk). Some transactions may be **redone** twice; this does not create inconsistency because **REDO** is "**idempotent**", that is, one REDO for an AFIM is equivalent to multiple REDO for the same AFIM.

Deferred Update (NO-UNDO/REDO) Recovery Protocol (cont.)

Advantage: Only **REDO** is needed during recovery.

Disadvantage: Many buffers may be pinned while waiting for transactions that updated them to commit, so system may run out of cache buffers when requests are made by new transactions.

UNDO/NO-REDO Recovery Protocol

In this method, **FORCE** rule is imposed by system (AFIMs of a transaction are flushed to the database on disk under Write Ahead Logging *before the transaction commits*).

Transactions in active list are **undone**; transactions in committed list are ignored (because based on FORCE rule, all their changes are already written to the database on disk).

Advantage: During recovery, only **UNDO** is needed.

Disadvantages: 1. Commit of a transaction is delayed until all its changes are force-written to disk. 2. Some buffers may be written to disk multiple times if they are updated by several transactions.

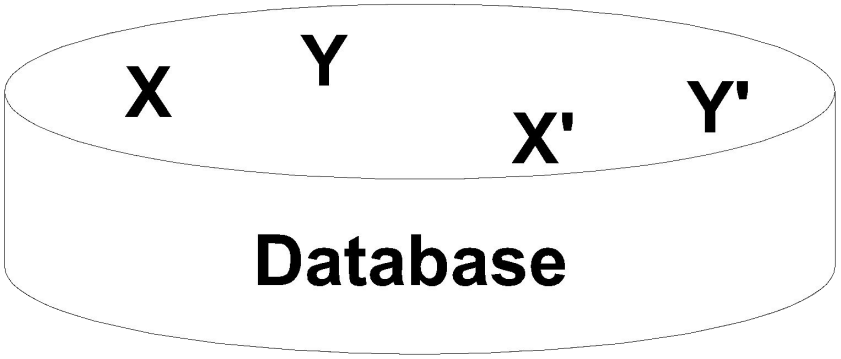
UNDO/REDO Recovery Protocol

Recovery can require both UNDO of some transactions and REDO of other transactions (Corresponds to STEAL/NO-FORCE). Used most often in practice because of disadvantages of the other two methods. To minimize REDO, checkpointing is used. The recovery performs:

1. **Undo** of a transaction if it is in the active transaction list.
2. **Redo** of a transaction if it is in the list of transactions that committed since the last checkpoint.

Shadow Paging (NO-UNDO/NO-REDO)

The AFIM does not overwrite its BFIM but is recorded at another place (new version) on the disk. Thus, a data item can have AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.



X and Y: Shadow (old) copies of data items
X` and Y`: Current (new) copies of data items

ARIES Database Recovery

Used in practice, it is based on:

1. WAL (Write Ahead Logging)
2. Repeating history during redo: ARIES will retrace all actions of the database system prior to the crash to reconstruct the correct database state.
3. Logging changes during undo: It will prevent ARIES from repeating the completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process.

ARIES Database Recovery (cont.)

The ARIES recovery algorithm consists of three steps:

1. **Analysis**: step identifies the dirty (updated) page buffers in the cache and the set of transactions active at the time of crash. The set of transactions that committed after the last checkpoint is determined, and the appropriate point in the log where redo is to start is also determined.
2. **Redo**: necessary redo operations are applied.
3. **Undo**: log is scanned backwards and the operations of transactions active at the time of crash are undone in reverse order.

ARIES Database Recovery (cont.)

The Log and Log Sequence Number (LSN)

A log record is written for (a) data update (Write), (b) transaction commit, (c) transaction abort, (d) undo, and (e) transaction end. In the case of undo a *compensating* log record is written.

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.

ARIES Database Recovery (cont.)

The Log and Log Sequence Number (LSN) (cont.)

A log record stores:

1. LSN of previous log record for same transaction: It links the log records of each transaction.
2. Transaction ID.
3. Type of log record.

For a write operation, additional information is logged:

1. Page ID for the page that includes the item
2. Length of the updated item
3. Its offset from the beginning of the page
4. BFIM of the item
5. AFIM of the item

ARIES Database Recovery (cont.)

The Transaction table and the Dirty Page table

For efficient recovery, the 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 (buffer) in the cache, which includes the page ID and the LSN corresponding to the earliest update to that page.

ARIES Database Recovery (cont.)

"Fuzzy" Checkpointing

A checkpointing process does the following:

1. Writes a *begin_checkpoint* record in the log, then forces updated (dirty) buffers to disk.
2. Writes an *end_checkpoint* record in the log, along with the contents of transaction table and dirty page table.
3. Writes the LSN of the *begin_checkpoint* record to a special file. This special file is accessed during recovery to locate the last checkpoint information.

To allow the system to continue to execute transactions, ARIES uses "fuzzy checkpointing".

ARIES Database Recovery (cont.)

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 that were appended to the log. During this phase some other records may be written to the log and the transaction table may be modified. The analysis phase compiles the set of redo and undo operations to be performed and ends.
2. **Redo phase**: Starts from the point in the log where all dirty pages have been flushed, and move forward. Operations of committed transactions are 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.

Recovery in Multi-database Transactions (Two-phase commit)

A multidatabase transaction can access several databases: e.g. airline database, car rental database, credit card database. The transaction commits only when all these multiple databases agree to commit individually the part of the transaction they were executing. This commit scheme is referred to as "*two-phase commit*" (2PC). If any one of these nodes fails or cannot commit its part of the transaction, then the whole transaction is aborted. Each node recovers the transaction under its own recovery protocol.

Two-phase commit (cont.)

Phase 1: Coordinator (usually application program running in middle-tier of 3-tier architecture) sends “Ready-to-commit?” query to each participating database, then waits for replies. A participating database replies Ready-to-commit only after saving all actions in its local log on disk.

Phase 2: If coordinator receives Ready-to-commit signals from all participating databases, it sends Commit to all; otherwise, it send Abort to all.

This protocol can survive most types of crashes.

Chapter 23 Summary

1 Purpose of Database Recovery
2 Types of Failure
3 The Log File
4 Data Cache and Buffers
5 Data Updating
6 Roll-back (Undo) and Roll-Forward (Redo)
7 Checkpointing
8 Recovery Techniques
9 Example: ARIES Recovery Scheme
10 Recovery in Multidatabase Transactions