Common Database Recovery Techniques

Dr. Rafal A. Angryk

Outline

Database Recovery Techniques

- **The system log, commit points and checkpoints**
- **Caching and writing to disk**
- **Two techniques for recovery**
  - **Deferred updates**
  - **Immediate updates**

Why do we need Database Recovery?

1. **To bring the database into the last consistent state, which existed prior to the failure.**
2. **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 consistent state before the transaction modified any of the accounts.

Types of Failure

- **Transaction failure: caused by:**
  - **incorrect input, deadlock, incorrect synchronization.**
- **System failure: caused by:**
  - **addressing error, application error, operating system fault, RAM failure, etc.**
- **Media failure: power disruption, etc.**

Common types of Data Update techniques

- **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.**
- **In-place update: The disk version of the data item is overwritten by the cache version.**

Transaction Log

- **For recovery from failure the logs contain the following data values.**
  - **BFIM (BeFore Image) - the value prior to modification and**
  - **AFIM (AFter Image) - the new value after modification**
- **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	2	Begin			
T1	1	4	Write	X	X = 100	X = 200
T2	0	8	Begin			
T1	2	5	Write	Y	Y = 50	Y = 100
T1	4	7	Read	M	M = 200	M = 200
T3	0	9	Read	N	N = 400	N = 400
T1	5	null	End			

The DBMS caching (1)

Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.

The DBMS caching (2)

- Caching of disk pages is traditionally an operating system function
  - Since WRITE's are so important, the DBMS handles it usually by calling the low-level OS routines
- The DBMS keeps a directory of RAM buffers (copies of disk blocks stored in the memory). It is called the DBMS cache.
- The buffer directory contains a table of <disk block address, buffer location>
  - Similar to page tables in OS's with paging

The DBMS caching (3)

- It is often necessary to replace (flush) some of the cache buffers to make space for new items.
  - Different strategies are used
    - LRU – least recently used
    - FIFO – first in first out
- Each buffer has several bits or flags associated with it
  - The dirty bit – set to 0 if the buffer has not been modified since last write to disk, set to 1 if it has
  - The pin-unpin bit – set to 1 if it cannot be written back to disk yet (usually waiting for a commit)

Writing buffers to disk (1)

Terms used to specify when a buffer can be written to disk

- Steal/no-steal modes:
  - Steal: Cache can be flushed before transaction commits.
  - No-Steal: Cache cannot be flushed before transaction commit.
  - Steal is used when the DBMS cache manager steals one buffer frame to use for another transactions
    - Of course, they have to be written to disk before the are overwritten
  - Some buffers must not be written to disk yet because the transaction has not yet committed
  - Buffers that are no-steal have the pin-unpin bit set

Writing buffers to disk (2)

Terms used to specify when a buffer can be written to disk

- Force/no-force modes:
  - Force: Cache is immediately flushed (forced) to disk.
  - No-Force: Cache is deferred until transaction commits.
  - If all pages updated by a transaction are immediately written to disk when the transaction commits, this is a force

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) and No-Steal/Force (No-undo/No-redo).

Writing buffers to disk (3)

- Typical database systems use a steal/no-force strategy
  - The advantage of steal is that less RAM has to be set aside for buffers, since DBMS is not forced to keep all updated pages in the RAM
  - The advantage of no-force is that an updated buffer may still be in RAM when anther transaction needs to update it (less I/O operations)

Flushing the buffers

Two common strategies, used when a buffer is written to disk:

- **In-place updating**
  - **Writes the buffer back to the same original disk location**
  - **It overwrites the old values of any changed data items**
  - **Recovery requires a log to undo and/or redo !!!**
- **Shadowing**
  - **Write an updated buffer to a different disk location**
  - **Thus the old values are not overwritten**
  - **May take a lot of disk space, but then it is not strictly necessary to keep a log of the BFIM (Before Image) and AFIM**

Transaction Roll-back (Undo) and Roll-Forward (Redo)

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

Undo: Restore all BFIMs (Before Images) on the disk (Remove all AFIMs).

Redo: Restore all AFIMs (After Images) on the 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.

Write ahead logging

- **If in-place updating is used, a log is necessary for recovery**
- **The log must be flushed to disk before the BFIM is replaced by the AFIM in the database**
- **Character of information kept in the log depends on whether the recovery mechanism needs to redo or undo, or both**
  - **A redo needs the AFIM (after image)**
  - **An undo needs the BFIM (before image)**

Checkpoints in the System Log

- **Another type of entry in the log is called a checkpoint.**
- **Checkpoints are often used with a steal/no-force protocol.**
- **Taking a checkpoint involves:**
  - **"force-writing" the contents of the DB buffers to disk**
  - **writing a checkpoint record to the log on disk. It contains**
    - **all transactions that were in progress at the time the checkpoint was taken.**

How checkpoints are used(1)

Time to time (randomly or under some criteria) the database flushes its buffer to database disk to minimize the task of recovery. The following steps defines a checkpoint operation:

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

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

How checkpoints are used(2)

1. **During a system crash, the contents of the db buffers are lost.**
2. **If a transaction had not successfully completed, it must be undone.**
3. **If it did complete, but had not yet been written to disk, it must be redone.**
4. **Whether to undo or redo needs to be decided – this is when we use the checkpoint record**

Deferred update (1)

- The database is not actually updated until after a transaction commits.
- Before commit, the updates are recorded in the transaction workspace. (memory buffer)
  - This is impractical for large DBs with many large transactions
- During commit, the updates are first recorded to the log which is written to disk
  - Only the AFIM is needed in the log
- After the log is written to disk, the buffers containing all the updates are written to the database

Deferred update (2)

- If a transaction fails, there is no need for UNDO.
  - the DB has not been changed
- May have to REDO, if the failure happened after the commit, before writing the update was complete.
- Known as no-undo/redo algorithm
- The redo operation is required to be idempotent
  - executing it over and over is equivalent to executing it just once.
  - This is necessary because the system may fail during recovery.

Immediate Update (1)

- The database may be updated before the transaction reaches a commit point.
- The operations are first recorded in the log on disk before they are applied to the DB system.
- If a transaction fails after recording changes but before committing, the transaction must be rolled back. (operations undone)

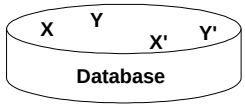
Immediate Update (2)

Two main categories

- undo/no-redo uses steal-force
  - all updates are recorded on the disk before the transaction commits
- undo/redo uses steal/no-force
  - The transaction is allowed to commit before all its changes are written to the DB system
  - This leaves a committed transaction not written to disk yet.

Shadow Paging (1)

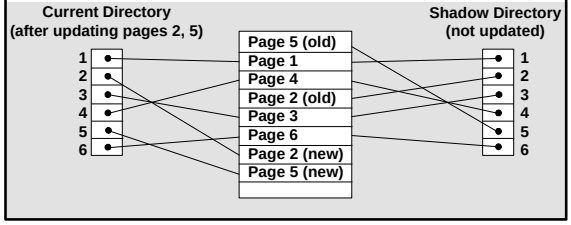
The AFIM does not overwrite its BFIM but recorded at another place on the disk. Thus, at any time a data item has AFIM and BFIM (Shadow copy of the data item) at two different places on the disk.



X and Y: Shadow copies of data items
X` and Y`: Current copies of data items

Shadow Paging (2)

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.



Practice (1)

- The following log corresponds to a particular schedule at the point of a system crash for the four transactions T1, T2, T3, and T4.

```
[start_transaction, T1]
[read_item, T1, A]
[read_item, T1, D]
[write_item, T1, D, 20]
[commit, T1]
[checkpoint]
[start_transaction, T2]
[read_item, T2, B]
[write_item, T2, B, 12]
[start_transaction, T4]
[read_item, T4, B]
[start_transaction, T3]
[write_item, T3, A, 30]
[read_item, T4, A]
[write_item, T4, A, 20]
[commit, T3]
[read_item, T2, D]
[write_item, T2, D, 25]
<---- System crash
```

Practice (2)

- Suppose we use the immediate update protocol (undo/redo category) with checkpointing.
- Specify which transactions are rolled back, which operations in the log are redone, and which (if any) are undone, and
- whether any cascading rollback takes place.