Database recovery techniques

Instructor: Mr Mourad Benchikh

**Text Books:** Database fundamental -Elmesri & Navathe Chap. 21
Database systems the complete book – Garcia, Ullman & Widow Chap. 17
Oracle9i Documentation

First-Semester 1427-1428

- Recovery algorithms are techniques to ensure transaction atomicity and durability despite failures
  - The recovery subsystem, using recovery algorithm, ensures atomicity by undoing the actions of transactions that do not commit and durability by making sure that all actions of committed transactions survive even if failures occur.

- Two main approaches in recovery process
  - Log-based recovery using WAL protocol.
  - Shadow-paging
Recovery Outline

– Recovery from transaction failures means restores the DB to the most recent consistent state just before the time of failure.

– Usually the system log –the log, sometimes called trail or journal- keeps the information about the changes that were applied to the data items by the various transactions.

– A typical strategy for recovery:
  
  • If there is a catastrophic failure (se chap. 19) –i.e. a disk crash-, restores a past copy of the database that was backed up to archival storage –typically tape- and reconstructs a more current state by reapplying the operations of committed transactions from the backed up log up to the time of failure.

  • If there is an inconsistency due to non-catastrophic failure, reverse any changes that caused the inconsistency and if necessary, reapply some operations in order to restore a consistent state of the database.
Recovery Outline (cont’d)

– Main recovery techniques.

1. Deffered update techniques.
   – 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.
Recovery Outline (cont’d)

2. Immediate update techniques.
   – 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-
Caching –i.e. buffering- of disk blocks

- Cache the disk pages (blocks)–containing DB items to be updated– into main memory buffers.
- Then update the main memory buffers before being written back to disk.
- For the efficiency of recovery purpose, the caching of disk pages is handled by the DBMS instead of the OS.
- Typically, a collection of in-memory buffers, called DBMS cache kept under the control of the DBMS.
- A directory for the cache is used to keep track of which DB items are in the buffers.
  - A table of <disk page address, buffer location> entries.
- The DBMS cache holds the database disk blocks including
  - Data blocks
  - Index blocks
  - Log blocks
Caching of disk blocks (cont’d)

- When DBMS requests action on some item
  - First checks the cache directory to determine if the corresponding disk page is in the cache.
  - If no, the item must be located on disk and the appropriate disk pages are copied into the cache.
  - It may be necessary to replace (flush) some of the cache buffers to make space available for the new item.
    - FIFO or LRU can be used as replacement strategies.
- Dirty bit.
  - Associate with each buffer in the cache a dirty bit.
    - The dirty bit can be included in the directory entry.
  - It indicates whether or not the buffer has been modified.
    - Set dirty bit=0 when the page is first read from disk to the buffer cache.
    - Set dirty bit=1 as soon as the corresponding buffer is modified.
  - When the buffer content is replaced –flushed- from the cache, write it back to the corresponding disk page only if dirty bit=1
Caching of disk blocks (cont’d)

- Pin-unpin bit.
  - A page is pinned –i.e. pin-unpin bit value=1-, if it cannot be written back to disk as yet.

- Strategies that can be used when flushing occurs.
  - In-place updating
    - Writes the buffer back to the same original disk location –overwriting the old value on disk-
  - Shadowing
    - Writes the updated buffer at a different disk location.
      - Multiple versions of data items can be maintained.
      - The old value called BFIM –before image-
      - The new value AFIM –after image-
        - The new value and old value are kept on disk, so no need of log for recovery.
Write-Ahead Logging (WAL)

- Two types of log entry –log record- information for a write command.
  - The information needed for UNDO.
    - A UNDO-type log entries including the old values (BFIM).
      - Since this is needed to undo the effect of the operations from the log.
  - The information needed for REDO.
    - A REDO-type log entries including the new values (AFIM).
      - Since it is needed to redo the effect of the operations from the log.
- In UNDO/REDO algorithms, both types of log entries are combined.
- The log includes read commands only when cascading rollback is possible.
Write-Ahead Logging-WAL- (cont’d)
• Write-Ahead Logging (WAL) is the fundamental rule that ensures that a record of every change to the DB is available while attempting to recover from a crash.
  – Suppose that the BFIM of a data item on disk has been overwritten by the AFIM on disk and a crash occurs.
    • Without ensuring that this BFIM is recorded in the appropriate log entry and the log is flushed to disk before the BFIM is overwritten with the AFIM in the DB on disk, the recovery will not be possible.
  – Suppose a transaction made a change and committed with some of its changes not yet written to disk.
    • Without a record of these changes written to disk, there would be no way to ensure that the changes of the committed transaction survive crashes.
• The log is a simple sequential (append-only) file
  – When a log record is written, it is stored in the current log in the DBMS cache and after written to disk as soon as is feasible.
  – With Write-Ahead Logging, the log blocks containing the associated log records for a particular data block update must first be written to disk before the data block itself can be written back to disk.
Write-Ahead Logging-WAL- (cont’d)

- WAL protocol for a recovery algorithm that requires both UNDO and REDO.

1. The before image of an item cannot be overwritten by its after image in the database on disk until all UNDO-type log records for the updating transaction—up to this point in time—have been force-written to disk.
   - Ensures atomicity.

2. The commit operation of a transaction cannot be completed until all the REDO-type and UNDO-type log records for that transaction have been force-written to disk.
   - Ensures durability.

- Commercial DBMSs and WAL
  - IBM DB2, Informix, Microsoft SQL Server, Oracle 8, and Sybase ASE all use a WAL scheme for recovery.

- To facilitate the recovery process, the DBMS recovery subsystem may need to maintain a number of lists.
  - List of active transactions: transactions started but not committed yet
  - List of committed transactions since last checkpoint.
  - List of aborted transactions since last checkpoint.
Steal/no-steal— Force/no-force Approaches

• No-steal approach
  – A cache page updated by a transaction cannot be written to disk before the transaction commits. -deferred update follows this approach-
    • The pin-unpin bit indicates if a page cannot be written back to disk

• Steal approach
  – An updated buffer can be written before the transaction commits.
    • Used when the buffer manager replaces an existing page in the cache, that has been updated by a transaction not yet committed, by another page requested by another transaction.
  – Advantage: avoid the need for a very large buffer space.

• Force/No-Force approaches
  – Force approach if all pages updated by a transaction are immediately written to disk when the transaction commits
  – No-force approach otherwise.
    • Advantage: an updated page of a committed transaction may still be in the buffer when another transaction needs to update it.-save in I/O cost-

• Typical database systems follow a steal/no-force strategy.
Checkpoints

- In case of failure, the recovery manager requires that the entire log be examined to process recovery → time consuming
  - A quick way to limit the amount of log to scan on recovery can be achieved using checkpoints.

- A [checkpoint] record is written into the log periodically at that point when the system writes out to the database on disk all DBMS buffers that have been modified.

- Hence, all transactions with [commit, T] entry in the log before [checkpoint] entry do not need to have their WRITE operations redone in case of crash.
  - Since all their update have been recorded in the DB on disk during checkpointing.

- The recovery manager must decides at what intervals to take a checkpoint.
  - The intervals measured in time, say every $m$ minutes.
  - Or the intervals measured in the number of committed transactions since the last checkpoint, say $t$ transactions.
    - $m$ & $t$ are system parameter
Checkpoints (cont’d)

• Take a checkpoint consists of the following
  1. Suspend execution of transactions temporarily.
  2. Force-write all main memory buffers that have been modified to disk.
  3. Write a [checkpoint] record to the log, and force-write the log to disk.
  4. Resume executing transactions.

• The time needed to force-write all modified memory buffers may delay transaction processing
  • Because of step 1.

• Use fuzzy checkpointing to reduce this delay.
  – The system can resume transaction processing after [checkpoint] record written to the log without having to wait for step 2 to finish.
  – However, until step 2 is completed, the previous [checkpoint] record should remain to be valid.
    • The system maintains a pointer to the valid checkpoint which points to the previous [checkpoint] recording the log
  – Once step 2 is conducted, that pointer is changed to point to the new checkpoint in the log.
Transaction Rollback

- If a transaction fails, roll back this transaction.
  - Any data item values changed by this transaction and written to the DB must be restored to their previous values –BFIM-
  - The UNDO-type log entries are used to restore the old values of these data items.
  - Only the write-item operations need to be undone during transaction rollback.

- If the recovery protocol ensure recoverable schedules without to ensure strict or cascadless schedules, cascading rollback can occur
  - If a transaction T is rolled back, roll back every transaction T’ that read a value of a data item written by T, and so on.
  - Read-item operations are recorded in the log only to determine whether cascading rollback of additional transactions is necessary
  - Cascading rollback is complex and time-consuming.
    - Almost all recovery mechanisms are designed such that cascading rollback is never required –they guarantee strict or cascadless schedules-
Transaction Rollback (cont’d)

LOG

A
30

B
15

C
40

D
20

[\text{start\_transaction}, T3]
[\text{read\_item}, T3, C]
[\text{write\_item}, T3, B, 15, 12]
[\text{start\_transaction}, T2]
[\text{read\_item}, T2, B]
[\text{write\_item}, T2, B, 12, 18]
[\text{start\_transaction}, T1]
[\text{read\_item}, T1, A]
[\text{read\_item}, T1, D]
[\text{write\_item}, T1, D, 20, 25]
[\text{read\_item}, T2, D]
[\text{write\_item}, T2, D, 25, 26]
[\text{read\_item}, T3, A]

$\leftarrow$ System crash
When a crash occurs, transaction T3 has not terminated and must be rolled back.

- The WRITE operations of T3, marked by a single *, are the ones to be undone during transaction rollback.

We check now for cascading rollback.

- T2 reads the value of item B that was written by transaction T3.
  - This can also be determined by examining the log.
- T3 rolled back so T2 rolled back too.
- The write operations of T2 to be rolled back are the ones marked by ** in the log.

Rollback actions

- The DB item D is restored to its old value 25.
- The DB item B is first restored to its old value 12 and finally to 15.
- Insert the following log records into the log, [T2, abort] and [T3, abort].
Recovery Techniques based on Deferred update

A typical deferred update protocol can be stated as follows

1. A transaction cannot change the database on disk until it reaches its commit point.
2. A transaction does not reach its commit point until all its update operations are recorded in the log and the log is force-written to disk –i.e. WAL-

- There is no need to UNDO any operations because the DB is never updated on disk until after the transaction commits.
- REDO is needed in case the system fails after a transaction commits but before all its changes are recorded in the DB on disk.
  - In this case, the transactions operations are redone from the log entries.
- It is known as NO-UNDO/REDO algorithm.

In multiuser systems, the method of recovery from failure is usually closely related to the concurrency control method.

- We discuss recovery in single-user –no concurrency control- then we discuss how concurrency control may affect the recovery process.
Deferred update in a Single-User Environment

In a single user environment, the recovery algorithm is rather simple

Procedure RDU_S - Recovery Using Deferred Update in a Single User Environment -

- Use a list of committed transaction since last checkpoint and a list of active transactions –at most only one transaction because in single user-
- Apply the REDO operation to all the write_item operations of the committed transaction from the log in the order in which they were written to the log.
- For uncommitted –active- transaction T writes the log record [abort,T] into the log.
- Restart the active transactions.
  - The only transaction in the active list will have had no effect on the DB because none of its operations were reflected in the DB on disk. However, this transaction must now be restarted, either automatically by the recovery process or manually by the user.

REDO(write_opr)

- Examine the log entry [write_item,T,X,new_value] and setting the value of item X in the database to new_value, which is the after image (AFIM).

The REDO operation –and the whole recovery process- is required to be idempotent.

- Executing it over and over is equivalent to execute it just once.
- So, the result of recovery from a system crash during recovery should be the same as the result of recovering when there is no crash during recovery.
Deferred update on Multiuser Environment

- In many cases, the concurrency control and recovery process are interrelated.
  - The recovery process may be more complex depending on the protocols used for concurrency control.
  - In general, the greater the degree of concurrency we wish to achieve, the more time consuming the task of recovery becomes.

- Assume
  - Strict 2PL used as concurrency control protocol.
  - [checkpoint] entries are included in the log.

- Procedure RDU_M
  - Use a list of committed transactions since the last checkpoint and a list of active transactions.
  - REDO all the write operations of the committed transactions from the log, in the order in which they were written in the log.
  - The transactions that are active and didn’t commit are effectively canceled and must be resubmitted.
    - Before to restart each uncommitted transaction T, writes the log record [abort,T] into the log.
Deferred update on Multiuser Environment

• Make the NO-UNDO/REDO algorithm more efficient by only REDO the last update of X.
  – Start from the end of the log and only REDO the first occurrence of X in the log.

• Advantages
  1. A transaction does not record any changes in the DB on disk until after it commits –never rollback because of transaction failure during transaction execution-
  2. A transaction will never read the value of an item that is written by an uncommitted transaction, hence no cascading rollback will occur.

• Drawbacks
  – Limits the concurrent execution of transactions because all items remain locked until the transaction reaches its commit point –due to 2PL-
  – Require excessive buffer space to hold all updated items until the transactions commit
Recovery Techniques based on Immediate update

- In these techniques, the DB on disk can be updated immediately without any need to wait for the transaction to reach its commit point.
- However, the update operation must still be recorded in the log (on disk) before it is applied to the database *-using WAL protocol- so that we can recover in case of failure.
- Undo the effect of update operations that have been applied to the DB by a failed transaction.
  - Rollback the transaction and UNDO the effect of its write_operations
- If the recovery technique ensures that all updates of a transaction are recorded in the DB on disk before the transaction commits, there is never a need to REDO any operations of committed transactions – UNDO/NO-REDO recovery algorithm-
- If the transaction is allowed to commit before all its changes are written to the DB, REDO all the operations of committed transactions –UNDO/REDO recovery algorithm-
UNDO/REDO Immediate Update in a Single-User Environment

– Procedure RIU_S – Recovery Immediate Update in Single-User environment -
• Use two lists of transactions maintained by the system: the committed transactions since the last checkpoint and the active transactions – at most one because single-user –
• Undo all write_item operations of the active transaction from the log, using the UNDO procedure.
  – The operations should be undone in the reverse of the order in which they were written into the log
  – After making these changes, the recovery subsystem writes a log record [abort,T] to each uncommitted transaction into the log.
• Redo the write_item operations of the committed transactions from the log, in the order in which they were written in the log, using the REDO procedure.
– UNDO(write_op)
  • Examine the log entry [write_item,T,X,old_value,new_value] and setting the value of item X in the DB to old_value which is the before image (BFIM).
UNDO/REDO Immediate Update with Concurrent Execution

- Assume
  - Log includes checkpoints.
  - Strict 2PL concurrency control protocol is used.

- Procedure RIU_M
  - Use two list maintained by the system: the committed transactions since the last checkpoint and the active transactions.
  - Undo all the write_item operations of the active (uncommitted) transactions using the UNDO procedure.
    - The operations should be undone in the reverse of the order in which they were written into the log
    - After making these changes, the recovery subsystem writes a log record [abort,T] to each uncommitted transaction into the log.
  - Redo all the write_operations of the committed transactions from the log in the order in which they were written into the log.
    - More efficiently done by starting from the end of the log and redoing only the last update of each item X.
Shadow Paging

• A recovery scheme
  – In a single-user environment, doesn’t require the use of log.
  – In multi-user environment, the log may be needed for concurrency control method.

• The DB is made up of $n$ fixed-size disk pages -blocks-

• A directory with $n$ entries where the $i^{th}$ entry points to the $i^{th}$ DB page on disk.
  • All references –reads or writes- to the DB pages on disk go through the directory.
  • The directory is kept in main memory if not too large.

• When a transaction begins executing, the current directory is copied into a shadow directory and the shadow directory is saved on disk
  • The current directory entries point to the most recent or current DB pages on disk

• During transaction execution, all updates are performed using the current directory and the shadow directory is never modified.
Shadow Paging (cont’d)

• When a write_item operation is performed
  – A new copy of the modified DB page is created and the old copy is not overwritten.
    • Two version, of the pages updated by the transaction, are kept.
  – The new page is written elsewhere on some unused disk block.
  – The current directory entry is modified to point to the new disk block.
    • The shadow directory is not modified.

• To recover from a failure
  – Free the modified database pages and discard the current directory.
  – Reinstate the shadow directory to recover the state of the DB before transaction execution.
    • Return to the state prior to the transaction that was executing when the crash occurred.

• To commit a transaction
  – Discard the previous shadow directory.

• NO-UNDO/NO-REDO technique since neither undoing or redoing of data items
Shadow Paging (cont’d)
Shadow Paging (cont’d)

• In a multiuser environment, logs and checkpoints must be incorporated.

• Drawbacks
  – The updated pages change location on disk.
    – Difficult to keep related DB pages close together on disk without complex storage management strategies.
    – Destroy clustering/contiguity of pages; data get fragmented.
  – The overhead of writing shadow directories to disk as transactions start (commit??) is significant.
  – A complicated garbage collection when a transaction commits
    – The old pages referenced by the shadow directory that have been updated must be released and added to a list of free pages for future use.
  – The migration between current and shadow directories must be implemented as an atomic operation
Recovery in Multidatabase Systems

- A multidatabase transaction require access to multiple databases.
  - The DBs may even be stored on different types of DBMS.
    - Some DBMS may be relational, whereas others are object-oriented, etc.
    - Each DBMS involved in the multidatabase transaction may have its own recovery technique and transaction manager separate from those of the other DBMSs.

- Use a two-level recovery mechanism to maintain the atomicity of a multidatabase transaction.
  - A global recovery manager, or coordinator.
  - The local recovery managers.
Recovery in Multidatabase Systems
(cont’d)

• The coordinator usually follows a two-phase commit protocol.
  – Phase 1
    – When all participating databases signal the coordinator that the part of the multidatabase transaction has concluded, the coordinator sends a message «prepare to commit» to each participant to get ready for committing the transaction.
    – Each participating database receiving that message will force-write all log records and needed information for local recovery to disk and then send a «ready to commit» -or OK- signal to the coordinator or «cannot commit» -or not OK- if it fails for some other reasons.
    – If the coordinator does not receive a reply from a database within a certain time out interval, it assumes a «not OK» response.
Recovery in Multidatabase Systems (cont’d)

– Phase 2

• If all the participants DB reply «OK» and also the coordinator, the transaction is successful and the coordinator sends a «commit» signal for the transaction to the participating databases.
  – Each participating database completes transaction commit by writing a [commit] entry for the transaction in the log and permanently updating the database if needed.

• If one or more participating DBs or the coordinator sends «not OK» message, the transaction fails and the coordinator sends a message to «rollback» -or UNDO- the local effect of the transaction to each participating database.
  – The UNDO of the local effect is done by using the log at each participating database
DB Backup and Recovery from Catastrophic Failures

- All the techniques discussed till now apply to noncatastrophic failures.
  - The system log is maintained on the disk and it is not lost as result of the failure.
  - Similarly, the shadow directory is not lost when shadow paging is used.
- The recovery manager must also have to handle more catastrophic failures such as disk crashes.
  - Main technique used is that of DB backup.
- The whole DB and the log are periodically copied into a cheap media as tapes.
- It is customary to backup the system log at more frequent intervals than full database backup.
  - The log is substantially small and hence can be backed up more frequently than DB itself.
  - Thus users do not lose all transactions they have performed since the last DB backup.
  - A new log is started after each DB backup.
- To recover from disk crash
  - Recreate the DB on disk from the latest backup copy on tape.
  - Reconstruct the effect of all committed transactions from the backup copies of log
Overview of Oracle9i recovery

- During a transaction execution and for any changed DB data by this transaction, Oracle does the following:
  - Stores the undo records of this transaction in either rollback segment or undo tablespace assigned to this transaction.
  - Stores also the redo records of this transaction in the redo log buffers of the SGA.
  - Changes the corresponding DB buffers on the SGA.

- **Undo records** contain the old value –i.e. BFIM- and are used to rollback this transaction when a rollback statement is issued or during DB recovery. Undo records are also used to provide read consistency among multiple users.

- **Redo records** contain (i.e. describe) the changes to DB data blocks and the changes to rollback blocks made by a transaction. Redo records are used to reconstruct all changes made to the DB (including rollback segment) when there is a DB recovery.
Overview of Oracle9i recovery

- DBWn may save the changes made to DB buffers (in SGA) by a transaction before this transaction commits but after the LGWR has written the redo entries describing these modified DB buffers and their corresponding undo records to the redo log files -i.e. WAL1-.

- LGWR is responsible on writing the redo log buffers of the SGA sequentially to the redo log files. This is done when the redo log buffer fills or a transaction commits.

- In case the transaction is rolled back, Oracle uses the corresponding undo records of this transaction to restore the old values, i.e. BFIM, of the data changed by this transaction. Locks are also released.

- When a transaction commits, LGWR writes the transaction’s redo records from the redo log buffer of the SGA to the redo log file –i.e. WAL2- and an SCN is assigned to identify the redo records for each committed transaction. This allows to redo transactions in the correct sequence.
  - The user is notified that the transaction is committed only when all the redo log records associated to his transaction have been safely saved on disk.
  - Locks held by the transaction are released.
Overview of Oracle9i recovery

- Because Oracle uses the WAL, DBWn does not need to write the modified (i.e. dirty) DB buffers, from SGA to data files, changed by a transaction when this transaction commits.

- Instead, DBWn performs batch writes only when more data needs to be read into the SGA and too few DB buffers are free. The last recently used data is written to the datafiles first.

- DBWn also performs writes of data dictionary cache. When a checkpoint is activated by the CKPT process, DBWn writes all the modified DB buffers that have been modified since the last checkpoint to the datafiles.
Overview of recovery in Oracle9i

• In general two types of failures can occur:
  – Media (disk) failure: an error occurs that prevent to read/write from a file (datafiles, redo log files, etc.). Media recovery deals with such types of failures based on backups.
  – Instance failure: a problem arises that prevents an instance from continuing working (power outage, etc.). Crash recovery or instance recovery deals with such type of failures with no need of backups.

• When a crash occurs, two steps are always used by Oracle during recovery from instance or media failure: rolling forward and rolling back (I.e. Undo/Redo recovery).

• Rolling forward
  – Reapply to the datafiles all the changes recorded in the redo log.
  – After roll forward, the datafiles contain all committed changes as well as any uncommitted changes that were recorded in the redo log.

• Rolling back
  – After the roll forward, any changes that were not committed must be undone.
  – The undo records are used to identify and undo transactions (I.e restoring BFIM values) that were never committed yet (and?) were recorded in the redo log.