Fundamentals of Database Systems
Database Recovery Systems

Arnab Bhattacharya
Dept. of Computer Science and Engineering,
Indian Institute of Technology, Kanpur

NPTEL
https://onlinecourses.nptel.ac.in/noc15_cs14/

July-September, 2015

Recovery management system

- Recovery management system of a database ensures that atomicity and durability properties are maintained despite failures
- Fail-stop assumption: Data on non-volatile storage is *not* lost due to system crash or power failure
- Recovery algorithms have two main parts
  1. Action taken *during normal execution* to ensure that enough information is collected to recover from failures
  2. Action taken *after a failure* that utilizes information collected so far to recover the database contents and maintain atomicity and durability
- Assume that transactions run serially, i.e., one after another

Log-based recovery

- Log is maintained on *stable storage*
  - Stable storage: data is *never* lost
- Log records for every operation of a transaction are recorded
- When transaction *T* starts, (start, T) record is logged
- *Before* read or write, the corresponding log record is written
- When *T* finishes successfully, (commit, T) is logged
- If *T* fails, (abort, T) is logged
- Two approaches of recovery based on logs
  1. Deferred database modification
  2. Immediate database modification

Deferred database modification

- Defers *all* writes to after partial commit
- Write log record is of the form (write, T, x, new)
  - Old value of *x* is not needed
- After *T* partially commits, (commit, T) is recorded in the log and log is written to the stable storage
  - (abort, T) record is not needed
- Log records are then read to actually perform the writes on the database
- When a transaction crashes, its (commit, T) record is searched
- If not found, nothing to do
  - No write has been performed on the database
- If found, redo all operations
  - Not sure if crash occurred while performing the deferred writes or after them
- Redo operation *must* be idempotent
  - Idempotent: Multiple operations has the same effect as single
- Also called a no-undo/redo recovery scheme
- Redos must be done in the order of serial transactions

Example

- (start, T0); (write, T0, A, 9); (commit, T0); (start, T1); (write, T1, A, 7); (commit, T1)
- Crash after (write, T0) statement
  - Nothing needs to be done
- Crash after (write, T1) statement
  - T0 is redone
- Crash after (commit, T1) statement
  - Both T0 and T1 are redone
  - redo(T0) must be done before redo(T1)
  - Otherwise, value of A will be wrong

Immediate database modification

- Allows writes to modify database even for uncommitted transactions
- Write log record is of the form (write, T, x, old, new)
  - Old value of *x* is needed
- Log record of write must be written to stable storage before the corresponding write is performed on the database
- When a transaction crashes, its (commit, T) record is searched
- If not found, undo all operations
  - Uncommitted writes have been performed on the database
- If found, redo all operations
  - If (commit, T) record is deferred till all writes are performed on the database, redo is not required
- Both undo and redo operations *must* be idempotent
  - Crash may occur multiple times
- Also called a undo/redo (or undo/no-redo) recovery scheme
- Undos must be done in the reverse order of serial transactions
- Redos must be done in the forward order of serial transactions
- All undos must be done before any redo

Example

- (start, T0); (write, T0, A, 3, 9); (commit, T0); (start, T1); (write, T1, B, 2, 5); (write, T1, A, 9, 7); (commit, T1);
- Crash after (write, T0, A) statement
  - T0 is undone
  - Value of A is set back to 3
- Crash after (write, T1, B) statement
  - T1 is undone
  - T0 is redone
  - undo(T1) must be done before redo(T0)
  - Value of A is restored to 9 and that of B to 2
- Crash after (commit, T1) statement
  - Both T0 and T1 are redone
  - redo(T0) must be done before redo(T1)
  - Value of A is set correctly to 7

Checkpoints

- Logs can become very big
- Searching in logs become time-consuming
- Transactions may be re-done unnecessarily
- Checkpointing alleviates these problems
- Log records are flushed to stable storage
- All pending writes are performed on the database
- An entry (checkpoint) is made in the log, and it is written to the stable storage
- During recovery, any transaction that has completed successfully before the checkpoint, i.e., have a (commit, T) entry need not be considered
- Any T_i with (commit, T_i) after checkpoint is redone
- Any T_j with (start, T_j) but no (commit, T_j) is undone

Concurrent transactions

- (checkpoint L) entry contains a list L of transactions active at that point
- After crash, scan backwards to last checkpoint
- Add T_i to redo-list when (commit, T_i) entry is found
- Add T_j to undo-list when (start, T_j) entry is found
- Add T_k in L to undo-list if not present in redo-list
- Undos done first in reverse order
- Redos done later in forward order
- Only operations after checkpoint need to be undone or redone

Example

- (start, T1); (write, T1, B, 2, 3); (start, T2); (commit, T1); (write, T2, C, 5, 7); (checkpoint, {T2}); (start, T3); (write, T3, A, 1, 9); (commit, T3); (start, T4); (write, T4, C, 7, 2)
- Undo-list: T4, T2
- Redo-list: T3
- Order: T4, T2, T3
- Order of operations:
  - T4: Revert value of C to 7
  - T2: No operation
  - T3: Re-write value of A to 9

Log record buffering

- Log records are buffered in memory before a block is output to the stable storage
- Records are flushed in the order of appearance in the log
- Force-writing is used to flush log records to stable storage before a transaction enters the commit point
- The (commit, T) entry is also flushed
- Before a block of data is written to the database, all log records pertaining to it are flushed to stable storage
- This rule is called write-ahead logging or WAL rule

Shadow paging

- Maintain two page tables: current page table and shadow page table
- Shadow page table is maintained on disk without any change
- When a page requires change, it is copied to a new location
- Current page table points to the new location
- Updates are performed on the copy
- When a transaction commits, flush all changed pages to disk
- Modify shadow page table with contents of current page table
- No recovery needed
- Also called a no-undo/no-redo recovery scheme
- Disadvantages
  - Entire page table is copied even though only some pages are modified
  - Commit overhead may be high
  - Works for serial transactions only
- Advantages
  - Recovery is built-in
  - No overhead of writing log records