Database Management Systems

Recovery

Recovery

- Two goals:
    - allow transactions to be either committed (with a guarantee that the effects are permanent) or aborted (with a guarantee that the effects disappear)
    - allow the database to be recovered to a consistent state in case of some kind of failure
- input a schedule of operations produced by scheduler (including commit and abort requests)
- output: a schedule of reads, writes and forced writes

Failures

- System Failure
    - database server halted abruptly
    - processing of transaction(s) halted abruptly
    - connections to clients broken
        - contents of memory buffers lost, however, database files are not damaged.
- Media failure
    - one or more database files become damaged or inaccessible
    - a media failure may cause a system failure, or possibly an orderly system system shutdown.
    - solution: Duplication! (by archiving (backup) or redundant (maybe distributed) copies)

Atomicity and Durability

- Overall, a failure can not cause a transaction to be partially executed.

- After a failure occurs, active transactions should be aborted automatically

- Committed transactions should be durable, i.e., changes they made to the database should not be lost as a result of the failure.

Unrealistically simple way

- keep all changes in memory and write them all back to disk when a transaction commits
    - guaranteed atomicity
    - Problems:
        - too many writes when a transaction commits, what if system failure happened in the middle of those writes?
            - what if we run out of memory buffers?
- keep no changes in memory and write each change to disk right away
    - guaranteed durability
    - Problems:
        - very poor response time
        - what if the transaction aborts
        - what if the system crashed in the middle of executing a transaction

Log based approaches

- a log is a read/append only data structure (file).
- when transactions are running, log records are appended to the log.
- Log records contain several types of information:
  - UNDO information: old values of objects that have been modified by a transaction.
  - REDO information: new values of objects that have been modified by a transaction.
  - BEGIN/COMMIT/ABORT: records are recorded whenever a transaction begins, commits or aborts.

Write-Ahead Logging (WAL)

- WAL protocol makes sure that LOG is consistent with the main database

- WAL protocol requires:
    - UNDO rule: a log record for an update is written to disk before the corresponding data page is written to disk.

    - REDO rule: all log records for a transaction are written to disk before the transaction is considered to be committed.

- UNDO guarantees Atomicity

- REDO guarantees Durability

Using WAL in operations

- Read(Ti, x);
    - read value v of x (if x is not in cache, load it in) and return it to the upper manager
- Write(Ti, x);
    - add Ti to the set of active transactions
    - read the old value v of x (load x to cache if it is not in)
    - LOG [Ti, x, v, v’]
    - write v’ to x (x is still in the cache) and then ACK write
- Commit(Ti);
    - LOG [Ti, commit]
    - flush all log records and then Acknowledge commit
    - LOG [Ti, end]
- Abort(Ti);
    - scan LOG backwards for data items updated by Ti, restore the old value of x (only restores in cache)
    - LOG [Ti, abort]
    - Acknowledge abort
    - LOG [Ti, end]

Recovery Algorithm (ARIES)

- ARIES (Algorithms for Recovery and Isolation Exploiting Semantics) Algorithm Assumptions
    - WAL used
    - cascadeless schedules
    - only single copy of any data item in the cache
- When system accidentally crashes and we need to restart the system:
    - Let R = emptySet
    - scan LOG to find all active and committed transactions
    - scan LOG backwards for records [Ti, x, v, v'] and for each such record such that x not in R:
        - read x; if Ti was committed then write v' into x and put x to R; if Ti was active then write v into x;
    - when done, write abort and end records for all active transactions
    - restart completed
- Restart operation is idempotent (if system crashes during restart, we don't mind)

Checkpoint

- Logs can grow very big over time — place a checkpoint to mark a new start
- A simple checkpoint algorithm
  - prevent new transactions from starting, wait for active ones to finish
  - copy modified memory blocks to database files
  - append a checkpoint record to the log
  - allow new transactions to begin