Discussion 9

Recovery

Binbin Chen
chenbb@shanghaitech.edu.cn

Motivation of Recovery

- Atomicity: all actions in a Xact should be either happen or none happen.
- Durability: If a Xact, commits, its effect persist permanently.

- Xact abort & DB crash -> recovery

No-Steal/Force Scheme for A&D

- NO STEAL: Xact locks the page and pins it in the buffer pool
  - Can’t be stolen by replacement policy, no dirty write go to DB
  - enables Atomicity
  - but not scalable and IO inefficient
- FORCE: every update is “forced” onto the DB disk before commit.
  - Provides Durability but somehow violates Atomicity for many IOs will have crash when forcing

Preferred Scheme: Steal/No-Force

- STEAL (enforcing Atomicity)
  - we allow buffer-pool frames with uncommited updates to be replaced or flushed to disk.
  - but has problems like Xact aborts and DB crash before end when dirty pages have flushed into DB already
  - so we need UNDO updates that should not happen
- NO FORCE (enforcing Durability)
  - we allow commit without flushing pages to the disk
  - but has problem that System crash before dirty buffer page of a committed transaction is flushed to DB disk.
  - so we need REDO

Buffer Management Summary

	No Steal	Steal
No Force		Fastest
Force	Slowest	

Performance Implications

	No Steal	Steal
No Force	No UNDO REDO	UNDO REDO
Force	No UNDO No REDO	UNDO No REDO

Logging/Recovery Implications

Simplified policy:
REDO: deal with commit before flush
UNDO: deal with flush before commit

Write-Ahead Logging (WAL)

- Log: An ordered list of log records to allow REDO/UNDO, with a write buffer (“tail”) in RAM.
    - Each log record has a unique Log Sequence Number (LSN)
    - we can only write records to tail, tail buffer will periodically flush to log’s end on disk
- The Write-Ahead Logging Protocol:
    - Must force the log record write to log device before the corresponding data page gets to the DB device every time.
    - Must force all log records write to log device for a Xact before commit.

LSN

- flushedLSN (= largest diskLSN) tracked in RAM
- Each data page in the DB contains a pageLSN, i.e. the LSN of the most recent log record for an update to that page.
- Before page i is flushed to DB, log must satisfy: pageLSNi ≤ flushedLSN
  - because pageLSN is now larger than flushLSN, we cannot write page to DB, but with diskLSN increasing, when pageLSN appears in log device, we can flush page to disk
  - this allows steal
- prevLSN is the LSN of the previous log record written by this XID

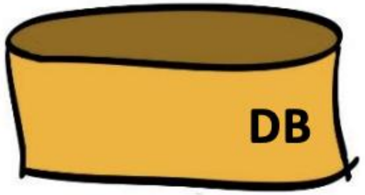
State in Memory

- Transaction Table
  - One entry per currently active Xact
  - Contains: XID, Status (running, committing, aborting), lastLSN (most recent LSN written by Xact)
- Dirty Page Table
  - One entry per dirty page currently in buffer pool.
  - •Contains recLSN (record which first caused the page to be dirty)

ARIES Big Picture: What's Stored Where

LogRecords

LSN
prevLSN
XID
type
pageID
length
offset
before-image
after-image



Data pages
each with a
pageLSN

Master record

Xact Table

xid
lastLSN
status

Dirty Page Table

pid
recLSN

Log tail
flushedLSN

Buffer pool

Execution of Xact

- commit: All log records up to Xact's commit record are flushed to disk
- abort: need CLR (compensation log record) with undonextLSN for each undone operation.
    - CLR contains REDO info, CLRs never Undone so exactly undo once
- checkpoint: Store LSN of most recent chkpt record in a safe place (master record)

Recovery Protocol: 3-phase

- Analysis - Scan log forward from checkpoint.
  - end, commit, update records
  - if commit then remove the Xact from Xact table, the remaining is all active and need abort for Atomicity
- REDO all actions including abort and CLRs. (repeat all history)
  - unless updates are flushed into DB already, like:
    - Affected page is not in the Dirty Page Table, or
    - Affected page is in D.P.T., but has recLSN > LSN, or
    - pageLSN (in DB) >= LSN. (this last case requires I/O)
- UNDO effects of failed Xacts.
  - to deal with Atomicity, we will backward from end to first LSN of oldest Xact alive (running, aborting)after Redo.
  - can optimize via CLR loops