Lec 20 Database Logging

Want atomicity and durability — all/none done and saved on disk
Recovery: want data to be safe even when bad things happen
crash

Faiture types

Volatile storage (buf pool, DRAM, ...) + power goes off

Ideally, stable storage — non-volatile, survives all possible failures
└ Doesn't exist, but we try to replicate this

1. Txn failure
   - Logical failure — constraint violation, etc.
   - Internal state " — deadlock, etc.
2. Sys Failure
   - Software failure — bugs, etc.
   - Hardware failure — power, corrupt storage, etc.
3. Storage failure
   - Disk failure
   - Controller problem

Want: upon successful commit, the changes are safe

Operations: - Undo — useful if txn aborts
- Redo — useful if disk write fails

Buf pool policies

Steal policy: whether uncommitted txn can overwrite ("steal")
most recently committed value in volatile storage

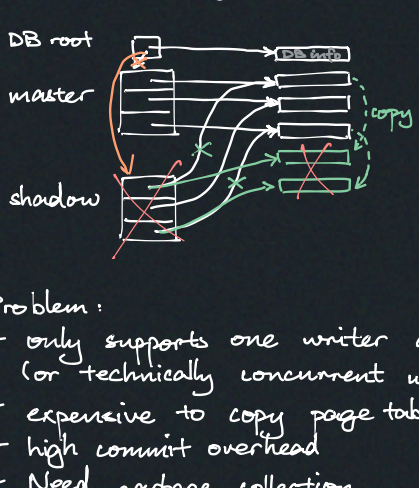
If need to write, may need to copy the page

Force policy: whether changes must write to non-volatile
before txn can commit

▸ No steal + force
+ Simple
- Slow
- Can't write for stuff larger than buf pool
- Changing one byte needs copy of whole page
- Multiple txn can each copy a page

Shadow paging

Like file table after fork
Keep master page table & per-txn shadow page table



when write
when commit — switch to the shadow table
when abort — throw away shadow copy

Problem:
- only supports one writer at a time
(or technically concurrent writers not touching same pages)
- expensive to copy page table
- high commit overhead
- Need garbage collection
- Fragments data

SQLite: offers rollback mode (special case shadow paging)
└ Journals the orig page before overwriting master
then can restore the journalled page if needed

Write-ahead log (WAL)

Log the changes each txn make
└ in disk, has pages, but buffered (" log buffer ")
Want: no force + steal

- Force the log only
- When stealing, can use log to undo
- Write log to stable before writing any pages
- Buf pool can evict as usual as long as logs have been flushed

The log: - <begin> and <commit>
- Txn id
- Before and after value

Ex. (sequential action list for each txn)

<T1, begin>
<T1, A, 1, 8> A=1 → A=8
<T1, B, 5, 9> B=5 → B=9
<T1, commit>] at this point, flush log to disk

Optimisation:
- Group commits and batch flush
  → Wait for a bit before flush to wait for other things to flush
  → Background process flushing every few ms

We get: fast runtime, harder recovery

Logging schemes ← Still need looking

- Physical — byte changes like git diff ← low level more data
- Logical — " update all records satisfying ... " ← high level more complicated
- Physiological — physical across pages, logical within page ← middleground

Checkpoints

Use log to redo and undo as needed

Conceptually: stop all queries
flush WAL } to disk
flush dirty pages }
write <checkpoint> to log
resume