CS186 Discussion Section Week 11

Logging and Recovery

ACID

- Atomicity
- Consistency
- Isolation
- Durability

Last week - Isolation

- Isolation: transactions aren’t affected by the operations of other transactions.

- We solved this: Strict 2PL gives us serializability

This week - Atomicity & Durability

- Atomicity: transactions are all or nothing
- Durability: after commit, data never disappears

How can we ensure these if the DB can crash at any time?

Buffer Policies

- NO STEAL: don’t let the system “steal” frames with uncommitted updates from the buffer pool and write them to disk
  - Which do we lose if we steal pages with uncommitted data?

    A. Atomicity

    B. Durability

Buffer Policies

- NO STEAL: don't let the system "steal" frames with uncommitted updates from the buffer pool and write them to disk
  - Which do we lose if we steal pages with uncommitted data?

    A. Atomicity

    B. Durability
  - if there's a crash with STEAL, transaction is incomplete, but some updates are on disk
  - To preserve atomicity, we must UNDO those changes.

Buffer Policies

- FORCE: “force” the buffer manager to write every updated page to disk before committing
  - Which do we lose if we don’t force flushing before committing?

    A. Atomicity

    B. Durability

Buffer Policies

- FORCE: “force” the buffer manager to write every updated page to disk before committing
  - Which do we lose if we don’t force flushing before committing?

    A. Atomicity

    B. Durability: commits may not survive a crash

Buffer Policies

- FORCE: “force” the buffer manager to write every updated page to disk before committing
    - Which do we lose if we don’t force flushing before committing?

      A. Atomicity: may have written only some of the pages

      B. Durability: commits may not survive a crash
    - if there’s a crash with NO FORCE, we have no guarantee that all committed pages are on disk.
    - To preserve atomicity/durability, we must REDO those changes.

Buffer Policies

- NO STEAL / FORCE (SimpleDB):
  - Don’t steal pages with uncommitted data.
  - Always force committed pages to disk.
  - Gives us atomicity and durability! Why not use?
    - Really slow at commit time, especially with both…
- STEAL / NO FORCE (Real World):
  - Can write uncommitted data to disk.
  - Don’t have to write data to disk on commit.
  - Essentially, no guarantees about A or D. Why use?
    - Much faster. Can use **write-ahead logging** to get A & D!

Write-Ahead Logging (WAL)

- Don’t guarantee data gets to disk, but guarantee that our **logs** about that data do.
  - Protection from STEAL: Force log record for update out before corresponding data page gets to disk.
  - Protection from NO FORCE: Force all logs for transaction out before commit finishes.
- Log everything!
  - Transaction start
  - Transaction updates
  - Transaction commit
  - Transaction abort

The Log

- <LSN, pageID, offset, old data, new data, prevLSN>
    - LSN (“Log Sequence Number”): globally increasing ID for log records
    - prevLSN: LSN of the last operation for **this txn**

More Transaction State

- Transaction Table
  - Answers question: which transactions are currently running?
  - Contains:
    - XID: Transaction ID
    - Status: Running/Committing/Aborting
    - lastLSN: **most recent** LSN created by the txn
- Dirty Page Table
  - Answers question: which buffer pages are dirty?
  - Contains:
    - pageID
    - recLSN: LSN of **first** update that dirtied this page
- Checkpoints: Occasionally save these tables in the log (helpful if we crash)

Normal Execution

- Transactions happening, everything bright and cheery.
- Commit occurs: what do we do?
  - Flush the logs to disk!
- Abort occurs: what do we do?
  - We need to undo all the txn’s changes.

Example Log (only 1 txn running)

LSN	Log	prevLSN
10	T1 Start	null

Example Log (only 1 txn running)

LSN	Log	prevLSN
10	T1 Start	null
20	T1 writes P5	10

Example Log (only 1 txn running)

LSN	Log	prevLSN
10	T1 Start	null
20	T1 writes P5	10
30	T1 writes P6	20

Example Log (only 1 txn running)

LSN	Log	prevLSN
10	T1 Start	null
20	T1 writes P5	10
30	T1 writes P6	20
40	T1 Abort	30

Example Log (only 1 txn running)

LSN	Log	prevLSN
10	T1 Start	null
20	T1 writes P5	10
30	T1 writes P6	20
40	T1 Abort	30
50	CLR: Undo 30, undoNextLSN=20	40

Example Log (only 1 txn running)

LSN	Log	prevLSN
10	T1 Start	null
20	T1 writes P5	10
30	T1 writ[illegible]	20
40	T1[illegible]	30
50	CLR: Undo 30, undoNextLSN=20	40

Compensation
Log Record

Example Log (only 1 txn running)

LSN	Log	prevLSN
10	T1 Start	null
20	T1 writes P5	10
30	T1 writes P6	20
40	T1 Abort	30
50	CLR: Undo 30, undoNextLSN=20	40
60	CLR: Undo 20, undoNextLSN=null	50

Example Log (only 1 txn running)

LSN	Log	prevLSN
10	T1 Start	null
20	T1 writes P5	10
30	T1 writes P6	20
40	T1 Abort	30
50	CLR: Undo 30, undoNextLSN=20	40
60	CLR: Undo 20, undoNextLSN=null	50
70	T1 End	60

Next Week: Crash Recovery

- System crashes. What do we do?
- General plan:
  - Make sure our in-memory txn state is also up to date (analysis)
  - Re-apply changes made by committed txns to make sure they got to disk (needed with NO FORCE)
  - Undo uncommitted changes on disk (needed with STEAL, or for in-flight txns).
- This is called “ARIES”