Recitation 18: Databases
Plan
* DBs & Transactions
* Durability
* Concurrent access

Logistics
* DP pres grades out next week
* DB hands on out 4/14
* No lecture 4/18 for Patriots Day
Database

- Collection of tables (rows, cols)
- High-level language (SQL) for reading/writing data in tables

If you haven't yet used it, you will be surprised & delighted?

- Client can group together a sequence of actions into a transaction

```
BEGIN TRANSACTION
A = 50
B = A
B = B + 1

In reality, use SQL

COMMIT / ABORT
```

- All that the DB system cares about is the reads & writes.
Two things to worry about

1. Crashes / Durability
   - Want committed data to persist on disk ("non-volatile" storage)

2. Concurrent access to data
   - Want each transaction to appear to execute in sequence (*sort of*)
Durability

Simplest implementation

Problem: Performance?

RAM read: 100 ns
Disk seek: 10,000,000 ns

Aggressive impl

Crash!
Write-ahead log

(Very common / useful idea)

- Write changes as log entry before commit
- Why better?
  - (a) Writes are large & sequential
  - (b) Repeated access to same obj doesn't touch disk

After crash, inspect log
  - "Undo" uncommitted partial txns
  - "Redo" committed txns

To undo, need to store old & new value of each record.

Similar ideas show up in other contexts...
Recovery
- **REDO** actions from log
  - Disk now in state as before crash.
- **UNDO** aborted trans
  - Find first such one, roll back

In groups, walk through this recovery process:

What happens if you crash during recovery?

Log can grow LARGE
- Checkpoints to speed up recovery.
Concurrent access

- Two different transactions should ideally appear to execute serially (conflict serializability)

\[ \text{Abal} = \text{Read}(A) \]
\[ \text{Abal} = 50 \]
\[ \text{Write}(A, \text{Abal}) \]

\[ \text{Abal} = \text{Read}(A) \]
\[ \text{Bbal} = \text{Read}(B) \]
\[ \text{Bbal} = 50 \]
\[ \text{Write}(B, \text{Bbal}) \]

- Most common technique: Locking
  - Covered in lecture.

- Two-phase locking
  - Each data item has lock

I. Growing: Acquire all locks needed → In fixed order? o.w. deadlock
   - Do stuff

II. Shrinking: Release all locks
Locking

Why are we not done? Performance!

If your transactions sum value of all rows in a table, you will prevent any other action on table for a while.

→ Must hold locks on all table rows
→ Action of locking & unlocking can be costly.

How to address this?
→ Discuss in groups

Two ideas:

1. Coarse-grained locks (hierarchical)
   → Reduce # of locks you need to acquire

2. Lock less
   → Release locks early to increase amount of concurrency
Hierarchical Locking

Can lock/unlock entire region at once.
- Typically a "page" of values

"Intent locks" (see paper) allow extra concurrency.
Relaxed Isolation

Postgres DB uses "read committed" by default

Let $Txn$ sees values of writes committed by other $ttxns$

```
BEGIN
  $Aval = $Read(A)
  Write(B, 0)
  $Aval = $Read(A)
COMMIT
```

→ Useful b/c allows releasing read locks early
Wrap up

Two challenges:

- Recovering from failures
- Doing many things at once

Both are easy if you don’t care about performance.

If you do, ... trade-offs!