CSE 444: Database Internals

Lectures 15
Transactions: Snapshot Isolation

Where We Are

- ACID properties of transactions
- Concept of serializability
- How to provide serializability with locking
- Lowers level of isolation with locking
- How to provide serializability with optimistic cc
  - Timestamps/Multiversion or Validation
- Today: lower level of isolation with multiversion cc
  - Snapshot isolation

Snapshot Isolation

- Not described in the book, but good overview in Wikipedia

Snapshot Isolation

- A type of multiversion concurrency control algorithm
- Provides yet another level of isolation
- Very efficient, and very popular
  - Oracle, PostgreSQL, SQL Server 2005
- Prevents many classical anomalies BUT…
- Not serializable (!), yet ORACLE and PostgreSQL use it even for SERIALIZABLE transactions!
  - But "serializable snapshot isolation" now in PostgreSQL

Snapshot Isolation Rules

- Each transactions receives a timestamp TS(T)
- Transaction T sees snapshot at time TS(T) of the database
- When T commits, updated pages are written to disk
- Write/write conflicts resolved by "first committer wins" rule
  - Loser gets aborted
- Read/write conflicts are ignored

Snapshot Isolation (Details)

- Multiversion concurrency control:
  - Versions of X: X_{t1}, X_{t2}, X_{t3}, . . .
- When T reads X, return $X_{TS(T)}$.
- When T writes X: if other transaction updated X, abort
  - Not faithful to "first committer" rule, because the other transaction U might have committed after T. But once we abort T, U becomes the first committer ☺

What Works and What Not

- No dirty reads (Why ?)
- No inconsistent reads (Why ?)
  - A: Each transaction reads a consistent snapshot
- No lost updates ("first committer wins")
- Moreover: no reads are ever delayed
- However: read-write conflicts not caught !

Write Skew

```
T1:
  READ(X);
  if X >= 50
      then Y = -50; WRITE(Y)
  COMMIT
```

```
T2:
  READ(Y);
  if Y >= 50
      then X = -50; WRITE(X)
  COMMIT
```

In our notation:

$R_1(X), R_2(Y), W_1(Y), W_2(X), C_1, C_2$

Starting with X=50,Y=50, we end with X=-50, Y=-50.
Non-serializable !!!

Write Skews Can Be Serious

- Acidicland had two viceroys, Delta and Rho
- Budget had two registers: taXes, and spendYng
- They had high taxes and low spending…

```
Delta:
  READ(taXes);
  if taXes = 'High'
      then { spendYng = 'Raise';
             WRITE(spendYng) }
  COMMIT
```

```
Rho:
  READ(spendYng);
  if spendYng = 'Low'
      then {taXes = 'Cut';
            WRITE(taXes) }
  COMMIT
```

… and they ran a deficit ever since.

Questions/Discussions

- How does snapshot isolation (SI) compare to repeatable reads and serializable?
  - A: SI avoids most but not all phantoms (e.g., write skew)
- Note: Oracle & PostgreSQL implement it even for isolation level SERIALIZABLE
  - But most recently: "serializable snapshot isolation"
- How can we enforce serializability at the app level ?
  - A: Use dummy writes for all reads to create write-write conflicts… but that is confusing for developers!!!

Commercial Systems

Always check documentation as DBMSs keep evolving and thus changing! Just to get an idea:

- DB2: Strict 2PL
- SQL Server:
  - Strict 2PL for standard 4 levels of isolation
  - Multiversion concurrency control for snapshot isolation
- PostgreSQL: Multiversion concurrency control
- Oracle: Multiversion concurrency control

Important Lesson

- ACID transactions/serializability make it easy to develop applications
- BUT they add overhead and slow things down
- Lower levels of isolation reduce overhead
- BUT they are hard to reason about for developers!