Transactions

Intel (TX memory):
Transactional
Synchronization
Extensions (TSX)

MySQL

PostgreSQL
Goal – A Distributed Transaction

- We want a transaction that involves multiple nodes
- Review of transactions and their properties
- Things we need to implement transactions
  * Locks
  * Achieving atomicity through logging
    - Roll ahead, roll back, write ahead logging
- Finally, 2 Phase Commit (aka 2PC) and 3PC
- Lead into Paxos (again!)
Transactions - Definition

A transaction is a sequence of data operations with the following properties:

* **Atomic**
  - All or nothing

* **Consistent**
  - Consistent state in => consistent state out

* **Independent**
  - Partial results are not visible to concurrent transactions

* **Durable**
  - Once completed, new state survives crashes
Transactional API

- Interface
  * `tran = TranMonitor.begin()`
  * `tran.commit()`
  * `tran.abort()`

```sql
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```
Importance of independence

- Possible problems if we don’t have it
  - lost update
    - t1 and t2 read x and then write x, t1’s update is lost
  - inconsistent retrieval
    - Intermediate state may be inconsistent
  - dirty read
    - t1 updates x, t2 reads x, t1 aborts; t2 has dirty value of x
  - premature write
    - t1 and t2 update x, t1 aborts; t2’s update is lost
Serializability

● A set of transactions is serializable iff
  * resulting state is equivalent to that produced by some serial ordering of those transactions

● They don’t actually have to run in serial order
  * system just ensures that actual outcome is the same as if they had
Two Possible Approaches

- Two Phase Locking
- Strict Two Phase Locking
Two Phase Locking

- **Locks**
  * reader/writer locks
  * acquired as transaction proceeds
  * no more acquires after first release

- **Phase 1**
  - acquire locks and access data, but release no locks

- **Phase 2**
  - access data, release locks, but acquire no new locks
Semantics of two-phase locking

- Does the Two-Phase Locking protocol ensure
  * serializability?
  * independence?

- How?
Semantics of two-phase locking

- Ensures serializability
  * if transactions have no conflicting lock access
    - order arbitrarily
  * for any transactions with conflicting lock access
    - order transactions based on order lock is acquired
  * transactions are serialized
    - because, no lock is acquired after first release
    - deadlocks are still possible

- Does not ensure independence
  * we still have premature write problem
  * t1 releases x, t2 acquires x, then t1 aborts
Strict two phase locking

- Like two-phase locking, but
  * release no locks until transaction commits
- Phase 1:
  · acquire locks and access data, but release no locks
- Phase 2:
  · Commit/abort transaction and then release all locks
- Ensures both serializability and independence