Implementing Transaction

Friday, May 31, 2019 2:15 PM

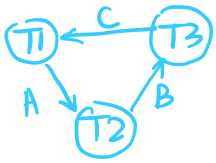
- Scheduler a.k.a. Concurrency Control Manager
  - approaches
    - Locking scheduler
      - Pessimistic concurrency control
      - e.g., SQLite, SQL Server, DB2
    - Multiversion Concurrency Control (MVCC)
      - Optimistic concurrency control ← more careful
      - Postgres, Oracle: Snapshot Isolation (SI)

- Locking Scheduler (we will only discuss this in class)
  - Each element has a unique lock
  - Each transaction must acquire the lock to read/write that element
  - If lock is taken by other transaction, wait.
  - Txn must release the lock.
  - By using lock we can ensure a conflict-serializable schedule.
- Action on Locks
  - $l_i(X)$ Lock element X by transaction T_i.
  - $u_i(X)$ Unlock element X by T_i.

- Two phase locking (2PL)
  - In every transaction, every lock requests must precede all unlock requests.
    - ➡ then it will be conflict-serializable.
  - Theorem 2PL ensures conflict serializability.

Proof: Suppose it is not serializability.

then ∃ a cycle in the precedence graph.



then there's temporal cycle:

$U_1(A) \to L_2(A) \Rightarrow U_1(A)$ happened strictly before $L_2(A)$.

$L_2(A) \to U_2(B) \Rightarrow$ because 2PL.

$U_2(B) \to L_3(B)$

$L_3(B) \to U_3(C)$

$U_3(C) \to L_1(C)$

$L_1(C) \to U_1(A)$

these form a contradiction.

∴ It must be conflict serializable.

- Non-recoverable Schedule when commit comes before rollback.
  recoverable
  - Commit: Commit the change; When a transaction is committed, it only commit its own effects.
  - Rollback: roll the state back to this transaction start

- Strict 2PL

- Strict 2PL
  - All locks are held until commit/abort
  - All unlocks are done together with commit/abort.

  → Strict 2PL make schedule conflict serializable and recoverable.

- Deadlock — another problem
  - e.g., T_1: R(A), W(B)
    T_2: R(B), W(A)
    } holds each others lock.
  - Relatively expensive: check periodically. if deadlock is found, then abort one TXN; re-check for deadlock more often.
  - To find deadlock: search for a cycle.

    T_1 waits for a lock held by T_2,

    T_2 waits for a lock held by T_3,

    ...

    T_n waits for a lock held by T_1.

- Lock Modes
  - S = shared lock (for read)

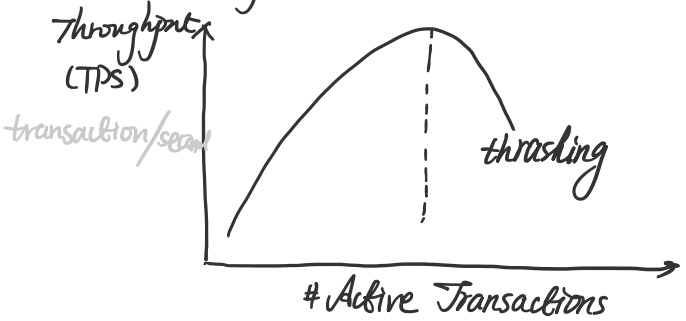
    X = exclusive lock (for write)
  - Lock compatibility matrix

txn1 \ txn2	None	S	X
None	✓	✓	✓
S	✓	✓	X
X	✓	X	X

← because can't read/write simultaneously

- Lock Granularity
  - Fine granularity locking (e.g. tuples)
    - high concurrency
    - high overhead in managing locks.
    - e.g., SQL Server
  - Coarse grain locking
    - Many false conflicts
    - Less overhead in managing locks
    - e.g. SQLite

- Lock Performance



- Phantom Problem appears if tuples are inserted/deleted
  - Phantom: A tuple that is invisible during part of a transaction execution but not invisible during the entire execution.
  - Conflict-serializability assumes DB is static.
  - When DB is dynamic then c-s is not serializable.
  - Static database
    - Conflict serializability implies serializability.
  - Dynamic database
    - Conflict serializability + phantom management = serializability.
  - (Strict) 2PL guarantees conflict serializability
- Dealing with phantoms
  - Lock the entire table
  - Lock the index entry - if index is available.
  - Predicate locks (Complex)
    - A lock on an arbitrary predicate.

➡ Dealing with phantoms is expensive.