Lecture 16: Concurrency Control Theory

15-445/645 Database Systems (Fall 2017)
Carnegie Mellon University
Prof. Andy Pavlo

Motivation

1. **Lost Update Problem (Concurrency Control):** How can we avoid race conditions when updating records at the same time?

2. **Durability Problem (Recovery):** How can we ensure the correct state in case of a power failure?

Transactions

1. A **transaction** is the execution of a sequence of one or more operations (e.g. SQL queries) on a shared database to perform some higher level function. They are the basic unit of change in a DBMS. Partial transactions are not allowed.

2. Example: Move $100 from Andy's bank account to his bookie's account

   (a) Check wether Andy has $100.

   (b) Deduct $100 from his account.

   (c) Add $100 to his bookie's account.

3. Strawman System (SQLite):

   (a) Execute one transaction one-by-one as they arrive to the DBMS. Only one transaction can be running at a time.

   (b) Before a transaction starts, copy the entire database to a new file and make all changes to that file. If transaction succeeds, overwrite original file. If transaction fails, toss dirty copy.

   (c) Drawback: This approach doesn't support concurrent transactions

4. Executing concurrent transactions in a DBMS is challenging. It is difficult to ensure correctness while also executing transactions quickly. We need formal correctness criteria:

   (a) Temporary inconsistency is ok.

   (b) Permanent inconsistency is bad.

5. **The scope of a transaction is only inside the database. It can't make changes to the outside world because it can't roll those back.**

Definitions

1. A database is a set of named data objects ($A, B, C, ..$).
2. A transaction is a sequence of read and write operations ($R(A), W(B)$). The outcome of a transaction is either `COMMIT` or `ABORT`.
   (a) If `COMMIT`, all of the transactions changes are saved to the database.
   (b) If `ABORT`, all changes are undone so that it is like the transaction never happened.
   (c) Aborts can be either self-inflicted or caused by the DBMS
3. Correctness Criteria: **ACID**
   (a) **A**tomicity: All actions in the transaction happen, or none happen.
   "All or Nothing"
   (b) **C**onsistency: If each transaction is consistent and the DB starts consitent, then it ends up consistent.
   "It looks correct to me..."
   (c) **I**solation: Execution of one transaction is isolated from that of other transactions.
   "As if alone"
   (d) **D**urability: If a transaction commits, its effects persist.
   "The transaction's changes can survive failures..."

ACID: Atomicity

The DBMS guarantees that transactions are **atomic**. The transaction either executes all its actions or none of them.

1. There are two possible outcomes of executing a transaction:
   (a) COMMIT after completing all its actions
   (b) ABORT after executing some actions
2. **Approach #1: Logging**
   (a) DBMS logs all actions so that it can undo the actions of aborted transactions.
   (b) Think of this like the black box in airplanes.
   (c) Logging is used by all modern systems for audit and efficiency reasons.
3. **Approach #2: Shadow Paging**
   (a) DBMS makes copies of pages and transactions make changes to those copies. Ony when the transaction commits is the page made visible to others.
   (b) Originally from System R but abandoned in the 1980s. Few systems do this today (CouchDB, LMDB).

ACID: Consistency

The "world" represented by the database is **consistent** (e.g., correct). All questions (i.e., queries) that the application asks about the data will return correct results.

1. **Database Consistency:**
   (a) The database accurately represents the real world entity it's modeling and follows integrity constraints.
   (b) Transactions in the future see the effects of transactions committed in the past inside of the database.
2. **Transaction Consistency:**
   (a) If the database is consistent before the transaction starts, it will also be consistent after.
   (b) Ensuring transaction consistency is the application's responsibility.

ACID: Isolation

The DBMS wants to transactions the illusion that they are running alone in the system. They do not see the effects of concurrent transactions. This is equivalent to a system where transactions are executed in serial order (i.e., one at a time). But in order to get better performance, the DBMS has to interleave the operations of concurrent transactions.

Concurrency Control

1. A **concurrency control protocol** (traffic cop) is how the DBMS decides the proper interleaving of operations from multiple transactions.
2. Two categories of concurrency control protocols:
   (a) **Pessimistic:** The DBMS assumes that transactions will conflict, so it doesn't let problems arise in the first place.
   (b) **Optimistic:** The DBMS assumes that conflicts between transactions are rare, so it chooses to deal with conflicts when they happen.
3. An execution schedule is correct if it is equivalent to some serial execution
   (a) **Serial Schedule:** A schedule that does not interleave the actions of different transactions.
   (b) **Equivalent Schedules:** For any database state, the effect of execution the first schedule is identical to the effect of executing the second schedule.
   (c) **Serializable Schedule**: A schedule that is equivalent to some serial execution of the transactions.
4. Interleaved execution conflicts:
   (a) **Read-Write Conflicts ("Unrepeatable Reads"):** A transaction is not able to get the same value when reading the same object multiple times.
   (b) **Write-Read Conflicts ("Dirty Reads"):** A transaction sees the write effects of a different transaction before that transaction committed its changes.

(c) **Write-Write conflict ("Lost Updates"):** One transaction overwrites the uncommitted data of another concurrent transaction.

5. Two types for serializability: Conflict vs. View

   (a) Neither definition allows all schedules that you would consider serializable

   (b) In practice, Conflict serializability is what systems support because it can be enforced efficiently.

   (c) To allow more concurrency, some special schedules are handled at the application level.

Conflict Serializability

Schedules are equivalent to some serial schedule. This is what (almost) every DBMS supports when you ask for the `SERIALIZABLE` isolation level.

1. Schedule S is conflict serializable if you are able to transform S into a serial schedule by swapping consecutive non-conflicting operations of different transactions.

2. **Dependency Graphs** (aka "precedence graph"):

   (a) One node per transaction.

   (b) Edge from T_i to T_j if an operation O_i of T_i conflicts with an operation O_j of T_j and O_i appears earlier in the schedule than O_j.

   (c) **A schedule is conflict serializable if and only if its dependency graph is acyclic.**

View Serializability

Allows for all schedules that are conflict serializable and "blind writes". Thus allows for slightly more schedules than Conflict serializability, but difficult to enforce efficiently. This is because the DBMS does not now how the application will "interpret" values.

ACID: Durability

All of the changes of committed transactions must be **durable** (i.e., persistent) after a crash or restart. The DBMS can either use logging or shadow paging to ensure that all changes are durable

Conclusion

1. Concurrency control and recovery are among the most important functions provided by a DBMS.

2. Concurrency control is automatic:

   (a) System automatically inserts lock/unlock requests and schedules actions of different transactions.

   (b) Ensures that resulting execution is equivalent to eecutin the transactions one after the other in some order.