Transactions: Atomic, Durable, and Distributed

Tom Kelliher, CS 318

May 1, 2002

1 Administrivia

Announcements

Final will only cover material since the midterm. It will be one hour. We will begin with a demonstration of the three projects.

Assignment

From Last Time

Transactions and isolation.

Outline

1. Atomic transactions.

2. Durable transactions.

3. Distributed transactions, serializability and replication.

Coming Up

Course evaluation, review.
2 Atomic Transactions

1. Atomicity: what is it?

2. How do we achieve atomicity when:

   (a) a transaction aborts?

   (b) the system crashes while transactions are active?

3. Maintain a write-ahead log. Records contained therein:

   (a) Begin record for each transaction.

   (b) Commit or abort record for each transaction.

   (c) Before- (undo) and after-image (redo) records for each database item modified.

   (d) Checkpoint records.

Example:

...  
Begin T1  
Undo T1 X <val>  
Redo T1 X <val'>  
Begin T2  
Begin T3  
Checkpoint T0 T1 T2 T3  
Commit T1  
Undo T2 X <val>  
Redo T2 X <val'>  
Undo T2 Y <val>  
Redo T2 Y <val'>  
Undo T3 Z <val>  
Redo T3 Z <val'>  
Abort T2

Using the WAL:
(a) when a transaction aborts. Scan backwards.

(b) when a crash occurs.

   Locate and rollback active transactions. Use of checkpoint records.

4. Write ahead: log records written before data updated. Why?

3  Durable Transactions

1. Durability: what is it?

2. How do we achieve durability when:

   (a) machines crash?

   (b) disks fatally crash?

3. Strategies:

   (a) Maintain data and log on different drives.

   (b) Mirror data on different drives.

   (c) Take database offline and dump.

      “Fuzzy” dumps of online databases.

4. Recovery from lost data:

   (a) Scan forwards using redo records, to re-establish data at time of crash.

   (b) Scan backwards using undo records, to rollback uncompleted transactions.
4 Distributed Transactions

The other day I was configuring a Sun server. Transactions involved:

1. Checking warehouse database for inventory.
2. Checking logistics database for shipping times.
3. Checking corporate database for pricing.

These are distributed multidatabase transactions.

1. Consider a transaction of a transfer of $1,000 from one bank branch to another. Each branch locally maintains its own customer database.

2. Two subtransactions:
   (a) Withdrawal $1,000 from branch A.
   (b) Deposit $1,000 to branch B.

What can go wrong?

3. How do we ensure atomicity?
   (a) A transaction manager manages the distributed transaction. It is the coordinator.
       Responsible for atomicity.
   (b) The individual DBMS' are known as cohorts.
   (c) Two phase commit protocol:
       i. Coordinator sends Prepare message to cohorts.
       ii. Each cohort responds with a vote: Commit or Abort.

       Silence is interpreted as abort. How long to wait?

       Once a cohort votes to commit, it cannot reverse the decision.
iii. Depending upon the outcome of the vote, the coordinator sends a **Commit** or **Abort** message to each cohort.

iv. Upon completion, each cohort responds with a **Done** message.

4. Strict two phase locking at the cohorts and two-phase commit at the transaction manager ensure global serializability!

### 4.1 Replication

1. Replicate distributed databases to:
   
   (a) improve response time.

   (b) improve availability.

2. Synchronous update systems: all replicas are updated before any DB modifications commit.

   Lower throughput when modification rate high. Global serializability ensured.

3. Asynchronous update systems: one replica update before any modifications commit. Other replicas get updated later.

   (a) Closest replica updated on commit (or other metric).

   Conflict resolution required.

   (b) Primary copy replica updated on commit.

   No conflict resolution necessary.

Greater throughput. Can produce non-serializable schedules.