Transactions

Jan Chomicki
University at Buffalo

Transactions

Transaction

Execution of a user program in a DBMS.

Transaction properties

- Atomicity: all-or-nothing execution
- Consistency: database consistency is preserved
- Isolation: concurrently executing transactions have no effect on one another
- Durability: results survive failures.

Schedules

Transaction (DBMS view)

- list of actions (read or write)
- terminated by commit or abort

Schedule

- interleaving of multiple transactions
- action order within transaction preserved
- complete: commit/abort for every transaction
- serial: no interleaving of actions from different transactions
- serializable: equivalent to a serial schedule (assuming all transactions commit).

Conflicts

Conflict

- a pair of actions of different transactions on the same object
- one action is a write
- a conflict orders the transactions

Conflicts influence serializability

- WR: reading uncommitted data
- RW: unrepeatable reads
- WW: overwriting uncommitted data.

Reading uncommitted data

T_1	debit(A,1000), credit(B,1000)
T_2	increase A by 10%, increase B by 10%

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

Unrepeatable read

T_3	credit(A,1000)
T_4	credit(A,2000)

T_3	T_4
R(A)	
	R(A)
	W(A)
	Commit
W(A)	
Commit	

Overwriting uncommitted data

T_5	book(F1,AA), book(F2,AA)
T_6	book(F1,Delta), book(F2,Delta)

T_5	T_6
W(F1)	
	W(F1)
	W(F2)
	Commit
W(F2)	
Commit	

Aborted transactions

The effect of aborted transactions has to be completely undone.

Problems

- a transaction depending on an aborted transaction may have already committed (unrecoverable schedule)
- aborting a transaction requires aborting other transactions (cascading aborts).

Unrecoverable schedule

T_7	debit(A,100)
T_8	increase A by 10%, increase B by 10%

T_7	T_8
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
Abort	

Strict two-phase locking

Rules

1. before an object is accessed, an appropriate lock on the object(read: shared mode, write: exclusive mode) needs to be obtained
2. lock in exclusive mode: no other transaction can lock the object in any mode
3. lock in shared mode: other transactions can lock the object in shared mode
4. a transaction cannot lock an object more than once
5. all the locks are held until the end of transaction.

Guarantees

- schedule serializability
- schedule recoverability
- no cascading aborts

Locking

Locks are stored in a lock table (managed by DBMS), lock requests are queued.

Lock/unlock: atomic operations.

Problems

- deadlocks
- starvation.

Deadlocks

Deadlock

A set of transactions such that each waits for a lock held by another one.

Handling deadlocks

- prevention:
  - object ordering
  - transaction priorities
  - obtaining all the locks at the beginning
- detection:
  - identifying cycles in the waits-for graph or timeout, and
  - abort transaction.

Handling starvation

- FIFO lock queues.

Database recovery

Types of failures

- transaction abort
- system crash
- media failure

Memory levels

- disk blocks
- main memory buffers
- local variables
- the same object may have a copy at each level

Basic transaction operations

Operations

- INPUT(X): Copy the disk block containing the object X to a buffer.
- READ(X,A): Copy the object X to a local variable A (preceded by INPUT(X) if necessary).
- WRITE(X,A): Copy the value of the local variable A to the object X (preceded by INPUT(X) if necessary).
- OUTPUT(X): Copy the block containing X from buffer to disk.

Assumption: each object fits into one block.

Logging

Recording all the operations in an append-only log (also stored on disk).

Log records

- <START T>
- <COMMIT T>
- <ABORT T>
- <T,X,old,new>

UNDO/REDO logging

UNDO/REDO rule

Before modifying an object X on disk on behalf of a transaction T, a log update record <T,X,old,new> needs to be written to disk.

Recovery

1. Redo all the committed transactions in the order earliest-first.
2. Undo all the incomplete transactions in the order latest-first.

Checkpointing

1. Write <START CKPT (T1,...Tk)> log record, where T1,...Tk are all the active transactions, and flush the log.
2. Flush all dirty buffers.
3. Write <END CKPT> log record, and flush the log.

Distributed transactions

Transactions

- subtransactions executing at different sites
- all subtransactions commit or none does (commit protocol)
- site and link failures.

Two-phase commit

A site is designated as a coordinator, other participating sites are subordinates.

Protocol

1. *Coordinator:* send a PREPARE message to each subordinate
2. *Subordinate:* receive PREPARE and decide to commit or abort:
   - commit: write a prepare log record, flush log, reply YES;
   - abort: write an abort log record, flush log, reply NO.
3. *Coordinator:*
   - all subordinates reply YES: write a commit log record, flush log, send a COMMIT message to each subordinate;
   - one replies NO or times out: write an abort log record, flush log, send an ABORT message to each subordinate.
4. *Subordinate:*
   - receive COMMIT: write a commit log record, flush log, send ACK to coordinator, commit;
   - receive ABORT: write an abort log record, flush log, send ACK, abort.
5. *Coordinator:* receive ACK from all subordinates: write end log record.