Transactions

Jan Chomicki
University at Buffalo

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

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>debit(A,1000), credit(B,1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_2$</td>
<td>increase A by 10%, increase B by 10%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(A)</td>
<td>R(A)</td>
</tr>
<tr>
<td>W(A)</td>
<td>W(A)</td>
</tr>
<tr>
<td>R(B)</td>
<td>R(B)</td>
</tr>
<tr>
<td>W(B)</td>
<td>W(B)</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
</tr>
</tbody>
</table>

Unrepeatable read

<table>
<thead>
<tr>
<th>$T_3$</th>
<th>credit(A,1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_4$</td>
<td>credit(A,2000)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$T_3$</th>
<th>$T_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(A)</td>
<td>R(A)</td>
</tr>
<tr>
<td></td>
<td>W(A)</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
</tr>
<tr>
<td>W(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Commit</td>
</tr>
</tbody>
</table>
Overwriting uncommitted data

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_5$</td>
<td>book(F1,AA), book(F2,AA)</td>
</tr>
<tr>
<td>$T_6$</td>
<td>book(F1,Delta), book(F2,Delta)</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_5$</td>
<td>$T_6$</td>
</tr>
<tr>
<td>W(F1)</td>
<td>W(F1)</td>
</tr>
<tr>
<td></td>
<td>W(F2)</td>
</tr>
<tr>
<td>W(F2)</td>
<td>Commit</td>
</tr>
<tr>
<td>Commit</td>
<td></td>
</tr>
</tbody>
</table>

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_7$</td>
<td>debit(A,100)</td>
</tr>
<tr>
<td>$T_8$</td>
<td>increase A by 10%, increase B by 10%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$T_7$</th>
<th>$T_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(A)</td>
<td></td>
<td>R(A)</td>
</tr>
<tr>
<td>W(A)</td>
<td></td>
<td>W(A)</td>
</tr>
<tr>
<td></td>
<td>R(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W(B)</td>
<td>Commit</td>
</tr>
<tr>
<td></td>
<td>Abort</td>
<td></td>
</tr>
</tbody>
</table>

---

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