Distributed Database Systems
Locking Protocols

Instructor: Randal Burns
Lecture for March 5, 2000
Computer Science 600.416
Johns Hopkins University

Locking Protocols
Overview

• Goals of locking
• Shared and exclusive locks
• Two-phase locking
  – Simple locking is non-serializable and wormhole transactions
• Locking theorems
• 2PL in distributed databases
  – Centralized
  – Primary copy
  – Distributed
• Degrees of Isolation
Goals of Locking

• Implement serializable schedules
  – For concurrent transactions
  – On-line, as transactions occur

• Part of a DBMS
  – Under the applications layer
    • Applications use transactions
    • DBMS implement transactional guarantees through locking
Shared and Exclusive Locks

• A shared lock, also called a read lock, on data item x allows a transaction to read that item: SL(x) or SLOCK(x)
  – Shared refers to its semantics
  – Read refers to the operation
  – In other data models, one may be able to take more actions than read based on a shared lock

• An exclusive lock, also called a write lock, on data item x allows a transaction to write that item: XL(x) or XLOCK(x)
  – Again, exclusive refers to its semantics

• Locks are released by RXL(x) or RSL(x)
# Shared and Exclusive Locks

<table>
<thead>
<tr>
<th>Requested Mode</th>
<th>Lock Compatibility Table</th>
<th>Granted Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL</td>
<td>SL</td>
<td>Yes</td>
</tr>
<tr>
<td>XL</td>
<td>XL</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requested Mode</th>
<th>Lock Compatibility Table</th>
<th>Granted Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL</td>
<td>SL</td>
<td>Yes</td>
</tr>
<tr>
<td>XL</td>
<td>XL</td>
<td>No</td>
</tr>
</tbody>
</table>

## Locking Protocols
Lock Compatibility

• Compatibility refers whether locks may be outstanding at the same time
  – Many shared locks may be outstanding
  – Data does not change under a shared lock
  – A single exclusive lock may be outstanding
  – Data may be modified under an exclusive lock

• Locks are managed entities
  – A lock manager uses the compatibility table to resolve outcomes
  – Transactions make requests of this lock manager
  – The lock manager function may limit the scalability of locking
Legal and Well-Formed

• A transaction is *well-formed* if every read, write and unlock action is protected by a suitable corresponding lock
• A history/schedule is *legal* if it does not grant conflicting locks to two transactions at the same time.
• Suprisingly, well-formed transactions conducted with legal histories are not serializable as a rule
  – Not enough to achieve isolation
Wormhole Transactions

• Simple locking rules
  – Before every read acquire an SLOCK
  – Before every write acquire an XLOCK
  – After the last read/write release the SLOCK/XLOCK

• This technique does not provide serial equivalent schedules
  – Recall that the goal of locking is to implement serial equivalent schedules
Wormhole Transactions

• Consider the following two transactions
  – T1: R(x), x=x+1,W(x), R(y), y=y-1,W(y)
  – T2: R(x), x=x*2,W(x), R(y), y=y*2,W(y)
  – Can be serialized by locking as
    • XL1(x), R1(x), x=x+1, W1(x), RXL1(x)
    • XL2(x), R2(x), x=x*2, W2(x), RXL2(x)
    • XL2(y), R2(y), y=y*2, W2(y), RXL2(y)
    • XL1(y), R1(y), y=y-1, W1(y), RXL1(y)
  • Outcome: x=(x+1)*2, y=y*2-1
  • Legal (serial) outcomes
    – x=(x+1)*2, y=(y-1)*2
    – x=x*2+1, y=y*2-1
Wormhole Transactions

• What happened?
  – Two parts to T1 and T2: part X and part Y

<table>
<thead>
<tr>
<th>Illegal</th>
<th>Legal</th>
<th>Legal</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1:PartX</td>
<td>T1:PartX</td>
<td>T2:PartX</td>
</tr>
<tr>
<td>T2:PartX</td>
<td>T2:PartX</td>
<td>T1:PartX</td>
</tr>
<tr>
<td>T2:PartY</td>
<td>T1:PartY</td>
<td>T2:PartY</td>
</tr>
<tr>
<td>T1:PartY</td>
<td>T2:PartY</td>
<td>T1:PartY</td>
</tr>
</tbody>
</table>

• This is called a wormhole
  – Not serial equivalent, neither T1->T2 nor T2->T1
  – Not isolated, T1 sees the DB state before and after T2
  – Hence, wormhole, time-travel in an atomic action

Locking Protocols
Two-Phase Locking (2PL)

- 2PL Rule: no transaction should request a lock after it releases one of its locks
  - Separates locking into a shrinking phase and a growing phase
  - Diagram
  - The “Lock Point” is when all locks are held for the whole transactions

- **What is the significance of the lock point? What function does it serve in serializability?**
  - Binds all of the data used in a transaction together, in our example x and y
  - It, therefore, prevents a transaction being split into parts, creating wormholes

Locking Protocols
Locking Theorems

- **Wormhole theorem:** a history is isolated if and only if it has no wormhole transactions
- **Locking theorem:** if all transactions are well-formed and two-phase, then any legal history will be isolated
- **Locking theorem converse:** if a transaction is not well-formed or not two-phase, then it is possible to write another transaction such that the resulting pair is a wormhole
- **Rollback theorem:** an update transaction that does an unlock and then a rollback is not two-phase
  - Cannot rollback after an unlock, motivation for strict 2PL
  - Rollback performs data writes, for which locks must be held
Cascading Aborts

• In 2PL, if locks are released before the transaction outcome is determined, other transactions can run concurrently and an abort in one transaction will abort others.

• Consider our example in which we have:
  – T1:Part1, T2:Part1, T1:Part2, T1:rollback
  – This rollback has several consequences
    • T2 holds XL(x) on the data item that T1 needs to write as part of its rollback.
    • T2 cannot commit unless T1 also commits, dynamic data dependency
  – What happens? T2’s locks are broken and T2 aborts

• Aborts may affect any number or transactions and are therefore called cascading aborts
  – Imagine T1 aborts T2 which aborts T3 …
Strict and Rigorous 2PL

• **Strict 2PL**: hold all exclusive locks until transaction outcome has been determined
  - Prevents cascading rollbacks because all data modified in a transaction are locked until its outcome.

• **Rigorous 2PL**: hold all locks until transaction outcome has been determined
  - *Since strict is adequate to prevent cascading aborts, how is rigorous different?*
  - Makes sure that commit order and serialization order are the same.
  - *Is this important? When?*
    • Consider two transactions conducted at the same site in which a long running transaction **T1** which reads **x** is ordered before a short transaction **T2** that writes **x**. **T2** returns first, showing an updated version of **x** long before **T1** completes on the old version.
2PL in Distributed Systems

• If data are distributed or replicated how does it affect locking protocols
  – Centralized: have a lock manager for all data
    • Scalability problems
  – Primary copy: each fragment, replica has an owning site that maintains locks for that site
    • Lockings is distributed in the same fashion as data
  – Distributed: all replica owners keep lock state
  – *Is distributed locking necessary for high availability?*
  – *What advantages might distributed locking have over primary copy locking?*
Degrees of Isolation

- Degree 0: does not overwrite another transactions dirty data if the other transaction is Deg 1 or greater
- Degree 1: has no lost updates
- Degree 2: has no lost updates and no dirty reads
- Degree 3: has no lost updates, repeatable reads (which means no dirty reads). This is true isolation
- Note that the isolation level is an external to the transaction concept
  - *E.g.* the system could allow dirty reads if it will perform cascading aborts in response to the writing transaction’s rollback

Locking Protocols
Locking Protocols

Isolation and Naming

• Degree 0: anarchy
• Degree 1: browse
• Degree 2: cursor stability
• Degree 3: isolated, serializable, or repeatable reads
• Degrees of isolation theorem:
  – If a transaction observes the 0,1,2,3 degree protocol, then any legal history will give that transaction 1,2,3, degree isolation, as long as other transactions are at least degree 1.
  – What this means? If everybody runs 1 or higher, everyone gets their needs met.
Isolation and Locking

• Degree 0: protocol is well-formed w.r.t. writes
• Degree 1: two-phase for XLOCK and well-formed w.r.t writes
• Degree 2: two-phase for XLOCK locks and well-formed
• Degree 3: two-phase and well-formed
• Different trans. have different concurrency and consistency needs.
  – Trans map needs to levels of isolation
  – Allows application to specify its needs
Isolation and SQL

• SQL queries run at degree 2, not 3, by default
  – Degree 2 is used so that the database may be scanned without locking whole relations for the duration of the scan
    • Consider 2 trans scanning the DB at the same time
• Actually, SQL does slightly better by keeping a shared lock on the cursor
  – Hence the name of the isolation
2 Isolation and Lost Updates

• 2 degree isolation causes lost updates
  – After the select, all read locks are released
  – Balance can be written by another transaction
  – The update overwrites the balance based on the wrong value

Exec sql select balance into :balance from account where account_id = :id;
balance = balance + 10;
Exec sql update account set balance = :balance where account_id = :id;
Solving with a Cursor

- SQL has the concept of a cursor, on which a read lock is maintained.
  - Holds the read value stable

```
Exec sql declare cursor c for
    select balance
    from account
    where account_id = :id;

Sql open c;

Sql fetch c into :balance;
balance = balance + 10;

Exec sql update account
    set balance = :balance
    where account_id = :id;

Sql close c;
```
But Searched Update is OK

- Since the action occurs in a single update, the DB correctly updates the value
  - Fine for 1 or 2 degree isolation
  - Each SQL statement is atomic

```
Exec sql update account
    set balance = balance+10
    where account_id=:id;
```
Problems with <4 Degree

• **What is wrong with the lower degrees of isolation?**
  – Breaks isolation in the ACID semantics
  – Requires cursors to prevent lost updates

• **What are the tradeoffs?**
  – Violation of transaction as a programming abstraction
  – Better models applications needs
  – Improve performance (more concurrency, fewer conflicts to be modeled, lower overhead)