Concurrency Control

Instructor: Matei Zaharia

cs245.stanford.edu
Outline

What makes a schedule serializable?
Conflict serializability
Precedence graphs

Enforcing serializability via 2-phase locking
  » Shared and exclusive locks
  » Lock tables and multi-level locking

Optimistic concurrency with validation
Concurrency control + recovery
Lock Modes Beyond S/X

Examples:

(1) increment lock

(2) update lock
Example 1: Increment Lock

Atomic addition action: \( \text{IN}_i(A) \)

\[
\{\text{Read}(A); \ A \leftarrow A+k; \ \text{Write}(A)\}
\]

\( \text{IN}_i(A), \ \text{IN}_j(A) \) do not conflict, because addition is commutative!
## Compatibility Matrix

<table>
<thead>
<tr>
<th>compat</th>
<th>S</th>
<th>X</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>X</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>I</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Update Locks

A common deadlock problem with upgrades:

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-S_1(A)</td>
<td>I-S_2(A)</td>
</tr>
<tr>
<td>I-X_1(A)</td>
<td>I-X_2(A)</td>
</tr>
</tbody>
</table>

--- Deadlock ---
Solution

If Ti wants to read A and knows it may later want to write A, it requests an **update lock** (not shared lock)
## Compatibility Matrix

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>X</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>U</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lock already held in

New request

compat
**Compatibility Matrix**

Note: asymmetric table!

<table>
<thead>
<tr>
<th>compat</th>
<th>S</th>
<th>X</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>X</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>U</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
How Is Locking Implemented In Practice?

Every system is different (e.g., may not even provide conflict serializable schedules)

But here is one (simplified) way ...
Sample Locking System

1. Don’t ask transactions to request/release locks: just get the weakest lock for each action they perform

2. Hold all locks until transaction commits
Sample Locking System

Under the hood: lock manager that keeps track of which objects are locked

» E.g. hash table

Also need a good way to block transactions until locks are available, and find deadlocks
Which Objects Do We Lock?

Table A

Table B

Tuple A

Tuple B

Tuple C

Disk block A

Disk block B

DB

DB

DB
Which Objects Do We Lock?

Locking works in any case, but should we choose small or large objects?
Which Objects Do We Lock?

Locking works in any case, but should we choose small or large objects?

If we lock large objects (e.g., relations)
  – Need few locks
  – Low concurrency

If we lock small objects (e.g., tuples, fields)
  – Need more locks
  – More concurrency
We Can Have It Both Ways!

Ask any janitor to give you the solution...

<table>
<thead>
<tr>
<th>Stall 1</th>
<th>Stall 2</th>
<th>Stall 3</th>
<th>Stall 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restroom</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hall</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example
Example

R1

- T_1(IS)
- T_1(S)

- t_1
- t_2
- t_3
- t_4
Example

R1

$T_1(\text{IS}), T_2(\text{S})$
Example 2
Example 2

R1

- \( t_1 \)
- \( t_2 \)
- \( t_3 \)
- \( t_4 \)

- \( T_1(IS) \)
- \( T_2(IX) \)
- \( T_1(S) \)
- \( T_2(X) \)
Example 3

R1

\[ T_1(\text{IS}), T_2(\text{S}), T_3(\text{IX})? \]
# Multiple Granularity Locks

<table>
<thead>
<tr>
<th>compat</th>
<th>Requestor</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>IX</td>
</tr>
<tr>
<td>IS</td>
<td></td>
</tr>
<tr>
<td>IX</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
</tr>
<tr>
<td>SIX</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
# Multiple Granularity Locks

<table>
<thead>
<tr>
<th>Holder</th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>SIX</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>IX</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>S</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>SIX</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>X</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
## Rules Within A Transaction

<table>
<thead>
<tr>
<th>Parent locked in</th>
<th>Child can be locked by same transaction in</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>IS, S</td>
</tr>
<tr>
<td>IX</td>
<td>IS, S, IX, X, SIX</td>
</tr>
<tr>
<td>S</td>
<td>none</td>
</tr>
<tr>
<td>SIX</td>
<td>X, IX, SIX</td>
</tr>
<tr>
<td>X</td>
<td>none</td>
</tr>
</tbody>
</table>
Multi-Granularity 2PL Rules

1. Follow multi-granularity compat function
2. Lock root of tree first, any mode
3. Node Q can be locked by Ti in S or IS only if parent(Q) locked by Ti in IX or IS
4. Node Q can be locked by Ti in X, SIX, IX only if parent(Q) locked by Ti in IX, SIX
5. Ti is two-phase
6. Ti can unlock node Q only if none of Q’s children are locked by Ti
Exercise:

Can $T_2$ access object f2.2 in X mode? What locks will $T_2$ get?
Exercise:
Can \( T_2 \) access object f2.2 in X mode? What locks will \( T_2 \) get?
Exercise:

Can $T_2$ access object $f_{3.1}$ in X mode? What locks will $T_2$ get?
Exercise:

Can $T_2$ access object $f2.2$ in S mode? What locks will $T_2$ get?
Exercise:

Can $T_2$ access object f2.2 in X mode? What locks will $T_2$ get?
# Insert + Delete Operations

<table>
<thead>
<tr>
<th>A</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>α</td>
</tr>
</tbody>
</table>

![Diagram showing insertion and deletion operations]

Insert
Changes to Locking Rules:

1. Get exclusive lock on A before deleting A

2. At insert A operation by Ti, Ti is given exclusive lock on A
Still Have Problem: Phantoms

Example: relation $R$ (id, name,…)
constraint: id is unique key
use tuple locking

\[ R \]
\[
<table>
<thead>
<tr>
<th>o1</th>
<th>id</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>o1</td>
<td>55</td>
<td>Smith</td>
</tr>
<tr>
<td>o2</td>
<td>75</td>
<td>Jones</td>
</tr>
</tbody>
</table>
\]
**T1:** Insert `<12, Mary, ...>` into R

**T2:** Insert `<12, Sam, ...>` into R

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1(o1)$</td>
<td>$S_2(o1)$</td>
</tr>
<tr>
<td>$S_1(o2)$</td>
<td>$S_2(o2)$</td>
</tr>
<tr>
<td>Check Constraint</td>
<td>Check Constraint</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>Insert $o_3[12, Mary, ..]$</td>
<td>Insert $o_4[12, Sam, ..]$</td>
</tr>
</tbody>
</table>
Solution

Use multiple granularity tree

Before insert of node N, lock parent(N) in X mode
## Back to Example

<table>
<thead>
<tr>
<th>T₁: Insert&lt;12,Mary&gt;</th>
<th>T₂: Insert&lt;12,Sam&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>X₁(R)</strong></td>
<td><strong>X₂(R)</strong></td>
</tr>
</tbody>
</table>

**Check constraint**

Insert<12,Mary>

**U₁(R)**

**X₂(R)**

**Check constraint**

Oops! e# = 12 already in R!
Instead of Using R, Can Use Index Nodes for Ranges

Example:
Outline

What makes a schedule serializable?
Conflict serializability
Precedence graphs

Enforcing serializability via 2-phase locking
  » Shared and exclusive locks
  » Lock tables and multi-level locking

Optimistic concurrency with validation
Concurrent control + recovery
Validation Approach

Transactions have 3 phases:

1. Read
   » Read all DB values needed
   » Write to temporary storage
   » No locking

2. Validate
   » Check whether schedule so far is serializable

3. Write
   » If validate OK, write to DB
Key Idea

Make validation atomic

If $T_1, T_2, T_3, \ldots$ is the validation order, then resulting schedule will be conflict equivalent to $S_s = T_1, T_2, T_3, \ldots$
Implementing Validation

System keeps track of two sets:

FIN = transactions that have finished phase 3 (write phase) and are all done

VAL = transactions that have successfully finished phase 2 (validation)
Example That Validation Must Prevent:

\[ RS(T2) = \{B\} \quad \text{RS}(T3) = \{A, B\} \neq \emptyset \]

\[ WS(T2) = \{B, D\} \quad WS(T3) = \{C\} \]
Example That Validation Must Prevent:

$RS(T2) = \{B\}$  $RS(T3) = \{A, B\} \neq \emptyset$

$WS(T2) = \{B, D\}$  $WS(T3) = \{C\}$

Diagram:

- $T_2$ start
- $T_2$ validated
- $T_3$ start
- $T_2$ finish phase 3
- $T_3$ validated
- Time line

Allow
Another Thing Validation Must Prevent:

\[ RS(T_2) = \{A\} \quad RS(T_3) = \{A, B\} \]

\[ WS(T_2) = \{D, E\} \quad WS(T_3) = \{C, D\} \]
Another Thing Validation Must Prevent:

$$RS(T_2) = \{A\} \quad RS(T_3) = \{A, B\}$$

$$WS(T_2) = \{D, E\} \quad WS(T_3) = \{C, D\}$$

BAD: $w_3(D)$ $w_2(D)$
Another Thing Validation Must Prevent:

\[ RS(T_2) = \{A\} \quad RS(T_3) = \{A,B\} \]
\[ WS(T_2) = \{D,E\} \quad WS(T_3) = \{C,D\} \]

T2 validated

T2 finish

T3 validated

T2 finish

Time
Validation Rules for Tj:

when Tj starts phase 1:
    ignore(Tj) $\leftarrow$ FIN

at Tj Validation:
    if Check(Tj) then
        VAL $\leftarrow$ VAL $\cup$ \{Tj\}
        do write phase
        FIN $\leftarrow$ FIN $\cup$ \{Tj\}
Check(Tj)

for Ti ∈ VAL – ignore(Tj) do
    if (WS(Ti) ∩ RS(Tj) ≠ ∅ or
        (Ti ∉ FIN and WS(Ti) ∩ WS(Tj) ≠ ∅))
        then return false

return true
Exercise

U: $RS(U) = \{B\}$
$WS(U) = \{D\}$

W: $RS(W) = \{A, D\}$
$WS(W) = \{A, C\}$

T: $RS(T) = \{A, B\}$
$WS(T) = \{A, C\}$

V: $RS(V) = \{B\}$
$WS(V) = \{D, E\}$

△ start
⊕ validate
☆ finish
Is Validation = 2PL?
S: $w_2(y) \ w_1(x) \ w_2(x)$

Achievable with 2PL?

Achievable with validation?
S: \( w_2(y) \; w_1(x) \; w_2(x) \)

S can be achieved with 2PL:
\[ l_2(y) \; w_2(y) \; l_1(x) \; w_1(x) \; u_1(x) \; l_2(x) \; w_2(x) \; u_2(x) \; u_2(y) \]

S cannot be achieved by validation:
The validation point of \( T_2 \), \( \text{val}_2 \), must occur before \( w_2(y) \) since transactions do not write to the database until after validation. Because of the conflict on \( x \), \( \text{val}_1 < \text{val}_2 \), so we must have something like:

\[ \text{S: } \text{val}_1 \; \text{val}_2 \; w_2(y) \; w_1(x) \; w_2(x) \]

With the validation protocol, the writes of \( T_2 \) should not start until \( T_1 \) is all done with writes, which is not the case.
Validation Subset of 2PL?

Possible proof (Check!):

» Let S be validation schedule

» For each T in S insert lock/unlocks, get S’:
  • At T start: request read locks for all of RS(T)
  • At T validation: request write locks for WS(T);
    release read locks for read-only objects
  • At T end: release all write locks

» Clearly transactions well-formed and 2PL

» Must show S’ is legal (next slide)
Validation Subset of 2PL?

Say $S'$ not legal (due to w-r conflict):
$S'$: ... $l_1(x)$ $w_2(x)$ $r_1(x)$ val1 $u_1(x)$ ...
  » At val1: $T_2$ not in Ignore($T_1$); $T_2$ in VAL
  » $T_1$ does not validate: $WS(T_2) \cap RS(T_1) \neq \emptyset$
  » contradiction!

Say $S'$ not legal (due to w-w conflict):
$S'$: ... val1 $l_1(x)$ $w_2(x)$ $w_1(x)$ $u_1(x)$ ...
  » Say $T_2$ validates first (proof similar if $T_1$ validates first)
  » At val1: $T_2$ not in Ignore($T_1$); $T_2$ in VAL
  » $T_1$ does not validate:
    $T_2 \notin FIN$ AND $WS(T_1) \cap WS(T_2) \neq \emptyset$
  » contradiction!
Is Validation = 2PL?
When to Use Validation?

Validation performs better than locking when:
» Conflicts are rare
» System resources are plentiful
» Have tight latency constraints
Summary

Have studied several concurrency control mechanisms used in practice
  » 2 PL
  » Multiple granularity
  » Validation

Next: how does concurrency control interact with failure recovery?
Outline

What makes a schedule serializable?
Conflict serializability
Precedence graphs

Enforcing serializability via 2-phase locking
  » Shared and exclusive locks
  » Lock tables and multi-level locking

Optimistic concurrency with validation

Concurrency control + recovery