Chapter 18: Concurrency Control

(Slides by Hector Garcia-Molina, http://www-db.stanford.edu/~hector/cs245/notes.htm)

Chapter 18

Chapter 18

Concurrency Control

T1

T2

... Tn

DB

(consistency constraints)

What’s Concurrency Control

• The general process of assuring that transactions preserve consistency when executing simultaneously.

• Scheduler may delay the requests, or abort the transaction

Transaction Manager

Read/write requests

Scheduler

Reads writes

Buffer
Assumption

- Every individual transaction transforms the DB from a consistent state to another consistent state
  - Transactions maintain DB consistency
- Problem: Concurrently running transactions
  - Schedules of concurrent actions of transactions make a difference
  - A schedule is a time-ordered actions taken by one or more transactions

Example:

T1: Read(A) T2: Read(A)
    A ← A + 100  A ← A × 2
    Write(A)     Write(A)
    Read(B)      Read(B)
    B ← B + 100  B ← B × 2
    Write(B)     Write(B)

Constraint: A = B

Schedule A

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read(A); A ← A + 100</td>
<td></td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Write(A);</td>
<td></td>
<td>125</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(B); B ← B + 100</td>
<td></td>
<td>125</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write(B);</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(A); A ← A × 2;</td>
<td></td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td></td>
<td>Write(A);</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(B); B ← B × 2;</td>
<td></td>
<td>250</td>
<td>250</td>
</tr>
</tbody>
</table>
### Schedule B

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A); A ← A×2; Write(A);</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Read(A); B ← B×2; Write(B);</td>
<td></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>Read(A); A ← A+100 Write(A);</td>
<td></td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Read(B); B ← B+100; Write(B);</td>
<td></td>
<td></td>
<td>150</td>
</tr>
<tr>
<td></td>
<td></td>
<td>150</td>
<td>150</td>
</tr>
</tbody>
</table>

### Schedule C

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A); A ← A+100 Write(A);</td>
<td></td>
<td>125</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>250</td>
</tr>
<tr>
<td>Read(A); A ← A×2; Write(A);</td>
<td></td>
<td></td>
<td>250</td>
</tr>
<tr>
<td>Read(B); B ← B+100; Write(B);</td>
<td></td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>Read(B); B ← B×2; Write(B);</td>
<td></td>
<td></td>
<td>250</td>
</tr>
<tr>
<td></td>
<td></td>
<td>250</td>
<td>250</td>
</tr>
</tbody>
</table>

### Schedule D

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A); A ← A+100 Write(A);</td>
<td></td>
<td>125</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>250</td>
</tr>
<tr>
<td>Read(A); A ← A×2; Write(A);</td>
<td></td>
<td></td>
<td>250</td>
</tr>
<tr>
<td>Read(B); B ← B+100; Write(B);</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Read(B); B ← B×2; Write(B);</td>
<td></td>
<td></td>
<td>150</td>
</tr>
<tr>
<td></td>
<td></td>
<td>250</td>
<td>150</td>
</tr>
</tbody>
</table>
### Schedule E

Same as Schedule D but with new T2'

<table>
<thead>
<tr>
<th>T1</th>
<th>T2'</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(A); A ← A + 100</td>
<td>Read(A); A ← A x 1;</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Write(A);</td>
<td>Write(A);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read(B); B ← B x 1;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write(B);</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>125</td>
<td>125</td>
</tr>
</tbody>
</table>

- **Want schedules that are “good”, regardless of**
  - initial state and
  - transaction semantics
- **Only look at order of read and writes**

**Example:**

\[
Sc = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)
\]

**Example:**

\[
Sc' = r_1(A)w_1(A) r_1(B)w_1(B)r_2(A)w_2(A)r_2(B)w_2(B)
\]
However, for Sd:
\[ Sd = r_1(A)w_1(A)r_2(A)w_2(A) r_2(B)w_2(B)r_1(B)w_1(B) \]

- \( T_2 \rightarrow T_1 \)
- Also, \( T_1 \rightarrow T_2 \)

\[ T_1 \xrightarrow{\text{\bigcirc}} T_2 \quad \Rightarrow \quad Sd \text{ cannot be rearranged into a serial schedule} \]
\[ \Rightarrow Sd \text{ is not “equivalent” to any serial schedule} \]
\[ \Rightarrow Sd \text{ is “bad”} \]

Returning to Sc
\[ Sc = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B) \]

- no cycles \( \Rightarrow \) Sc is “equivalent” to a serial schedule
  (in this case \( T_1, T_2 \))

In a serial schedule, actions from different transactions do not mix.
Conflict Actions

• A pair of actions such that if their order is changed, then the behavior of at least one of the transactions can change
• Actions that do not conflict
  – r_i(x), r_j(y), even if x=y
  – r_i(x), w_j(y) is not a conflict provided x!=y
  – w_i(x), w_j(y) is not a conflict provided x!=y

Conflict Actions

• Actions that do conflict
  – Two actions of the same transaction
    • r_i(x), r_j(y)
  – A read and a write of the same DB element by two different transactions
    • r_i(x), w_j(x)
  – Two writes of the same DB element by two different transactions
    • w_i(x), w_j(x)

Conflict Actions

• Simple rules
  – Two actions of different transactions conflict if
    • They involve the same DB element, and
    • At least one is a write
  • Conflict actions cannot be swapped without affecting the involved transactions
Chapter 18

Concepts

Transaction: sequence of \( r(x), w(x) \) actions

Conflicting actions:

\[
\begin{align*}
& r_1(A) \quad w_2(A) \quad w_1(A) \\
& w_2(A) \quad r_1(A) \quad w_2(A)
\end{align*}
\]

Schedule: represents chronological order in which actions are executed

Serial schedule: no interleaving of actions or transactions

Chapter 18

Definition

\( S_1, S_2 \) are conflict equivalent schedules if \( S_1 \) can be transformed into \( S_2 \) by a series of swaps on non-conflicting actions.

Chapter 18

Exercise

- \( S_1: r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B) \)
- \( S_2: r_1(A)w_1(A)r_1(B)w_1(B)r_2(A)w_2(A)r_2(B)w_2(B) \)
- \( S_3: r_2(A)w_2(A)r_2(A)w_1(A)r_1(B)w_1(B)r_1(B)w_2(B) \)

- Is \( S_1 \) conflict equivalent to \( S_2 \)?
- How about \( S_1 \) and \( S_3 \)?
- How about \( S_2 \) and \( S_3 \)?
Definition
A schedule is conflict serializable if it is conflict equivalent to some serial schedule.

Exercise
• S1: r₁(A)w₁(A)r₂(A)w₁(A)r₁(B)w₁(B)r₂(B)w₂(B)
• S2: r₁(A)w₁(A)r₁(B)w₁(B)r₂(A)w₁(A)r₂(B)w₂(B)
• S3: r₂(A)w₂(A)r₁(A)w₁(A)r₁(B)w₁(B)r₂(B)w₂(B)

• Is S1 conflict serializable?
• How about S2?
• How about S3?

Precedence graph P(S) (S is schedule)
Nodes: transactions in S
Arcs: Tᵢ → Tⱼ whenever
- p(A), q(A) are actions in S
- p(A) <ₛ q(A)
- at least one of pᵢ, qᵢ is a write
Exercise:

- What is $P(S)$ for $S = w_3(A) w_2(C) r_1(A) w_1(B) r_1(C) w_2(A) r_4(A) w_4(D)$

- Is $S$ serializable?

Lemma

$S_1, S_2$ conflict equivalent $\Rightarrow P(S_1) = P(S_2)$

Note: $P(S_1) = P(S_2) \neq S_1, S_2$ conflict equivalent

Counter example:

$S_1 = w_1(A) r_2(A) w_2(B) r_1(B)$

$S_2 = r_2(A) w_1(A) r_1(B) w_2(B)$
**Theorem**

\[ P(S_1) \text{ acyclic } \iff S_1 \text{ conflict serializable} \]

**Proof:**

\[ \iff \text{ by cycle } \Rightarrow \text{ not conflict serializable} \]

\[ \Rightarrow \text{ induction} \]

**Key observation:** acyclic implies that at least one node Ti has no incoming arc; all actions in Ti can be moved to the front of \( S_1 \)

---

**Conflict Serializable v.s. Serializable**

- \( w_1(y); w_2(y); w_3(x); w_4(x) \)
  - Conflict serializable?

- Serializable?
  - Writes of X by T1 and T2 have no effect

---

**How to enforce serializable schedules?**

*Option 1:* run system, recording \( P(S) \);
  at end of day, check for \( P(S) \) cycles and declare if execution was good
How to enforce serializable schedules?

Option 2: prevent P(S) cycles from occurring

\[ T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \ldots \rightarrow T_n \]

Scheduler

DB

A locking protocol

Two new actions:
- lock (exclusive): \( l_i (A) \)
- unlock: \( u_i (A) \)

\[ T_1, T_2, \ldots \]

lock
table

scheduler

Rule #1: Consistent transactions

\( T_i: \ldots \ l_i(A) \ldots \ p_i(A) \ldots \ u_i(A) \ldots \)
Chapter 18

Rule #2  Legal scheduler

\[ S = \ldots l_i(A) \ldots u_i(A) \ldots \rightarrow \]
o\!\!\!\!l_i(A)

Exercise:

- What schedules are legal?
- What transactions are consistent?

\[ S_1 = l_1(A)l_1(B)r_1(A)w_1(B)l_2(A)u_1(A)r_2(B)w_2(B)u_2(B) \]
\[ l_3(B)r_3(B)w_3(B)u_3(B) \]
\[ S_2 = l_1(A)r_1(A)w_1(B)u_1(A)r_1(B)w_1(B) \]
\[ l_2(B)r_2(B)w_2(B)u_2(B) \]
\[ S_3 = l_1(A)r_1(A)u_1(A)l_1(B)w_1(B)u_1(B) \]
\[ l_2(B)r_2(B)w_2(B)u_2(B) \]

Schedule F

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_1(A);\text{Read}(A)</td>
<td>l_2(A);\text{Read}(A)</td>
</tr>
<tr>
<td>A \leftarrow A+100;\text{Write}(A);u_1(A)</td>
<td>A \leftarrow Ax2;\text{Write}(A);u_1(A)</td>
</tr>
<tr>
<td>l_1(B);\text{Read}(B)</td>
<td>l_2(B);\text{Read}(B)</td>
</tr>
<tr>
<td>B \leftarrow Bx2;\text{Write}(B);u_1(B)</td>
<td>B \leftarrow B+100;\text{Write}(B);u_1(B)</td>
</tr>
</tbody>
</table>
Schedule F

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>l:(A);Read(A)</td>
<td></td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>A=A+100;Write(A);u(A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l:(A);Read(A)</td>
<td></td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>A=A+x2;Write(A);u(A)</td>
<td></td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>l:(B);Read(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B=B+100;Write(B);u(B)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rule #3 Two phase locking (2PL)
for transactions

Ti = ........ l:(A) ........... u:(A) ........

no unlocks no locks

# locks held by Ti

Growing Shrinkng
Phase Phase
Schedule G

T1                  T2
ls(A); Read(A)     ls(A); Read(A)
A ← A + 100; Write(A) A ← A + 2; Write(A); delayed
ls(B); u(A)         l(B); u(A)

Read(B); B ← B + 100
Write(B); u(B)
Schedule H (T2 reversed)

T1
i(A); Read(A)
A←A+100; Write(A)

T2
i(B); Read(B)
B←B×2; Write(B)

deadlock

• Assume deadlocked transactions are rolled back
  – They have no effect
  – They do not appear in schedule

E.g., Schedule H on previous slide

We can show that rules #1,2,3 ⇒ conflict-serializable schedules

(see textbook)
• Beyond this simple 2PL protocol, it is all a matter of improving performance and allowing more concurrency... 
  – Shared locks
  – Multiple granularity
  – Inserts, deletes and phantoms
  – Other types of C.C. mechanisms

Shared locks

So far:
S = ...l1(A) r1(A) u1(A) ... l2(A) r2(A) u2(A) ...

Do not conflict

Instead:
S=... ls1(A) r1(A) ls2(A) r2(A) .... us1(A) us2(A)

Lock actions
l-t(A): lock A in t mode (t is S or X)
u-t(A): unlock t mode (t is S or X)

Shorthand:
u(A): unlock whatever modes
T_i has locked A
• What about transactions that read and write same object?

Option 1: Request exclusive lock
\[ T_i = ...l-X_i(A) \ldots r_i(A) \ldots w_i(A) \ldots u_i(A) \ldots \]

Option 2: Upgrade
(E.g., need to read, but don’t know if will write...)
\[ T_i = ... l-S_i(A) \ldots r_i(A) \ldots l-X_i(A) \ldots w_i(A) \ldots u_i(A) \ldots \]

The same as:
- Getting 2nd lock on A, or
- Dropping S, getting X lock

Compatibility matrix

<table>
<thead>
<tr>
<th>Comp</th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>X</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
Lock types beyond S/X

Examples:
(1) increment lock
   (see textbook)
(2) update lock

Update locks

Common deadlock problem with upgrades:

\[
\begin{array}{c|c}
T1 & T2 \\
\hline
l-S_1(A) & l-S_2(A) \\
l-X_1(A) & l-X_2(A) \\
\end{array}
\]

--- Deadlock ---

Solution

If \( T_1 \) wants to read \( A \) and knows it may later want to write \( A \), it requests **update** lock (not shared)
Chapter 18

How does locking work 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 trust transactions to request/release locks
(2) Hold all locks until transaction commits

# locks vs. time

Scheduler, part I

Scheduler, part II

Ti

Read(A), Write(B)

l(A), Read(A), l(B), Write(B)...

Read(A), Write(B)

DB
Lock table
Conceptually

Every possible object

<table>
<thead>
<tr>
<th>A</th>
<th>Λ</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Λ</td>
</tr>
<tr>
<td></td>
<td>Λ</td>
</tr>
</tbody>
</table>

If null, object is unlocked

Lock info for B

Lock info for C

What are the objects we lock?

Relation A

Tuple A

Disk block A

Relation B

Tuple B

Disk block B

Tuple C

Disk block C

DB

DB

DB
• 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

Summary

Have looked at 2PL, a concurrency-control mechanism commonly used in practice

Others (in the textbook):
  - Multiple granularity
  - Tree (index) protocols
  - Timestamping
  - Validation