Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

Lecture #22: Concurrency Control
Part 2 (R&G ch. 17)

Outline

• conflict/view serializability
• Two-phase locking (2PL); strict 2PL (== 2PL-C, for 'Commit')
• deadlocks prevention & detection
• Locking granularity
• Tree locking protocols
• Phantoms & predicate locking

Review questions

• conflict serializability?
• 2PL theorem?
• what is strict 2PL? why do we need it?
  – ‘dirty read’?
  – cascading aborts?
• who generates the lock requests?
Not in book: ‘Lost update’ problem

\begin{tabular}{ll}
T1 & T2 \\
Read(N) & Read(N) \\
N=N-1 & N= N-1 \\
Write(N) & Write(N) \\
\end{tabular}

Major conclusions so far:

- (strict) 2PL: extremely popular
- Deadlock may still happen
  - detection: wait-for graph
  - prevention: abort some xacts, defensively
- philosophically: concurrency control uses:
  - locks
  - and aborts

Outline

- conflict/view serializability
- Two-phase locking (2PL); strict 2PL (== 2PL-C, for ‘Commit’)
- deadlocks prevention & detection
- Locking granularity
- Tree locking protocols
- Phantoms & predicate locking
Lock granularity?
- lock granularity
  - field? record? page? table?
- Pros and cons?
  - (Ideally, each transaction should obtain a few locks)

Multiple granularity
• Eg:
  ![Diagram of a database structure with DB, Tables, Records, and Attributes]

What would you do?
• T1: read Smith’s salary,
  • while T2: give 10% raise to everybody
  • what locks should they obtain?
  ![Diagram of a database structure with DB, Tables, Records, and Attributes]
What types of locks?

• X/S locks for leaf level +
• ‘intent’ locks, for higher levels

What types of locks?

• X/S locks for leaf level +
• ‘intent’ locks, for higher levels
• IS: intent to obtain S-lock underneath
• IX: intent …. X-lock …
• S: shared lock for this level
• X: ex- lock for this level
• SIX: shared lock here; + IX

Protocol

- each xact obtains appropriate lock at highest level
- proceeds to desirable lower levels
### Compatibility matrix

<table>
<thead>
<tr>
<th>T1 wants</th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>SIX</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>IX</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>X</td>
</tr>
<tr>
<td>S</td>
<td>N</td>
<td></td>
<td>N</td>
<td>N</td>
<td>X</td>
</tr>
<tr>
<td>SIX</td>
<td>N</td>
<td></td>
<td>N</td>
<td>N</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>N</td>
<td></td>
<td>N</td>
<td>N</td>
<td>X</td>
</tr>
</tbody>
</table>
Compatibility matrix

<table>
<thead>
<tr>
<th>wants</th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>SIX</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>IX</td>
<td></td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIX</td>
<td></td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Multiple Granularity Lock Protocol

- Each Xact: lock root.
- To get S or IS lock on a node, must hold at least IS on parent node.
  - What if Xact holds SIX on parent? S on parent?
- To get X or IX or SIX on a node, must hold at least IX on parent node.
- Must release locks in bottom-up order.
Multiple granularity protocol

X
- SIX
  - S
    - IX
      - IS

- stronger
  - (more privileges)
- weaker

Examples – 2 level hierarchy

- T1 scans R, and updates a few tuples:
- T1 gets an SIX lock on R, then get X lock on tuples that are updated.
Examples – 2 level hierarchy

- T2: find avg salary of ‘Sales’ employees
  - T2 gets an IS lock on R, and repeatedly gets an S lock on tuples of R.

Examples – 2 level hierarchy

- T3: sum of salaries of everybody in ‘R’:
Examples – 2 level hierarchy

- T3: sum of salaries of everybody in ‘R’;
- T3 gets an S lock on R.
- OR, T3 could behave like T2; can use lock escalation to decide which.
  - Lock escalation dynamically asks for coarser-grained locks when too many low level locks acquired

Multiple granularity

- Very useful in practice
- each xact needs only a few locks

Outline

- ...
- Locking granularity
  - Tree locking protocols
  - Phantoms & predicate locking
Locking in B+ Trees

- What about locking indexes?

Example B+tree

- T1 wants to insert in H
- T2 wants to insert in I
- why not plain 2PL?

Example B+tree

- T1 wants to insert in H
- T2 wants to insert in I
- why not plain 2PL?
- Because: X/S locks for too long!
Two main ideas:

- 'crabbing': get lock for parent; get lock for child; release lock for parent (if 'safe')
- 'safe' nodes == nodes that won’t split or merge, ie:
  - not full (on insertion)
  - more than half-full (on deletion)

Example B+tree

- T1 wants to insert in H
- crabbing:

Example B+tree

- T1 wants to insert in H
Example B+tree

- T1 wants to insert in H
- (if 'B' is 'safe')

A Simple Tree Locking Algorithm: “crabbing”

- **Search:** Start at root and go down; repeatedly,
  - S lock child
  - then unlock parent
- **Insert/Delete:** Start at root and go down, obtaining X locks as needed. Once child is locked, check if it is **safe**:
  - If child is safe, release all locks on ancestors.
Example

\begin{itemize}
\item Do:
  \begin{enumerate}
  \item Search 38*
  \item Delete 38*
  \item Insert 45*
  \item Insert 25*
  \end{enumerate}
\end{itemize}

Answers:

1. Search 38*
2. Delete 38*
   - X A, X B, X C; U A, U B, X D, U C
3. Insert 45*
   - X A, X B; U A, X C, X E, U C
4. Insert 25*
   - X A, X B, U A, X F, U B, X H

Can we do better?
Can we do better?

- Yes [Bayer and Schkolnik]:
  - Idea: hope that the leaf is ‘safe’, and use S-locks & crabbing to reach it, and verify
  - (if false, do previous algo)


A Better Tree Locking Algorithm (From Bayer-Schkolnick paper)

- Search: As before.
- Insert/Delete:
  - Set locks as if for search, get to leaf, and set X lock on leaf.
  - If leaf is not safe, release all locks, and restart Xact using previous Insert/Delete protocol.
- Gambles that only leaf node will be modified; if not, S locks set on the first pass to leaf are wasteful. In practice, better than previous alg.
Example

Do:
1) Delete 38*
2) Insert 25*

Answers:

1. Delete 38*
   - S A, S B, U A, S C, U B, X D, U C
2. Insert 25*
   - S A, S B, U A, S F, U B, X H; U H;
   - X A, X B, U A, X F, U B, X H

Notice:

- Textbook has a third variation, that uses lock-upgrades (and may lead to deadlocks)
Outline

- Locking granularity
- Tree locking protocols
- Phantoms & predicate locking

Dynamic Databases – The “Phantom” Problem

- so far: only reads and updates – no insertions/deletions
- with insertions/deletions, new problems:

The phantom problem

```
<table>
<thead>
<tr>
<th>Time</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>71</td>
<td>select max(age) ... where rating=1</td>
<td>insert ... age=96 rating=1</td>
</tr>
<tr>
<td>96</td>
<td>select max(age) ... where rating=1</td>
<td></td>
</tr>
</tbody>
</table>
```
Why?

• because T1 locked only *existing* records – not ones under way!
• Solution?

Solution

theoretical solution:
• ‘predicate locking’: e.g., lock all records (current or incoming) with rating=1
  – VERY EXPENSIVE

practical solution:
• index locking: if an index (on ‘rating’) exists, lock the appropriate entries (rating=1 in our case)
• otherwise, lock whole table (and thus block insertions/deletions)
Transaction Support in SQL-92

**SERIALIZABLE** – No phantoms, all reads repeatable, no "dirty" (uncommitted) reads.
- REPEATABLE READS – phantoms may happen.
- READ COMMITTED – phantoms and unrepeatable reads may happen
- READ UNCOMMITTED – all of them may happen.

---

Transaction Support in SQL-92

- **SERIALIZABLE** : obtains all locks first; plus index locks, plus strict 2PL
- **REPEATABLE READS** – as above, but no index locks
- **READ COMMITTED** – as above, but S-locks are released immediately
- **READ UNCOMMITTED** – as above, but allowing ‘dirty reads’ (no S-locks)

---

Transaction Support in SQL-92

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE READ ONLY

Defaults:
- **SERIALIZABLE**
- **READ WRITE**
Summary

- Multiple granularity locking: leads to few locks, at appropriate levels
- Tree-structured indexes:
  - ‘crabbing’ and ‘safe nodes’

Summary

- “phantom problem”, if insertions/deletions
  - (Predicate locking prevents phantoms)
  - Index locking, or table locking