CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #16: Transactions 2: 2PL and Deadlocks
Reminders

- On Thursday Nov 1
  - PA2 due
  - PA3 and HW7 out
  - Recitation for PA3 by Deepika – don’t miss!

- On Tuesday Nov 6
  - No lecture, but Deepika will be in the classroom during lecture time for extra office hours.
Review (last lecture)

- DBMSs support ACID Transaction semantics.
- Concurrency control and Crash Recovery are key components.
Review

- For Isolation property, serial execution of transactions is safe but slow
  - Try to find schedules equivalent to serial execution
- One solution for “conflict serializable” schedules is Two Phase Locking (2PL)
Outline

- 2PL/2PLC
- Lock Management
- Deadlocks
  - detection
  - Prevention
- Specialized Locking
Serializability in Practice

- DBMS does not test for conflict serializability of a given schedule
  - Impractical as interleaving of operations from concurrent Xacts could be dictated by the OS

- Approach:
  - Use specific protocols that are known to produce conflict serializable schedules
  - But may reduce concurrency
Solution?

- One solution for “conflict serializable” schedules is Two Phase Locking (2PL)
Answer

- (Full answer:) use locks; keep them until commit (‘strict 2 phase locking’)  
- Let’s see the details
<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(N)</td>
<td>Read(N)</td>
</tr>
<tr>
<td>$N = N - 1$</td>
<td>$N = N - 1$</td>
</tr>
<tr>
<td>Write(N)</td>
<td>Write(N)</td>
</tr>
</tbody>
</table>
Solution – part 1

- with locks:
- lock manager: grants/denies lock requests
Lost update problem – with locks

T1
- lock(N)
- Read(N)
- N=N-1
- Write(N)
- Unlock(N)

T2
- lock(N)
- grants lock
- T2: waits
- grants lock to T2

lock manager
- denies lock

Prakash 2018
Locks

- Q: I just need to read ‘N’ - should I still get a lock?
Solution – part 1

- Locks and their flavors
  - exclusive (or write-) locks
  - shared (or read-) locks
  - <and more ... >

- Compatibility matrix

<table>
<thead>
<tr>
<th>T2 wants</th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 has</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solution – part 1

- Locks and their flavors
  - exclusive (or write-) locks
  - shared (or read-) locks
  - <and more ... >

- compatibility matrix

<table>
<thead>
<tr>
<th>T2 wants</th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 has</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td><strong>Yes</strong></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solution – part 1

- transactions request locks (or upgrades)
- lock manager grants or blocks requests
- transactions release locks
- lock manager updates lock-table
Solution – part 2

locks are not enough – eg., the ‘inconsistent analysis’ problem
Inconsistent analysis

\[
\begin{align*}
T1 & : & \text{Read}(A) \\
& & A = A - 10 \\
& & \text{Write}(A) \\
T2 & : & \text{Read}(A) \\
& & \text{Sum} = A \\
& & \text{Read}(B) \\
& & \text{Sum} += B \\
& & \text{Read}(B) \\
& & B = B + 10 \\
& & \text{Write}(B)
\end{align*}
\]
‘Inconsistent analysis’ – w/ locks

<table>
<thead>
<tr>
<th>time</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L(A)</td>
<td>L(A)</td>
</tr>
<tr>
<td>Read(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>U(A)</td>
<td>L(B)</td>
<td></td>
</tr>
</tbody>
</table>

the problem remains!

T2 reads an inconsistent DB state

Solution??
General solution:

- Protocol(s)
- Most popular protocol: 2 Phase Locking (2PL)
2PL

X-lock version: transactions issue no lock requests, after the first ‘unlock’

THEOREM: if **ALL** transactions in the system obey 2PL --> all schedules are serializable
2PL – example

- ‘inconsistent analysis’ – how does 2PL help?
- how would it be under 2PL?
2PL – X/S lock version

transactions issue no lock/upgrade request, after the first unlock/downgrade

In general: ‘growing’ and ‘shrinking’ phase
transactions issue no lock/upgrade request, after the first unlock/downgrade

In general: ‘growing’ and ‘shrinking’ phase

violation of 2PL
Two-Phase Locking (2PL), cont.

- 2PL on its own is sufficient to guarantee conflict serializability (i.e., schedules whose precedence graph is acyclic), but, it is subject to Cascading Aborts.

![Diagram showing the number of locks held over time with acquisition and release phases.]
2PL

- Problem: Cascading Aborts
- Example: rollback of T1 requires rollback of T2!

| T1: R(A), W(A), R(B), W(B), Abort |
| T2: R(A), W(A) |

- Solution: Strict 2PL, i.e,
- keep all locks, until ‘commit’
Strict 2PL

- Allows only conflict serializable schedules, but it is actually stronger than needed for that purpose.

Graph:
- # locks held vs. time
- Acquisition phase:
- Release all locks at end of xact
In effect, “shrinking phase” is delayed until
– Transaction commits (commit log record on disk), or
– Aborts (then locks can be released after rollback).

# locks held

acquisition phase

release all locks at end of xact

time

Prakash 2018
Non-2PL, $A=1000$, $B=2000$, Output =?

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock_X(A)</td>
<td></td>
</tr>
<tr>
<td>Read(A)</td>
<td></td>
</tr>
<tr>
<td>$A := A-50$</td>
<td></td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
</tr>
<tr>
<td>Unlock(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lock_S(A)</td>
</tr>
<tr>
<td>Read(A)</td>
<td></td>
</tr>
<tr>
<td>Unlock(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lock_S(B)</td>
</tr>
<tr>
<td>Read(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRINT(A+B)</td>
</tr>
<tr>
<td>Lock_X(B)</td>
<td></td>
</tr>
<tr>
<td>Read(B)</td>
<td></td>
</tr>
<tr>
<td>$B := B +50$</td>
<td></td>
</tr>
<tr>
<td>Write(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>Action</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Lock_X(A)</td>
<td></td>
</tr>
<tr>
<td>Read(A)</td>
<td></td>
</tr>
<tr>
<td>A := A - 50</td>
<td></td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
</tr>
<tr>
<td>Lock_X(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(A)</td>
<td></td>
</tr>
<tr>
<td>Lock_S(A)</td>
<td></td>
</tr>
<tr>
<td>Read(A)</td>
<td></td>
</tr>
<tr>
<td>Read(B)</td>
<td></td>
</tr>
<tr>
<td>B := B + 50</td>
<td></td>
</tr>
<tr>
<td>Write(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
</tr>
<tr>
<td>Lock_S(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(A)</td>
<td></td>
</tr>
<tr>
<td>Read(B)</td>
<td></td>
</tr>
<tr>
<td>Unlock(B)</td>
<td></td>
</tr>
<tr>
<td>PRINT(A + B)</td>
<td></td>
</tr>
</tbody>
</table>

2PL, A = 1000, B = 2000, Output = ?
Strict 2PL, A = 1000, B = 2000, Output = ?

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Lock_X(A)</td>
<td></td>
</tr>
<tr>
<td>2. Read(A)</td>
<td></td>
</tr>
<tr>
<td>3. A := A - 50</td>
<td></td>
</tr>
<tr>
<td>4. Write(A)</td>
<td></td>
</tr>
<tr>
<td>5. Lock_X(B)</td>
<td></td>
</tr>
<tr>
<td>6. Read(B)</td>
<td></td>
</tr>
<tr>
<td>7. B := B + 50</td>
<td></td>
</tr>
<tr>
<td>8. Write(B)</td>
<td></td>
</tr>
<tr>
<td>9. Unlock(A)</td>
<td></td>
</tr>
<tr>
<td>10. Unlock(B)</td>
<td></td>
</tr>
<tr>
<td>11. Lock_S(A)</td>
<td></td>
</tr>
<tr>
<td>12. Read(A)</td>
<td></td>
</tr>
<tr>
<td>13. Lock_S(B)</td>
<td></td>
</tr>
<tr>
<td>14. Read(B)</td>
<td></td>
</tr>
<tr>
<td>15. Read(B)</td>
<td></td>
</tr>
<tr>
<td>16. PRINT(A + B)</td>
<td></td>
</tr>
<tr>
<td>17. Unlock(A)</td>
<td></td>
</tr>
<tr>
<td>18. Unlock(B)</td>
<td></td>
</tr>
</tbody>
</table>
Venn Diagram for Schedules

All Schedules

Conflict Serializable

Avoid Cascading Abort

Serial
Q: Which schedules does Strict 2PL allow?

- All Schedules
- Conflict Serializable
  - Avoid
  - Cascading
  - Abort
  - Serial
Q: Which schedules does Strict 2PL allow?

All Schedules

Conflict Serializable

Avoid Cascading Abort

Serial
Another Venn diagram

- Serializable schedules
- 2PL schedules
- 2PLC
- Serial schedules
Outline

- 2PL/2PLC
- Lock Management
  - detection
  - Prevention
- Specialized Locking
Lock Management

- Lock and unlock requests handled by the Lock Manager (LM).
- LM contains an entry for each currently held lock.
- Q: structure of a lock table entry?
Lock Management

- Lock and unlock requests handled by the Lock Manager (LM).
- LM contains an entry for each currently held lock.
- Lock table entry:
  - Ptr. to list of transactions currently holding the lock
  - Type of lock held (shared or exclusive)
  - Pointer to queue of lock requests
Lock Management, cont.

- When lock request arrives see if any other xact holds a conflicting lock.
  - If not, create an entry and grant the lock
  - Else, put the requestor on the wait queue

- Lock upgrade: transaction that holds a shared lock can be upgraded to hold an exclusive lock
Lock Management, cont.

- Two-phase locking is simple enough, right?
- We’re not done. There’s an important wrinkle ...
Example: Output = ?

<table>
<thead>
<tr>
<th>Lock_X(A)</th>
<th>Lock_S(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read(B)</td>
</tr>
<tr>
<td></td>
<td>Lock_S(A)</td>
</tr>
<tr>
<td>Read(A)</td>
<td></td>
</tr>
<tr>
<td>A: = A-50</td>
<td></td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
</tr>
<tr>
<td>Lock_X(B)</td>
<td></td>
</tr>
</tbody>
</table>
Example: Output = ?

<table>
<thead>
<tr>
<th>Lock_X(A)</th>
<th>Lock_S(B)</th>
<th>Read(B)</th>
<th>Lock_S(A)</th>
<th>Read(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: = A-50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write(A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lock_X(B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

lock mgr:
- grant
- grant
- wait
Outline

- Lock Management
- Deadlocks
  - detection
  - Prevention
- Specialized Locking
Deadlocks

- **Deadlock**: Cycle of transactions waiting for locks to be released by each other.

- **Two ways of dealing with deadlocks:**
  - Deadlock prevention
  - Deadlock detection

- **Many systems just punt and use Timeouts**
  - What are the dangers with this approach?
Deadlock Detection

- Create a waits-for graph:
  - Nodes are transactions
  - Edge from Ti to Tj if Ti is waiting for Tj to release a lock

- Periodically check for cycles in waits-for graph
Deadlock Detection (Continued)

Example:

T1: S(A), S(D), S(B)
T2: X(B) X(C)
T3: S(D), S(C), X(A)
T4: X(B)
Another example

- is there a deadlock?
- if yes, which xacts are involved?
Another example

- now, is there a deadlock?
- if yes, which xacts are involved?
Deadlock detection

- how often should we run the algo?
- how many transactions are typically involved?
Deadlock handling

• Q: what to do?
Deadlock handling

• Q0: what to do?
  • A: select a ‘victim’ & ‘rollback’

• Q1: which/how to choose?
Deadlock handling

- Q1: which/how to choose?
  - A1.1: by age
  - A1.2: by progress
  - A1.3: by # items locked already...
  - A1.4: by # xacts to rollback

- Q2: How far to rollback?
• **Q2: How far to rollback?**  
  • **A2.1: completely**  
  • **A2.2: minimally**  
• **Q3: Starvation?**
Deadlock handling

- Q3: Starvation??
- A3.1: include #rollbacks in victim selection criterion.
Outline

- Lock Management
- Deadlocks
  - detection
  - Prevention
- Specialized Locking
Deadlock Prevention

- Assign priorities based on timestamps (older -> higher priority)
- We only allow ‘old-wait-for-young’
- (or only allow ‘young-wait-for-old’)
- and rollback violators. Specifically:
- Say Ti wants a lock that Tj holds - two policies:
  - Wait-Die: If Ti has higher priority, Ti waits for Tj; otherwise Ti aborts (ie., old wait for young)
  - Wound-wait: If Ti has higher priority, Tj aborts; otherwise Ti waits (ie., young wait for old)
Deadlock Prevention

Wait-Die

Ti wants → Tj has

Priorities

Wound-Wait

Ti wants → Tj has

Prakash 2018
Deadlock Prevention

- Q: Why do these schemes guarantee no deadlocks?
- A:
- Q: When a transaction restarts, what is its (new) priority?
- A:
Deadlock Prevention

- Q: Why do these schemes guarantee no deadlocks?
  - A: only one ‘type’ of direction allowed.

- Q: When a transaction restarts, what is its (new) priority?
  - A: its original timestamp. -- Why?
SQL statement

- usually, conc. control is transparent to the user, but
- LOCK <table-name> [EXCLUSIVE|SHARED]
Quiz:

- is there a serial schedule (= interleaving) that is not serializable?
- is there a serializable schedule that is not serial?
- can 2PL produce a non-serializable schedule? (assume no deadlocks)
Quiz - cont’d

- is there a serializable schedule that can not be produced by 2PL?
- a xact obeys 2PL - can it be involved in a non-serializable schedule?
- all xacts obey 2PL - can they end up in a deadlock?
Outline

- Lock Management
- Deadlocks
  - detection
  - Prevention
- Specialized Locking
Things we will not study

- We assumed till now DB objects are fixed and independent---not true in many cases!
- Multi-level locking
  - Lock db or file or pages or record?
- What about locking indexes?
  - E.g. B+-trees
  - Crabbing Algorithm
- What about dynamic databases?
  - ‘phantom’ problem
  - Solution: predicate locking
- Non-locking based Techniques
  - Timestamp based Concurrency Control
- All these are in the textbook though
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

  isolation level
  access mode
Conclusions

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