Concurrency, 2PL, and 2PC
Today’s Topics

• Continue concurrency
  • Review two-phase locking (2PL)
  • 2PL with shared locks
  • Deadlocks
  • Timestamped concurrency

• Implementing distributed transactions
  • Transaction manager
  • Two-phase commit (2PC)
Recap from last class

Serial Equivalence: combined effect of two (or more) tx is equivalent to a serial execution

T1
read X
read Y
write Z

T2
read A
write Y
write W
write Z
Recap from last class

Serial Equivalence: combined effect of two (or more) tx is equivalent to a serial execution.

Conflicts are operations in two tx on same data whose combined effect depends on order.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>read X</td>
<td>read A</td>
</tr>
<tr>
<td>read Y</td>
<td>write Y</td>
</tr>
<tr>
<td>write Z</td>
<td>write W</td>
</tr>
<tr>
<td>write Z</td>
<td>write Z</td>
</tr>
</tbody>
</table>
Recap from last class

Serial Equivalence: combined effect of two (or more) tx is equivalent to a serial execution

Conflicts are operations in two tx on same data whose combined effect depends on order

• Read/write or write/write
Recap from last class

Serial Equivalence: combined effect of two (or more) tx is equivalent to a serial execution

Conflicts are operations in two tx on same data whose combined effect depends on order

If all conflicts follow transaction ordering, execution is serially equivalent

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>read X</td>
<td>read A</td>
</tr>
<tr>
<td>read Y</td>
<td>write Y</td>
</tr>
<tr>
<td>write Z</td>
<td>write W</td>
</tr>
<tr>
<td>write W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>write Z</td>
</tr>
</tbody>
</table>
Recap from last class

Serial Equivalence: combined effect of two (or more) tx is equivalent to a serial execution

Conflicts are operations in two tx on same data whose combined effect depends on order

If all conflicts follow transaction ordering, execution is serially equivalent

T1
read X
read Y
write Z
write W

T2
read A
write Y
write Z
Recap from last class

Serial Equivalence: combined effect of two (or more) tx is equivalent to a serial execution

Conflicts are operations in two tx on same data whose combined effect depends on order

If all conflicts follow transaction ordering, execution is serially equivalent

Two-phase locking (2PL): lock variable before access, unlock only a commit/abort time

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock X; read X</td>
<td>lock A; read A</td>
</tr>
<tr>
<td>lock Y; read Y</td>
<td>try lock Y…</td>
</tr>
<tr>
<td>lock Z; write Z</td>
<td>lock W; write W</td>
</tr>
<tr>
<td>commit; unlock all</td>
<td>commit; unlock all</td>
</tr>
<tr>
<td></td>
<td>lock Y; write Y</td>
</tr>
<tr>
<td></td>
<td>lock Z; write Z</td>
</tr>
</tbody>
</table>
Exclusive locks: missed parallelism

T1
read A
read B
write C

T2
read A
read B
write D
2P Locking: Non-exclusive lock (per object)

- A read lock is **promoted** to a write lock when the transaction needs write access to the same object.
- A read lock shared with other transactions’ read lock(s) cannot be promoted. Transaction waits for other read locks to be released.
- Cannot demote a write lock to read lock during transaction – violates the 2P principle

<table>
<thead>
<tr>
<th>Lock set</th>
<th>Lock requested</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Read</td>
<td>OK</td>
</tr>
<tr>
<td>Read</td>
<td>Write</td>
<td>Wait</td>
</tr>
<tr>
<td>Write</td>
<td>Read</td>
<td>Wait</td>
</tr>
<tr>
<td>Write</td>
<td>Write</td>
<td>Wait</td>
</tr>
</tbody>
</table>
Locking Procedure in 2P Locking

• When an operation accesses an object:
  • if the object is not already locked, lock the object in the lowest appropriate mode & proceed.
  • if the object has a conflicting lock by another transaction, wait until object has been unlocked.
  • if the object has a non-conflicting lock by another transaction, share the lock & proceed.
  • if the object has a lower lock by the same transaction,
    • if the lock is not shared, promote the lock & proceed
    • else, wait until all shared locks are released, then lock & proceed

• When a transaction commits or aborts:
  • release all locks that were set by the transactions
R/W 2PL

T1: add 1% dividend to C based on balances of A and B
Read Lock A
x := A.getBalance()
Read Lock B
y := B.getBalance()
Read Lock C
z := C.getBalance()
Promote C to write
C.setBalance((x+y) * 0.01 + z)
Unlock A, B, C

T2: transfer 100 from A to B
Read Lock A
t := A.getBalance()
Read Lock B
u := B.getBalance()
Try to promote A to write lock
Promote A to write lock
A.setBalance(t - 100)
Promote B to write lock
B.setBalance(u + 100)
Unlock A, B
Why lock promotion is necessary

T1: add 1% dividend to C based on balances of A and B
Read Lock A
x := A.getBalance()
Read Lock B
y := B.getBalance()
Read Lock C
z := C.getBalance()
Promote C to write
C.setBalance((x+y)*0.01+z)
Unlock A, B, C

T2: transfer 100 from A to B
Read Lock A
t := A.getBalance()
Read Lock B
u := B.getBalance()
Try to promote A to write lock
Promote A to write lock
A.setBalance(t-100)
Promote B to write lock
B.setBalance(u+100)
Unlock A, B
Why we need lock promotion

T1:

acquire R-lock on a
a.read()

release R-lock on a
acquire W-lock on a
a.write()
commit
release W-lock on a

T2:

acquire R-lock on a
a.read()

release R-lock on a
acquire W-lock on a
a.write()
commit
release W-lock on a
Deadlocks

- Necessary conditions for deadlocks
  - Non-shareable resources (locked objects)
  - No preemption on locks
  - Hold & Wait
  - Circular Wait (Wait-for graph)
Deadlock Resolution Using Timeout

<table>
<thead>
<tr>
<th>Transaction T</th>
<th>Operations</th>
<th>Locks</th>
<th>Transaction U</th>
<th>Operations</th>
<th>Locks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\textit{a.deposit(100);}</td>
<td>write lock \textit{a}</td>
<td></td>
<td>\textit{b.deposit(200)}</td>
<td>write lock \textit{b}</td>
</tr>
<tr>
<td></td>
<td>\textit{b.withdraw(100)}</td>
<td>waits for \textit{U}'s lock on \textit{b}</td>
<td></td>
<td>\textit{a.withdraw(200)}</td>
<td>waits for \textit{T}'s lock on \textit{a}</td>
</tr>
<tr>
<td></td>
<td>\bullet\bullet\bullet</td>
<td>(timeout elapses)</td>
<td></td>
<td>\bullet\bullet\bullet</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\text{\textit{T}'s lock on \textit{A} becomes vulnerable, unlock \textit{a}, abort \textit{T}}</td>
<td>unlock \textit{a}, abort \textit{T}</td>
<td></td>
<td>\textit{a.withdraw(200)}; commit</td>
<td>write locks \textit{a}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>unlock \textit{a}, \textit{b}</td>
</tr>
</tbody>
</table>
Deadlock Strategies

• **Timeout: how to set value?**
  • Too large -> long delays
  • Too small -> false positives

• **Deadlock prevention**
  • Lock all objects at transaction start
  • Use lock ordering

• **Deadlock Detection (later)**
  • Maintain wait-for graph, look for cycle
  • Abort one transaction in cycle
Identify all conflicts

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>read X</td>
<td>read X</td>
</tr>
<tr>
<td>write Y</td>
<td>write Y</td>
</tr>
<tr>
<td>read Z</td>
<td>read A</td>
</tr>
<tr>
<td>read W</td>
<td>read B</td>
</tr>
<tr>
<td>write V</td>
<td>write C</td>
</tr>
</tbody>
</table>
Timestamp Ordering

- Assign each transaction a unique timestamp \( ts \)
  - Serialize transactions according to timestamps
- Keep track of timestamp last transaction to read and write an object
- Maintain two invariants:
  - If T writes O, last read and write timestamp must be lower than T's
  - If T reads O, last write timestamp must be lower than T
- If T tries to read/write object with higher timestamp, abort and rollback

\[
\begin{align*}
T & (1) & U & (2) & V & (3) \\
\text{read } X & (X.\text{rts}=1) & & \text{write } Y & (Y.\text{wts}=1) \\
\text{read } X & (X.\text{rts}=2) & \text{read } Y & (Y.\text{rts}=3) & \text{write } X & (X.\text{wts}=3) \\
\text{write } X & \text{ abort!} & \end{align*}
\]