Database Management and Performance Tuning

Concurrency Tuning

Pei Li

University of Zurich
Institute of Informatics

Unit 9

Acknowledgements: The slides are provided by Nikolaus Augsten and adapted from "Database Tuning" by Dennis Shasha and Philippe Bonnet.

Outline

Undesirable Phenomena of Concurrent Transactions

- Dirty read
  - transaction reads data written by concurrent uncommitted transaction
  - problem: read may return a value that was never in the database because the writing transaction aborted
- Non-repeatable read
  - different reads on the same item within a single transaction give different results (caused by other transactions)
  - e.g., concurrent transactions T_1: $x = R(A)$, $y = R(A)$, $z = y - x$ and T_2: $W(A = 2 * A)$, then z can be either zero or the initial value of A (should be zero!)
- Phantom read
  - repeating the same query later in the transaction gives a different set of result tuples
  - other transactions can insert new tuples during a scan
  - e.g., "Q: get accounts with $balance > 1000$" gives two tuples the first time, then a new account with $balance > 1000$ is inserted by an other transaction; the second time Q gives three tuples

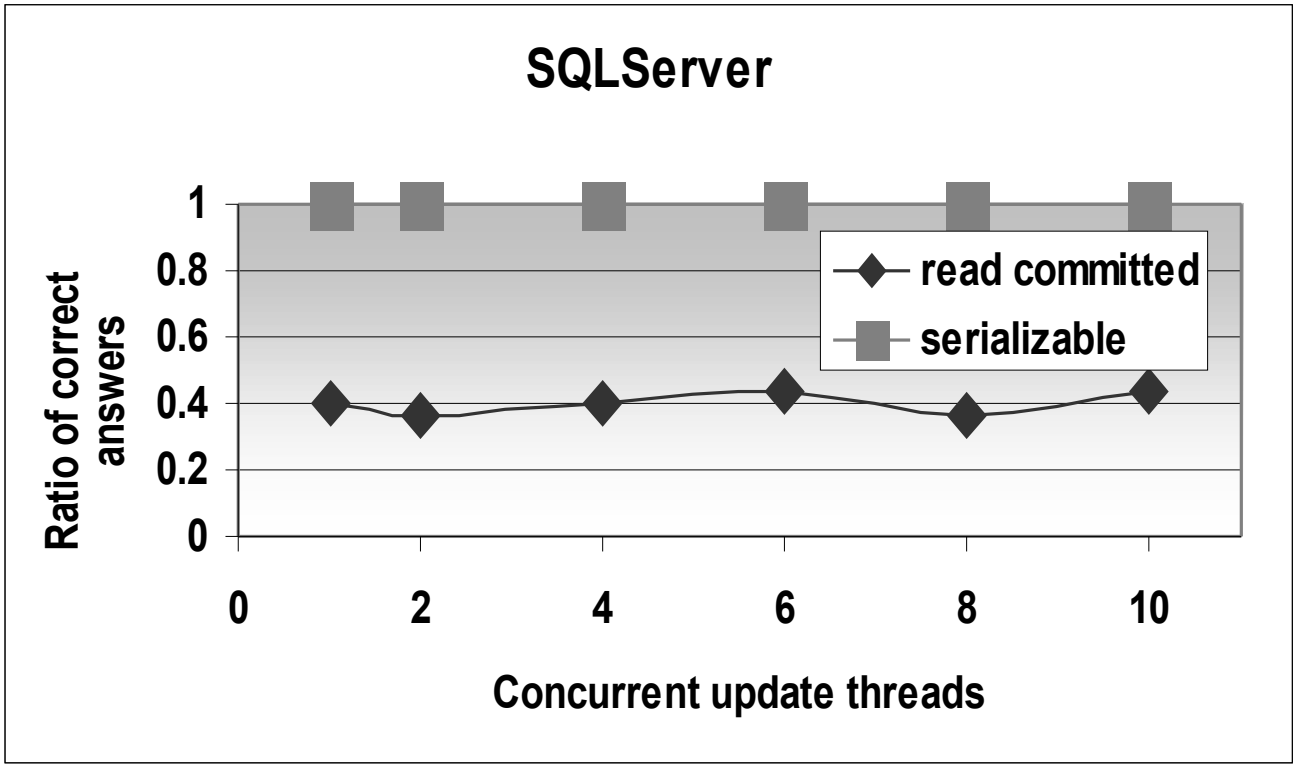
Isolation Guarantees (SQL Standard)

- Read uncommitted: dirty, non-repeatable, phantom
  - read locks released after read; write locks downgraded to read locks after write, downgraded locks released according to 2-phase locking
  - reads may access uncommitted data
  - writes do not overwrite uncommitted data
- Read committed: non-repeatable, phantom
  - read locks released after read, write locks according to 2-phase locking
  - reads can access only committed data
  - cursor stability: in addition, read is repeatable within single `SELECT`
- Repeatable read: phantom
  - 2-phase locking, but no range locks
  - phantom reads possible
- Serializable:
  - none of the undesired phenomenas can happen
  - enforced by 2-phase locking with range locks

Experiment: Read Commit vs. Serializable

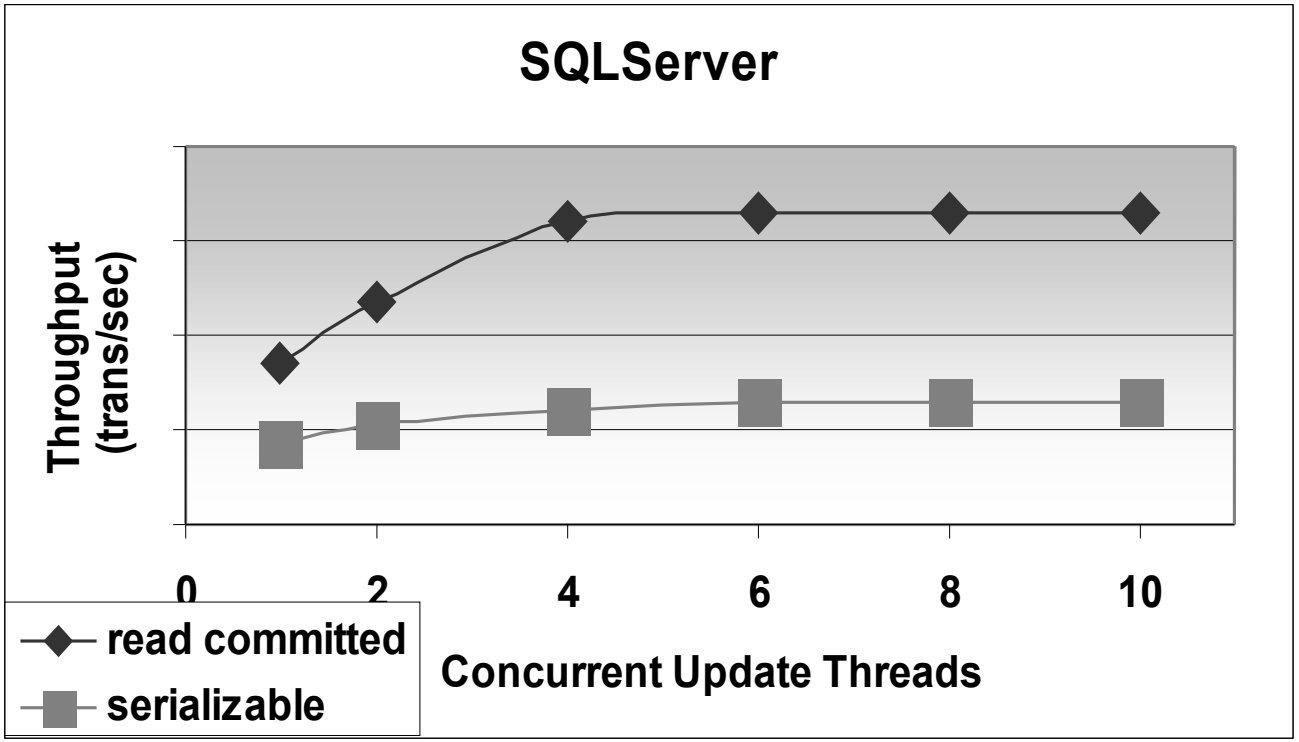
- Experimental setup:
  - T1: summation query: `SELECT SUM(balance) FROM Accounts`
  - T2: money transfers between accounts
  - row level locking
- Parameter: number of concurrent threads
- Measure:
  - percentage of correct answers (over multiple tries)
  - measure throughput

Experiment: Read Commit vs. Serializable



- Read committed allows sum of account balances after debit operation has taken place but before corresponding credit operation is performed – incorrect sum!

Experiment: Read Commit vs. Serializable



- Read committed: faster, but incorrect answers
- Serializable: always correct, but lower throughput

When To Weaken Isolation Guarantees?

- Query does not need exact answer (e.g., statistical queries)
  - example: count all accounts with balance$>$ \$1000.
  - read committed is enough!
- Transactions with human interaction
  - example: flight reservation system
  - price for serializability too high!

Example: Flight Reservation System

- Reservation involves three steps:
  1. retrieve list of available seats
  2. let customer decide
  3. secure seat
- Single transaction:
  - seats are locked while customer decides
  - all other customers are blocked!
- Two transactions: (1) retrieve list, (2) secure seat
  - seat might already be taken when customer wants to secure it
  - more tolerable than blocking all other customers

Snapshot Isolation for Long Reads – The Problem

- Consider the following scenario in a bank:
  - read-only query Q: `SELECT SUM(deposit) FROM Accounts`
  - update transaction T: money transfer between customers A and B
- 2-Phase locking inefficient for long read-only queries:
  - read-only queries hold lock on all read items
  - in our example, T must wait for Q to finish (Q blocks T)
  - deadlocks might occur:
    $T.xL(A)$, $Q.sL(B)$, $Q.sL(A)$ - wait, $T.xL(B)$ - wait
- Read-committed may lead to incorrect results:
  - Before transactions: $A = 50, B = 30$
  - $Q : sL(A), R(A) = 50, uL(A)$
  - $T : xL(A), xL(B), W(A \leftarrow A + 20), W(B \leftarrow B - 20), uL(A), uL(B)$
  - $Q : sL(B), R(B) = 10, uL(B)$
  - sum computed by Q for $A + B$ is 60 (instead of 80)

Snapshot Isolation for Long Reads

- Snapshot isolation: correct read-only queries without locking
  - read-only query Q with snapshot isolation
  - remember old values of all data items that change after Q starts
  - Q sees the values of the data items when Q started
- Example: bank scenario with snapshot isolation
  - Before transactions: $A = 50, B = 30$
  - $Q : R(A) = 50$
  - $T : xL(A), xL(B), W(A \leftarrow A + 20), W(B \leftarrow B - 20), uL(A), uL(B)$
  - $Q : R(B) = 30$ (read old value)
  - sum computed by Q for $A + B$ is 80 as it should be

Concurrency in Oracle

- "Read committed" in Oracle means:
  - non-repeatable and phantom reads are possible at the transaction level, but not within a single SQL statement
  - update conflict: if row is already updated, wait for updating transaction to commit, then update new row version (or ignore row if deleted) – no rollback!
  - possibly inconsistent state: transaction sees updates of other transaction only on the rows that itself updates
- "Serializable" in Oracle means:
  - phenomena: none of the three undesired phenomena can happen
  - update conflict: if two transactions update the same item, the transaction that updates it later must abort – rollback!
  - not serializable: snapshot isolation does not guarantee full serializability (skew writes)
- Similar in PostgreSQL.

Skew Writes: Snapshot Isolation Not Serializable

- Example: $A = 3, B = 17$
  - $T1 : A \leftarrow B$
  - $T2 : B \leftarrow A$
- Serial execution:
  - order $T1, T2$: $A = B = 17$
  - order $T2, T1$: $A = B = 3$
- Snapshot isolation:
  - $T1 : R(B) = 17$
  - $T2 : R(A) = 3$
  - $T1 : W(A \leftarrow 17)$
  - $T2 : W(B \leftarrow 3)$
  - result: $A = 17, B = 3$ (different from serial execution)

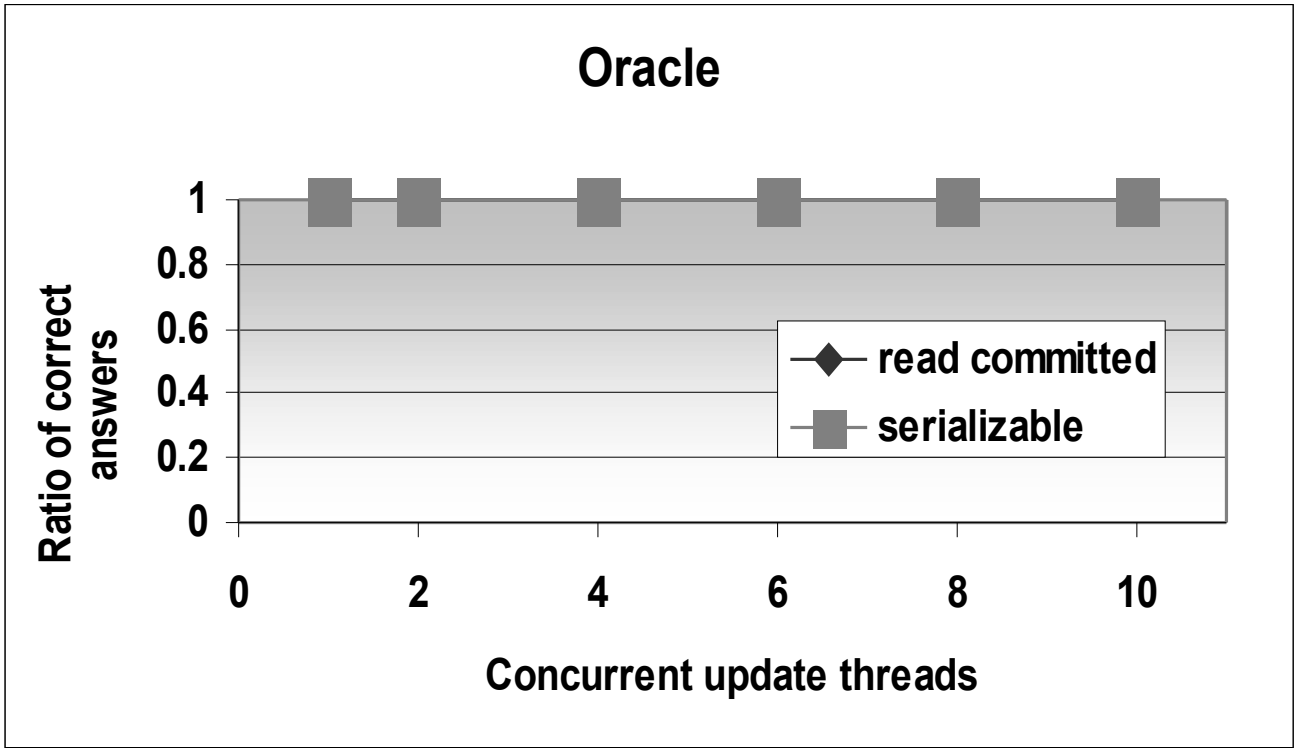
Snapshot Isolation

- Advantages: (assuming "serializable" of Oracle)
  - readers do not block writers (as with locking)
  - writers do not block readers (as with locking)
  - writers block writers only if they update the same row
  - performance similar to read committed
  - no dirty, non-repeatable, or phantom reads
- Disadvantages:
  - system must write and hold old versions of modified data (only date modified between start and end of read-only transaction)
  - does not guarantee serializability for read/write transactions
- Implementation example: Oracle 9i
  - no overhead: leverages before-image in rollback segment
  - expiration time of before-images configurable, "snapshot too old" failure if this value is too small

Snapshot Isolation – Summary

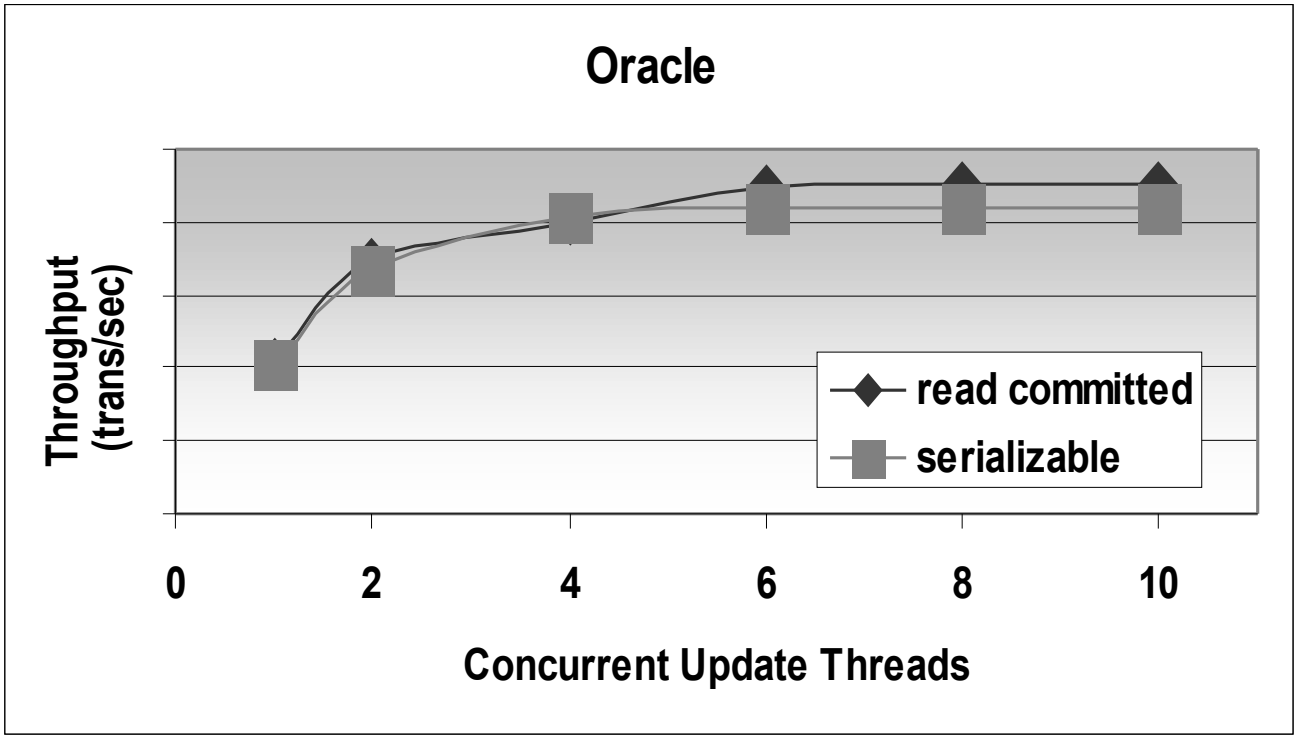
- Considerable performance advantages since reads are never blocked and do not block other transactions.
- Not fully serializable, although no dirty, non-repeatable, or phantom reads.

Experiment: Read Commit vs. Serializable



- Summation query with concurrent transfers between bank accounts.
- Oracle snapshot isolation: read-only summation query is not disturbed by concurrent transfer queries
- Summation (read-only) queries always give exact answer.

Experiment: Read Commit vs. Serializable



- Both "read commit" and "serializable" use snapshot isolation.
- "Serializable" rolls back transactions in case of write conflict.
- Summation queries always give exact answer.