Database Tuning

Concurrency Tuning

Nikolaus Augsten

University of Salzburg
Department of Computer Science
Database Group

Unit 7 – WS 2013/2014

Adapted from "Database Tuning" by Dennis Shasha and Philippe Bonnet.

Outline

1. Concurrency Tuning
   - Introduction to Transactions

What is a Transaction?[1]

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- Example: transfer \$50 from account A to account B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- Two main issues:
  1. concurrent execution of multiple transactions
  2. failures of various kind (e.g., hardware failure, system crash)

[1] Slides of section "Introduction to Transactions" are adapted from the slides "Database System Concepts", 6^{th} Ed., Silberschatz, Korth, and Sudarshan

ACID Properties

- Database system must guarantee ACID for transactions:
  - Atomicity: either all operations of the transaction are executed or none
  - Consistency: execution of a transaction in isolation preserves the consistency of the database
  - Isolation: although multiple transactions may execute concurrently, each transaction must be unaware of the other concurrent transactions.
  - Durability: After a transaction completes successfully, changes to the database persist even in case of system failure.

Atomicity

- Example: transfer \$50 from account A to account B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- What if failure (hardware or software) after step 3?
  - money is lost
  - database is inconsistent
- Atomicity:
  - either all operations or none
  - updates of partially executed transactions not reflected in database

Consistency

- Example: transfer \$50 from account A to account B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- Consistency in example: sum $A + B$ must be unchanged
- Consistency in general:
  - explicit integrity constraints (e.g., foreign key)
  - implicit integrity constraints (e.g., sum of all account balances of a bank branch must be equal to branch balance)
- Transaction:
  - must see consistent database
  - during transaction inconsistent state allowed
  - after completion database must be consistent again

Isolation – Motivating Example

- Example: transfer \$50 from account A to account B
  1. $R(A)$
  2. $A \leftarrow A - 50$
  3. $W(A)$
  4. $R(B)$
  5. $B \leftarrow B + 50$
  6. $W(B)$
- Imagine second transaction T_2:
  - $T_2 : R(A), R(B), print(A + B)$
  - T_2 is executed between steps 3 and 4
  - T_2 sees an inconsistent database and gives wrong result

Isolation

- Trivial isolation: run transactions serially
- Isolation for concurrent transactions: For every pair of transactions T_i and T_j, it appears to T_i as if either T_j finished execution before T_i started or T_j started execution after T_i finished.
- Schedule:
  - specifies the chronological order of a sequence of instructions from various transactions
  - equivalent schedules result in identical databases if they start with identical databases
- Serializable schedule:
  - equivalent to some serial schedule
  - serializable schedule of $T1$ and $T2$ is either equivalent to $T1, T2$ or $T2, T1$

Durability

- When a transaction is done it commits.
- Example: transaction commits too early
  - transaction writes A, then commits
  - A is written to the disk buffer
  - then system crashes
  - value of A is lost
- Durability: After a transaction has committed, the changes to the database persist even in case of system failure.
- Commit only after all changes are permanent:
  - either written to log file or directly to database
  - database must recover in case of a crash

Locks

- A lock is a mechanism to control concurrency on a data item.
- Two types of locks on a data item A:
  - exclusive – $xL(A)$: data item A can be both read and written
  - shared – $sL(A)$: data item A can only be read.
- Lock request are made to concurrency control manager.
- Transaction is blocked until lock is granted.
- Unlock A – $uL(A)$: release the lock on a data item A

Lock Compatibility

- Lock compatibility matrix:

$T_1 \downarrow \quad T_2 \rightarrow$	shared	exclusive
shared	true	false
exclusive	false	false

- T_1 holds shared lock on A:
  - shared lock is granted to T_2
  - exclusive lock is not granted to T_2
- T_2 holds exclusive lock on A:
  - shared lock is not granted to T_2
  - exclusive lock is not granted to T_2
- Shared locks can be shared by any number of transactions.

Locking Protocol

- Example transaction T_2 with locking:
  1. $sL(A)$, $R(A)$, $uL(A)$
  2. $sL(B)$, $R(B)$, $uL(B)$
  3. $print(A + B)$
- T_2 uses locking, but is not serializable
  - A and/or B could be updated between steps 1 and 2
  - printed sum may be wrong
- Locking protocol:
  - set of rules followed by all transactions while requesting/releasing locks
  - locking protocol restricts the set of possible schedules

Pitfalls of Locking Protocols – Deadlock

- Example: two concurrent money transfers
  - T_1: $R(A), A \leftarrow A + 10, R(B), B \leftarrow B - 10, W(A), W(B)$
  - T_2: $R(B), B \leftarrow B + 50, R(A), A \leftarrow A - 50, W(A), W(B)$
  - possible concurrent scenario with locks:
    $T_1.xL(A), T_1.R(A), T_2.xL(B), T_2.R(B), T_2.xL(A), T_1.xL(B), \ldots$
  - T_1 and T_2 block each other – no progress possible
- Deadlock: situation when transactions block each other
- Handling deadlocks:
  - one of the transactions must be rolled back (i.e., undone)
  - rolled back transaction releases locks

Pitfalls of Locking Protocols – Starvation

- Starvation: transaction continues to wait for lock
- Examples:
  - the same transaction is repeatedly rolled back due to deadlocks
  - a transaction continues to wait for an exclusive lock on an item while a sequence of other transactions are granted shared locks
- Well-designed concurrency manager avoids starvation.

Two-Phase Locking

- Protocol that guarantees serializability.
- Phase 1: growing phase
  - transaction may obtain locks
  - transaction may not release locks
- Phase 2: shrinking phase
  - transaction may release locks
  - transaction may not obtain locks

Two-Phase Locking – Example

- Example: two concurrent money transfers
  - T_1: $R(A), A \leftarrow A + 10, R(B), B \leftarrow B - 10, W(A), W(B)$
  - T_2: $R(A), A \leftarrow A - 50, R(B), B \leftarrow B + 50, W(A), W(B)$
- Possible two-phase locking schedule:
  1. $T_1 : xL(A), xL(B), R(A), R(B), W(A \leftarrow A + 10), uL(A)$
  2. $T_2 : xL(A), R(A), xL(B)$ *(wait)*
  3. $T_1 : W(B \leftarrow B - 10), uL(B)$
  4. $T_2 : R(B), W(A \leftarrow A - 50), W(B \leftarrow B + 50), uL(A), uL(B)$
- Equivalent serial schedule: T_1, T_2