Snapshot Isolation and Serializable Execution

Alan Fekete
University of Sydney (Australia)

Ack: slides from Michael Cahill, Uwe Röhm and Mohammad Alomari

This is joint work

• A. Fekete in PODS2005: "Allocating Isolation Levels to Transactions"
• S. Jorwekar, A. Fekete, K. Ramamritham, S. Sudarshan in VLDB2007: "Automating the Detection of Snapshot Isolation Anomalies"

Databases

• Data that is shared among several applications, can be stored and managed centrally in a complex software system with dedicated hardware and staff
  – Organizational benefits (accountability, economies of scale, etc)
• Database: a collection of shared data
• Database management system (DBMS): the complex software that controls access to the database

Database Research

• Study issues related to managing substantial amounts of data
• Storage, query processing, data mining, schema management, data integration
  – Hot topics in 2008: streaming, privacy, internet-scale processing, scientific data
• Combine approaches from infrastructure systems, programming languages, data structures, theory, AI, etc
• Large, unified and well-established research international community
  – 2009 is 35th VLDB conference, 29th SIGMOD, 25th ICDE
• Great commercialization record

Transaction Processing

• A powerful model from business data processing
• Each real-world change is performed through a program which executes multiple database operations
  – Some ops modify the database contents, based on data already there and on program parameters
• Eg customer purchases goods
• ACID properties:
  – Atomic (all or nothing, despite failures)
  – Consistent (maintains data integrity)
  – Isolated (no problems from concurrency)
  – Durable (changes persist despite crashes)

Serializability (academic def)

• Used to define the correctness of an interleaved execution of several transactions (formalize "isolated")
  – Same values read, same final values as in serial (batch) execution of the same transactions
• For every integrity condition C: if eachtxn acting alone preserves C, then a serializable execution will preserve C
  – That is: programmer makes sure txn does the right thing on its own, then platform makes sure no problems from concurrency
• Can be assessed by absence of cycles in a graph showing conflicts/dependencies
  – When different txns access the same items, and at least one txn modifies
But…. Vendor advice

- **Oracle**: “Database inconsistencies can result unless such application-level consistency checks are coded with this in mind, even when using serializable transactions.”

- “PostgreSQL’s Serializable mode does not guarantee serializable execution…”

Our research agenda

- Guide DBAs (or application designers) so that they do not risk data corruption from concurrency between application programs
  - Each application preserves data integrity when run alone
  - DBA can check if applications will run correctly, together
  - DBA can change things to get to this situation

- Provide assurance that all executions will be serializable
  - Running on platforms that don’t provide this guarantee in general

- Help understand the performance implications of different ways to have this assurance

Isolation Levels

- SQL standard offers several isolation levels
  - Each transaction can have level set separately
  - Problematic definitions, but in best practice done with variations in lock holding

- **Serializable**
  - (ought to be default, but not so in practice)
  - Traditionally done with Commit-duration locks on data and indices
  - We’ll call this “Two Phase Locking (2PL)”

- **Repeatable Read**
  - Commit-duration locks on data

- **Read Committed**
  - short duration read locks, commit-duration write locks

- **Read Uncommitted**
  - no read locks, commit-duration write locks

Snapshot Isolation (SI)

- A multiversion concurrency control mechanism was described in SIGMOD `95 by H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O’Neil, P. O’Neil
  - Does not guarantee serializable execution!

- Supplied by Oracle, PostgreSQL for “Isolation Level Serializable”

- Available in Microsoft SQL Server 2005 as “Isolation Level Snapshot”

First committer wins (FCW)

- T will not be allowed to commit a modification to an item if any other transaction has committed a changed value for that item since T’s start (snapshot)

- Similar to optimistic CC, but only write-sets are checked

- T must hold write locks on modified items at time of commit, to install them.
  - In practice, commit-duration write locks may be set when writes execute. These simplify detection of conflicting modifications when T tries to write the item, instead of waiting till T tries to commit.
Benefits of SI

- Reading is never blocked, and reads don’t block writes
- Avoids common anomalies
  - No dirty read
  - No lost update
  - No inconsistent read
  - Set-based selects are repeatable (no phantoms)
- Matches common understanding of isolation: concurrent transactions are not aware of one another’s changes

Is every execution serializable?

- For any set of txns, if they all run with Two Phase Locking, then every interleaved execution is serializable
- For some sets of txns, if they all run with SI, then every execution is serializable
  - Eg the txns making up TPC-C
- For some sets of txns, if they all run with SI, there can be non-serializable executions
  - Undeclared integrity constraints can be violated

Example

- Table Duties(Staff, Date, Status)
- Undeclared constraint: for every Date, there is at least 1 Staff with Status='Y'
- Transaction TakeBreak(S, D) running at SI

```sql
SELECT COUNT(*) INTO :tmp
FROM Duties
WHERE Date=:D AND Status='Y';
IF :tmp < 2 ROLLBACK;
UPDATE Duties
SET Status = 'N'
WHERE Staff =:S AND Date =:D;
COMMIT;
```

Example continued

- Possible execution, starting when two staff (S101, S103) are on duty for 2004-06-01
  - Concurrently perform
    TA: TakeBreak(S101, 2004-06-01)
    TB: TakeBreak(S103, 2004-06-01)
      - Each succeeds, as each sees snapshot with 2 on duty
      - No problem committing, as they update different rows!
  - End with no staff on duty for that date!

Write Skew

- SI breaks serializability when txns modify different items in each other’s read sets
  - Neither sees the other, but in a serial execution one would come later and so see the other’s impact
- This is fairly rare in practice
- Eg the TPC-C benchmark always runs correctly under SI
  - whenever its txns conflict (eg read/write same data), there is also a write-conflict: a shared item they both modify (like a total quantity) so SI will abort one of them

Interaction effects

- You can’t think about one program, and say “this program can use SI”
- The problems have to do with the set of application programs, not with each one by itself
- Example where T1, T2, T3 can all be run under SI, but when T4 is present, we need to fix things in T1
- Non-serializable execution can involve read-only transactions, not just updaters
Overview

1. Review of databases, isolation levels and serializability
2. Theory to determine whether an application will have serializable executions when running at SI
3. Modifying applications
4. Fixing the DBMS
5. Summary and Future work
6. Database and Middleware Research at USydney

Multiversion Serializability Theory

• From Y. Raz in RIDE’93
  – Suitable for multiversion histories
• WW-conflict from T1 to T2
  – T1 writes a version of x, T2 writes a later version of x
    • In our case, succession (version order) defined by commit times of writer txns
• WR-conflict from T1 to T2
  – T1 writes a version of x, T2 reads this version of x (or a later version)
• RW-conflict from T1 to T2 (Adya et al ICDE’00 called this "antidependency")
  – T1 reads a version of x, T2 writes a later version of x
• Serializability tested by acyclic conflict graph

Interference Theory

• We produce the "static dependency graph"
  – Node for each application program
  – Draw directed edges each of which can be either
    • Non-vulnerable interference edge, or
    • Vulnerable interference edge
• Based on looking at program code, to see what sorts of conflict situations can arise
• More complicated with programs whose accesses are controlled by parameters
• A close superset of SDG can be calculated automatically in some cases

Edges in the SDG

• Non-vulnerable interference edge from T1 to T2
  • Conflict, but it can’t arise
    transactions can run concurrently
    • Eg "ww" conflict
      • Concurrent execution prevented by FCW
    • Or "wr" conflict
      • conflict won’t happen in concurrent execution due to reading old version
  • Eg
    – T1 = R1(x) R1(y) W1(x)
    – T2 = R2(x) R2(y) W2(x) W2(y)
• Vulnerable interference edge from T1 to T2
  • Conflict can occur when transactions run concurrently
    • Eg "rw without ww": rset(T1) intersects wset(T2), and wset(T1) disjoint from wset(T2)
  • Eg
    – T1 = R1(x) R1(y) W1(x)
    – T2 = R2(x) R2(y) W2(y)
• Shown as dashed edge on diagram

Paired edges

• In SDG, an edge from X to Y implies an edge from Y to X
• But the type of edge is not necessarily the same
  – Both vulnerable, or
  – Both non-vulnerable, or
  – One vulnerable and one non-vulnerable

Dangerous Structures

• A dangerous structure is two edges linking three application programs, A, B, C such that
  – There are successive vulnerable edges (A,B) and (B,C)
  – (A, B, C) can be completed to a cycle in SDG
    • Call B a pivot
  – Special case: pair A, B with vulnerable edges in both directions

Path through zero or more edges from C to A

A ------- B ------- C

Pivot

Dangerous structure
The main result

- Theorem: If the SDG does not contain a dangerous cycle, then every execution is serializable (with all transactions using SI for concurrency control)

Example: SmallBank Benchmark

- Traditional benchmarks (e.g. TPC-C) are already serializable under SI
- SmallBank benchmark: designed to have non-serializable executions under SI
  - three tables: Account, Saving, Checking
  - five transactions of a banking scenario: Balance, WriteCheck, DepositChecking, TransactionSaving, Amalgamate

SmallBank Dependencies

- **Read-Dependencies(WR):**
  - **Balance(N):**
    - `SELECT bal FROM account WHERE custid=x;
    - `COMMIT`
  - **WriteCheck (N,V):**
    - `UPDATE Account SET bal=bal-V WHERE custid=x;
    - `COMMIT`

- **Write-Dependency(WW):**
  - **Balance(N):**
    - `SELECT bal FROM account WHERE custid=x;
    - `COMMIT`
  - **WriteCheck (N,V):**
    - `UPDATE Account SET bal=bal-V WHERE custid=x;
    - `COMMIT`

- **Anti-Dependencies(RW):**
  - **Balance(N):**
    - `SELECT bal FROM account WHERE custid=x;
    - `COMMIT`
  - **WriteCheck (N,V):**
    - `UPDATE Account SET bal=bal-V WHERE custid=x;
    - `COMMIT`

Analysis of SmallBank's SDG

What is the dangerous structure???
- nodes A, B, and C:
  - anti-dependency A → B
  - anti-dependency B → C
  - path from C to A or A=C
- In this case, only dangerous structure is Bal → WC → TS

SDG of SmallBank

Main theorem: Proof Sketch I (Find crucial feature in CSG)
- In any cycle in CSG, there exists
  - TA to TB have rw-dependency, and are concurrent
  - TB to TC have rw-dependency, and are concurrent
- Here TC is earliest committer among the cycle
- Case analysis relating types of dependency edge to ordering between start/commit times
Main theorem: Proof Sketch II (Relate CSG and SDG)
• If TA to TB is in CSG, then TA to TB is in SDG
• If edge in CSG has rw-dependency and transactions are concurrent, then edge in SDG is vulnerable

Main theorem: Proof Sketch III
• Assume existence of non-serializable execution
• So exists cycle in CSG
• So has special structure
  – TA to TB to TC, each being (rw and concurrent)
• So cycle in SDG with consecutive vulnerable edges
  – dangerous structure
• Contradiction, if SDG has no dangerous structure

Overview
1. Review of databases, isolation levels and serializability
2. Theory to determine whether an application will have serializable executions when running at SI
3. Modifying applications
4. Fixing the DBMS
5. Summary and Future work
6. Database and Middleware Research at USydney

Modifying application code
• DBA modifies one or more of the programs that make up the mix
• Modifications should not alter the observed semantics of any program
• Modified set of programs should have all executions serializable
  – So modified SDG has no dangerous structure

Decisions
• Decide WHERE: choose a set of edges containing at least one from each a dangerous structure
  – Finding a minimal set is NP-Hard
  – One easy choice: choose ALL vulnerable edges
• Decide HOW: introduce ww conflict on chosen edges
  – Without changing program semantics
  – Materialize or Promotion or External Locking
• Outcome: modified application mix has SDG where each chosen edge is not vulnerable
  – Modified application SDG has no dangerous structure

Approach 1: Materialize the Conflict
Both programs in the chosen edge get an extra update to a new table that is not used elsewhere in the application
• target row parameterized so FCW conflict happens exactly when trans have rw-dependency
School of Information Technologies

In Oracle, can use SELECT FOR UPDATE to get the FCW check as if this actually did a write. Doesn’t work in PostgreSQL, MS SQL Server 2005.

---

**Approach 2: Promote a Read**

Source program of chosen edge gets an extra update to the row which is in rw-dependency
- identity update: sets row to current value

In Oracle, can use SELECT FOR UPDATE to get the FCW check as if this actually did a write. Doesn’t work in PostgreSQL, MS SQL Server 2005.

**Approach 3: External Lock (ELM)**

Each transaction in the chosen edge is surrounded by explicitly lock/unlock on a suitable set of parameters.

---

**Why ELM is different from 2PL?**

- Transactions that are not involved in chosen edges do not set locks at all
- There are only exclusive locks, no shared locks
- Even if a transaction touches many objects, it may need to lock only one or a few string values
- All locking is done at the start of the transaction, before any database activity has occurred
- It can be implemented without risk of deadlock

---

**Performance impact**

- Does modification impact much on performance?
- For SmallBank, DBA could
  - Choose a minimal edge set which is just Bal – WT
  - Choose a minimal edge set WT – TS
  - Choose ALL vulnerable edges
- Each can be done by Materialize or Promotion or ELM
- This gives at least 9 options for DBA to modify application; which gives best performance?
- We take performance of SI as “target” (but we try to get this level of performance as well as serializability)

---

**Experiment Setup**

- Evaluating techniques on PostgreSQL 8.2 and a commercial platform offering SI
- Multi-threaded client executing SmallBank transactions using stored procedures
  - Each thread chooses one transaction type randomly
  - a ramp-up period 30 second followed by one minute measurement interval
- **Parameters:**
  Choice of SDG edges on which to introduce conflict, technique to introduce conflict, low & high contention scenarios (controlled by size of hotspot getting 90% of accesses)

---

**PostgreSQL-Low contention**

![Graph showing performance impact](image-url)
### Modifying applications: Lessons

- Choice of edge set really matters with promote or Materialize.
- Some choices can suffer substantial loss of performance compared to SI.
  - It is not wise to place write operations in a previously read-only txn.
- ELM gets good performance for all the various edge sets.
  - ELM can even get better performance than SI under contention, because locks on an edge also lead to blocking on self-loops of SDG, where ww-conflicts lead to frequent aborts with SI.

### Mixing isolation levels

- Theory usually assumes one cc mechanism for the dbms.
- But in fact different txns can use different mechanisms.
- Either declaratively, by setting "isolation level".
- Or programatically, by explicit LOCK TABLE and UNLOCK TABLE statements.
**Alternative: allocate isolation levels**

- Can we ensure serializable execution without modifying application code?
  - Just set isolation level for each transaction appropriately
  - In configuration, or at session establishment
- Potential advantage: don’t need to modify application source

**Extension of theory**

- Allocate some transactions to use 2PL and others to use SI
  - Eg on MS SQLServer 2005
- Theorem: If every pivot uses 2PL, then every execution is serializable (with other transactions using either 2PL or SI for concurrency control)
  - Minimal set of transactions to run with 2PL is the set of pivots (call this approach Pivot2PL)
  - Of course, using 2PL for ALL transactions guarantees serializable execution, this is a maximal set

**Mixing Isolation Levels; Low Contention**

**Compare to application modification**

**High Contention case**

**High Update Rates (few Bal transactions)**
Allocating Isolation Levels: Lessons

- Can lose quite a bit of SI's performance
- Generally, it would be better for the DBA to get the information needed and make a wise choice of how to modify application code
  - If they have permissions etc to do so

Overview

1. Review of databases, isolation levels and serializability
2. Theory to determine whether an application will have serializable executions when running at SI
3. Modifying applications
4. Fixing the DBMS
5. Summary and Future work
6. Database and Middleware Research at USydney

Serializable SI

- If we can alter the DBMS, we could provide a new algorithm for serializable isolation
  - Online, dynamic
  - Modifications to standard Snapshot Isolation
- To do so:
  - Keep versions, read from snapshot, FCW (like SI)
  - Detect read-write conflicts at runtime
  - Abort transactions with consecutive rw-edges
    - Much less often than traditional optimistic CC
    - Don't do full cycle detection

Challenges

- During runtime, rw-pairs can interleive arbitrarily
- Have to consider begin and commit timestamps:
  - which snapshot is a transaction reading?
  - can conflict with committed transactions
- Want to use existing engines as much as possible
- Low runtime overhead
- But minimize unnecessary aborts

SI anomalies: a simple case

Algorithm in a nutshell

- Add two flags to each transaction (in & out)
- Set T0.out if rw-conflict T0 \rightarrow T1
- Set T0.in if rw-conflictTN \rightarrow T0
- Abort T0 (the pivot) if both T0.in and T0.out are set
  - If T0 has already committed, abort the conflicting transaction
Detection: write before read

\[
\begin{array}{c}
\text{b1} \rightarrow w1(y) \rightarrow c1 \\
\text{b0} \rightarrow r0(y) \rightarrow c0
\end{array}
\]

read old y

T1.in = true

T0.out = true

Detection: read before write

\[
\begin{array}{c}
\text{lock x, SIREAD} \\
\text{bN} \rightarrow rN(y) \\
\text{b0} \rightarrow w0(x) \rightarrow c0
\end{array}
\]

write lock x

TN.out = true

T0.in = true

How can we detect this?

SIREAD mode lock doesn’t block anything

Just for record keeping

Kept even after transaction commits

Main Disadvantage: False positives

\[
\begin{array}{c}
\text{bN} \rightarrow rN(x) \rightarrow wN(y) \rightarrow cN \\
\text{b0} \rightarrow r0(y) \rightarrow w0(x)
\end{array}
\]

no cycle

unnecessary abort

Prototype in Oracle InnoDB

- Implemented in Oracle InnoDB plugin 1.0.1
  - Most popular transactional backend for MySQL
  - Already includes multiversion concurrency control
- Added:
  - True Snapshot Isolation with first-committer-wins (InnoDB’s “repeatable read” isolation has non-standard semantics)
  - Serializable SI, including phantom detection (uses InnoDB’s next-key locking)
- Added 230 lines of code to 130K lines in InnoDB
  - Most changes related to transaction lifecycle management

Experimental scenarios

- sibench – synthetic microbenchmark
  - conflict between sequential scan and updating a row
  - table size determines write-write conflict probability and CPU time required for scan
- TPC-C++ - modified TPC-C to introduce an SI anomaly
  - added a “credit check” transaction type to the mix
  - measured throughput under a variety of conditions
  - most not sensitive to choice of isolation level, but we found a mix favoring “stock level” transactions that demonstrates the tradeoff

sibench: 10 reads per write
Overview

1. Review of databases, isolation levels and serializability
2. Theory to determine whether an application will have serializable executions when running at SI
3. Modifying applications
4. Fixing the DBMS
5. Summary and Future work
6. Database and Middleware Research at USydney

Serializable SI: Lessons

- New algorithm for serializable isolation
  - Online, dynamic, and general solution
  - Modification to standard Snapshot Isolation
  - Keeps the features that make SI attractive:
    - Readers don’t block writers, much better scalability than 2PL
- In most cases, performance is comparable with SI
- Never worse than locking serializable isolation
- Feasible to add to an RDBMS using Snapshot Isolation (such as Oracle) with modest changes

Summary

- Can we change engine internals?
  - Yes: Fix cc
- Can we modify application design?
  - Yes: Allocate pivots to use 2PL
- Do 2PL for all transactions; Performance often very poor
- Find and analyse SDG; Choose modifications wisely; Good performance
- Find and analyse SDG; Allocate pivots to use 2PL; Performance often suffers
Future Work

- How does Read Committed fit in the story?
  - Consider programs some of which use RC (either traditional, with short-duration shared-locks, or snapshot-based), others use SI or 2PL
- What about lazy replication?
  - This has impact very like RC
  - Some existing work (Breitbart et al, late 90s)
- What about partitioned consistency models in cloud computing (shards)?
- Intended outcomes:
  - guidance for developers, so they can use what the platform provides, and know whether they risk anomalies
  - new platforms that are easier to use correctly without losing performance or scalability

Database & Middleware @ USydney

- Leaders
  - Uwe Röhm, PhD (ETH Zurich)
  - Alan Fekete, PhD (Harvard)
  - Sanjay Chawla, PhD (Tennessee)
  - David Levy, PhD (Natal)
- Collaborators from CSIRO
  - Paul Greenfield
  - Shiping Chen, PhD (UNSW)
- and 15-20 PhD and MPhil students

Major Publications Since 2006

- SIGMOD ‘06 and ’08 (including Best Paper in 2008)
- VLDB ’07
- ICDE ’08 and ’09
- CIDR ’07, ’09 and ’09
- ICDM ’06 and ’07
- KDD ’06
- SDM ’06 (Best Paper award) and ’07
- ICWS ’06, ’07 and ’07; WISE ’07
- Data Mining and Knowledge Discovery (2008)
- IEEE Transactions on Knowledge and Data Engineering (2008)
- IEEE/ACM Transactions on Bioinformatics (2007)

Collaborators from Microsoft Research, Microsoft SQL Server team, Sun Microsystems, IIT Bombay

Scientific Data Management

- How to efficiently manage scientific data with a DBMS?
  - Data design? Physical optimisations? Indexing?
  - Best way to bring data analysis close to data?
  - Efficient management of data provenance?
- Several Projects
  - dbBLAST
  - NextGen Gene Sequencing
  - GeneExpressionDB
- Collaborations with Microsoft Research & Sanger Institute
  - also direct contact to SQL Server Development Team
  - and CSIRO Sydney

Data Mining Topics

- Spatial and Temporal Data Mining and Outlier Detection
- Examples
  - Outlier Detection for processing insurance claims workers; all NSW workers compensation claims are now processed through a system designed by Chawla
  - Relationship between outliers in El Nino and rainfall in South America
  - The use of random projections for pattern discovery in data streams

Corona – Sun SPOT Distributed Query Processor

- Data Abstraction
  - virtual relation; horizontally partitioned
  - SQL-like queries
- Classic WSN Architecture
  - query engine on each node
  - sensing, filtering, communication
  - time synchronized, multi-tasking
  - multiple queries
- Routing tree connecting to base station
- control system on the host
  - query parsing and compilation
  - query dissemination
- GUI client

Collaborators from Microsoft Research, Microsoft SQL Server team, Sun Microsystems, IIT Bombay