The University of Sydney

Topics in Database Isolation
IITB, January 2006
Lecture 2: Safe Use of Low Isolation

Alan Fekete
(University of Sydney)
fekete@it.usyd.edu.au

Road Map

- Lecture 1: Isolation levels
- Lecture 2: Safe Use of Low Isolation
  - Safe use of Read Committed
  - Safe Use of Snapshot Isolation
  - Applying the theory to SQL applications
  - Application Modification
- Lecture 3: Replication Management

Safe use of Read Committed

- Application semantics
  - Old data is acceptable
- Data semantics
  - Write-once data
  - No inter-item integrity constraints

Recently changed data

- Many applications can work well with slightly old data
  - Data used mainly as "hint"
    - Eg choose shipper based on expected delay
  - Social processes fix problems
    - Eg mail forwarding with obsolete address
    - Eg merchant honors old prices

Write-once data

- Data that never changes after commit of the transaction that inserted the record
  - Audit trail
  - Immutable attributes
    - Eg manufacturer of a product
- Note that we don't need whole record to be constant, just those fields this particular application cares about

Unrelated items

- Application may deal with only one record
- Or it may deal with several records, but there are no integrity constraints between them
- Eg often each constraint concerns a single product, so an application that processes several products can safely release read locks on each product record before moving to the next one.

Safe Use of SI

- Given a collection of transactions
  - Decide which should use SI as their isolation mechanism, and which should use 2PL
- Goal: *all* executions must be serializable
- Difficulty: it's a non-local decision; the proper allocation of isolation level to one transaction may depend on which other transactions are present
- Contrast to use of read committed, based on semantics of one application or of the data

Reference

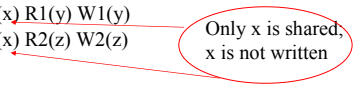
- A. Fekete "Allocating Isolation Levels to Transactions", Proc ACM PODS, 2005.

Interference Theory

- We produce the "interference graph" for the txns
- Draw directed edges each of which can be either
  - Protected interference edge, or
  - Exposed interference edge

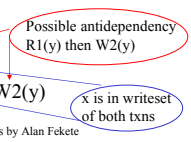
No interference edge

- No edge from T1 to T2, nor from T2 to T1
  - No conflicts: any shared item isn't updated by either
  - rset(T1) disjoint from wset(T2), and
  - wset(T1) disjoint from rset(T2), and
  - wset(T1) disjoint from wset(T2)
- Eg
  - P1 = R1(x) R1(y) W1(y)
  - P2 = R2(x) R2(z) W2(z)



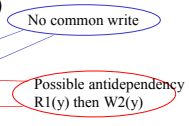
Protected interference edge

- Protected interference edge from T1 to T2
- Conflict exists, and "if rw then also ww"
- wset(T1) intersects wset(T2), or
- rset(T1) disjoint from wset(T2)
  - But wset(T1) intersects rset(T2) so there is a conflict
- Eg
  - T1 = R1(x) R1(y) W1(x)
  - T2 = R2(x) R2(y) W2(x) W2(y)



Exposed Interference Edge

- Exposed interference edge from T1 to T2
- "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

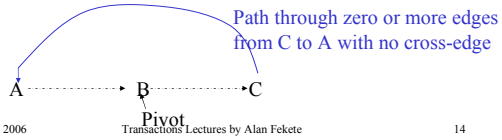


Paired edges

- In IG, an edge from X to Y implies an edge from Y to X
- But the type of edge is not necessarily the same
  - Both exposed, or
  - Both protected, or
  - One exposed and one protected

Interference Graph

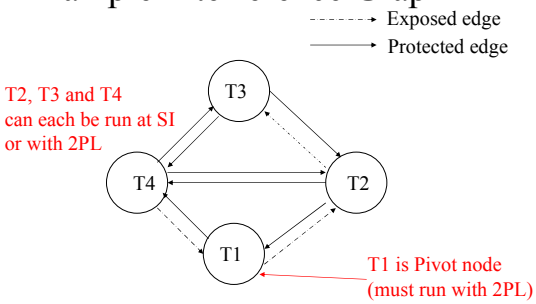
- A *pivot* node is one representing a txn B such that
  - There is an incoming exposed interference edge (A,B)
  - There is an outgoing exposed interference edge (B,C)
  - (A, B, C) can be completed to a chord-free cycle in IG
    - Eg A and B are pivots when there are reverse exposed edges (A to B and B to A)



The key result

- Theorem: Every execution is conflict-serializable, exactly when every *pivot* node P is allocated to use 2PL, and others are allocated to either SI or 2PL (independently and arbitrarily).
- NB: it is possible that *some* executions are serializable when a pivot uses SI

Example Interference Graph



Proof Structure

- Prove both directions
- A: if allocation assigns each pivot to use 2PL, then every execution is serializable
  - Done by contradiction: take arbitrary non-serializable execution, find pivot that uses SI
- B: if some pivot uses SI, then there exists an execution which is not conflict-serializable

Proof Sketch A,I
Find crucial feature in conflict graph

- In any cycle in conflict graph of an execution, there exists
  - T_A to T_B is rw-conflict, and not ww-conflict
  - T_B is using SI
  - T_B to T_C is rw-conflict, and not ww-conflict
- Here T_C is earliest committer among the cycle
- Case analysis relating types of conflict edge to ordering between start/commit times
- If this is true for every cycle, then it is true for a minimal cycle, which is chord-free

Proof Sketch A,II
Relate conflict graph and IG

- If T_A to T_B is in conflict graph of an execution, then T_A to T_B is in Interference Graph of the transaction set
- If edge in conflict graph is due to rw-conflict and not ww-conflict, then corresponding edge in IG is exposed
- If cycle in conflict graph is chord-free then cycle in IG is chord-free

Proof Sketch A,III

- Assume existence of non-serializable execution
- So exists cycle in conflict graph for that execution
- So exists chord-free cycle with special structure
  - T_A to T_B to T_C, each being (rw and not ww), T_B using SI
- So exists chord-free cycle in IG with special structure
  - T_B is pivot and uses SI
- Contradiction, if allocation has each pivot use 2PL

Proof Sketch B

- Take chord-free cycle with a pivot T_B using SI: T_B T_β … T_η T_B
- Construct execution
- Start(T_B), then all of T_β then… then all of T_η then operations of T_B
- The conflict graph of this execution is exactly the given cycle

Theory vs practice

- There is a mismatch between the model of serializability theory, and the structure of application programs
  - each application isn't well represented as a sequence of reads and writes on items
  - each application program is really code with SQL statements (which may mention program arguments) and complicated control flow
- How can we apply the theory?

Applying the theory to applications

- Sometimes it is straightforward
  - Each SQL statement deals with a fixed set of rows, based on primary key
  - Control is straightline
  - Construct the interference graph for all possible txns that arise from the programs
    - Eg one txn for each value of input parameters in the program

Set-oriented operations

- Extend serializabilty theory for set-oriented SQL
- Model a "predicate read" operation; it determines which items satisfy a where condition
  - Txn will then read or write these items
- Treat all items as existing forever, with special non-existent value before insertion, and special deleted value after deletion
- Predicate read conflicts with any write that changes the set of items satisfying the condition

Static analysis

- Sometimes it is too hard to work out all possible txns and their conflicts
- Instead, we can use a *condensed graph*, with one node per application program
- In the condensed graph, draw an exposed edge from P_A to P_B if there might be some parameter values x and y so that $P_A(x)$ and $P_B(y)$ have an exposed edge
- Draw a protected edge from P_A to P_B if there is not an exposed edge, but there might be some parameter values x and y so that $P_A(x)$ and $P_B(y)$ have a protected edge

Approximations

- There is a mapping from the interference graph of any collection transactions that arise from the programs, to the condensed graph of the programs
  - This maps exposed edge in IG to exposed edge in condensed graph
  - It maps protected edge in IG to either protected or exposed edge in condensed graph

Sufficient but not necessary

- A *potential pivot* node is one representing a program P such that
  - There is an incoming exposed interference edge (Q,P)
  - There is an outgoing exposed interference edge (P,R)
  - (Q, P, R) can be completed to a cycle in condensed graph

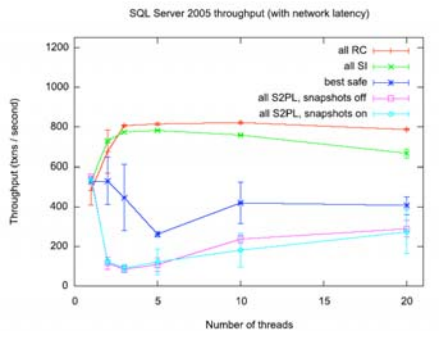
Not necessarily chord-free

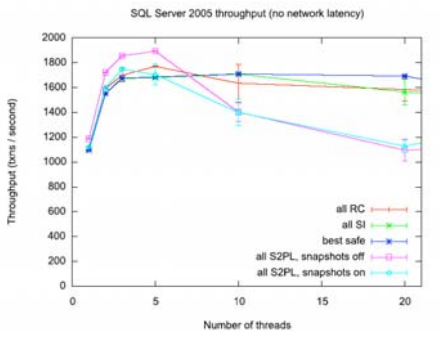
- Theorem: Every execution is conflict-serializable, provided every *potential pivot* node P is allocated to use 2PL, and others are allocated to either SI or 2PL (independently and arbitrarily).

NB. This is not "exactly when"

Impact

- Substantial performance gains are possible for a mixed allocation that guarantees serializability, compared to using 2PL for all txns
- In some circumstances, performance for a mixed, always-serializable allocation will approach that offered by using SI (or even Read Committed) for all txns (which may lead to integrity violation)
- Following slides show measurements on SQLServer 2005, from research of Michael Cahill (USyd)





Application Modification

- Declarative use of 2PL for pivots may not be easy or cheap
  - Eg on Oracle or PostgreSQL, 2PL can only be done by setting and holding explicit (table-level) locks
  - This might introduce lots of extra unnecessary blocking
- An alternative is to modify the application to change an exposed edge to protected
  - However, we should not change the application semantics!

Conflict introduction

- When there is an exposed edge from T1 to T2
  - i.e. there is rw on some item x, without any ww
- Introduce ww conflict
  - Aim to have extra conflict only between these transactions
- Extend T1 with an identity write (UPDATE T SET T.f=T.f WHERE…) on the item x
  - In Oracle, you can simply make a read "act like a write" by using "SELECT FOR UPDATE" instead of "SELECT"
- Or, invent a new data item eg CONFLICT(Id,Val)
  - Add "UPDATE CONFLICT Set Val = Val+1 WHERE Id = 10" to each of T1 and T2
  - Use a different Id for each edge you need to fix

Future Work

- Extend performance studies to know when to choose each of the different ways to ensure serializability
  - Also consider performance issues in choosing between semantic-preserving modifications of the txns
- Tool support to identify a (hopefully small) set of potential exposed edges
  - Look at case studies to find what sort of reasoning is needed for detecting common cases of protected edges or no edges
- Extension of theory to incorporate other isolation levels
  - Eg how to mix read-committed with SI, 2PL