CMU SCS

15-721 DB Sys. Design & Impl.

Locking and Consistency

Christos Faloutsos

www.cs.cmu.edu/~christos

CMU SCS

Roadmap

1) Roots: System R and Ingres
2) Implementation: buffering, indexing, q-opt
3) Transactions: locking, recovery
   - **locking & degrees of consistency**
   - optimistic C.C
   - B-trees and locking
   - ...
4) Distributed DBMSs
5) Parallel DBMSs: Gamma, Alphasort

CMU SCS

Paper

Granularity of locks and degrees of consistency in a shared data base

Gray, Lorie, Putzolu, Traiger

IFIP Working Conf. On Modelling of DBMS pp 1-29, 1997

CMU SCS

Detailed Roadmap

- Reminders
  - transactions / ACID properties
  - serializability; Locking; 2PL
- Multiple Granularity locks
- Degrees of consistency

CMU SCS

Reminders:

- (see undergrad book, eg., Silberschatz, Korth + Sudarshan)
- transaction - DFN?
- ACID properties
- serializability - DFN
- locking and 2PL
- (deadlocks)

CMU SCS

Transactions - dfn

= unit of work, eg.

move $10 from savings to checking

Atomicity (all or none) ← **recovery**
Consistency
Isolation (as if alone) ← **concurrency control**
Durability

Isolation

other transactions should not affect us

counter-example: lost update problem:

read(N) **1**	
	read(N) **1**
N = N - 1	
	N=N-1
write(N)	
	write(N)

Interleaved execution

time		
Read(X)		
	Read(X)	
X=X-10 Write(X) Read(Y) Y=Y+10 Write(Y)		**'correct'?**
	X = X * 1.1 Write(X) Read(Y) Y=Y*1.1 Write(Y)	

How to define correctness?

A: Serializability:

A schedule (=interleaving) is 'correct' if it is serializable,

ie., equivalent to a serial interleaving

(regardless of the exact nature of the updates)

examples and counter-examples:

'Lost update' case

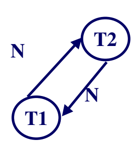
T1	T2
Read(N)	
	Read(N)
N=N-1	
	N= N-1
Write(N)	
	Write(N)

How to check for correctness?

Precedence graph

T1	T2
Read(N)	
	Read(N)
N=N-1	
	N= N-1
Write(N)	
	Write(N)

RW, WR, WW conflicts



Cycle -> not serializable

(counter) example: 'Inconsistent analysis'

T1	T2
Read(A) A=A-10 Write(A)	
	Read(A) Sum = A Read(B) Sum += B
Read(B) B=B+10 Write(B)	

Precedence graph?

CMU SCS

Locking

- Q: how to automatically create correct interleavings?
- A: locks to the rescue
  - lock(X); unlock(X)
  - exclusive/shared locks; compatibility matrix
  - locks are not enough:

CMU SCS

Locks are not enough

- (counter) examle?

CMU SCS

‘Inconsistent analysis’

time

T1	T2
Read(A) A=A-10 Write(A)	
	Read(A) Sum = A Read(B) Sum += B
Read(B) B=B+10 Write(B)	

Precedence graph?

CMU SCS

‘Inconsistent analysis’ – w/ locks

time

T1	T2
L(A)	
Read(A)	
...	
U(A)	
	L(A)
	....
	L(B)
	....

the problem remains!

Solution??

CMU SCS

General solution:

- Protocol(s)
- Most popular one: 2 Phase Locking (2PL)
- X-lock version: transactions issue no lock requests, after the first ‘unlock’

THEOREM: if all transactions obey 2PL -> all schedules are serializable (*)

* but deadlocks are possible

CMU SCS

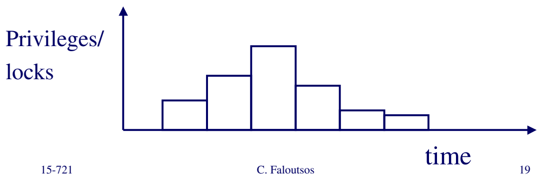
2PL – X/S lock version

Q: how to modify 2PL, for the shared/exclusive lock case?

2PL – X/S lock version

A: transactions issue no lock/upgrade request, after the first unlock/downgrade

In general: ‘growing’ and ‘shrinking’ phase



2PL – observations

- limits concurrency
- may lead to deadlocks (what to do, then?)
- 2PLC (keep locks until ‘commit’)

Q1: lock granularity?

Q2: how to trade-off correctness for concurrency?

Detailed Roadmap

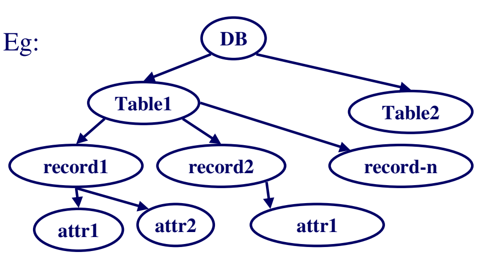
- Reminders
  - transactions / ACID properties
  - serializability; Locking; 2PL
- Multiple Granularity locks
- Degrees of consistency

Motivation

- lock granularity – field? record? page? table?
- Pros and cons?
- (Ideally, each transaction should obtain a few locks)

Multiple granularity

- Eg:



what types of locks?

- X/S locks for leaf level
- higher levels? X/S are too restrictive!
  - Why not go directly to the proper level?

what types of locks?

- X/S locks for leaf level +
- ‘intent’ locks, for higher levels
- IS: intent to obtain S-lock underneath
- IX: intent .... X-lock ...
- S: shared lock for this level
- X: ex- lock for this level
- (SIX: shared lock here; + IX)

Protocol

- each xact obtains appropriate lock at highest level
- proceeds to desirable lower levels
  - must have IS/IX lock on parent, for IS/S/IX lock on children
  - must have IX/SIX lock on parent, for IX/X/SIX on childre
- when done, unlock items, bottom-up

Compatibility matrix

T2 wants / T1 has	IS	IX	S	SIX	X
IS					
IX					
S					
SIX					
X					

Examples

- T1 wants to update “Smith”s record
  - IX on DB
  - IX on EMPLOYEE table
  - X on “Smith”s record

Examples - cont’d

- T2 wants to give 10% raise to everybody that is below average salary
  - IX on DB
  - SIX on EMPLOYEE
  - X on appropriate employee tuples
- OR:
  - IX on DB
  - X on EMPLOYEE

Consistency

DFN: “Dirty” data: updates of un-committed xacts

DFN: long locks: held until commit

Q: what is the impact of long/short S-locks, and long X-locks on correctness

CMU SCS

Consistency levels:

Degree 0: short write locks on updated items

Degree 1: long write locks on updated items

("long" means to hold until the transaction finishes)

Degree 2: long write locks on updated items, and short read locks on items read

Degree 3: long write locks on updated items, and long read locks on items read

CMU SCS

Consistency levels:

(no locks: ERRORS!)

Degree 0: short write locks on updated items

-> *we may update uncommitted data -> cascaded aborts*

CMU SCS

Consistency levels:

Degree 0: short write locks on updated items

Degree 1: long write locks on updated items

-> *we may read uncommitted data*

CMU SCS

Consistency levels:

Degree 0: short write locks on updated items

Degree 1: long write locks on updated items

Degree 2: long write locks on updated items, and short read locks on items read

-> *we read clean data, but repeated reads may give different results*

CMU SCS

Consistency levels:

Degree 0: short write locks on updated items

Degree 1: long write locks on updated items

Degree 2: long write locks on updated items, and short read locks on items read

Degree 3: long write locks on updated items, and long read locks on items read

-> *(= 2PLC): 'correct'*

CMU SCS

Consistency Levels

- Concurrency increases conversely with 'correctness'
- **Degree 3** is the default.

Conclusions

- (locks and 2PL for consistency)
- multiple granularity locks
- levels of consistency