15-721 Database Management Systems

Locking and Consistency

Anastassia Ailamaki
http://www.cs.cmu.edu/~natassa

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

Detailed Roadmap

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

Reminders:

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

Definitions

- **Database**
  - a fixed set of named resources (entities)
- **Consistency constraints**
  - must be true for DB to be considered consistent
  - **Example:**
    Σ(ACCT-BALS) = Σ(ASSETS)
    ACCT-BAL >= 0
- **Key point**

consistent database S1 → transaction T → consistent database S2

Transactions - definition

= unit of work, eg.
move $10 from savings to checking

Atomicity (all or none)
Consistency
Isolation (as if alone)
Durability

recovery → Atomicity, Durability
concurrency control → Consistency, Isolation

Problem statement

- Concurrent execution of independent transactions
  - utilization/throughput (“hide” waiting for I/Os.)
  - response time
  - fairness
- Isolation example (lost update):

	T1:	T2:
t0:	tmp1 := read(X)	
t1:		tmp2 := read(X)
t2:	tmp1 := tmp1 – 20	
t3:		tmp2 := tmp2 + 10
t4:	write (tmp1, X)	
t5:		write (tmp2, X)

Problem statement

- Arbitrary interleaving can lead to
  - Temporary inconsistency (ok, unavoidable)
  - “Permanent” inconsistency
- Need correctness criteria:
  - **schedule**: a particular action sequencing for a set of transactions
  - **consistent schedule**: each transaction sees consistent view of DB

Example: Interleaved execution

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)	

time ↓

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)

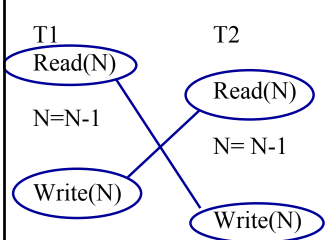
examples and counter-examples:

'Lost update' case

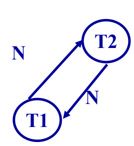
T1	T2
Read(N)	
	Read(N)
N=N-1	
	N= N-1
Write(N)	
	Write(N)

How to check for correctness?

Serialization graph



RW, WR, WW conflicts



Cycle -> not serializable

Serializability

Assumption: all serial schedules are consistent

- Dependencies:
  - T1 reads X, …, T2 writes X --- **RW**
  - T1 writes X, …, T2 reads X --- **WR**
  - T1 writes X, …, T2 writes X --- **WW**
- Serialization graph
  - Nodes are Transactions T1, T2, …
  - Edges: Ti → Tj if there is RW, WR, or WW from Ti to Tj

Theorem: schedule S serializable ⇔ SG(S) acyclic

- suggests (bad) technique for CC:
  build SG(S), topological sort, see if it works

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:

Locks are not enough

- (counter) example?

‘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?

‘Inconsistent analysis’ – w/ locks

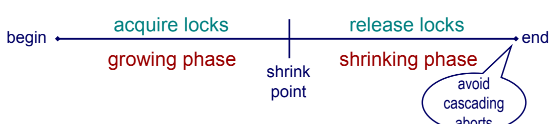
time	T1	T2
	L(A)	
	Read(A)	
	...	
	U(A)	
		L(A)
		....
		L(B)
		....

the problem remains!

Solution??

Solution: Locking+protocols

- **Well-formed Xact**: lock, action, unlock, lock…
  - Basic idea: **lock** <entity> / **unlock** <entity>
- **Two-phased Xact**: <lock> <actions> <unlock>



Theorem:

all Xacts well-formed and 2-phased ⇒ any S is serializable

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

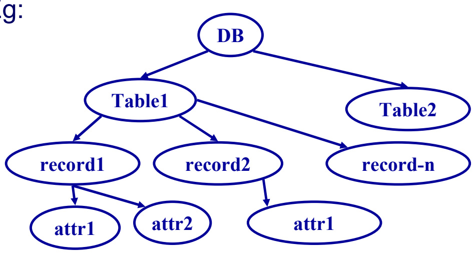
- Reminders
  - transactions / ACID properties
  - serializability; Locking; 2PL
- Multiple Granularity locks
- Degrees of consistency

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 children
- when done, unlock items, bottom-up

Compatibility matrix

T1 \ T2	IS	IX	S	SIX	X
IS					
IX					
S					
SIX					
X					

Compatibility matrix

T1 \ T2	IS	IX	S	SIX	X
IS	ok	ok	ok	ok	no
IX					
S					
SIX					
X					

Compatibility matrix

T1 \ T2	IS	IX	S	SIX	X
IS	ok	ok	ok	ok	no
IX	ok	ok	no	no	no
S					
SIX					
X					

Compatibility matrix

T1 \ T2	IS	IX	S	SIX	X
IS	ok	ok	ok	ok	no
IX	ok	ok	no	no	no
S	ok	no	ok	no	no
SIX					
X					

Compatibility matrix

T1 \ T2	IS	IX	S	SIX	X
IS	ok	ok	ok	ok	no
IX	ok	ok	no	no	no
S	ok	no	ok	no	no
SIX	ok	no	no	no	no
X					

Compatibility matrix

T1 \ T2	IS	IX	S	SIX	X
IS	ok	ok	ok	ok	no
IX	ok	ok	no	no	no
S	ok	no	ok	no	no
SIX	ok	no	no	no	no
X	no	no	no	no	no

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

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

Definition: long locks: held until commit

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

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

Consistency levels (0)

(no locks: ERRORS!)

Degree 0: short write locks on updated items

-> we may update uncommitted data -> cascaded aborts

Examples (0/1)

- Garbage reads

  T1: update(X); T2: update(X)

  - Who knows what value X will wind up holding?
  - Solution: set short write locks. **(→ degree 0)**

- Lost Updates

  T1: update(X);

  T2: update(X);

  T1: abort (restoring X to pre-T1 value)

  - At this point the update due to T2 is lost. (note: log contains (T1, X, [oldval, newval])
  - Solution: set long write locks. **(→ degree 1**)

Consistency levels (1)

Degree 0: short write locks on updated items

Degree 1: long write locks on updated items

-> we may read uncommitted data

Prevention of Inconsistency (1/2)

- Dirty Reads

  T1: update(X)

  T2: read(X)

  T1: abort

  - Now T2's read is bogus
  - Solution: long exclusive locks + short read locks **(→ degree 2)**
  - Systems often run long queries at level 2

Consistency levels (2)

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

Prevention of Inconsistency (2/3)

- Unreapeatable Reads

T1: update(X)
T1: complete transaction
T2: read(X)
T3: update(X)
T3: complete transaction
T2: read(X)

- Now T2 has read two different values for X
- Solution: long read locks. **(→ degree 3)**

2-phase well-formed → degree 3 consistent

Consistency levels (3)

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'

Consistency Levels

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

Conclusions

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