CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

Concurrency Control

(R&G ch. 17)

CMU SCS

Review

- DBMSs support ACID Transaction semantics.
- Concurrency control and Crash Recovery are key components

CMU SCS

Review

- For Isolation property, serial execution of transactions is safe but slow
  - Try to find schedules equivalent to serial execution
- One solution for "conflict serializable" schedules is Two Phase Locking (2PL)

CMU SCS

Outline

- Serializability - concepts and algorithms
- One solution: Locking
  - 2PL
  - variations
- Deadlocks

CMU SCS

Conflicting Operations

- We need a formal notion of equivalence that can be implemented efficiently…
  - Base it on the notion of "conflicting" operations
- <u>Definition</u>: Two operations conflict if:
  - They are by different transactions,
  - they are on the same object,
  - and at least one of them is a write.

CMU SCS

Conflict Serializable Schedules

- Definition: Two schedules are conflict equivalent iff:
  - They involve the same actions of the same transactions, and
  - every pair of conflicting actions is ordered the same way
- Definition: Schedule S is conflict serializable if:
  - S is conflict equivalent to some serial schedule.
- Note, some "serializable" schedules are NOT conflict serializable (see example #4', later)

Conflict Serializability – Intuition

- A schedule S is conflict serializable if:
  - You are able to transform S into a serial schedule by swapping **consecutive non-conflicting** operations of different transactions.
- *Example:*

R(A) W(A) R(B) W(B)

R(A) W(A) R(B) W(B)

≡

R(A) W(A) R(B) W(B)

R(A) W(A) R(B) W(B)

Conflict Serializability (Continued)

- Here's another example:

R(A) W(A)

R(A) W(A)

- Serializable or not????

Conflict Serializability (Continued)

- Here's another example:

R(A) W(A)

R(A) W(A)

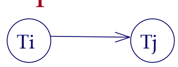
- Serializable or not????

NOT!

Serializability

- Q: any faster algorithm? (faster than transposing ops?)

Dependency Graph



- One node per Xact
- Edge from Ti to Tj if:
  - An operation Oi of Ti conflicts with an operation Oj of Tj and
  - Oi appears earlier in the schedule than Oj.

Dependency Graph

- Theorem: Schedule is conflict serializable if and only if its dependency graph is acyclic.

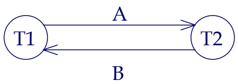
('dependency graph': a.k.a.'precedence graph')

CMU SCS

Example #1

- A schedule that is not conflict serializable:

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	



Dependency graph

- The cycle in the graph reveals the problem. The output of T1 depends on T2, and vice-versa.

CMU SCS

Example #2 (Lost update)

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

CMU SCS

Example #2 (Lost update)

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

R/W

CMU SCS

Example #2 (Lost update)

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

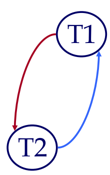
R/W

CMU SCS

Example #2 (Lost update)

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

R/W



CMU SCS

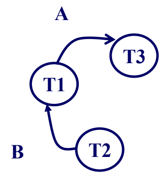
Example #3

T1	T2	T3
Read(A)		
…		
write(A)		
		Read(A)
		…
		Write(A)
	Read(B)	
	…	
	Write(B)	
Read(B)		
…		
Write(B)		

CMU SCS

Example #3

T1	T2	T3
Read(A)		
…		
write(A)		
		Read(A)
		…
		Write(A)
	Read(B)	
	…	
	Write(B)	
Read(B)		
…		
Write(B)		



equivalent serial execution?

CMU SCS

Example #3

A: T2, T1, T3

(Notice that T3 should go after T2, although it starts before it!)

Q: algo for generating serial execution from (acyclic) dependency graph?

CMU SCS

Example #3

A: T2, T1, T3

(Notice that T3 should go after T2, although it starts before it!)

Q: algo for generating serial execution from (acyclic) dependency graph?

A: Topological sorting

CMU SCS

Example #4 (Inconsistent Analysis)

T1	T2
R (A)	
A = A-10	
W (A)	
	R(A)
	Sum = A
	R (B)
	Sum += B
R(B)	
B = B+10	
W(B)	

dependency graph?

CMU SCS

Example #4 (Inconsistent Analysis)

T1	T2
R (A)	
A = A-10	
W (A)	
	R(A)
	Sum = A
	R (B)
	Sum += B
R(B)	
B = B+10	
W(B)	

create a 'correct' schedule that is not conflict-serializable

CMU SCS

Example #4' (Inconsistent Analysis)

T1	T2
R (A)	
A = A-10	
W (A)	
	R(A)
	if (A>0), count=1
	R (B)
	if (B>0), count++
R(B)	
B = B+10	
W(B)	

A: T2 asks for the count of my active accounts

CMU SCS

An Aside: View Serializability

- Alternative (weaker) notion of serializability.
- Schedules S1 and S2 are view equivalent if:
  1. If Ti reads initial value of A in S1, then Ti also reads initial value of A in S2
  2. If Ti reads value of A written by Tj in S1, then Ti also reads value of A written by Tj in S2
  3. If Ti writes final value of A in S1, then Ti also writes final value of A in S2

T1: R(A) W(A)
T2: W(A)
T3: W(A)

view ≡

T1: R(A),W(A)
T2: W(A)
T3: W(A)

CMU SCS

View Serializability

- Basically, allows all conflict serializable schedules + "blind writes"

T1: R(A) W(A)
T2: W(A)
T3: W(A)

view ≡

T1: R(A),W(A)
T2: W(A)
T3: W(A)

CMU SCS

View Serializability

- Basically, allows all conflict serializable schedules + "blind writes"

T1: R(A) W(A)
T2: W(A)
T3: W(A)

view ≡

T1: R(A),W(A)
T2: W(A)
T3: W(A)

A: 5 10 8 25

A: 5 8 10 25

CMU SCS

Notes on Serializability Definitions

- View Serializability allows (slightly) more schedules than Conflict Serializability does.
  - Problem is that it is difficult to enforce efficiently.
- Neither definition allows all schedules that you would consider "serializable".
  - This is because they don't understand the meanings of the operations or the data (recall example #4')

CMU SCS

Notes on Serializability Definitions

- In practice, **Conflict Serializability** is what gets used, because it can be enforced efficiently.
  - To allow more concurrency, some special cases do get handled separately, such as for travel reservations, etc.

CMU SCS

Outline

- Serializability - concepts and algorithms
- ➡ One solution: Locking
  - 2PL
  - variations
- Deadlocks

CMU SCS

Two-Phase Locking (2PL)

Lock Compatibility Matrix

	S	X
S	√	–
X	–	–

- Locking Protocol
  - 'S' (shared) and 'X' (eXclusive) locks
  - A transaction can not request additional locks once it releases any locks.
  - Thus, there is a "growing phase" followed by a "shrinking phase".

CMU SCS

2PL

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

CMU SCS

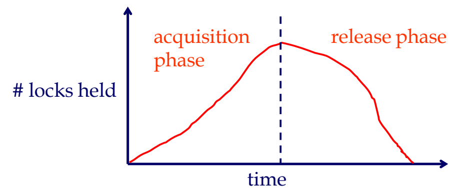
2PL

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

(if even one violates 2PL, non-serializability is possible -example?)

CMU SCS

Two-Phase Locking (2PL), cont.



- 2PL on its own is sufficient to guarantee conflict serializability (i.e., schedules whose precedence graph is acyclic), but, it is subject to **Cascading Aborts**.

CMU SCS

2PL

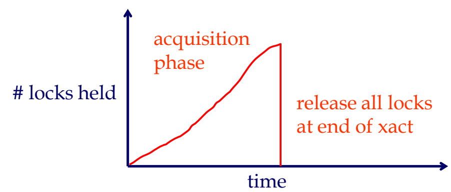
- Problem: Cascading Aborts
- Example: rollback of T1 requires rollback of T2!

T1:	R(A), W(A),		R(B), W(B), Abort
T2:		R(A), W(A)	

- Solution: Strict 2PL, i.e,
- keep all locks, until 'commit'

CMU SCS

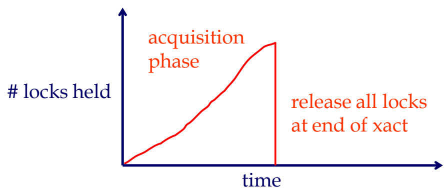
Strict 2PL



- Allows only conflict serializable schedules, but it is actually stronger than needed for that purpose.

CMU SCS

Strict 2PL (continued)



- In effect, "shrinking phase" is delayed until
  - Transaction commits (commit log record on disk), or
  - Aborts (then locks can be released after rollback).

Faloutsos SCS 15-415 37

CMU SCS

Next ...

- A few examples

Faloutsos SCS 15-415 38

CMU SCS

Non-2PL, A= 1000, B=2000, Output =?

Lock_X(A)	
Read(A)	Lock_S(A)
A: = A-50	
Write(A)	
Unlock(A)	
	Read(A)
	Unlock(A)
	Lock_S(B)
Lock_X(B)	
	Read(B)
	Unlock(B)
	PRINT(A+B)
Read(B)	
B := B +50	
Write(B)	
Unlock(B)	

CMU SCS

2PL, A= 1000, B=2000, Output =?

Lock_X(A)	
Read(A)	Lock_S(A)
A: = A-50	
Write(A)	
Lock_X(B)	
Unlock(A)	
	Read(A)
	Lock_S(B)
Read(B)	
B := B +50	
Write(B)	
Unlock(B)	Unlock(A)
	Read(B)
	Unlock(B)
	PRINT(A+B)

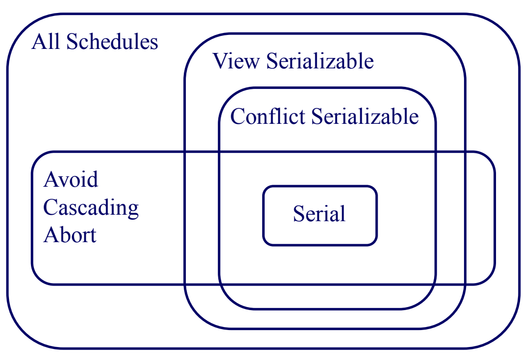
CMU SCS

Strict 2PL, A= 1000, B=2000, Output =?

Lock_X(A)	
Read(A)	Lock_S(A)
A: = A-50	
Write(A)	
Lock_X(B)	
Read(B)	
B := B +50	
Write(B)	
Unlock(A)	
Unlock(B)	
	Read(A)
	Lock_S(B)
	Read(B)
	PRINT(A+B)
	Unlock(A)
	Unlock(B)

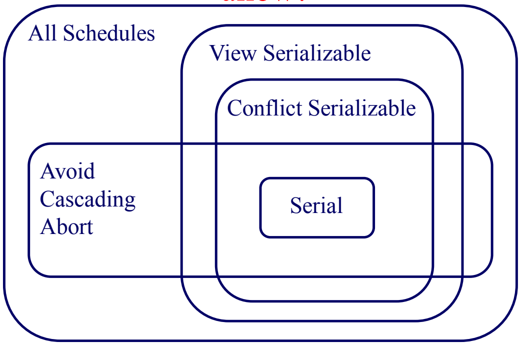
CMU SCS

Venn Diagram for Schedules



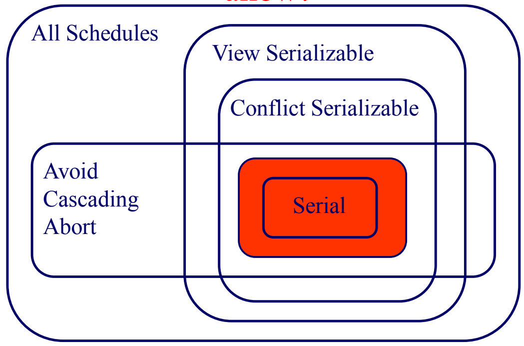
CMU SCS

Q: Which schedules does Strict 2PL allow?



CMU SCS

Q: Which schedules does Strict 2PL allow?



CMU SCS

Lock Management

- Lock and unlock requests handled by the Lock Manager (LM).
- LM contains an entry for each currently held lock.
- Q: structure of a lock table entry?

CMU SCS

Lock Management

- Lock and unlock requests handled by the Lock Manager (LM).
- LM contains an entry for each currently held lock.
- Lock table entry:
  - Ptr. to list of transactions currently holding the lock
  - Type of lock held (shared or exclusive)
  - Pointer to queue of lock requests

CMU SCS

Lock Management, cont.

- When lock request arrives see if any other xact holds a conflicting lock.
  - If not, create an entry and grant the lock
  - Else, put the requestor on the wait queue
- **Lock upgrade**: transaction that holds a shared lock can be upgraded to hold an exclusive lock

CMU SCS

Lock Management, cont.

- Two-phase locking is simple enough, right?
- We're not done. There's an important wrinkle …

CMU SCS

Example: Output = ?

Lock_X(A)	
	Lock_S(B)
	Read(B)
	Lock_S(A)
Read(A)	
A: = A-50	
Write(A)	
Lock_X(B)	

CMU SCS

Example: Output = ?

		lock mgr:
Lock_X(A)		grant
	Lock_S(B)	grant
	Read(B)	
	Lock_S(A)	wait
Read(A)		
A: = A-50		
Write(A)		
Lock_X(B)		wait

CMU SCS

Outline

- Serializability - concepts and algorithms
- One solution: Locking
  - 2PL
  - variations
- Deadlocks
  - detection
  - prevention

CMU SCS

Deadlocks

- **Deadlock**: Cycle of transactions waiting for locks to be released by each other.
- Two ways of dealing with deadlocks:
  - Deadlock prevention
  - Deadlock detection
- Many systems just punt and use Timeouts
  - What are the dangers with this approach?

CMU SCS

Deadlock Detection

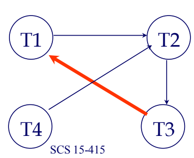
- Create a waits-for graph:
  - Nodes are transactions
  - Edge from Ti to Tj if Ti is waiting for Tj to release a lock
- Periodically check for cycles in waits-for graph

CMU SCS

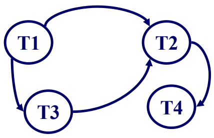
Deadlock Detection (Continued)

Example:

T1: S(A), S(D), S(B)
T2: X(B) X(C)
T3: S(D), S(C), X(A)
T4: X(B)

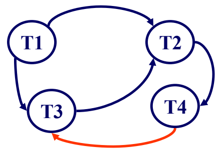


Another example



- is there a deadlock?
- if yes, which xacts are involved?

Another example

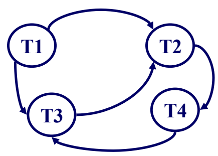


- now, is there a deadlock?
- if yes, which xacts are involved?

Deadlock detection

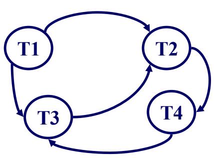
- how often should we run the algo?
- how many transactions are typically involved?

Deadlock handling



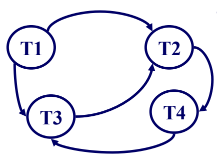
- Q: what to do?

Deadlock handling



- Q0: what to do?
  - A: select a 'victim' & 'rollback'
- Q1: which/how to choose?

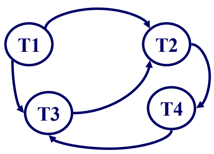
Deadlock handling



- Q1: which/how to choose?
  - A1.1: by age
  - A1.2: by progress
  - A1.3: by # items locked already...
  - A1.4: by # xacts to rollback
- Q2: How far to rollback?

CMU SCS

Deadlock handling

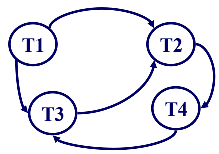


- Q2: How far to rollback?
  - A2.1: completely
  - A2.2: minimally
- Q3: Starvation??

Faloutsos SCS 15-415 61

CMU SCS

Deadlock handling



- Q3: Starvation??
- A3.1: include #rollbacks in victim selection criterion.

Faloutsos SCS 15-415 62

CMU SCS

Outline

- Serializability - concepts and algorithms
- One solution: Locking
  - 2PL
  - variations
- Deadlocks
  - detection
  - prevention

Faloutsos SCS 15-415 63

CMU SCS

Deadlock Prevention

- Assign priorities based on timestamps (older -> higher priority)
- We only allow 'old-wait-for-young'
- (or only allow 'young-wait-for-old')
- and rollback violators. Specifically:
- Say Ti wants a lock that Tj holds - two policies:

Wait-Die: If Ti has higher priority, Ti waits for Tj; otherwise Ti aborts (ie., old wait for young)

Wound-wait: If Ti has higher priority, Tj aborts; otherwise Ti waits (ie., young wait for old)

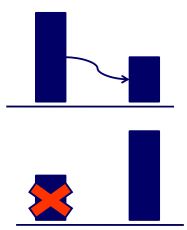
Faloutsos SCS 15-415 64

CMU SCS

Deadlock prevention

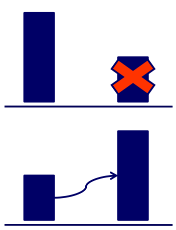
Wait-Die

Ti wants Tj has



Wound-Wait

Ti wants Tj has



Faloutsos SCS 15-415 65

CMU SCS

Deadlock Prevention

- Q: Why do these schemes guarantee no deadlocks?
- A:
- Q: When a transaction restarts, what is its (new) priority?
- A:

Faloutsos SCS 15-415 66

CMU SCS

Deadlock Prevention

- Q: Why do these schemes guarantee no deadlocks?
- A: only one ‘type’ of direction allowed.
- Q: When a transaction restarts, what is its (new) priority?
- A: its original timestamp. -- Why?

CMU SCS

SQL statement

- usually, conc. control is transparent to the user, but
- LOCK <table-name> [EXCLUSIVE|SHARED]

CMU SCS

Concurrency control - conclusions

- (conflict) serializability <-> correctness
- automatically correct interleavings:
  - locks + protocol (2PL, 2PLC, ...)
  - deadlock detection + handling
    - (or deadlock prevention)

CMU SCS

Quiz:

- is there a serial schedule (= interleaving) that is not serializable?
- is there a serializable schedule that is not serial?
- can 2PL produce a non-serializable schedule? (assume no deadlocks)

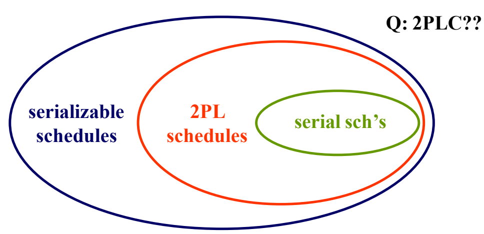
CMU SCS

Quiz - cont’d

- is there a serializable schedule that can not be produced by 2PL?
- a xact obeys 2PL - can it be involved in a non-serializable schedule?
- all xacts obey 2PL - can they end up in a deadlock?

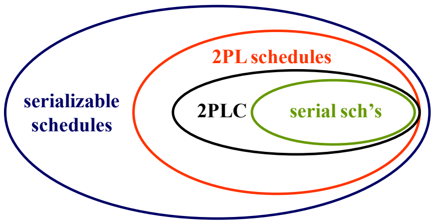
CMU SCS

Quiz - hints:



CMU SCS

Quiz - hints:



Faloutsos SCS 15-415 73