CIS 560 – Database System Concepts

Lecture 20

Concurrency Control

October 21, 2013

Credits for slides: Chang, Ullman, Whitehead.

Outline

Last:

- Undo logging 17.2
- Redo logging 17.3
- Redo/undo 17.4

Today:

- Serial and serializable schedules 18.1
- Conflict serializability 18.2

Next:

- Locks 18.3

2

Concurrency Control: The Problem

- Multiple concurrent transactions T_1, T_2, …
- They read/write common elements A_1, A_2, …
- How can we prevent unwanted interference ?

The SCHEDULER is responsible for that

3

Some Famous Anomalies

- Dirty reads
- Inconsistent reads
- Unrepeatable reads
- Lost updates

Many other things can go wrong too

4

Conflicts

- Write-Read – WR
- Read-Write – RW
- Write-Write – WW

5

Dirty Reads

Write-Read Conflict

T_1: WRITE(A)

T_2: READ(A)

T_1: ABORT

6

Inconsistent Read

Write-Read Conflict

T_1: A := 20; B := 20;
T_1: WRITE(A)

T_2: READ(A);
T_2: READ(B);

T_1: WRITE(B)

7

Unrepeatable Read

Read-Write Conflict

T_2: READ(A);

T_1: WRITE(A)

T_2: READ(A);

8

Lost Update

Write-Write Conflict

T_1	T_2
T_1: READ(A)	
	T_2: READ(A);
T_1: A := A+5	
	T_2: A := A*1.3
T_1: WRITE(A)	
	T_2: WRITE(A);

9

Schedules

- Given multiple transactions

A *schedule* is a sequence of interleaved actions from all transactions

10

Example

T1	T2
READ(A, t)	READ(A, s)
t := t+100	s := s*2
WRITE(A, t)	WRITE(A,s)
READ(B, t)	READ(B,s)
t := t+100	s := s*2
WRITE(B,t)	WRITE(B,s)

11

A Serial Schedule

T1	T2	A	B
READ(A, t)		25	25
t := t+100			
WRITE(A, t)		125	
READ(B, t)			
t := t+100			125
WRITE(B,t)			
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

Serial schedule: (T1, T2)

12

A Serial Schedule

T1	T2	A	B
	READ(A,s)	25	25
	s := s*2		
	WRITE(A,s)	50	
	READ(B,s)		
	s := s*2		50
	WRITE(B,s)		
READ(A, t)			
t := t+100			
WRITE(A, t)		150	
READ(B, t)			
t := t+100			
WRITE(B,t)			150

Serial schedule: (T2, T1)

13

Serializable Schedule

A schedule is ***serializable*** if it is equivalent to a serial schedule

A schedule S is serializable, if there is a serial schedule S', such that for every initial database state, the effects of S and S' are the same

14

A Serializable Schedule

T1	T2	A	B
READ(A, t)		25	25
t := t+100			
WRITE(A, t)		125	
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
READ(B, t)			
t := t+100			
WRITE(B,t)			125
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

This is NOT a serial schedule

15

A Non-Serializable Schedule

T1	T2	A	B
READ(A, t)		25	25
t := t+100			
WRITE(A, t)		125	
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		50
READ(B, t)			
t := t+100			
WRITE(B,t)			150

16

Transaction Semantics

T1	T2	A	B
READ(A, t)		25	25
t := t+100			
WRITE(A, t)		125	
	READ(A,s)		
	s := s+200		
	WRITE(A,s)	325	
	READ(B,s)		
	s := s+200		
	WRITE(B,s)		225
READ(B, t)			
t := t+100			
WRITE(B,t)			325

Is this serializable?

17

Ignoring Details

- Serializability is undecidable !
- Scheduler should not look at transaction details
- Assume worst case updates
  - Only care about reads r(A) and writes w(A)
  - Not the actual values involved

18

Notation

actions

T_1: $r_1(A); w_1(A); r_1(B); w_1(B)$
T_2: $r_2(A); w_2(A); r_2(B); w_2(B)$

transactions

schedule

$r_1(A); w_1(A); r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B)$

19

Conflict Serializability

Conflict: pair of consecutive actions in schedule s.t. if swapped, then behavior changes

Two actions by same transaction T_i: $r_i(X); w_i(Y)$

Two writes by T_i, T_j to same element $w_i(X); w_j(X)$

Read/write by T_i, T_j to same element $w_i(X); r_j(X)$ $r_i(X); w_j(X)$

20

Conflict Serializability

- A schedule is *conflict serializable* if it can be transformed into a serial schedule by a series of swappings of adjacent non-conflicting actions

Example:

$r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B)$

$\Downarrow$

$r_1(A); w_1(A); r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B)$

21

Example

$r_1(A); w_1(A); r_2(A); w_2(A); r_1(B); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_2(A); r_1(B); w_2(A); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_1(B); r_2(A); w_2(A); w_1(B); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_1(B); r_2(A); w_1(B); w_2(A); r_2(B); w_2(B)$

$r_1(A); w_1(A); r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B)$

22

The Precedence Graph Test

Is a schedule conflict-serializable?

Simple test:

- Build a graph of all transactions T_i
- Edge from T_i to T_j if T_i makes an action that conflicts with one of T_j and comes first
  - T_i must *precede* T_j iff an action from T_i conflicts with a later action from T_j
- The test: if the graph has no cycles, then it is conflict serializable!

23

Example 1

$r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B)$

$$1 \xrightarrow{B} 2 \xrightarrow{A} 3$$

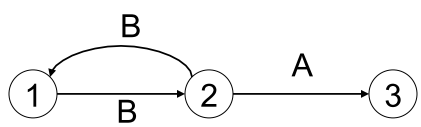
This schedule is conflict-serializable

Any transaction order which follows the precedences shown in the graph gives an equivalent serial schedule.

24

Example 2

$r_2(A); r_1(B); w_2(A); r_2(B); r_3(A); w_1(B); w_3(A); w_2(B)$



This schedule is NOT conflict-serializable

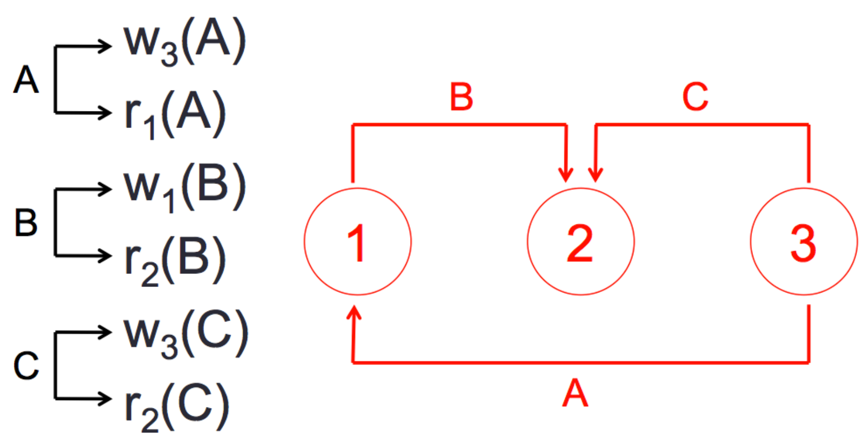
25

Exercise: construct precedence graph

$w_3(A)$
$r_1(A)$
$w_1(B)$
$r_2(B)$
$w_3(C)$
$r_2(C)$

26

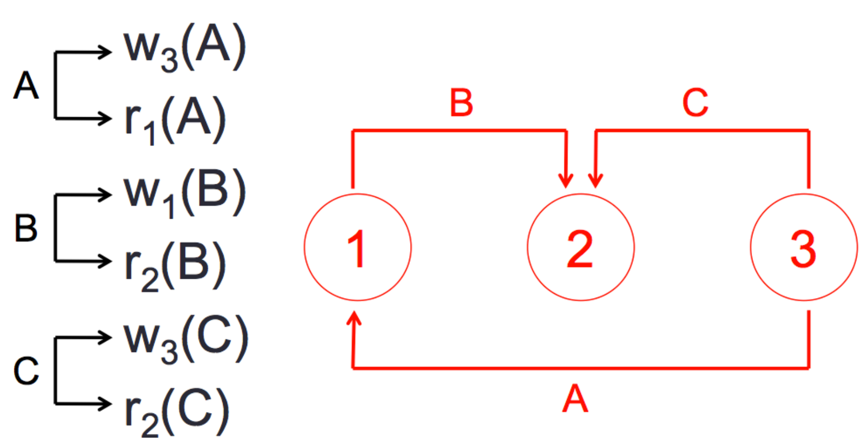
Exercise: construct precedence graph



Is conflict-serializable?

27

Exercise: construct precedence graph



T_3, T_1, T_2

28

Serial Equivalent

T_3, T_1, T_2

$w_3(A)$	$w_3(A)$
$r_1(A)$	$w_3(C)$
$w_1(B)$	$r_1(A)$
$r_2(B)$	$w_1(B)$
$w_3(C)$	$r_2(B)$
$r_2(C)$	$r_2(C)$

29

Home exercise

$r_1(A)$
$r_2(A)$
$r_1(B)$
$r_2(B)$
$r_3(A)$
$r_4(B)$
$w_1(A)$
$w_2(B)$

30

Conflict Serializability

- A serializable schedule need not be conflict serializable, even under the "worst case update" assumption

$w_1(X); w_2(X); w_2(Y); w_1(Y); w_3(Y);$

Lost writes

$w_1(X); w_1(Y); w_2(X); w_2(Y); w_3(Y);$

Equivalent, but can't swap

So, conflict serializability is a stronger condition.

Scheduler

- The scheduler is the module that schedules the transaction's actions, ensuring serializability.
- How? Three main techniques:
  - Locks
  - Timestamps
  - Validation

Locking Scheduler

Simple idea:

- Each element has a unique lock
- Each transaction must first acquire the lock before reading/writing that element
- If the lock is taken by another transaction, then wait
- The transaction must release the lock(s)

33

Notation

$L_i(A)$ = transaction T_i acquires lock for element A

$U_i(A)$ = transaction T_i releases lock for element A

34

Example

T1	T2
L_1(A); READ(A, t)	
t := t+100	
WRITE(A, t); U_1(A); L_1(B)	
	L_2(A); READ(A,s)
	s := s*2
	WRITE(A,s); U_2(A);
	L_2(B); **DENIED…**
READ(B, t)	
t := t+100	
WRITE(B,t); U_1(B);	
	…**GRANTED;** READ(B,s)
	s := s*2
	WRITE(B,s); U_2(B);

Is this a conflict-serializable schedule?

35

Example

T1	T2
L_1(A); READ(A, t)	
t := t+100	
WRITE(A, t); U_1(A); L_1(B)	
	L_2(A); READ(A,s)
	s := s*2
	WRITE(A,s); U_2(A);
	L_2(B); **DENIED…**
READ(B, t)	
t := t+100	
WRITE(B,t); U_1(B);	
	…**GRANTED;** READ(B,s)
	s := s*2
	WRITE(B,s); U_2(B);

Scheduler has ensured a conflict-serializable schedule

36

Example

T1	T2
$L_1(A)$; READ(A, t)	
t := t+100	
WRITE(A, t); $U_1(A)$;	
	$L_2(A)$; READ(A,s)
	s := s*2
	WRITE(A,s); $U_2(A)$;
	$L_2(B)$; READ(B,s)
	s := s*2
	WRITE(B,s); $U_2(B)$;
$L_1(B)$; READ(B, t)	
t := t+100	
WRITE(B,t); $U_1(B)$;	

Is this a conflict-serializable schedule?

37

Example

T1	T2
$L_1(A)$; READ(A, t)	
t := t+100	
WRITE(A, t); $U_1(A)$;	
	$L_2(A)$; READ(A,s)
	s := s*2
	WRITE(A,s); $U_2(A)$;
	$L_2(B)$; READ(B,s)
	s := s*2
	WRITE(B,s); $U_2(B)$;
$L_1(B)$; READ(B, t)	
t := t+100	
WRITE(B,t); $U_1(B)$;	

Locks did not enforce conflict-serializability!!!

38

Two Phase Locking (2PL)

The 2PL rule:

- In every transaction, all lock requests must precede all unlock requests
- This ensures conflict serializability! (why?)

39

Example: 2PL transactions

T1	T2
L_1(A); L_1(B); READ(A, t)	
t := t+100	
WRITE(A, t); U_1(A)	
	L_2(A); READ(A,s)
	s := s*2
	WRITE(A,s);
	L_2(B); **DENIED…**
READ(B, t)	
t := t+100	
WRITE(B,t); U_1(B);	
	…**GRANTED;** READ(B,s)
	s := s*2
	WRITE(B,s); U_2(A); U_2(B);

Now it is conflict-serializable

40