CSE 444: Database Internals

Section 5:

Transactions

Review in this section

- Serializability and conflict Serializability
  - Precedence graph
- Two-Phase Locking
  - Strict two phase locking
- Concurrency control by timestamp

Problem 1: Serializability and Locking

- Is this schedule conflict serializable?

What is
- Serializability
- Conflict Serializability?

T_0	T_1
$R_0(A)$	
$W_0(A)$	
	$R_1(A)$
	$R_1(B)$
	C_1
$R_0(B)$	
$W_0(B)$	
C_0	

Review: (Conflict) Serializable Schedule

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

- 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:

$$\boxed{r_1(A);\ w_1(A);\ r_2(A);\ \overbrace{w_2(A);\ r_1(B)};\ w_1(B);\ r_2(B);\ w_2(B)}$$

$$\boxed{r_1(A);\ w_1(A);\ r_1(B);\ w_1(B);\ r_2(A);\ w_2(A);\ r_2(B);\ w_2(B)}$$

Problem 1: Serializability and Locking

- Is this schedule conflict serializable?

T_0	T_1
$R_0(A)$	
$W_0(A)$	
	$R_1(A)$
	$R_1(B)$
	C_1
$R_0(B)$	
$W_0(B)$	
C_0	

- No.

- The **precedence graph** contains a cycle

$$\begin{tikzcd}
T_0 \arrow[r, bend right, "{R_1(B), W_0(B)}"'] & T_1 \arrow[l, bend right, "{W_0(A), R_1(A)}"']
\end{tikzcd}$$

•Why does precedence graph test work?

•Proof by induction (sec 18.2.3)

- Show how 2PL can ensure a conflict-serializable schedule
  - ❑ Original schedule below

What is
- Two Phase Locking
- Strict Two Phase Locking?

T_0	T_1
$R_0(A)$	
$W_0(A)$	
	$R_1(A)$
	$R_1(B)$
	C_1
$R_0(B)$	
$W_0(B)$	
C_0	

Review:
(Strict) Two Phase Locking (2PL)

The 2PL rule:

In every transaction, all lock requests must preceed all unlock requests

- Ensures conflict serializability
- Proof by induction (sec 18.3.4)

Strict 2PL:

All locks held by a transaction are released when the transaction is completed

- Ensures that schedules are <u>recoverable</u>
  - Transactions commit only after all transactions whose changes they read also commit
- Avoids <u>cascading rollbacks</u>

- Show how 2PL can ensure a conflict-serializable schedule
  - ❑ Original schedule below

T_0	T_1
$R_0(A)$	
$W_0(A)$	
	$R_1(A)$
	$R_1(B)$
	C_1
$R_0(B)$	
$W_0(B)$	
C_0	

T_0	T_1
$L_0(A)$	
$R_0(A)$	
$W_0(A)$	
	$L_1(A)$: Block
$L_0(B)$	
$R_0(B)$	
$W_0(B)$	
$U_0(A)$	
$U_0(B)$	
C_0	
	$L_1(A)$: Granted
	$R_1(A)$
	$L_1(B)$
	$R_1(B)$
	$U_1(A)$
	$U_1(B)$
	C_1

Is this strict 2PL?

No, replace C_0 by abort
-- Release locks after commit

- Show how the use of locks **without 2PL** can lead to a schedule that is NOT conflict-serializable
  - ❑ Original schedule below

T_0	T_1
$R_0(A)$	
$W_0(A)$	
	$R_1(A)$
	$R_1(B)$
	C_1
$R_0(B)$	
$W_0(B)$	
C_0	

T_0	T_1
$L_0(A)$	
$R_0(A)$	
$W_0(A)$	
$U_0(A)$	
	$L_1(A)$
	$R_1(A)$
	$U_1(A)$
	$L_1(B)$
	$R_1(B)$
	$U_1(B)$
	C_1
$L_0(B)$	
$R_0(B)$	
$W_0(B)$	
$U_0(B)$	
C_0	

Problem 2: Timestamp-based Concurrency Control

- Explain what happens when a time-stamp based concurrency control is used.
- ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

- Remember!
  - You need to mention any changes of RT, WT, Aand C bit of each element
  - Four rules in section 18.8.4
  - Four Possible actions: request is <u>accepted</u>, <u>ignored</u>, <u>delayed</u>, <u>rolledback/aborted</u>

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$						

C = 1 means C = true
C = 0 means C = false
(no space!)

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$					

1. Physically realizable: $TS(T_1) >= WT(X)$

2. C = 1: grant request

3. Update RT : $TS(T_1) > RT(X)$

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$					

1. Physically realizable:
$TS(T_2) >= WT(X)$

2. C = 1: grant request

3. Update RT : $TS(T_2) > RT(X)$

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C = 0		
$W_1(X)$						

1. Physically realizable:
$TS(T_2)$ >= $RT(X)$ and $TS(T_2)$ >= $WT(X)$

2. Update WT and C (not committed yet)

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$				

1. **NOT** Physically realizable:
$TS(T_1) < RT(X)$

Abort/rollback

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$					

1. Physically realizable:
$TS(T_3)$ >= $RT(X)$ and $TS(T_3)$ >= $WT(X)$

2. Update WT and C (not committed yet)

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$: **delay**					
		C_3				

1. Physically realizable:

$TS(T_3) >= RT(X)$ although $TS(T_2) < WT(X)$

2. We could not apply Thomas' write rule **(ignore $W_2(Y)$)** since C=0

$ST_1 \to ST_2 \to ST_3 \to ST_4 \to R_1(X) \to R_2(X) \to W_2(X) \to W_1(X) \to W_3(Y) \to W_2(Y) \to C_3 \to W_4(Z) \to C_4 \to R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$: **delay**					
		C_3			C=1	

What else?

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$: **delay**					
		C_3			C=1	
	Ignore $W_2(Y)$ and proceed					
			$W_4(Z)$			

A later write by T_3 has been committed

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **al**						
					WT=3, C=0	
					C=1	
	Ignore $W_2(Y)$ and **proceed**					
			$W_4(Z)$			WT=4, C = 0
			C_4			

1. Physically realizable:
$TS(T_4)$ >= $RT(X)$ and $TS(T_4)$ >= $WT(X)$

2. Update WT and C (not committed yet)

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 ->$R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
	$R_2(X)$			RT=2		
	$W_2(X)$			WT=2, C=0		
$W_1(X)$: **abort**						
		$W_3(Y)$			WT=3, C=0	
	$W_2(Y)$: **delay**					
		C_3			C=1	
	Ignore $W_2(Y)$ and **proceed**					
			$W_4(Z)$			WT=4, C = 0
			C_4			C=1
	$R_2(Z)$					

ST_1 -> ST_2 -> ST_3 -> ST_4 -> $R_1(X)$ -> $R_2(X)$ -> $W_2(X)$ -> $W_1(X)$ -> $W_3(Y)$ -> $W_2(Y)$ -> C_3 -> $W_4(Z)$ -> C_4 -> $R_2(Z)$

T1	T2	T3	T4	X	Y	Z
1	2	3	4	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1	RT = 0, WT = 0, C = 1
$R_1(X)$				RT=1		
				RT=2		
				WT=2, C=0		
$W_1(X)$: **a**						
					WT=3, C=0	
		C_3			C=1	
	Igr $V_2(Y)$ and **ceed**					
			$W_4(Z)$			WT=4, C = 0
			C_4			C=1
	$R_2(Z)$: **abort**					

1. **NOT** Physically realizable:
$TS(T_2) < WT(Z)$

Abort/rollback

Four Rules

- Rule 1: Read request on X by T
  - TS(T) < WT(X), **abort**, not physically realizable (read too late)
  - TS(T) >= WT(X), physically realizable
    - If C = 1, **accept**, update RT(X) if necessary
    - If C = 0, **delay** T

Note:

- If a request is not physically realizable, we abort
  - for read request, check WT
  - for write request, check RT
- If it is physically realizable
  - we accept, delay, or (only for write request) ignore

Four Rules

- Rule 2: Write request on X by T
  - TS(T) < RT(X), not physically realizable (write too late)
    - **abort**
  - TS(T) >= RT(X), physically realizable
    - TS(T) >= WT(X)
      - **accept**, update WT(X), set C = 0
    - TS(T) < WT(X)
      - If C = 1, **ignore**
      - If C = 0, **delay**

Four Rules

- Rule 3: Commit request by T
  - Set C = 1 for all X written by T
  - Allow waiting transactions to proceed

- Rule 4: Abort T
  - Check if the waiting transactions can proceed now.

You should try to understand the rules before applying them to solve problems ☺

More Timestamp-based Concurrency Control

What will happen at the last request?

- ST_1 -> ST_2 -> $R_1(A)$ -> $R_2(A)$ -> $W_1(B)$ -> **$W_2(B)$**
- ST_1 -> ST_2 -> $R_2(A)$ -> C_2 -> $R_1(A)$ -> **$W_1(A)$**
- ST_1 -> ST_2 -> ST_3-> $R_1(A)$ -> $W_3(A)$ -> C_3 -> **$W_2(A)$**
- ST_1 -> ST_2 -> ST_3-> $R_1(A)$ -> $W_1(A)$ -> **$R_2(A)$**

More Timestamp-based Concurrency Control

What will happen at the last request?

- ST_1 -> ST_2 -> $R_1(A)$ -> $R_2(A)$ -> $W_1(B)$ -> **$W_2(B)$**
  - **ACCEPTED** [no need to check C(B)]
- ST_1 -> ST_2 -> $R_2(A)$ -> C_2 -> $R_1(A)$ -> **$W_1(A)$**
  - **ROLLED BACK** [$R_2(A)$ precedes]
- ST_1 -> ST_2 -> ST_3-> $R_1(A)$ -> $W_3(A)$ -> C_3 -> **$W_2(A)$**
  - **IGNORED** [$W_3(A)$ committed]
- ST_1 -> ST_2 -> ST_3-> $R_1(A)$ -> $W_1(A)$ -> **$R_2(A)$**
  - **DELAYED** [$W_1(A)$ not committed yet]