LECTURE 08

CONCURRENCY CONTROL
(PART 2/3)

OUTLINES

18.3 Enforcing Serializability by Locks

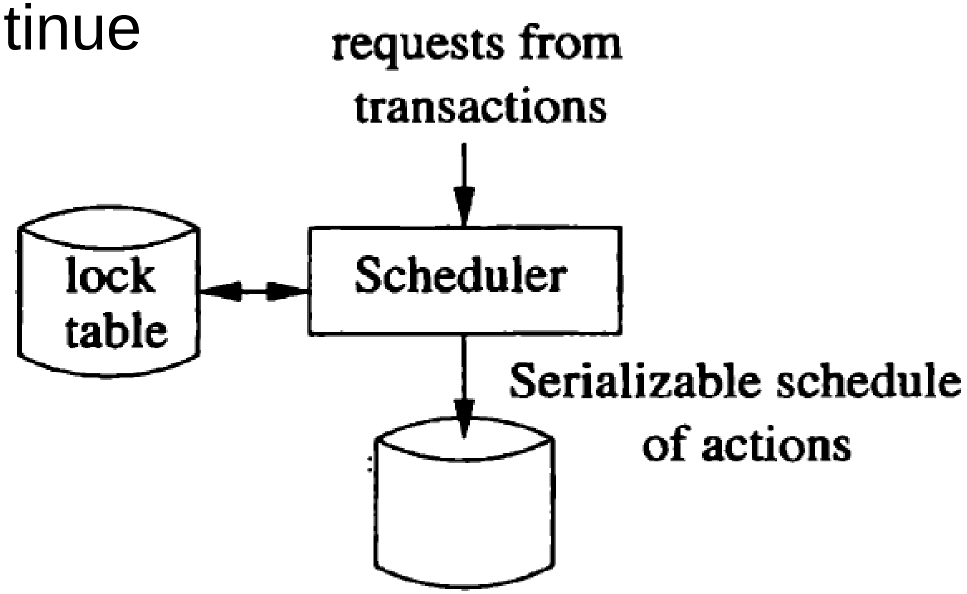
Introduction

- The most common architecture for a scheduler
  - “locks” are maintained on database elements to prevent unserializable behavior
- <u>In this section</u>
  - The concept of locking
  - One kind of lock
    - Which transactions must obtain on a database element if they want to perform any operation on that element

Locks

- The responsibility of the scheduler
  - Take requests from transactions
    - Either allow them to operate on the database
    - Or block the transaction until such time as it is safe to allow it to continue

A lock table will be used to guide this decision



Locking Scheduler

- A locking scheduler
  - – Enforces conflict-serializability
    - Which is more stringent condition than serializability
  - – Transactions must request and release locks
    - The structure of transactions
    - The structure of schedules

Locking Scheduler

- A locking scheduler
  - Consistency of Transactions:
    - Actions and locks must relate in the expected ways
      - A transaction can only read or write an element if it previously was granted a lock on that element and has not yet released the lock
      - If a transaction locks an element, it must later unlock that element
  - Legality of Schedules:
    - Locks must have their intended meaning:
      - No two transactions may have locked the same element without one having first released the lock

Notations

- Notations for locking and unlocking

  $l_i(X)$

  - Transaction T_i requests a lock on database element X

  $u_i(X)$

  - Transaction T_i releases (unlocks) its lock on database element X

Consistency Condition & Legality of Schedules

- Consistency Condition
  - Whenever a transaction T_i has an action $r_i(X)$ or $w_i(X)$, then there is a previous action $l_i(X)$ with no intervening action $u_i(X)$, and there is a subsequent $u_i(X)$
- Legality of Schedules
  - If there are actions $l_i(X)$ followed by $l_j(X)$ in a schedule, then somewhere between these actions there must be an action $u_i(X)$

Example

- Each of these transactions is consistent

T_1: $l_1(A)$; $r_1(A)$; `A := A+100;` $w_1(A)$; $u_1(A)$; $l_1(B)$; $r_1(B)$; `B := B+100;` $w_1(B)$; $u_1(B)$;

T_2: $l_2(A)$; $r_2(A)$; `A := A*2;` $w_2(A)$; $u_2(A)$; $l_2(B)$; $r_2(B)$; `B := B*2;` $w_2(B)$; $u_2(B)$;

Example

- A legal schedule of consistent transactions
- But it is not serializable

- The additional condition
- "Two-phase locking"
  - That we need to assure that legal schedules are conflict-serializable

T_1	T_2	A	B
		25	25
$l_1(A)$; $r_1(A)$;			
`A := A+100;`			
$w_1(A)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;		
	`A := A*2;`		
	$w_2(A)$; $u_2(A)$;	250	
	$l_2(B)$; $r_2(B)$;		
	`B := B*2;`		
	$w_2(B)$; $u_2(B)$;		50
$l_1(B)$; $r_1(B)$;			
`B := B+100;`			
$w_1(B)$; $u_1(B)$;			150

The Locking Scheduler

- The Locking Scheduler
  - It is the job of a scheduler based on locking to grant requests if and only if the request will result in a legal schedule
  - If a request is not granted, the requesting transaction is delayed
  - It waits until the scheduler grants its request at a later time

The Locking Scheduler

- Lock Table
  - The scheduler has a lock table that tells, for every database element, the transaction (if any) that currently holds a lock on that element
  - Only one kind of lock: simple case
    - A relation Locks (element, transaction)
      - Consisting of pairs (X, T) such that transaction T currently has a lock on database element X

Example

- Sometimes it is not possible to grant requests

T_1: $l_1(A)$; $r_1(A)$; `A := A+100;` $w_1(A)$; $l_1(B)$; $u_1(A)$; $r_1(B)$; `B := B+100;` $w_1(B)$; $u_1(B)$;

T_2: $l_2(A)$; $r_2(A)$; `A := A*2;` $w_2(A)$; $l_2(B)$; $u_2(A)$; $r_2(B)$; `B := B*2;` $w_2(B)$; $u_2(B)$;

T_1	T_2	A	B
		25	25
$l_1(A)$; $r_1(A)$;			
`A := A+100;`			
$w_1(A)$; $l_1(B)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;		
	`A := A*2;`		
	$w_2(A)$;	250	
	$l_2(B)$ **Denied**		
$r_1(B)$; `B := B+100;`			
$w_1(B)$; $u_1(B)$;			125
	$l_2(B)$; $u_2(A)$; $r_2(B)$;		
	`B := B*2;`		
	$w_2(B)$; $u_2(B)$;		250

Two-Phase Locking

- Two-Phase Locking (2PL)
  - Guarantee that a legal schedule of consistent transactions is conflict-serializable
- 2PL
  - In every transaction, all lock actions precede all unlock actions

Two-Phase Locking

- 2PL
  - In every transaction, all lock actions precede all unlock actions

- “Two-Phase”
  - The first phase: locks are obtained
  - The second phase: locks are relinquished

Two-Phase Locking

- 2PL
  - is a condition on the order of actions in a transaction
  - A transaction that obeys the 2PL condition is said to be a two-phase-locked transaction, or 2PL transaction

Example

- The transactions do not obey the two-phase locking rule

T_1	T_2	A	B
		25	25
$l_1(A)$; $r_1(A)$;			
A := A+100;			
$w_1(A)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;		
	A := A*2;		
	$w_2(A)$; $u_2(A)$;	250	
	$l_2(B)$; $r_2(B)$;		
	B := B*2;		
	$w_2(B)$; $u_2(B)$;		50
$l_1(B)$; $r_1(B)$;			
B := B+100;			
$w_1(B)$; $u_1(B)$;			150

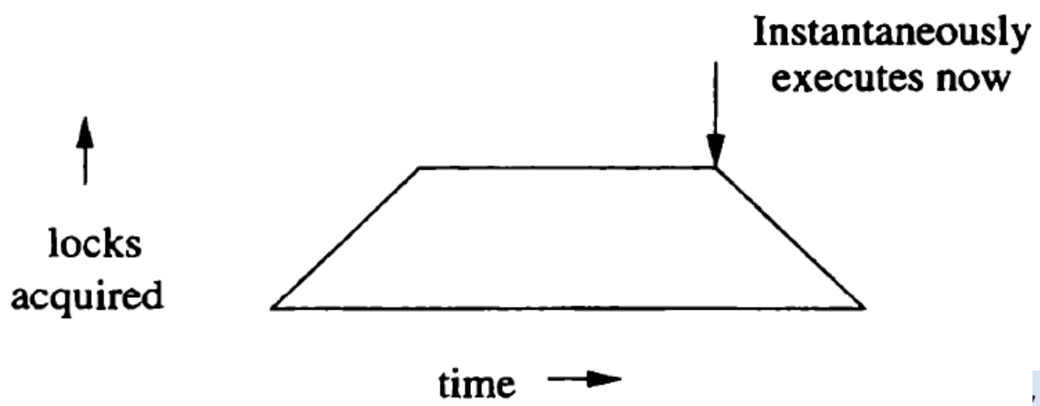
Example

- The transactions *do* obey the 2PL condition

T_1	T_2	A	B
		25	25
$l_1(A)$; $r_1(A)$;			
`A := A+100;`			
$w_1(A)$; $l_1(B)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;		
	`A := A*2;`		
	$w_2(A)$;	250	
	$l_2(B)$ **Denied**		
$r_1(B)$; `B := B+100;`			
$w_1(B)$; $u_1(B)$;			125
	$l_2(B)$; $u_2(A)$; $r_2(B)$;		
	`B := B*2;`		
	$w_2(B)$; $u_2(B)$;		250

Why Two-Phase Locking Works

- Intuitively
  - Each two-phase-locked transaction may be thought to execute in its entirety at the instant it issues its first unlock request
  - There is always at least one conflict-equivalent serial schedule for a schedule *S* of 2PL transactions
    - The one in which the transaction appear in the same order as their first unlocks



Why Two-Phase Locking Works

- How to convert any legal schedule *S* of consistent, two-phase-locked transactions to a conflict-equivalent serial schedule
  - An induction on the number of transactions in *S*
  - Issue of conflict-equivalence refers to the read and write actions only
    - As we swap the order of reads and writes, we ignore the lock and unlock actions

Why Two-Phase Locking Works

- An induction on the number of transactions in S
  - BASIS:
    - If $n = 1$, there is nothing to do; S is already a serial schedule

Why Two-Phase Locking Works

- An induction on the number of transactions in S
  - INDUCTION:
    - Suppose S involves n transactions T_1, T_2, …, T_n, and let T_i be the transaction with the first unlock action in the entire schedule S, say $u_i(X)$
    - We claim it is possible to move all the read and write actions of T_i forward to the beginning of the schedule without passing any conflicting reads and writes

Why Two-Phase Locking Works

- An induction on the number of transactions in S
  - INDUCTION:
    - Consider some action of T_i, say $w_i(Y)$.
    - Could it be preceded in S by some conflicting action, say $w_j(Y)$?
    - If so, then in schedule S, actions $u_j(Y)$ and $l_i(Y)$ must intervene, in a sequence of actions

$$\ldots\ w_j(Y);\ \ldots\ ;\ u_j(Y);\ \ldots\ ;\ l_i(Y);\ \ldots\ ;\ w_i(Y);\ \ldots$$

Why Two-Phase Locking Works

– INDUCTION:

- If so, then in schedule S, actions $u_j(Y)$ and $l_i(Y)$ must intervene, in a sequence of actions

… $w_j(Y)$; … ; $u_j(Y)$; … ; $l_i(Y)$; … ; $w_i(Y)$; …

- Since T_i is the first to unlock, $u_i(X)$ precedes $u_j(Y)$ in S; that is, S might look like:

… $w_j(Y)$; … ; $u_i(X)$; … ; $u_j(Y)$; … ; $l_i(Y)$; … ; $w_i(Y)$; …

or $u_i(X)$ could even appear before $w_j(Y)$.

In any case, $u_i(X)$ appears before $l_i(Y)$,

which means that T_i is *not* two-phase-locked, as we assumed

Why Two-Phase Locking Works

– INDUCTION:

- While we have only argued the nonexistence of conflicting pairs of writes, the same argument applies to any pair of potentially conflicting actions, one from T_i and the other from another T_j
- We conclude that it is indeed possible to move all the actions of T_i forward to the beginning of S, using swaps of nonconflicting read and write actions, followed by restoration of the lock and unlock actions of T_i. That is, S can be written in the form

(Actions of T_i) (Actions of the other n-1 transactions)

Why Two-Phase Locking Works

– INDUCTION:

(Actions of T_i) (Actions of the other n-1 transactions)

- The tail of n-1 transactions is still a legal schedule of consistent, 2PL transactions,
- so the inductive hypothesis applies to it.
- We convert the tail to a conflict-equivalent serial schedule, and now all of S has been shown conflict-serializable

A Risk of Deadlock

- The potential for deadlocks
  - that is not solved by two-phase locking
  - where several transactions are forced by the scheduler to wait forever for a lock held by another transaction

A Risk of Deadlock

- Example

T_1	T_2	A	B
		25	25
$l_1(A); r_1(A);$			
	$l_2(B); r_2(B);$		
A := A+100;			
	B := B*2;		
$w_1(A);$		125	
	$w_2(B);$		50
$l_1(B)$ **Denied**	$l_2(A)$ **Denied**		