CSE 444: Database Internals

Lectures 15
Transactions:
Optimistic Concurrency Control

Motivation

- Locking ensures serializability
  - BUT adds overhead and slows-down transactions
- Observations:
  - Many transactions are read-only
  - Many transactions touch unrelated parts of database
- Can we:
  - Assume no unserializable behavior will occur
  - Abort transactions in case of violations
- Yes: this is *optimistic concurrency* control

Locking vs Optimistic

- Locking prevents unserializable behavior from occurring: it causes transactions to wait for locks
- Optimistic methods assume no unserializable behavior will occur: they abort transactions if it does
- Locking typically better in case of high levels of contention; optimistic better otherwise

Outline

- Concurrency control by timestamps (18.8)
- Concurrency control by validation (18.9)

Timestamps

- Each transaction receives unique timestamp TS(T)

Could be:

- The system's clock
- A unique counter, incremented by the scheduler

Timestamps

Main invariant:

The timestamp order defines
the serialization order of the transaction

Main Idea

- For any two conflicting actions, ensure that their order is the serialized order:

In each of these cases

- $w_U(X) \ldots r_T(X)$ — Read too late ?
- $r_U(X) \ldots w_T(X)$ — Write too late ?
- $w_U(X) \ldots w_T(X)$ — Write too late ?

Answer: Check that TS(U) < TS(T)

When T wants to read X, $r_T(X)$, how do we know U, and TS(U) ?

Timestamps

With each element X, associate

- RT(X) = the highest timestamp of any transaction that read X
- WT(X) = the highest timestamp of any transaction that wrote X
- C(X) = the commit bit: true when transaction with highest timestamp that wrote X committed

If 1 element = 1 page,
these are associated with each page X in the buffer pool

Time-based Scheduling

- Note: simple version that ignores the commit bit

- Transaction wants to read element X
  - If TS(T) < WT(X) abort
  - Else read and update RT(X) to larger of TS(T) or RT(X)

- Transaction wants to write element X
  - If TS(T) < RT(X) abort
  - Else if TS(T) < WT(X) ignore write & continue (Thomas Write Rule)
  - Otherwise, write X and update WT(X) to TS(T)

Details

Read too late:

- T wants to read X, and TS(T) < WT(X)

START(T) … START(U) … $w_U(X) \ldots r_T(X)$

Need to rollback T !

Details

Write too late:

- T wants to write X, and TS(T) < RT(X)

START(T) … START(U) … $r_U(X) \ldots w_T(X)$

Need to rollback T !

Details

Write too late, but we can still handle it:

- T wants to write X, and
  TS(T) >= RT(X) but WT(X) > TS(T)

START(T) … START(V) … $w_V(X) \ldots w_T(X)$

Don't write X at all !
(but see later…)

More Problems

Read dirty data:

- T wants to read X, and WT(X) < TS(T)
- Seems OK, but…

START(U) … START(T) … $w_U(X)$. . . $r_T(X)$… ABORT(U)

If C(X)=false, T needs to wait for it to become true

More Problems

Write dirty data:

- T wants to write X, and WT(X) > TS(T)
- Seems OK not to write at all, but …

START(T) … START(U)… $w_U(X)$. . . $w_T(X)$… ABORT(U)

If C(X)=false, T needs to wait for it to become true

Timestamp-based Scheduling

- When a transaction T requests r(X) or w(X), the scheduler examines RT(X), WT(X), C(X), and decides one of:

- To grant the request, or
- To rollback T (and restart with later timestamp)
- To delay T until C(X) = true

Timestamp-based Scheduling

RULES including commit bit

- There are 4 long rules in Sec. 18.8.4
- You should be able to derive them yourself, based on the previous slides
- Make sure you understand them !

READING ASSIGNMENT: 18.8.4

Multiversion Timestamp

- When transaction T requests r(X) but WT(X) > TS(T), then T must rollback

- Idea: keep multiple versions of X: X_t, X_{t-1}, X_{t-2}, . . .

  $TS(X_t) > TS(X_{t-1}) > TS(X_{t-2}) > \ldots$

- Let T read an older version, with appropriate timestamp

Details

- When $w_T(X)$ occurs, create a new version, denoted X_t where t = TS(T)

- When $r_T(X)$ occurs, find most recent version X_t such that t < TS(T)
  Notes:
  - $WT(X_t)$ = t and it never changes
  - $RT(X_t)$ must still be maintained to check legality of writes

- Can delete X_t if we have a later version X_{t1} and all active transactions T have TS(T) > t1

Example using Basic Timestamps

T_1	T_2	T_3	T_4	A
150	200	175	225	RT=0 WT=0
$R_1(A)$				RT=150
$W_1(A)$				WT=150
	$R_2(A)$			RT=200
	$W_2(A)$			WT=200
		$R_3(A)$		
		Abort		
			$R_4(A)$	RT=225

Example using Multiversion

T_1	T_2	T_3	T_4	A_0	A_{150}	A_{200}
150	200	175	225			
$R_1(A)$				Read		
$W_1(A)$					Create	
	$R_2(A)$				Read	
	$W_2(A)$					Create
		$R_3(A)$			Read	
			$R_4(A)$			Read

Tradeoffs

- Locks:
  - Great when there are many conflicts
  - Poor when there are few conflicts
- Timestamps
  - Poor when there are many conflicts (rollbacks)
  - Great when there are few conflicts
- Compromise
  - READ ONLY transactions → timestamps
  - READ/WRITE transactions → locks

Outline

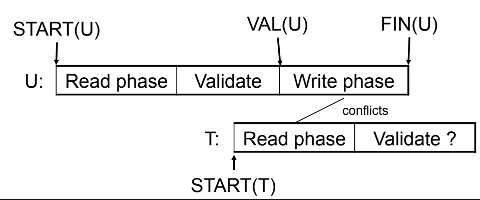
- Concurrency control by timestamps (18.8)
- Concurrency control by validation (18.9)

Concurrency Control by Validation

- Each transaction T defines a *read set* RS(T) and a *write set* WS(T)
- Each transaction proceeds in three phases:
  - Read all elements in RS(T). Time = START(T)
  - Validate (may need to rollback). Time = VAL(T)
  - Write all elements in WS(T). Time = FIN(T)

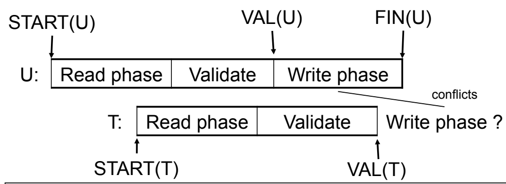
Main invariant: the serialization order is VAL(T)

Avoid $r_T(X)$ - $w_U(X)$ Conflicts



IF RS(T) ∩ WS(U) and FIN(U) > START(T)
(U has validated and U has not finished before T begun)
Then ROLLBACK(T)

Avoid $w_T(X)$ - $w_U(X)$ Conflicts



IF WS(T) ∩ WS(U) and FIN(U) > VAL(T)
(U has validated and U has not finished before T validates)
Then ROLLBACK(T)

Validation Rules Summary

- Check that RS(T) ∩ WS(U) is empty for any previously validated U that did not finish before T started (i.e., FIN(U) > START(T))
- Check that WS(T) ∩ WS(U) is empty for any previously validated U that did not finish before T validated (i.e, FIN(U) > VAL(T))