CS411
Database Systems

16: Final Review Session

Kazuhiro Minami

Concurrency Control

Concurrency Control – Basic concepts

- What is a transaction?
- Which actions do we consider in a transaction?
- How to represent a transaction?
- What is a schedule?
- What is the goal of concurrency control?
- What is a serial schedule?
- What is a serializable schedule?
- What is a conflict-serializable schedule?
- What are conflicting swaps?
- How to determine whether a schedule is conflict-serializable?

Basic Concepts on Locks

- What is a lock?
- What is a lock table? What kind of information is stored there?
- What is the *consistency* of transactions?
- What is the *legality* of transactions?
- What is the notations for actions of locking and unlocking?
- What is the job of the locking scheduler?
- What is the two-phase locking (2PL) condition? What type of serializable schedules are produced with this approach?

Two-phase locked schedule

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

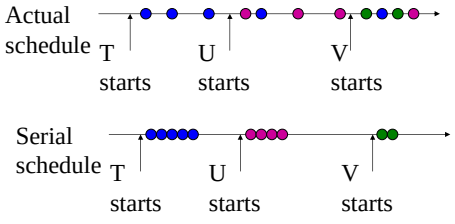
1. Make T_1 and T_2 consistent, but not 2PL by adding lock and unlock actions
2. Give a legal, but not serializable schedule of T_1 and T_2 in question 1
3. Make T_1 and T_2 consistent and 2PL by adding lock and unlock actions
4. Give a legal schedule of T_1 and T_2 in question 3

Concepts on Timestamp-based Concurrency Control

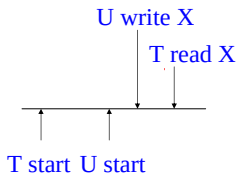
- What is a timestamp?
- When is a timestamp assigned to a transaction?
- What's the notation for transaction T's timestamp?
- What information do we maintain for each database element X?
- What type of serializable schedules are produced with this approach?
- How the timestamp-based approach solve the problem of "dirty" read?

Assumed Serial Schedule

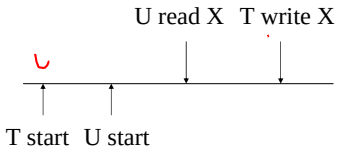
- Conflict serializable schedule that is equivalent to a serial schedule in which the timestamp order of transactions is the order to execute them



How to detect T's reading X too late?

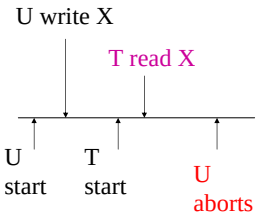


How can you detect T's writing to late?



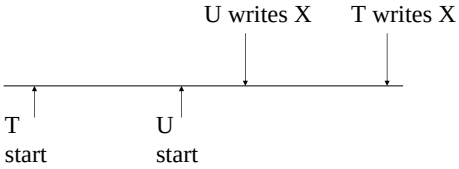
Prevention of Dirty Read

- How to prevent transaction T from reading data written by uncommitted transaction U?
- We want T to wait until U commits



Thomas Write Rule

- Why can we skip transaction T's write on element X, which is already modified by transaction U?
- What if there is another transaction V starting after T but before U?
- What if V starts after U?

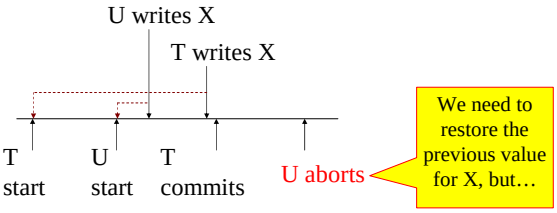


Another Problem with Dirty Data

- Thomas write rule: T's write can be skipped if

$$TS(T) < WT(X)$$

- But, we want T to wait until U commits



Concurrency Control by Timestamps

- Tell me what happens as each executes
  - st_1; $r_1(A)$; st_2; $w_2(B)$; $r_2(A)$; $w_1(B)$

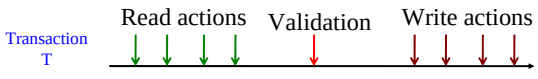
T1	T2	A	B
100	200	RT=0 WT=0	RT=0 WT=0
$r_1(A)$		RT=100	
	$w_2(B)$		WT=200
	$r_2(A)$	RT=200	
$w_1(B)$			

OK, but needs to wait until T2 commits

Concurrency Control by Validation

Concurrency Control by Validation

- Another type of optimistic concurrency control
- Maintain a record of what active transactions are doing
- Just before a transaction starts to write, it goes through a "validation phase"
- If a there is a risk of physically unrealizable behavior, the transaction is rolled back

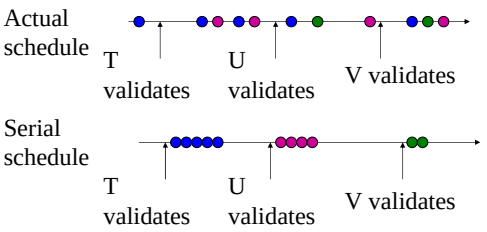


Validation-based Scheduler

- Keep track of each transaction T's
  - Read set RS(T): the set of elements T read
  - Write set WS(T): the set of elements T write
- Execute transactions in three phases:
  1. *Read*. T reads all the elements in RS(T)
  2. *Validate*. Validate T by comparing its RS(T) an WS(T) with those in other transactions. If the validation fails, T is rolled back
  3. *Write*. T writes its values for the elements in WS(T)

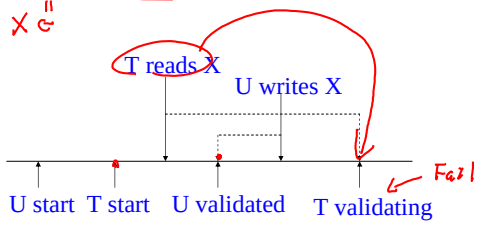
Assumed Serial Schedule for Validation

- We may think of each transaction that successfully validates as executing at the moment that it validates



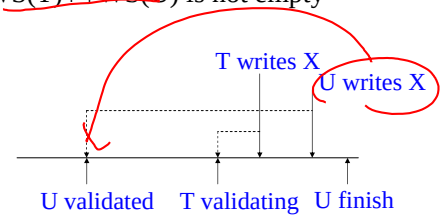
Potential Violation: Read too Early

- Transactions T and U such that
  1. U has validated
  2. START(T) < FIN(U)
  3. RS(T) ∩ WS(U) is not empty



Another Potential Violation: Write too Early

- Two transactions T and U such that
  - U is in VAL
  - VAL(T) < FIN(U)
  - WS(T) ∩ WS(U) is not empty



Validation Rules

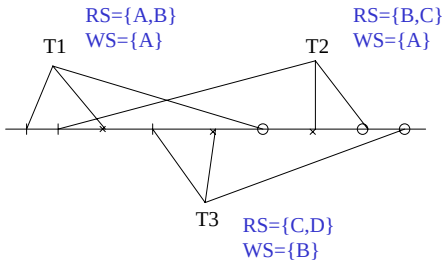
To validate a transaction T,

1. Check that RS(T) ∩ WS(U) is an empty set for any validated U and START(T) < FIN(U)
2. Check that WS(T) ∩ WS(U) is an empty set for any validated U that did not finish before T validated, i.e., if VAL(T) < FIN(U)

Example Problem

In the following sequence of events, tell what happens when each sequence is processed by a validation-based scheduler.

R1(A,B); R2(B,C); V1; R3(C,D); V3; W1(A); V2; W2(A); W3(B);



Storage

- How to store records in a block?
  - Fixed-length record
  - Variable-length record
- What extra information do we need to store a record in a block?
- What information is stored in a block header?
- What information is stored in a record header?
- In which situations do we create an overflow block?

Indexing

- Do you know which types of index can we use in different situations?
  - Dense/Sparse
  - Secondary (unclustered)
  - Multi-level index
- B-tree
  - Do you understand the structure of a B-tree?
    - Root node; internal node; leaf node
  - How to find/insert/delete a record?
- Dynamic hash table
  - Do you understand the structure of a dynamic hash table?
    - Pointer array, data bucket, nub, etc.
  - How to find/insert/delete a record?
  - What are differences from static hash tables?
  - What are different between extensible and linear hash tables?

Query execution

- What are cost parameters?
- Can you describe how different join algorithms work?
  - Nested loop join; sort-based join; hash-based join
- Do you know the memory requirement of each algorithm?
- Do you know in which situation we can use a simple-sort join or sort-merge-join?

Query Optimization

- Do you know how to apply algebraic laws to get a better logical plan?
- Do you know how to perform a cost-based optimization on a logical plan?
  - Dynamic programming
  - Left-deep or Right-deep join trees
- Do you know how to estimate the size of an intermediate operation?
  - Selection
  - Join

Logging & Recovery

- What's the correctness principle?
- What are primitive four operations of transactions?
- How does undo logging work?
  - Log records
  - Undo-logging rules
  - Recovery procedure
- How does redo logging work?
- How does undo/redo logging work?
- What is a checkpoint?
- What is a nonquiescent checkpoint?
- Do you know when <END CKPT> is added in each method?
- Do you know recovery procedures with a checkpointed log?
- What are advantages and disadvantages of each method?