CIS 560 – Database System Concepts

Lecture 22

Concurrency Control and Indexes

October 25, 2013

Credits for slides: Chang, Ullman, Whitehead.

Announcements

- HW6 due today
- HW7 will be posted tonight

2

Outline

Last:

- Locks 18.3
- Timestamps 18.8

Today:

- Timestamps 18.8
- Indexes and B-trees 14.1-14.2

Next:

- Indexes and B-trees 14.1-14.2
- Query execution 15.1-15.6
- Query optimization 16

3

Main Idea

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

In each of these cases

- $w_U(X) \ldots r_T(X)$
- $r_U(X) \ldots w_T(X)$
- $w_U(X) \ldots w_T(X)$

Possible conflicts

When T requests $r_T(X)$ or $w_T(X)$, need to check TS(U) <= TS(T)

4

Timestamps

With each element X, associate

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

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

5

Simplified Timestamp-based Scheduling

Note: simple version that ignores the commit bit

- Only for transactions that do not abort
- Otherwise, may result in non-recoverable schedule

Transaction wants to read element X
If TS(T) < WT(X) then ROLLBACK
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) then ROLLBACK
Else if TS(T) < WT(X) ignore write & continue (Thomas Write Rule)
Otherwise, WRITE and update WT(X) =TS(T)

6

Details

Read too late:

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

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

Need to rollback T!

7

Details

Write too late:

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

START(T) … START(U) … $r_U(X)$ . . . $w_T(X)$

Need to rollback T!

8

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(U) … w_U(X) . . . w_T(X)

Don't write X at all !
(Thomas' rule)

9

More problems when transactions can ABORT

Thomas' rule needs to be revised:

- 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

10

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 a new timestamp)
  - To delay T until C(X) = true

11

Timestamp-based Scheduling

Transaction wants to READ element X
    If TS(T) < WT(X) then ROLLBACK
    Else If C(X) = false, then WAIT
    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) then ROLLBACK
    Else if TS(T) < WT(X)
        Then If C(X) = false then WAIT
            else IGNORE write (Thomas Write Rule)
    Otherwise, WRITE, and update WT(X)=TS(T), C(X)=false

Textbook section 18.8.4

12

Exercise

- The following schedule is presented to a timestamp-based scheduler. Assume that the read and write timestamps of each element start at 0 (RT(X) = WT(X) = 0), and the commit bits for each element are set. Explain what happens as the schedule executes.

$st_1, st_3, st_2, r_1(A), r_2(B), w_1(C), r_3(B), r_3(C), w_2(B), w_3(A)$

13

T1	T2	T3	A	B	C
100	300	200	RT =0 WT=0	RT =0 WT=0	RT =0 WT=0
$r_1(A)$					
	$r_2(B)$				
$w_1(C)$					
		$r_3(B)$			
		$r_3(C)$			
	$w_2(B)$				
		$w_3(A)$			

14

T1	T2	T3	A	B	C
100	300	200	RT =0 WT=0	RT =0 WT=0	RT =0 WT=0
$r_1(A)$			RT=100		
	$r_2(B)$			RT=300	
$w_1(C)$					WT=100 C = 0
		$r_3(B)$ $r_3(C)$ Delay T3 until C(C) = 1 or T1 aborts, then recheck timestamps and retry this action.			
	$w_2(B)$			WT=300 C=0	
		$w_3(A)$ Wait until T3 is unblocked and r3(C) above succeeds. If T3 is later aborted, then do not execute this action.			

15

Summary of Timestamp-based Scheduling

- Conflict-serializable
- Recoverable
  - Even avoids cascading aborts
- Does NOT handle phantoms

16

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}, . . .

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

- Let T read an older version, with appropriate timestamp

This is what most commercial DBMSs implement.

17

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

18

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

19

Transaction Best Practices

20

READ-ONLY Transactions

```
Client 1: START TRANSACTION
          INSERT INTO SmallProduct(name, price)
                    SELECT pname, price
                    FROM Product
                    WHERE price <= 0.99

          DELETE  FROM Product
                     WHERE price <=0.99
          COMMIT

Client 2: SET TRANSACTION READ ONLY
          START TRANSACTION
          SELECT count(*)
          FROM Product

          SELECT count(*)
          FROM SmallProduct
          COMMIT
```

Can improve performance

21

Isolation Levels in SQL

1. "Dirty reads"
   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

2. "Committed reads"
   SET TRANSACTION ISOLATION LEVEL READ COMMITTED

3. "Repeatable reads"
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

4. Serializable transactions
   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

ACID

22

Choosing Isolation Level

- Trade-off: efficiency vs correctness
- DBMSs give user choice of level

Always read docs!

Beware!!
- Default level is often NOT serializable
- Default level differs between DBMSs
- Some engines support subset of levels!
- Serializable may not be exactly ACID

23

1. Isolation Level: Dirty Reads

Implementation using locks:

- "Long duration" WRITE locks
  - Strict Two Phase Locking
- No READ locks
  - Read-only transactions are never delayed

Possible problems: dirty and inconsistent reads

24

2. Isolation Level: Read Committed

Implementation using locks:

- "Long duration" WRITE locks
- "Short duration" READ locks
  - Only acquire lock while reading (not 2PL)

Possible problems - unrepeatable reads
When reading same element twice,
may get two different values

25

3. Isolation Level: Repeatable Read

Implementation using locks:

- "Long duration" READ and WRITE locks
  - Full Strict Two Phase Locking

Why ?

This is not serializable yet !!!

26

Isolation Summary

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

MySQL default: repeatable read
InnoDB - a combination of multiversion concurency control and locks

27

Indexes

28

Disks and Files

- Basic data abstraction – *data file* - collection of records.
- DBMS store data on ("hard") disks.
- Data is stored and retrieved in units called *disk blocks* or *pages.*
- Performance varies with time to retrieve disk pages
  - `SELECT * FROM Product WHERE BarCode = 10002121`
  - `SELECT * FROM Product WHERE Price BETWEEN 5 and 15`
  - Assume: 200,000 rows in table – 20,000 pages on disk

File Types

The **data file** can be one of:

- Heap file:
  - Set of records, partitioned into blocks
  - Unsorted
- Sequential file:
  - Set of records, partitioned into blocks
  - Sorted according to some attribute(s) called *sort key*

Note: "sort key" different from "primary key"

Index

- A (possibly separate) file, that allows fast access to records in the *data file* given a *search key*.
- The index contains (key, value) pairs:
  - The key = an attribute value
  - The value = either a pointer to the record, or the record itself

Note: “search key” different from “primary key”

Index Classification

- Clustered/unclustered
  - Clustered = records close in index are close in data
  - Unclustered = records close in index might be far in data
- Primary/secondary:
  - Primary = on primary key
  - Secondary = on any other key
- Dense/sparse
  - Dense = each record has an entry in the index
  - Sparse = only some records have
- Organization: B+ tree or Hashtable