Introduction to Database Systems
CSE 444

Lecture 14
Transactions: concurrency control
(part 2)

Magda Balazinska - CSE 444, Fall 2010

Outline

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

Magda Balazinska - CSE 444, Fall 2010

Timestamps

• Each transaction receives a unique timestamp
TS(T)

Could be:
• The system's clock
• A unique counter, incremented by the scheduler

Magda Balazinska - CSE 444, Fall 2010

Main invariant:

The timestamp order defines
the serialization order of the transaction

Magda Balazinska - CSE 444, Fall 2010

Main Idea

• For any two conflicting actions, ensure that
their order is the serialized order:
In each of these cases
• w_U(X) . . . r_T(X)
• r_U(X) . . . w_T(X)
• w_U(X) . . . w_T(X)

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

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

Magda Balazinska - CSE 444, Fall 2010

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

Magda Balazinska - CSE 444, Fall 2010
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) … \( w_0(X) \ldots r_1(X) \)

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) … \( w_U(X) \ldots (r_T(X)) \ldots ABORT(U) \)

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) … \( w_T(X) \ldots (r_U(X)) \ldots ABORT(U) \)

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

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}, ...  
  \[ TS(X_t) > TS(X_{t-1}) > TS(X_{t-2}) > ... \]
- 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_{t+1} and all active transactions T have TS(T) > t+1

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

START(U)  VAL(U)  FIN(U)

U: Read phase  Validate  Write phase

T: Read phase  Validate ?

START(T)  VAL(T)

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

START(U)  VAL(U)  FIN(U)

U: Read phase  Validate  Write phase

T: Read phase  Validate  Write phase ?

START(T)  VAL(T)

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