More Locking and Timestamp Concurrency Control

CS157B

Chris Pollett

May 9, 2005.

Outline

- The Tree Protocol
- TimeStamps

The Tree Protocol

- Standard locking protocols make it very hard to do concurrency when doing locking with B-trees.
- The problem is we would typically need to lock a whole path down the tree in case the tree changes during an update. This prevents other people from looking at the tree.
- Instead, to get concurrency for B-trees, we don't use two phase locking and:
  - a transaction's first lock in the tree may be at any node of the tree (typically lock first node that could be changed by your transaction.)
  - subsequent locks can be obtained iff the transaction has the parent lock.
  - nodes can be unlocked at any time
  - a transaction may not relock a node it has released.

Other Concurrency Control Techniques

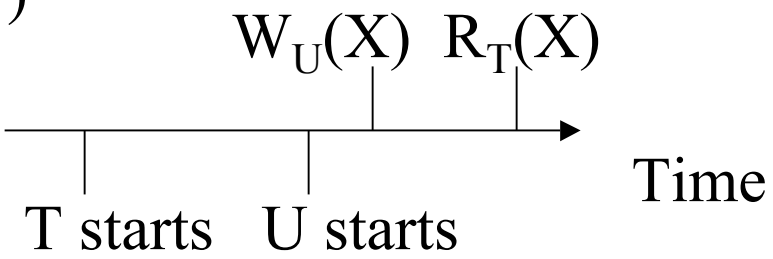
- Besides locking there are two other common methods for doing concurrency control:
  - timestamping -- give each transaction and database element a timestamp and compare these to determine if the schedule is equivalent to the serial schedule given by the transactions timestamps.
  - validation -- also uses timestamps but will yield schedules equivalent to the serial schedule give by transaction commit times.
- We'll focus on the first technique.

Timestamps

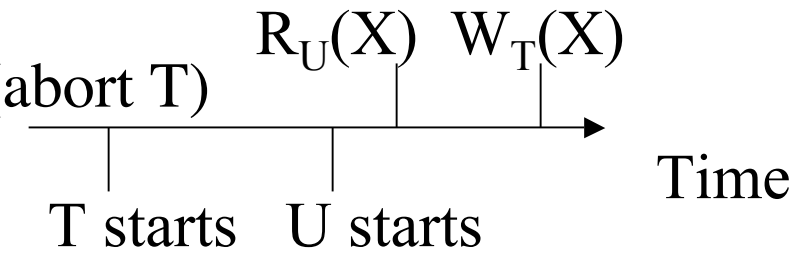
- We'll write TS(T) to denote a number (timestamp) assigned by the scheduler to transaction.
- The number must be such that later transaction receive a higher number.
- Such numbers could be based on the system clock or based on using a counter.
- For each database element X, the scheduler also maintains the following numbers: RT(X) -- timestamp of last T to read X, WT(X) -- timestamp of last T to write X, C(X) -- last transaction to write X has commited or not.

Physically Unrealizable Behaviors

- Suppose each operation in a schedule executed at the time of the timestamp assigned by the scheduler to DB items.
- Two problems could occur:
  - Read too late: (abort T)

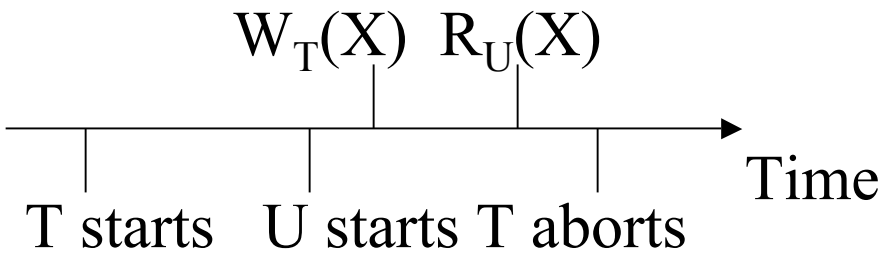


  - Write too late: (abort T)



Problems with Dirty Data

- The two problems of the last slide could be resolved by comparing TS(T) and WT(X) or RT(X).
- What is the C(X) used for? Consider the following sequence:



- The value read by U is bogus since T aborted. This is called a *dirty read*. We can check to make sure C(X) = true. Scheduler maintains C(X) so works. Can delay U to make a valid schedule.

Rules for Timestamp-Based Scheduling

- The scheduler when given a read or write request by T can either: grant it, abort T, or delay T until some other transaction commits or aborts.
- Next slide has the exact rules.

More rules

- If the request is a $R_T(X)$:
  - If TS(T) >WT(X), the read is physically realizable.
    - If C(X) is true, grant the request. If TS(T) >RT(X) set RT(X) :=TS(T)
    - If C(X) is false, delay T until either C(X) becomes true or the transaction that wrote X aborts.
  - If TS(T) < WT(X), the read isn't realizable. Rollback T, aand restart it with a larger timestamp.
- If the request is a $W_T(X)$:
  - If TS(T) >= RT(X) and TS(T) >= WT(X), the write is physically realizable and must be performed.
    - Set new value for X
    - Set WT(X) := TS(T)
    - Set C(X) := false …

Yet More Rule

  - If TS(T) >= RT(X) but TS(T) < WT(X), then the write is realizable but there is already a later value of X store in DB. If C(X) is true than can ignore write request. If C(X) is false must delay T until other transaction commits or aborts.
  - If TS(T) < RT(X), operation is unrealizable so rollback.
- If the scheduler receives a commit request from T then all of the elements written by X have C(X) marked true. Delayed transactions are notified.
- Similarly, if T aborts transactions waiting on T are notified.