Multiversion Concurrency Control, Transaction Management

CS157B
Chris Pollett
May 11, 2005.
Outline

• Multiversion Timestamps
• Timestamps versus Locking
• Logging and Concurrency
Multiversion Timestamps

• We want to allow for greater concurrency than basic timestamping allows.
• To do this we will reduce the number of reads that cause transactions to abort.
• Suppose transactions arrive in the order: T1, T3, T2, T4. Consider the schedule: r_1(A), w_1(A), r_2(A), w_2(A), r_3(A), r_4(A). T3 must abort according to timestamping.
• However, if we had kept an old value of A around for T3 we might not need to abort it.
More on Multiversion Timestamps

• How do we manage multiple versions of database elements?
  – When a write $w_T(X)$ occurs, if it is legal, a new version of $X$ is created. Its write time is $TS(T)$ and we will call it $X_t$ where $t=TS(T)$.
  – When a $r_T(X)$ occurs, the scheduler finds the first version of $X_t$ of $X$ such that $t \leq TS(T)$, and such that there is no $X_{t'}$ with $t < t' \leq TS(T)$.
  – Write times are now associated with versions of an element and they never change.
  – Read times are associated with version. Read times are used to reject certain writes as will be indicated on the next slide.
  – When a version $X_t$ has a time $t$ such that no active transaction has a timestamp less than $t$, then we may delete any version $X$ of previous to $X_t$. 
What kind of writing should be rejected?

\[ R_S(X) \text{ sets} \]
\[ RT(X_{50}) = 80 \]

\[ X_{50} \quad W_T(X) \quad X_{100} \]

\[ \text{attempted where} \quad TS(T) = 60 \]

- \( S \) should have read \( T \)'s value but reads \( X_{50} \) instead. So should abort \( T \).
Timestamps versus Locking

- Basic rule of thumb: timestamping works better when most transactions are read-only, or it is rare that concurrent transaction will read and write the same element.
- Locking works better in high conflict situations.
- The reasoning is:
  - Locking frequently delays transactions as they have to wait for locks and can lead to deadlocks.
  - If concurrent transactions have frequent reads and write in common then timestamping will tend to cause transactions to rollback frequently making the throughput less than with locking.
- Commercial DBMS systems try to get the best of both by allowing a read only isolation level which is handled using multiversion timestamping and otherwise use locking for other isolation levels.
More on Transaction Management

- We have now talked about recovery and about serializability but we haven’t said how to get these two components of the DBMS to work together.
- Our logging mechanisms make no mention of serializability and there is no guarantee when we do a recovery that the consistent state we get to corresponds to something that might have been produced by a serializable schedule.
- On the other hand, there is nothing about two phase locking that prevents a transaction from writing into the database uncommitted data.
- To finish up the semester we will give an example situation where logging and concurrency interact.
Cascading Rollbacks

• Consider the schedule:
  – L_1(A), R_1(A), W_1(A), L_1(B), U_1(A),
    L_2(A), R_2(A), W_2(A), L_2(B) denied,
    R_1(B), A_1, U_1(B), L_2(B), U_2(A), R_2(B),
    W_2(B), U_2(B).

• If we are using timestamping with a commit bit the above schedule without the locks couldn’t happen, but it is a legal 2PL schedule. However, T_2’s value for A is dirty so we should rollback T_2 when T_1 aborts. This rollback that causes another rollback is called a cascading rollback.

• To avoid this problem, a transaction must not release any write locks until it either commits or aborts and the commit or abort log record is flushed to disk. This locking protocol is called strict 2PL. It shows that logging and concurrency do need to interact.

• Aside: A quick trick when blocks are locked rather than rows --that does not require interaction with the log -- is to require blocks written (and locked) by uncommitted transaction be pinned in main memory until the transaction commits or aborts.