Conflict Serializability, Locking, Lock Modes

CS157B

Chris Pollett

Apr.27, 2005.

Outline

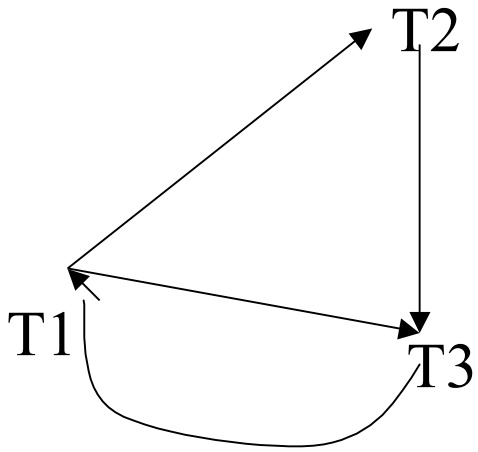
- Testing for Conflict Serializability
- Locking and Two Phase Locking
- Different Types of Lock Modes

Testing for Conflict Serializability

- From our definition of conflicting operation last day, two operations from different transactions are in conflict if they involve the same database element and one of the operations is a write.
- We can use conflicts to make a graph based on a schedule.
- We put an edge T_i -> T_j in the graph, if there is an operation O_i of T_i in the schedule which is in conflict with operation O_j of T_j in the schedule and O_i occurs before T_j.
- The schedule will be conflict serializable iff this precedence graph is acyclic.
- If it the graph has a topological ordering, the schedule given by this ordering will be conflict equivalent to the original schedule. This topologically ordered schedule will be a serial one.
- To topologically order the graph, we repeatedly take out of the graph those elements that have no predecessors and put them in our topological order.

Example Precedence Graph

W1(X), W2(X),
C2, W3(X),
W3(Y),
W1(Y), C3,
C1.



Has a cycle so not conflict serializable.

If we deleted the W1(Y,b) it would have been conflict serializable.

Serializable versus Conflict Serializable

- Conflict serializable implies serializable.
- However, serializable does not imply conflict serializable.
- For example, W1(Y), W1(X), W2(Y), W2(X), W3(X) is a serial schedule which has the same effect on the database as W1(Y), W2(Y), W2(X), W1(X), W3(X), but this latter is not conflict serializable.

Locking

- We are now going to consider a mechanism for guaranteeing conflict serializability based on locks.
- In this set-up, the scheduler keeps a table of locks.
- Entries in this table consist of the id of a transaction together with an id of a database element.
- The scheduler ensures that only transactions which have the appropriate lock on a database element to do an action can perform that action.
- Some kinds of locks will prevent other transactions from obtaining locks on an element.

More on Locking

- To make the locking mechanism of the last slide work, we want:
  - Consistency of transactions:

    (1) A transaction can only read or write an element ,if it has previously requested and obtained the lock and it hasn't since released the lock.

    (2) If a transaction locks an element it must release it a some point.
  - Legality of schedules:

    Locks must have their intended meaning -- In the simplest case, this means no two transactions may have the lock on the same element without one having first released the lock.
- In a schedule, we write $l_i(X)$ to say T_i locks X and we write $u_i(X)$ to say T_i releases the lock.

Two Phase Locking (2PL)

- The two phase locking condition is the following:
  - For every transaction, all lock requests must precede all unlock requests.
- It turns out that if this condition is satisfied for a schedule, then it will be conflict serializable.
- It is called “two-phase”, because for a given transaction there will be a phase when the number of locks it has is increasing, followed by a phase where the number of locks is decreasing.

Why two-phase locking works

- We will only give some intuition…A proof is in the book based on induction on the number of transactions in a schedule.
- The idea. though, is if T1 reads or writes X, it must first get a lock.
- At that point, it must hold on to the lock until it is done all of its operations on X. This is because T1 is not allowed in 2PL to get a lock, release it and re-acquire the lock.
- Thus, if T2 does anything with X it must have done all of its operations with X before or all of its operations after T1.
- Now suppose T1 and T2 also conflict on some element Y. Tobr concrete, suppose T2 has the lock on Y. T1 can't get this lock until T2 releases it, at which point we know T2 can't acquire more locks so, in particular, it couldn't get the lock for X.
- So T2 must do all of its operation of X and Y either before T1 or do them all after T1.

Different types of Lock Modes

- A more sophisticated locking mechanism allows for different levels of locks.
- For example, shared and exclusive locks.
- If T1 has a shared lock on X other transactions can also get a shared lock on X. Write $sl_i(X)$ in the schedule.
- However, only one transaction at a time can have an exclusive lock on X and if any transaction has an exclusive lock on X then no other locks on X can be held. One cannot get an exclusive lock until everyone else has released their locks. Write $xl_i(X)$ in the schedule.
- 2PL is an in the single lock mode case.