Transaction

Wei Wang

weiw AT cse.unsw.edu.au

School of Computer Science & Engineering
University of New South Wales

November 26, 2014
Things to note:

- **Why transaction theories?**
  - the need for concurrent execution of transactions.
  - even if all concurrently executed transactions are "correct", a "bad" schedule of them will bring the DB to an "incorrect" state.
    - Not new if you have done multi-threaded programming (i.e., you know the difference between HashMap and Hashtable)
    - Programmers only need to make sure their transactions are "correct" when running alone.
    - In Programming Languages, used to achieve CC manually, but it may be changing. See software transactional memory on wikipedia.

- For the analysis of schedules, a transaction is a function whose *sole* input are the DB objects it reads, and whose *sole* output is the DB objects it writes.
  - No side effects; hence, can be scheduled serially and can be *restarted* if needed!
  - Similar to the advantage of *functional programming*.

- Complex computation may occur in a transaction.
  - Need to acknowledge this complexity when reasoning about serialializability, but we do not need to model them.
  - OS will take care of scheduling the concurrent execution of these computations.
One way to understand the need for so many schedules:

1. **Serial schedule**: the obviously correct and reasonable schedule. However, no concurrent execution is allowed!

2. **Serializable schedule**: those that are equivalent to some serial schedule, but allows concurrent execution of transactions.
   - (Optional) View-serializable schedule.

3. **Conflict-serializable schedule**: those that are conflict-equivalent to some serial schedule. It (1) allows concurrent execution, (2) its decision problem can be solved in polynomial time using the precedence graph, and (3) there are concurrency control mechanisms that generate this categories of schedules.
   - $S_1$ is conflict-equivalent to $S_2$ means $S_1$ can be changed to $S_2$ by repeatedly swapping pairs of statements that are not in conflict.
   - (Optional) We actually only need the pair to be communicative. Hence incremental and decrement by 1 can be swapped!

4. All the above only considers committed transactions. Abort makes it more complex:
   - Recoverable schedule:
   - Cascadeless schedule: avoid cascading aborts.

Further reading: “Schedule (computer science)” page on wikipedia.
Transaction Support in SQL and DBMSs

SQL:
- Two access modes:
- Four isolation levels, and how they eliminate serialization anomalies.

DBMS's implementation varies:
- Oracle, PostgreSQL: SERIALIZABLE implemented using snapshot isolation, almost but not truly serializable (because of using Multi-version Concurrency Control (MVCC) and the prohibitive cost of implementing the predict locking).
  - Programmers need to use explicit locking by themselves in necessary.
- MS SQL Server: supports the additional SNAPSHOT isolation level.
- PostgreSQL: essentially only implements two levels: READ_COMMITTED and SERIALIZABLE. See http://www.postgresql.org/docs/8.4/static/transaction-iso.html.
Conflict-Serializable Schedule

T1: R(X) W(X) W(Z) R(Y) W(Y)
T2: R(Y) W(Y) W(Y) R(X) W(X) R(V) W(V)

1. Is the above (partial) schedule conflict serializable? Show your working.
2. Modify it to create a complete schedule that is conflict-serializable.
Consider the precedence edges created by each object:

- X: $T_1 \rightarrow T_2$.
- Y: $T_2 \rightarrow T_1$.
- Z and V: none

Hence the precedence graph has (is) a circle, and the schedule is therefore not conflict-serializable.

E.g., break the $T_1 \rightarrow T_2$ edge by moving the $R(X)$ and $W(X)$ operations of $T_2$ before the first $R(X)$ of $T_1$. 
Recoverable Schedule

Recoverability and serializability are both important properties of concurrent transaction schedules. They are also orthogonal. Serializability requires that the schedule be equivalent to some serial ordering of the transactions. Recoverability requires that each transaction commits only after all of the transactions from which it has read data have also committed.

Using the following two transactions:

T1:  W(A)  W(B)  C  
T2:  W(A)  R(B)  C

Give examples of schedules that are:

1. recoverable and serializable
2. recoverable but not serializable
3. not recoverable but serializable
Solution

1. T1: W(A) W(B) C  
   T2: W(A) R(B) C

2. T1: W(A) W(B) C  
   T2: W(A) R(B) C

3. T1: W(A) W(B) C  
   T2: W(A) R(B) C
Cascadeless schedules avoid the potential cascading rollbacks that can make recoverable schedules less than desirable. Using the transactions from the previous question, give an example of a cascadeless schedule.
Solution

T1: \[ W(A) \ W(B) \quad C \]
T2: \[ W(A) \quad R(B) \quad C \]