CS632 Final Exam Part A

A. Demers

2 May 2003

A couple more questions will appear by Monday. Nothing will be due less than a week after it appears here.

As usual, you may consult others for ideas and proof approaches, but please *reference your sources* and write up answers independently.

1 The Oracle(tm) Isolation Rules

Here is a somewhat simplified description of the isolation rules used by Oracle.

The system maintains an event counter called the System Change Number (SCN), which is effectively a clock. For any entity x and any SCN (or time) t, the system can return the last value written to x by a transaction that committed before t. Thus, for any time t a transaction can arrange to get a *consistent snapshot* of the database, consisting of the values written by those transactions that committed before t.

Dirty reads do not happen. Each entity x is tagged with the transaction that last wrote it. If that transaction has not (yet) committed, the value of the entity is never returned to a reader, but instead data from the *rollback segment* (which is essentially an undo log) is used to find and return the appropriate value written by a committed transaction.

The operations are supporting this model are

- R(x, t): Return the value written to entity x by the last transaction to commit before t.
- R(x): Equivalent to R(x, C) where C is the current SCN.
- L(x): Acquire an exclusive lock on entity x.
- U(x): Release the lock on entity x.

W(x): Write entity x.

C: Commit the transaction.

A: abort the transaction.

As usual, we shall assume a transaction is not allowed to re-read an entity that it has already written.

Problem 1a: Sketch how an ARIES-like recovery algorithm can support the operations described above. Obviously, the interesting operation is R(x, t). Don't worry about fine-granularity locking – you may assume a locked entity occupies exactly one page. But don't oversimplify either – your algorithm should support a transaction that modifies more entities than will fit in the buffer pool, and committing a transaction should require forcing the log but not waiting for dirty pages to be flushed from the buffer pool. Discuss checkpointing and archiving/truncating the log.

Problem 1b: By default, the locking protocol used by Oracle is strict 2-phase locking on writes, and *no locking at all* on reads. Each read is done at the current SCN (that is, transactions use R(x) rather than R(x, t) in the above description).

Show by example that the default Oracle locking protocol can generate nonserializable schedules.

(In practice, a transaction consists of multiple SQL queries, each of which accesses many entities. All the read access of a single SQL query are performed as of the same SCN, so that each SQL query sees a consistent snapshot. However, different SQL queries in the same transaction are executed at different SCNs and thus see different snapshots. Feel free to ignore this detail, and think of each SQL query as a single read.)

Problem 1c: An alternative rule associates with each transaction A a "snapshot timestamp" t_A, which is the SCN when the transaction was started. Every read in A is performed at time t_A – that is, each read R(x) is replaced by R(x, t_A). In addition, the following test is performed for each lock operation $L_A(x)$:

> After the lock has been granted, if the commit time of the last transaction that wrote x is greater than t_A, then abort A.

Show that this protocol can also generate nonserializable schedules.

Problem 1d: (We are trying desperately to find some way to guarantee serializable schedules in Oracle.) Consider *static* 2-phase locking, that is, strict 2-phase locking with the additional requirement that all locks must be acquired at the beginning of the transaction, before the first read or write operation. Let the snapshot timestamp of a transaction be the SCN after all locks have been acquired. Using this rule, are nonserializable schedules possible? Give an example, or prove that no such example exists.

Under rule (1d), what happens to transactions / schedules that caused transaction aborts under rule (1c)?