Solution notes

Concurrent Systems and Applications 2005 – Paper 6 Question 4 (JKF)

(*a*) Strict isolation really does ensure that transactions do not see the intermediate results of others. Non-strict isolation relaxes that but does not permit transactions to commit until any dirty reads (values that have been updated by a transaction that has not yet done commit/abort) are resolved by the other transaction calling commit. Similarly, if a dirty read is rolled back by the other transaction then you have to abort and retry—a cascading abort.

(*b*) (*i*) OCC algorithm: a transaction proceeds by taking shadow copies of each object it uses (when it accesses it for the first time). It works on these shadows so changes remain local—isolated from other transactions. Upon commit it must: Validate that the shadows were consistent and that no other transaction has committed an operation on an object which conflicts with the one intended by this transaction. If OK then commit the updates to the persistent objects, in the same transaction-order at every object. If not OK then abort: discard the shadows and retry.

(*ii*) No cascading aborts or deadlock.

(*iii*) Conflicts force transactions to retry—performance is poor if there are lots of conflicting operations happening on "popular" objects.

(*c*) A write-ahead log provides persistence by writing transaction start messages, commit/abort messages, and idempotent object manipulation messages that give the pre- and post-update values (so changes can be rolled back). To replay the log, make UNDO and REDO queues. UNDO partially-done operations that didn't commit, REDO operations whose transactions did commit because changes might not have hit the disks. Checkpoints save us from having to start at the very beginning of the log every time—essential. Instead, just process the log from the last checkpoint.