SOLUTION NOTES

Concurrent Systems and Applications 2002 Paper 6 Question 4 (TLH)

This question is about different schemes for providing isolation between transactions and the selection between them as the workload varies.

(*a*) For *strict isolation* one transaction should not see any updates from other transactions that have not yet committed. Ordinary *isolation* relaxes this: a transaction cannot commit until those whose updates it has seen have committed.

(*b*) A transaction using two-phase locking is divided into a first phase during which locks can be acquired and a second phase during which locks can be released. Strict two-phase locking requires that locks are only released when a transaction completes. This evidently enforces strict isolation – an update made to an object by one transaction will not be visible until after the lock protecting that object is released.

(*c*) Three impacts:

(*i*) Ordinary 2PL allows locks to be released as soon as the second phase is reached. During the second phase each object's lock can be released as soon as that object has been accessed for the final time.

(*ii*) A transaction must not commit until all transactions that made updates that it has observed have committed. This enforces *isolation*.

(*iii*) When one transaction aborts then any transaction that has observed updates that it made must also be aborted. This is the problem of *cascading aborts*.

(*d*) The observations that I intend:

(*i*) The system is becoming deadlocked. Contention is high so deadlock detection, aborting and retrying may not be appropriate: it does not guarantee progress. Deadlock avoidance could be performed by acquiring locks according to some total order.

(*ii*) Long-running transactions can harm performance in S-2PL: a transaction cannot release any locks until it has completed, extending the period of time for which objects remain locked. Two reasonable options: consider using 2PL or splitting the transaction into sections.

(*iii*) If contention is rare then lock acquisition / release may account for proportionately more execution time. Two schemes are covered in the course – timestamp ordering and optimistic concurrency control with a simple validator. A description of either is reasonable and TSO is probably simpler, e.g.:

Associate a unique ordered timestamp with each transaction (e.g. the time at which it began along with a suitable tie-break). Extend each object to record the timestamp of its most recent access. Permit a new access if it is from a later-timestamped transaction (storing that timestamp in the object). Otherwise reject the access as 'too-late'. This enforces *isolation* and serializes execution according to timestamp order. For *strict isolation* delay any access until after the previous accessor has committed or aborted.