Locking Protocols

1. Overview of Section
   (a) Goals of locking
   (b) Shared and exclusive locks
   (c) Two-phase locking
      i. Simple locking is non-serializable
      ii. wormhole transactions
   (d) Locking theorems
   (e) 2PL in databases
      i. Centralized
      ii. Primary-copy
      iii. Distributed
   (f) Degrees of Isolation
2. Goals of Locking
   (a) Implement serializable schedules
      i. For concurrent transactions
      ii. On-line, as transactions occur
   (b) Part of a DBMS
      i. Under the application layer, $i.e.$ transparent
         A. Databases use transactions
         B. DBMS implements transactional guarantees through locking
3. Shared and Exclusive Locks
   (a) A shared lock, also called a read lock, on data item x allows a transaction to read that item: $SL(x)$ or $SLOCK(x)$.
      i. Shared refers to its management semantics
      ii. Read describes the allowed operation by the lock holder
      iii. In other data models, one may be able to take more actions than read based on a shared lock
      iv. In serializable schedules, shared lock means read
   (b) An exclusive lock, also called a write lock, on data item x allows a transaction to write that item: $XL(x)$ or $XLOCK(x)$
      i. Again, exclusive refers to the management semantics
   (c) Locks are released by $RXL(x)$ or $RSL(x)$
4. Shared and exclusive locks

Compatibility table		Granted Mode	
		SL	XL
Requested Mode	SL	Yes	No
	XL	No	No

5. Lock Compatibility
   (a) Compatibility refers to whether lock may be outstanding at the same time

i. Many shared locks may be outstanding
ii. Data does not change under a shared lock
iii. A single exclusive lock may be outstanding
iv. Data may be modified under an exclusive lock

(b) Locks are managed entities
i. A lock manager uses the compatibility table to resolve outcomes
ii. Transactions make requests of this lock manager
iii. The lock manager function may limit the scalability of locking

6. Legal and well-formed

(a) A transaction is *well-formed* if every read, write, and unlock action is protected by a suitable corresponding lock
(b) A history/schedule is legal if it does not grant conflicting locks to two transactions at the same time
(c) surprisingly, well-formed transactions conducted with legal histories are not serializable as a rule
i. not enough to achieve isolation, good enough for atomicity and consistency

7. Wormhole transactions

(a) Under simple locking rules
i. Before every read of data item x acquire $SL(x)$
ii. Before every write of data item x acquire $XL(x)$
iii. After the last read/write of s release $SL(x)$/$RL(x)$
(b) This technique does not provide a serial equivalent schedule (the goal)
(c) Consider the following two transactions
i. T1: $R(x), x = x + 1, W(x), R(y), y = y - 1, W(y)$
ii. T2: $R(x), x = x * 2, W(x), R(y), y = y * 2, W(y)$
(d) The can be serialized in the following way by locking
i. $SL_1(x), R_1(x), x = x + 1, XL_1(x), W_1(x), RXL_1(x)$
ii. $SL_2(x), R_2(x), x = x * 2, XL_2(x), W_2(x), RXL_2(x)$
iii. $SL_2(y), R_2(y), y = y * 2, XL_2(y), W_2(y), RXL_2(y)$
iv. $SL_1(y), R_1(y), y = y - 1, XL_1(y), W_1(y), RXL_1(y)$
(e) Outcome: $x = (x + 1) * 2, y = y * 2 - 1$
(f) Legal (serial) outcomes:
i. $x = (x + 1) * 2, y = (y - 1) * 2$
ii. $x = x * 2 + 1, y = y * 2 - 1$
(g) What happened?
i. Two independent parts to each transactions, an x part and a y part

Illegal	**Legal**	**Legal**
T1: Part x	T1: Part x	T2: Part x
T2: Part x	T2: Part x	T1: Part x
T2: Part y	T1: Part y	T2: Part y
T1: Part y	T2: Part y	T1: Part y

(h) This is called a wormhole
i. Not serial equivalent, T1 does not precede T2, nor vice versa
ii. Not isolated, T1 sees the DB before and after T2
iii. Hence, wormhole, time-travel (multiple times) in an atomic action

Two-Phase Locking

1. Two-phase locking
   (a) 2PL Rule: no transaction should request a lock after it releases one of its locks
      i. Separates locking into a growing and a shrinking phase
      ii. Diagram (TODO)
      iii. The "Lock Point" is when all locks are held for the whole transaction
   (b) *What is the significance of the lock point?*
   (c) *What function does the lock point serve for serializability?*
      i. Binds all of the data used in a transaction together, in our example, x and y
      ii. It, therefore, prevents a transaction from being split into parts, creating wormholes
2. Locking Theorems
   (a) **Wormhole Theorem**: a history is isolated if and only if it has no wormhole transactions
   (b) **Locking Theorem**: if all transactions are well-formed and two-phase then any legal history will be isolated
   (c) **Locking Theorem Converse**: if a transaction is not well-formed or not two-phase then it is possible to write another transaction such that the resulting pair has a wormhole
   (d) **Rollback Theorem**: an update transaction that does an unlock and then a rollback is not two-phase
      i. Cannot rollback after an unlock, motivation for strict 2PL
      ii. Rollback performs data writes, for which locks must be held
      iii. Rollback is not intuitive, rollback does work and is part of the transaction going forward
3. Cascading Aborts
   (a) In 2PL, if locks are released before the transaction outcome is determined, other transactions can run concurrently and an abort in one transaction (the one that released locks) can cause aborts in other transactions
   (b) Consider our example in which we have, T1: Part x, T2: Part x T1: Rollback
   (c) This rollback has several consequences
      i. T2 holds $XL(x)$ on the data item that T1 needs to write as part of its rollback
      ii. T2 cannot commit unless T1 also commits, dynamic data dependency
   (d) What happens? before T1 can rollback, T2 must abort and complete its rollback
   (e) T2 may have a T3, etc. such that this abort dependencies cascade
4. Strict and Rigorous 2PL
   (a) Strict 2PL: hold all exclusive locks until transaction outcome has been determined
      i. Prevents cascading rollbacks because all data modified in a transaction are locked until the outcome
   (b) Rigorous 2PL: hold all locks until transaction outcome has been determined
      i. *Since strict is adequate to prevent cascading aborts, how is rigorous different or useful?*
      ii. Makes sure that commit order and serialization order are the same.
      iii. *When is this important?*
         A. Consider two transactions conducted at the same site in which a long running transaction T1 which reads x is ordered before a short transaction T2 that writes x. T2 returns first, showing an update version of x long before T1 completes based on the old version.
5. 2PL in distributed systems

(a) If data are distributed or replicated how does it affect locking protocols
    i. Centralized: have a lock manager for all data
        A. Scalability problems. One site for all computers
    ii. Primary copy: each fragment, replica has an owning site that maintains locks for that site
        A. Locking is distributed in the same fashion of data in fragmented databases
    iii. Distributed: all replica owners keep lock state

(b) *Is distributed locking necessary for high availability?*

(c) *What advantages might distributed locking have over primary copy locking? Disadvantages?*