Problem Set 4, Part I

Problem 1: Replication

1-1) 7 copies of each item, fully distributed locking

voting scheme	***would it work? (yes/no)***	***explanation***
update 6 read 3	yes	6 > 7 / 2 where 7 is the number of copies 3 > 7 - 6 where 7 is the number of copies and 6 is the number of writes. Having 3 reads guarantees at least one read has the most up to date copy
update 2 read 6	no	2 < 7 / 2 violates the constraint that writes must be at least greater than 7/2 where 7 is the number of copies of each item.
update 5 read 2	no	2 <= 7 - 5 violates the constraint that reads must be at least > 2 because you need at least 3 reads to guarantee a read on the most up to date value
update 4 read 4	yes	4 > 7/2 where 7 is the number of copies 4 > 7-4 where 7 is the number of copies and 4 is the number of writes. Having 4 reads guarantees at least 1 read contains the most up to date copy

1-2) 7 copies of each item, primary-copy locking

voting scheme	***would it work? (yes/no)***	***explanation***
update 6 read 3	yes	3 > 7 - 6 where 7 is the number of copies and 6 is the number of writes. Having 3 reads guarantees at least one read has the most up to date copy
update 2 read 6	yes	6 > 7 - 2, because reading 6 guarantees at least 1 read has the most up to date copy
update 5 read 2	no	5 <= 7-2 because reading 2 does not guarantee at least 1 most up to date copy
update 4 read 4	yes	4 > 7-4 because reading 4 guarantees at least 1 most up to date copy

1-3)

Any high write or low read configuration would work (1-1(a) and 1-2(a)). This is because by writing more, we are required to read less copies to ensure the most up to date data. Fully Distributed Locking would require the transaction to obtain global lock through the number of reads whereas Primary Copy would have the server in charge of that particular table/copy to handle the locking. Generally, Fully Distributed Locking would have more overhead, because you must obtain a global lock from local lock copies, however, it is more reliable and contains less of a bottleneck if the table/copy is less busy. On the other hand, Primary Copy locking would require less network overhead, because one server is in charge of a particular copy's locks. Personally, I would choose 1-2(a) Primary Copy locking with high write, low read configuration, because network overhead is a pretty big bottleneck. However, if primary copy server with lock for the specific read you are doing fails, the transaction gets blocked - which means primary copy is less reliable.

1-4)

No question, I would choose 1-2(b) Primary Copy Locking with 2 update and 6 reads. That has the lowest number of writes required. Because fully distributed lock with voting scheme's 1-1(b) doesn't qualify as a valid configuration, I am going with Primary locking 1-2(b). 2 update and 6 reads requirement basically means that each transaction only needs to write to 2 copies, which is the lowest compare to any other configuration.

Problem 2: Distributed locking with update locks

We can create a global shared lock, global exclusive lock, and global update lock with the following constraints. Global locks in general are acquired via a transaction with enough write/read actions on copies of one particular item acquires a local lock. Global locks require at least n/2 local locks where n is the number of copies. Global shared locks require at least $s > n - x$ where x is the number of local exclusive locks and s is number of local shared locks. The constraints are similar to non distributed locking - no two transactions can hold a global exclusive lock, any number of transactions can hold a global shared lock as long as the other transactions are locked by global shared lock. However, the difference is, global exclusive locks can't be acquired if there is already a global shared lock present. Adding in update locks may seems to be similar as non-distributed update locks: update locks must be acquired after a shared lock and global shared locks must wait until update lock has finished before being able to acquire a shared lock. Also, update locks would probably need to share the same constraint as global shared lock where $u > n - x$, because all the global shared locks would need to be updated together since update locks upgrades shared locks into exclusive locks, which lets copies be written.