Assignment 8 – Solution

Problem 1.

a) Database DB1 is read-only.

1. Every transaction accesses one copy only. Therefore, availability improves with more replicas.
2. There is no bound on system throughput insofar as the database system is concerned. More replicas give more throughput. However, the network may impose a bound, depending on the capacity of connections between clients and the database.

b) Database DB1 processes update transactions. The major challenge is synchronizing the updates, which have to be applied and synchronized at all replicas.

1. There are several ways availability can be reduced. Here is one way: The system uses an algorithm that requires a transaction to update more than one replica to commit. For example, the replication protocol could specify that *all replicas* must be updated within the transaction. In this case, the failure of one replica makes the whole system unavailable for updates. The more replicas, the higher the probability that one of them will fail.
2. System throughput is capped by the number of updates a single replica can perform because updates are performed on all replicas. This is an upper bound on system throughput for any number of replicas. In a primary-copy system with synchronous updates of replicas, network latency bounds the update rate, since each transaction must be acknowledged by (a quorum of) the replicas before committing at the primary. Two-phase commit imposes another limit. The latter two factors reduce the maximum throughput as the number of replicas increase.

Problem 2.

No. 1SR is not guaranteed even if each site uses 2PL (providing local serializability) and 2PC (providing atomic commitment). To show this, suppose the database has two elements x and y and is replicated on two servers A and B.

$T_1 = r_1(x), w_1(y)$
$T_2 = r_2(y), w_2(x)$

Site A	$rl_1(x_a)$	$r_1(x_a)$	$wl_1(y_a)$	$w_1(y_a)$	$ru_1(x_a)$	$wl_2(x_a)$	$w_2(x_a)$	C_1	C_2
Site B	$rl_2(y_b)$	$r_2(y_b)$	$wl_2(x_a)$	$w_2(x_a)$	$ru_2(y_a)$	$wl_1(y_b)$	$w_1(y_b)$	C_1	C_2

Each transaction obeys 2PL. It holds its read locks until just before it commits and holds its write locks until after it commits on both servers A and B. 2PC steps are omitted (since they don't change the example) as are unlock operations after commits.

The execution is not 1SR since in a serial execution on a one-copy database, one of the transactions would have read the other one's output, which did not occur in this execution.

The problem here is the early release of read locks. If read locks were held until after commit, then the execution couldn't arise.

Problem 3.

T is non-deterministic, writing different values with each execution.

If T is used for update propagation, replicas will contain different values for data item w. To remedy the non-determinism, time.now() can be replaced with a time value, such as Feb 29, 2012 12:00 am. Non-determinism should be resolved before execution on the resource manager, for example, at the client or in replication middleware.

The answer is unchanged when T executes multiple times. Ordinarily, it matters that transactions execute in the same order on both replicas. However, with T, the only effect of transaction order is the value returned by time.now(), which is the same problem noted above. However, if the system ran transactions that don't commute, then it would be important that they executed in the same order on both systems.