Problem 1

Suppose that we want to build a synchronous system of n nodes that can achieve Byzantine agreement in the presence of f Byzantine failure and t crash failure.

1. What is the minimum number of nodes required to achieve Byzantine agreement? Explain your answer.
2. Assume $f = 1$ and $t = 1$. Explain why at least 3 rounds are necessary to achieve Byzantine agreement in this case.

Answer:

1. minimum number of nodes is $3f + t + 1$. Achieving Byzantine agreement with f Byzantine failures requires that there are at least $3f+1$ nodes (out of which at least $2f+1$ are non-faulty). This problem states that there are also t nodes which might crash. Because the t crash failure nodes do not add to the number of invalid messages (messages that are sent from them are guaranteed to be valid), in the worst case, they will send no messages. As a result, the minimum number of nodes required is the sum of: $2f + 1$ (valid nodes), f (Byzantine failure nodes) and t (crash failure nodes), which equals $3f + t + 1$.
2. 3 rounds are necessary to achieve Byzantine agreement if $f = 1$ and $t = 1$. If there are a total of $F = f + t$ possible failures, $F + 1$ rounds are needed to ensure agreement among valid nodes. This is true because in the worst case, one failure could occur during each of the first two rounds. For example, the sender could experience a Byzantine failure in round one and one of the peers could experience a crash failure in round two. So, the value of the crashed node might be received by some nodes but not every node. So we need one more round. In this case, a third round is necessary to reach Byzantine agreement. In general, if there are a total of F failures, $F + 1$ rounds are needed to ensure Byzantine agreement.

Problem 2

State true or false with an explanation: If a sequentially consistent shared memory contains only 1 variable, then it is also a linearizable shared memory.

Answer: False. If a sequentially consistent shared memory contains only 1 variable, then it is not necessarily a linearizable shared memory. Linearizability implies sequential consistency, however, the reverse is not true. Sequential consistency for a replicated shared object service requires that there is some interleaving of clients operations that:

1. meets the specification of a single correct copy of the objects, end
2. is consistent with the program order in which each individual client executes those operations.

Linearizability requires that there is some interleaving of clients operations that:

1. meets the specification of a single correct copy of the objects, and
2. is consistent with the real times at which each operation occurred during the execution.

The key difference here is that sequential consistency is more relaxed and only requires that the ordering of each clients operations will be reflected in the sequentially consistent ordering while linearizability requires that operations are arranged in the single unique way that their execution actually occurred.

Problem 2 continued on next page...

The following is a counter example which is sequesntially consistent, but it is not linearizable.

P0: 2-write $x = 1$ 3-read x (which returns 1)

P1: 1-write $x = 0$ 4-read x (which returns 0)

$- >$

Physical time

This sequence is sequentially consistent, since $1, 4, 2, 3$ is the desired permutation, which respects the semantics of x and preservs the order of operations at the same node. On the other hand, the sequence is not linearizable, since to respect the semantics of x (which returns 0 from $P1$), event 4 must preced event 2. But this would violate the real-time ordering, because 2 precedes 4 in real time.

Problem 3

(Question 18.13 from the textbook-5th edition) In a gossip system, a front end has a timestamp (3,5,7) representing the data it has received from members of a group of three replica managers. The tree replica managers have vector timestamps (4,2,8), (4,5,6) and (4,5,8), respectively. Which replica managers could immediately satisfy a query from the front end, and what would the resultant timestamp of the front end be? Which could incorporate an update from the front end immediately?

Answer: The only replica manager that can satisfy a query from this front end is the third, with timestamp (4,5,8). The others have not yet processed at least one update seen by the front end. The resultant time stamp of the front end will be (4,5,8). Similarly, only the third replica manager could incorporate an update from the front-end immediately

Problem 4

Why Gossip-based system is not appropriate for updating replicas in near-real time? Provide an alternative approach.

Answer: Due to lazy approach to update propagation, a gossip-based system is inappropriate for updating replicas with near-real time. A multicast-based system will be more appropriate for this case.

Problem 5

In a replication system, the total number of servers is 4. 2 servers have an independent probability $p = 0.3$ of failing each, the $3rd$ server has $p = 0.5$ and the last one has $p = 1$. What is the availability of an object stored at each of these servers?

Answer: $(1 - p_1 \times p_2 \times p_3 \times p_4) = (1 - 0.3^2 \times 0.5 \times 1) = 0.955$ i.e., 95.5%