Exam Solution: Moed A

Question 1 (10 points):

You need to develop a replicated service to which clients communicate using the Web services standard so that a client is directed to one of the servers through TCP/IP. In case the server dies, the client needs to be redirected to another alive server.

Assuming that TCP/IP is implemented as a deterministic state machine inside the OS kernel, is it better to try masking a server’s failure by replicating the TCP/IP state machine (as well as transferring the IP address of the failed server to its backup), or to rely on reliable RPC to reopen the failed TCP connection and reissue the request to the backup server? Explain the benefits and pitfalls of each approach.

Answer: The main benefits of replicating the TCP/IP state machine is that it is completely transparent to the client and the fail-over can be made very quickly - in particular, no need to wait for the client to timeout and reissue the request.

The benefits of the R-RPC approach are: (a) no need to make any changes to the OS - not even drivers, which might not work across different versions of the Kernel, (b) can overcome TCP connections breakups that are not the result of a failure, which is why this functionality often exists in any case at the client’s middleware, (c) requires much fewer resources than TCP replication, since TCP state machine replication needs to be involved in each packet transmission and each timer expiration of TCP.

If there is absolutely no access to the client machine, and its middleware does not support reliable RPC in the first place, then TCP replication is better.

Otherwise, in the more common scenarios, R-RPC is usually the right solution.
Question 2 (45 points, 15 each):

Consider the implementation of Chandra&Toueg Consensus protocol as given below:

0) \( ts_i := (-1); \) \( r := (-1); \)
1) \( r := r + 1; \) \( c := r \mod n; \) \( ack := \text{ACK} \)
2) send(PH1,v_i,r,ts_i) to \( c \);
3) \text{if } c \neq i \text{ then goto line number 7 } \text{ endif;}

4) \text{wait until got a PH1 msg from at least half of the nodes (quorum);}
5) \( (v_i,ts_i) := (v_j,ts_j) \) for which \( ts_j \) is maximal for all received messages;
5.5) \( ts_i := r; \)
6) send(PROP,v_i,r,ts_i) to all;

7) \text{wait until either got a msg from } c \text{ or } c \text{ is suspected;}
8) \text{if got a msg then } (v_i,ts_i) := (v_c,ts_c) \text{ else } ack := \text{NACK } \text{ endif;}
9) send(PH2,v_i,r,ts_i,ack) to \( c \);
10) \text{if } ack := \text{NACK then goto line number 1;}
11) \text{if } c \neq i \text{ then goto line number 16 } \text{ endif;}

12) \text{wait until got a PH2 msg from at least half of the nodes (quorum);}
13) \text{if got a majority of ACKs then } \text{r-cast(DEC,v_i,r,ts_i) to all}
14) \text{else send(NEXT,r) to all}
15) \text{endif;}

16) \text{wait until either got a msg from } c \text{ or } c \text{ is suspected;}
17) \text{if the message is (DEC,v_c,r,ts_c) then}
18) \text{decide } v_c \text{ and return}
19) \text{goto line number 1}

i) Why is it OK to skip lines 2, 4, and 5 during the first round of the protocol?
ii) Why are these lines required in other rounds (r>0)?
iii) When comparing the communication pattern of Paxos and C&T, we can see that in Paxos the coordinator (during rounds r>0) sends a message to all other nodes before the equivalent of Line 2 in C&T. Why is it needed in Paxos?

Answers:

i) According to the validity requirement of the Consensus problem statement, it is legal to decide on any value proposed by any process. Thus, the initial value of the coordinator is a valid value and there is no need to consult anyone. Looking syntactically at the protocol, it is clear that in the first round all \( ts_j \) values are -1, so any \( v_j \) value can be chosen.

ii) In more advanced rounds, there is a risk that some process reaches a decision state in a previous round (line 13 or 18, depending on its role). Hence, the coordinator needs to give precedence to such a value, if exists, by consulting a quorum before proposing any value. Otherwise, it could violate agreement.

iii) In Paxos, processes consult the failure detector (of type \( \Omega \)) only at the beginning of a round, whereas in C&T they consult the failure detector (of type \( <>S \)) in each wait loop. Also, the coordinator in C&T is chosen deterministically based on the round number whereas in Paxos each process can potentially declare itself a coordinator of a given round (and also round numbers are not necessarily consecutive). Thus, the coordinator needs to notify at least a quorum of nodes about its existence and have this quorum nodes cooperate with it.
**Question 3 (10 points):**

Present a solution to the Consensus problem that is based on a uniform total ordering primitive. Write down the signature (methods) of the uniform total ordering service you are using and provide a pseudo-code for it. There is no need to implement the uniform total ordering mechanism - you can simply assume it exists.

**Answer:** There are many options here. In order to solve a single instance of Consensus, we can use the protocol from the home assignment:

```java
Semaphore validReturnValue = new Semaphore(0);
object result = null;
Boolean firstValue = true;

object Consensus(object v) {
    UTOBcast( v);
    validReturnValue.wait();
    return result;
}

synchronized UTODeliver(object m) {
    if (firstValue) {
        firstValue = false;
        result = (m);
        validReturnValue.signal();
    }
}
```

In order to support multiple instance, there is a need for an instance counter while result and the semaphore need to become arrays. A simpler polling based solution is as follows:

```java
object Consensus(object v) {
    static int myInstance = 0;
    int conInstance;
    object result;
    UTOBcast( (v,myInstance++));
    loop {
        (result,conInstance) = UTOReceive();
        if (conInstance == myInstance) then
            return result;
    }
}
```
Question 4 (15 points):

It is proposed to change the distance metric in the Kademlia protocol such that the distance between two points is defined as the absolute algebraic difference between them, that is \( d(x,y) = |x-y| \). This is instead of the XOR function used today.

How would this change affect the Kademlia protocol? In particular, refer to the lookup operation, the content of the k-buckets, initialization, etc. Describe at least one drawback of this new metric function compared to XOR.

Answer: The most significant difference is that the new function is not unidirectional, that is, for a given point \( x \) and distance \( d \), there are two points \( y' \) and \( y'' \) such that \( d(x,y') = d(x,y'') \) yet \( y' \neq y'' \).

Using the new function could compromise the protocol’s correctness, since now it is not guaranteed that a joining node would be able to get inserted into the routing table of another node. This is unlike XOR, in which it is ensured that the node would have room in the table of the k-bucket of its closest node. With the new function, this k-bucket could be full. Consequently, this new node would be isolated.

Even when nodes can be found, with the new function the triplets returned for a query during the search could arrive from different k-buckets. Hence, the search process would be longer, as it is no longer guaranteed that each search step brings us exponentially closer (one bit) to the target.

Finally, loss of the unidirectional property hurts the ability to employ effective caching of search results.

Question 5 (20 points, 10 each item):

i) How is the split brain behavior phenomenon being avoided in BigTables?

Answer: In BigTables, each tablet is served by a single tablet server at a time. A tablet server must maintain a lock (lease) in Chubby, which prevents multiple tablet servers from serving the same tablet. Similarly, there is a single master server, which must also maintain a lock in Chubby. Finally, the root of the tablets is stored in Chubby as well. Chubby is assumed to be a safe lock manager, which does not suffer from split brain. The persistent data itself is stored on GFS, which is also assumed to be resilient to split brain.

ii) What would have happened in Dynamo without the W+R>N requirements? When does this matter?

Answer: Without the W+R>N requirement, even without failures it would have been possible for a read to return old values of an object, since a read could return after obtaining replies from replicas that do not intersect with the ones that were involved in the last write. While this situation is currently possible when there are failures and network problems, the latter are relatively rare. Without the W+R>N, it could happen on every read!