Fabien Bessez
Homework 2
Distributed Systems

1. **How does an RPC call differ from a simple function call in a shared-memory system? What extra steps are necessary?**

   An RPC call differs from a simple function call in a shared-memory system in that the procedure definition does not exist on the client's computer, hence the term 'remote'. There are plenty of extra steps necessary when calling an RPC than a simple function call in a shared-memory system. First of all, the client must format its request in such a way so that the server that contains the function definition can properly use the arguments that the client used. For example, add(m, n) might be formatted as [‘add’, m, n]. This helps because the server will always read the first index of the request list as the function name and the following two as the two arguments. Furthermore, the client must convert the reformatted procedure call as a byte-stream so that it can be passed over a network. After the client stub has properly formatted the procedure call, a network connection must be established between the local OS and the remote OS (server). The server then unpacks the message and figures out which function the client wants to run. The server executes the function with the given arguments and returns a result to a stub function that will re-format that result so that it can be passed over the network. The stub function must also ensure that the results are in such an order that the client can understand what the value of the result is. For example, if the RPC was add(1,2), the server might return a byte-string representing [200, 3] where 200 is a status code that the function was executed successfully and the 3 is the value of add(1,2). The client's computer then has to use a stub function that extracts the desired value from the message sent by the server. Finally, after all those extra steps, the client has access to the result of the RPC.

2. **What kinds of data types can be marshaled and which cannot? Give two examples of each.**

   Marshaling is the process of converting an object in memory into a format that can be written to a disk or transmitted over a network connection. Scalar data types (Booleans, integers, strings…) can be marshaled whereas non-scalar data types like lists, tuples, dictionaries, sets and other user defined classes can't be marshaled.

3. **Consider the following two threads, executed in parallel. Assume that all variables are initialized to 0. Enumerate all the possible final values of x, given arbitrary interleavings of these instructions.**

Listing 1: Thread 1

```
x+=1
```

Listing 2: Thread 2

```
x*=2
x*=3
```

There are three possible ways that these two threads can be executed. Let us say that "x += 1" is on line 1, "x *= 2" is on line 2, and "x *=3" is on line 3. These 3 lines can be executed in the following orders: a) line1, line2, line3 b) line2, line1, line3 c) line2, line3, line1. Let's enumerate each of the possible values of x. In all cases x = 0 to begin with. In case a) line1 is performed first, resulting in x = 1. Then line2 is performed, resulting in x = 2. Then line3 is performed, resulting in 6, the final value. In case b) line2 is performed first, resulting in x = 0. Then line1 is performed, resulting in x = 1. Then line3 is performed, resulting in x = 3, the final value. In case c) line2 is performed first, resulting in x = 0. Then line3 is performed, resulting in x = 0. Then line1 is performed, resulting in x = 1, the final value. In the three cases, we got three different final values: 6, 3, and 1. This means that the output of these two threads is non-deterministic!

4. **Imagine a distributed system in a universe with no network faults. Can the system still become inconsistent? How?**

   I believe that this question asks us to consider the CAP theorem when Partition-tolerance is not a concern and therefore not prioritized when having to choose two of the three properties. This means that C (atomic consistency) and A (availability) can be prioritized. The system can still become inconsistent in a universe with no network faults because inconsistency can arise from system faults as well. Even when a network is working perfectly, the systems that are communicating might fail. For example, in a universe with no network faults, the computers that use these networks might be burnt in a fire; if you're computer is burnt in a fire, it is unlikely that it can communicate using the perfect networks.

5. **Imagine a version of Gmail that lacks the Durability property. What kind of behavior would you expect from this system? Give an example of a behavior illustrating the lack of Durability.**

Fabien Bessez
Homework 2
Distributed Systems

The durability property states that the state of a system will survive, even in the case of crashes. If a version of Gmail lacked this property, one could expect e-mails to go missing whenever the system crashes. In other words, if Gmail lacked the durability property, one could imagine a scenario in which I could send this homework to you, Gmail would then crash, and then you would have no knowledge of me ever sending this email. Durability requires that once a transaction has been completed, its effects are permanent.