Addis Ababa Institute of Technology

Reflection Report

Fundamentals of Distributed Systems

Nathan Mesfin Shiferaw
UGR/0534/14
Software Stream

Reflection Report

Discuss how RPC simplifies communication compared to socket programming.

Using RPC, procedures on remote machines can be called as if they are located locally. This massively simplifies the organization of distributed systems primarily because of how it abstracts the inner workings of sockets. Developers working in a distributed environment need not worry about the implementation on the server side, only the interfaces through which they communicate. Additionally, using socket programming means that the server and client stubs must be written manually. This includes any data serialization that is necessary along with whatever service interfaces are available. Moreover, maintaining a distributed environment that relies on RPC is easier as it abstracts most of the complexity in using a client-server model.

Describe challenges encountered (e.g., handling timeouts, concurrent clients).

There was only one challenge that was encountered in this lab, which was the handling of timeouts in the client code. I couldn't find a way of adding a context deadline like with socket programming. After trying a few alternatives with the server code, I ultimately ended up using the implementation in the lab 5 document. This used a *select* block with channels to make sure that the request arrived before the deadline.

```
call = client.Go("Calculator.GetLastResult", &args, &reply, nil)
select {
case <-call.Done:
    if call.Error != nil {
        log.Println("RPC error:", call.Error)
    } else {
        fmt.Printf("Stored result after method invocation: %d\n", reply)
    }
case <-time.After(2 * time.Second):
    log.Println("RPC call timed out")
}
```