CSE5306- Distributed Systems
Project1-Report

Team Members-
Megha Modi – 1001750561
Nishank Gujar – 1001861756

I have neither given nor received unauthorized assistance on this work
Signed: Megha Modi Date: 07 Oct 2021
Signed: Nishank Gujar Date: 07 Oct 2021

Implementation

We used Python3.8 as the programming language for the implementation. For the inter-process communication, socket library was used. Python sockets supports a number of address families under the network layer protocol. We created a TCP/IP protocol using socket.socket and used AF_INET for IPV4 for network addressing. time library was used for sleeping to mock work being done asynchronously, the threading & _thread libraries to handle threading and the json library to help serialize and deserialize arrays.
RPC implementation was done using the json library as it is stable across different platforms and lightweight as compared to others.
- Asynchronous RPC For this RPC, the server and the client were kept busy until the client performs a lookup query for the result on the server. The result of the same was stored in a dictionary as the time complexity of search operation or lookup would be constant or O(1). Resetting the table values after the operation is done to ensure that previous values are not accidentally read.
- Deferred Synchronous RPC Its a single request -response RPC which is split into two RPC's. The client triggers a asynchronous RPC on server, on completion, the server calls-back client to deliver the results.We used two threads, one for continuing to execute whatever the client was doing and the other one to provide a channel for an interruption from the server.

What we Learned

Serializing data is one of the most important aspects while implementing RPC. When using complex data types such as arrays or structs, passing the data through sockets is not an easy process. In our case, serializing - converting these objects to strings using

the json library was important for being able to pass the data from the client to the server. Understanding the difference between blocking and non-blocking calls and when to use the former or the latter. It helped us understand the concept of inter-process communication.

The Challenges We Faced

The conversion of a single threaded to a multithreaded server was actually relatively easy for us. We think this was a result of modularity of our server implementation which lent itself to the modification. Even though, we did found out that it would not be easy to implement distributed locking preventing dead-locks or race-conditions. This would be necessary if more than one client needed to read from or edit the same file simultaneously. Another challenge was how to handle the interruption from the server to the client in the deferred synchronous RPC. After considering and trying a number of options, we eventually settled on using two threads — one for the client server to continue executing whatever it was executing and another one - waiting for the server to interrupt. The other challenge here was how to communicate the interruption to the other thread so that we can interrupt the server and print the result.
Debugging errors when trying to get a response from the server and client getting stuck. One of the other challenges was getting stuck even after uploading the content/ download completion. The solution for this issue was - timeout limit and catching the error after waiting for a certain amount of time or upon the file transfer completion.

Conclusion

The project was challenging but gave a lot of practical knowledge while implementation. The concepts taught in the class were put to work while implementing this project.