UNIVERSITY OF TEXAS-ARLINGTON

UNIVERSITY OF
TEXAS
ARLINGTON

PROJECT-1 REPORT

Submitted in partial fulfilment of the requirement for the course-work
CSE-5306-003
DISTRIBUTED SYSTEMS

SUBMITTED BY

JAYACHANDRA JARAJAPU (1001964536)

PRUDHVI GURRAM (1001955767)

DECLARATION

I PRUDHVI GURRAM bearing UTA ID: 1001955767 and JAYACHANDRA JARAJAPU bearing UTA ID: 1001964536, of the Department of Computer Science, University of Texas at Arlington Confirm that we have neither given nor received unauthorized assistance on this work.

Prudhvi Gurram

Jayachandra Jarajapu

09/23/2022

INTRODUCTION:

The Requirement of the project is to build an upload and download service and a computation service using remote procedure call (RPC). The file server supports four operations such as upload, download, delete and rename of the file. We also created a computation server where we perform addition and sorting operations. We use multi-threaded file sharing system which performs all the above operations to the file.

IMPLEMENTATION OF FILE SERVER:

For part-1&2, we were asked to design a multi-threaded file system which can perform basic operations to the file. We have implemented a file sharing system using RPC communication. The execution operations are present in the server and the server is connected with an ip address and port. The client makes an rpc call and sends the instructions to be performed to the server, the server then executes the instructions and send back to the client with results. To perform this operation, the client will use the same ip address where the server is open and get the results with the adequate buffer size. If the data is more than the size then the RPC makes two calls to the server for the remaining result.

The program has been implemented using socket programming in python and we have used threading in python as well.

WORKFLOW:

- First we start a server which has execution instructions of the client. The server connects and listens on the ip address and port number and is open to requests.
- We have given the input operation as a command line argument to the client file so that the client takes the arguments as a input and parse the input and performs the required operation.
- We have created two folders client_data and server_data to handle client files and server files
- Initially we upload a local file to the server folder using CLI and later we can perform other file operations for server folder
- The client then sends the rpc call to the server to perform the operation using socket and is connected with server ip and waits for the result.
- The server sends back the results to the client and a response is updated in the server that the file operation is completed

- We use pyftpsync for synchronizing the folders between the client and the server. Once we run the server the connection is established and client file is run and the client is connected to the port with the targeted folder sync(server-sync).
- UTF-8 encoding is used to completely replace the file once the file is altered or removed.
- The synchronization is bidirectional and the requests are sent to the server are to the client using meta json file and this file checks for the latest dump in the folder catching the time and date and then sends a new file to the server to update the file
- The synchronization thread checks folders for every few seconds and if there are any changes made to the folder based on time and date, the request is sent to the server to completely replace the file and sends the acknowledgement to the client

IMPLEMENTATION OF COMPUTATION SERVER:

Part-3 of the project were asked to implement a computation server which performs addition and sorting of numbers. We have implemented a RPC server to perform synchronous and asynchronous RPC. we used separate classes for both RPC's because each has to call invoke and get the result from the server. The server file consists of computation instructions which performs the operations and sends back to the client with results.

For both RPC's we give input as command line arguments and the arguments are parsed using argparse and performs the required RPC. the client decides whether to perform an sync or async RPC of the program.

WORKFLOW:

Synchronous rpc:

- In Sync RPC, the client sends the request to the server and it is immediately processed and the result is sent back to the client.
- The input is given in the form of command line arguments and the client decides which rpc to perform in cli.
- We use argparse to parse the arguments from the client to either sync or the async rpc.
- In the client program, we have implemented async and sync rpc using different classes and these rpc's have invoke and result functions which will send a request to the server to perform the computation on the server side.

- Initially, The rpc call is sent to the server to perform the computation and the acknowledgement is sent back to the client with computation ID and a token is generated.
- Once the token is created, the client uses the token to get the result from the server and show the required result on the client side.

Asynchronous RPC:

- In Async RPC, the client makes the request to the server and once the request is sent it performs other computations instead of waiting for the result from the server
- Since the process does not block, We have created a function which performs generating random integers on the client side rather than waiting for the original call.
- once the operation is completed, the client makes another remote procedure call to the server using the token generated and the server sends the result back to the client using the computation ID
- In async RPC, the client makes two calls to the server, one for receiving the acknowledgement of the request and another to get the result from the server.

ISSUES ENCOUNTERED:

- Python socket programming are web sockets which just transfers bytes of data and there is no way to check the initial and end points of the file. Hence we have limited the buffer size of the file and limit to one file transfer for one call.
- To implement a file synchronizer, we had a little knowledge about python sync libraries and had to learn about the process and use the packages accordingly
- Python socket libraries doesn’t naturally identify events to be executed so we had to create custom classes to check which operation client wants to perform and execute those instructions.
- Instead of importing existing rpyc calls we had to write own python sync and async rpc classes for the computations. We had to make two remote procedure calls to the server for the async RPC using computation ID and tokens.

Team Collaboration:

Prudhvi Gurram:

- Worked on part-1 of the project
- Learnt encode and decode of the data in sockets between client and server and used it for file transfer for part-1
- Implemented synchronous RPC for part-3
- Prepared the report for the project

Jayachandra Jarajapu:

- Worked on part-2 of the project
- Learnt about directory synchronizers which include upload, download and bidirectional synchronizers and implemented in part-2
- Implemented asynchronous RPC for part-3
- Prepared the readme files for all the parts of the project.

KNOWLEDGE WE GAINED:

- After implementing the project, we gained knowledge about how to implement RPC between client and server and how RPC works over the network.
- We learnt about python socket programming and how we can use this knowledge to perform file operations and computation service
- We tried and learnt how asynchronous and synchronous RPC's work and what differs it shows when we are using it in a real-time environment.

References:

Jennings, Nathan. "Socket Programming in Python (Guide)." *Socket Programming in Python*, Real Python, 21 Feb. 2022, https://realpython.com/python-sockets/.

Tomar, Nikhil. "File Transfer Using TCP Socket in Python3 - Idiot Developer." *Medium*, Medium, 10 Nov. 2021, https://nikhilroxtomar.medium.com/file-transfer-using-tcp-socket-in-python3-idiot-developer-c5cf3899819c.

Rodola, Giampaolo. "Tutorial¶." *Tutorial - Pyftpdlib 1.5.4 Documentation*, 2009, https://pyftpdlib.readthedocs.io/en/latest/tutorial.html.

Wendt, Martin. "Release 1.0.3.Dev-20150609 - Media.readthedocs.org." *Pyftpsync Documentation*, Buildmedia.readthedocs.org, 9 June 2015, https://media.readthedocs.org/pdf/pyftpsync/master/pyftpsync.pdf.