# Report Project 1

# Implementing Client and Server using RPC

## Project Overview:

This project consists of client and server which communicates with each other using RPC to showcase multithreading, Sync and Async operations. The server constitutes of 2 parts, file handling and computation.

## Implementation:

It contains two types of server: FileHandling Server and Computation Server.

Server in Part 1 and Part 2 of the project code focus on FileHandling which is implemented using *Java RMI* API. Server in part 3 provides the functionality of computation server to ADD and SORT on numbers which is implemented using gRPC and also provides sync and async operations on ADD and SORT.

**Part 1**: This has been implemented using the FileInputStream class to read the files in bytes and FileOutputStream class to write the bytes in a class. This has been used for *download* and *upload* call. Java File class is used for *rename* and *delete* calls of a particular file. The client folder to download files and server folder to upload files has been made configurable and can be changed in the client1config.properties and server1config.properties file.

**Part 2:** For synchronizing two folders, the implementation on client side runs a thread which runs after every 1 min to send the call to the server for any changes in the client watched folder. The schedule time, client folder and server folder has been made configurable and can be changed in the client2config.properties and server2config.properties file. The method called by thread checks the modified date of a file and if there is any change since the last scheduled run then it updates the changes of *rename, delete and new file addition* to the server side. Server has all the methods available for making the sync to the output folder.

**Part 3:**

a. **Synchronous:** To implement synchronous calls for addition and sorting of user provided input, **Unary communication channel** was created between the server and the client. Once the protoc (compiler) generated stubs from the services defined in .proto files, we use the **BlockingStub** to prevent concurrent submission of requests from client.
b. **Asynchronous:** to implement async calls for addition and sorting of user provided input**, Bi-directional streaming communication channel** was created between the server and the client. Once the protoc (compiler) generated the stubs, we used the **newStub** to allow concurrent exchanges of request, acknowledgements and responses between client and server.

## Learning:

1. Designing a server and client application.
2. **Java RMI:** This is a Java API which is used for Remote Method Invocation from one JVM to another JVM. Servers implementing FileHandling methods are using Java RMI to invoke methods on JVM.
3. **File Handling:** FIleInputStream and FileOutputStream class has been used for file handling operations of file upload, download, rename and delete operations. These classes use bytes to read data from files.
4. **Threads:** Threads are used for executing small set of instructions. Java RMI implements multithreading in background due to which same server can handle multiple client request at same time. In this project thread has been implemented for file folder synchronization between client and server in part 2.
5. **gRPC:** It is an open-source framework which can run on any environment. It uses RPC infrastructure for invocation of methods between client and server. gRPC is a highly efficient language independent framework that provides unary and streaming(client, server and bi-directional) functionalities.
6. Team collaboration.

## Issues encountered

1. Creating and running the registry for RMI: first we were trying to start the registry using the command line which ran the registry on default port 1098. Now server was able to find the registry on default port and started successfully but client was not able to do lookup for the name bind by server because it was not able to find the path of the class bind to the registry. So we created the registry from the code which started the registry from current directory which resolved the path issue.
2. changing the buffer size on server side was causing issues while downloading the file on client as the size of buffer was fixed on client side. It was fixed by creating an DownloadDataType class which send the bytes read along with the buffer size used.
3. gRPC setup: GRPC setup consumed a lot of our time as the versions of the plugins used were not in sync.
4. Not able to retrieve acknowledgement with Unary GRPC calls, had to implement bi-directional gRPC.
5. gRPC is not well documented, being a new technology, a lot of information regarding streaming and callbacks was missing.

## Labor Division:

| | |
|---|---|
| Akanksha Tomar : | Worked on implementation of part 1 and part 2. |
| Devyani Singh : | Worked on implementation of part 3 (Synchronous and Asynchronous) |

## References:

https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html

https://grpc.io/docs/languages/java/quickstart/

https://www.youtube.com/c/Devoxx2015

https://www.youtube.com/watch?v=9ttYs4dXc9k&list=PL1F54YqZCTdxpGWMM7zyOigOQ_MVwSNc2

---

## Prepared By:

**Akanksha Tomar - 1002039215**

**Devyani Singh - 1001959376**