Multithreaded Server Key-Value Store using RPC

Server-client implementation in this assignment serves as a key value store. A key-value store is a storage system where each value (string) is storage against a unique key in the system. In a distributed system, key-value store is used in several scenarios of which one important role is to identify the different peers associated with the server. In aforementioned example, the key-value store works against a single master server which stores network addresses in its key-value store. But in this environment, the server acts as a multi-threaded system. What this means for the server is at any point multiple clients can interact with the server and perform the operation. This setup highly increases the efficiency and achieves the task of a distributed file information storage.

Mechanism

The Server and Client communicate through Remote Procedural Calls (RPCs) and the mechanism used in this project to implement RPCs is Apache Thrift. I have chosen Apache Thrift for the reasons that its multiple cross language references and has explicit support to RPCs. Moreover, the option for Server and Client to interact through secure TTLS communication channel provides an implicit security for the architecture.

Model

In this assignment, the key-value store in the server is represented by a Map of values. In this map, the server uniquely stores the keys associated with an individual client. This service implementation is implicitly generated by thrift mechanism and acts as the skeleton for the RPC.

Client is designed to take user inputs to perform any operation associated with the implementation. It is also designed to robust to handle mis-inputs from the user and provide descriptive messages to understand the error.

Client also shares the same service Skelton and uses the method stubs to invoke the RPCs calls for the server.

Implementation

The client interacts with the server using the connection address passed to it from the command line arguments. Similarly, the server initializes the connection using the details passed to it from the command line arguments.

The client and server share multiple operations among them. Each communicate with each other over the network using serialized data (unencrypted). This allows them to share objects between each other and establish a more object-oriented data flow between them. Multiple operations are:

- PUT
  - Used to save keys with the server
- DELETE
  - Used to delete the key from the server
- GET
  - Used to retrieve the key and its value from the server.

Client handles exceptions and resets to ask the user to re-enter the input again. The client adds a new functionality called GET KEYS which is used to retrieve the list of available from the server and display it to the client. It also saves this information locally in its instance. The data between server and client is serialized and this process right before the data is sent over the network. This gives the instances the flexibility to perform operations.

LOGGING

The server and client log their information in the log folder associated with the project. Each model of implementation stores the log in their respective locations and distinguished by their names. Each log is timestamped, and each type has its own purpose.

- INFO
  - This type of log informs the user about occurrence of a pre-defined actions.
- WARN
  - This type of log informs the user about occurrence of non-pre-defined actions.
- SEVERE
  - This type of log informs the user about the runtime exceptions that occur during the program execution.

Almost similarly, each action, warning and error is also printed to the screen in a human-readable way for both client and server.