**Northeastern University – Silicon Valley**

CS 6650 Scalable Dist Systems

**Project #2** [200 points]

**Multi-threaded Key-Value Store using RPC**

# Guidelines

Project #1 should be electronically submitted to Blackboard by midnight on the due date. A submission link is provided.

# Assignment Overview

For this project, you will extend Project #1 in two distinct ways.

1) You need to enable your client and server to communicate using Remote Procedure Calls (RPC) instead of sockets.  If you’ve implemented Project #1 in Java, you may want to look into and leverage Java RMI for RPC communication.  However, there are multiple other RPC frameworks you can leverage (with their own IDLs) to provide the stubs/skeletons necessary across the network.  An additional example that enables the use of multiple languages is Apache Thrift. (<http://thrift.apache.org/>

[(Links to an external site.)](http://thrift.apache.org/)

2) You need to make your server multi-threaded such that you can handle multiple outstanding client requests at once.  You may decide how to thread your server.  One approach may be to use thread pools similar to other servers, although there are certainly many ways to do this. The key result is that your servers should be able to handle requests from multiple running instances of you client doing concurrent PUT, GET, and DELETE operations.  Due to the addition of multi-threading, you will need to handle mutual exclusion.

As in project #1, you should use your client to pre-populate the Key-Value store with data and a set of keys.  The composition of data is up to you in terms of what you want to store.  Once the key-value store is populated, your client must do at least 5 of each operation: 5 PUTs, 5 GETs, 5 DELETEs.

**Evaluation**

Your multi-threaded Key-Value Store server will be evaluated on how well they interoperate with each other using RPC while doing concurrent operations. The grade for your executive summary is based on the effort you put into the assignment overview and technical impression. In general, if you put some effort into your writing, you will receive full credit for your executive summary (provided that it is properly formatted and submitted as a plain text file).

**Executive Summary**

Part of your completed assignment submission should be an executive summary containing an “assignment overview” (1 paragraph, up to about 250 words) explaining what you understand to be the purpose and scope of the assignment and a “technical impression” (1–2 paragraphs, about 200–500 words) describing your experiences while carrying out the assignment. The assignment overview shows how well you understand the assignment; the technical impression section helps to determine what parts of the assignment need clarification, improvement, etc., for the future.

Assignment Overview:

This assignment was focused on creating a single pair of client-server applications using Remote Procedure Calls which I implemented using JAVA RMI and making it multithreaded so that the server can handle multiple client requests at the same time and thus handling mutual exclusion. Comparing it to the first project, I found it much easier to implement than the Socket based implementations using TCP and UDP. So the purpose of this assignment according to me was to get understanding of how middleware helps to improve the coding aspects of distributed system. This was just one level of the middleware that we used and using the proper stack of middleware can help building complex distributed applications quite easily. This project also focused on making the server thread safe, thus enabling us to debug the issues of thread synchronization thus helping to handle multiple clients at the same time, which makes the project closer to a real world distributed system than the one implemented in the first project. Furthermore, the use of docker platform became much clearer by creating and using docker images and containers for this project as well. Hence the main aim of the project was to further improve upon the building of complex distributed systems using already built services and middleware that help productivity.

Technical Impression:

Even before starting the project, as I read the homework document, I had some questions such as how to create a simple protocol on our own to communicate between client and server. It was not clear if we must create it using XML or some other protocols. In regard to the unrequested datagram packets, as it is a simple interaction of client and server, I found it difficult to understand how to test this kind of cases. As a result, help from TA was required to clarify some of the requirements.  
  
After starting the assignment, I was unsure which of the Tanenbaum’s client-server organization to follow. I first had checks of client input on client side, but the code was getting redundant as the same check was applied on server side. I eventually ended up using the 2nd organization, only the user interface on client side and the processing (including all checks) and data layer on the server side. Since the clients and the servers have some methods in common, I decided to add AbstractClient and AbstractServer that each client and server can extend. The differences in TCP and UDP clients and servers is the way connection is established (in TCP), Datagrams created (in UDP) and how the data is sent in both protocols. All the backend APIs are written in their respective Abstract Classes. I even created separate log files of each application that store the logs present in the terminal. After running the applications locally, major issues occurred during the creation of Dockerfile and bash files to run docker images. Since it was a new concept, it took some time and tutorials to understand the importance of docker and how to use it. The resources provided to understand docker including sample Dockerfile and scripts were highly useful to create and modify my own Dockerfile the way I want it to run.  
  
In summary, this project required to make design choices, and after the project was completed, I was satisfied with the way I implemented the requirements and how the applications work.

I considered it as a relatively easier project since I reused most of the code that I wrote in first project with refactoring wherever necessary. For example, the protocol of taking input was same as before, and the conditions to check the proper input format and the outputs remained the same. Since this project required to use RMI and the how to use RMI was already covered in a homework assignment, it was easier to implement. Overall, I found that use of JAVA RMI (and thus the middleware) is very useful to create distributed applications.

For the multithreaded part also, the JAVA RMI already accepts multiple clients. So, the only thing to implement was to make the calls to database, thread-safe. I initially wrote all the 3 methods: GET/PUT/DELETE in the database individually synchronized which I tested by adding random sleeps on each of the call. But it led to a race condition for the sequence when delete is called before get by other client but gets executed before it. So, to handle such cases I created a single lock for all the three methods, which solved the issue.

Overall, I believed it was a simple assignment just on the facts that much of the implementation work was already covered in first project and homework assignments which helps to understand the concept of remote procedure calls and multithreaded in a better way.

The grade for your executive summary is based on the effort you put into the assignment overview and technical impression. In general, if you put some effort into your writing, you will receive full credit for your executive summary.

**Project Deliverables**

The following items should be archived together, e.g., placed in a .zip file or tarball file (\*.tgz or \*.tar.gz), and electronically submitted via the link is provided on the course Moodle page.

1. All novel Java, C, and/or other source code files implementing the client and server programs, i.e., plus any additional support code.
2. A simple README that includes
   * 1. How to build your server and client codes (including any external libraries necessary)
     2. How to run your server and client programs
     3. Your executive summary

**Other notes:**

You should use your client to pre-populate the Key-Value store with data and a set of keys. The composition of the data is up to you in terms of what you want to store there. Once the key-value store is populated, your client must do at least five of each operation: 5 PUTs, 5 GETs, 5 DELETEs.