Northeastern University

CS 6650 Scalable Dist Systems

Project #1 [200 points]

Single Server, Key-Value Store (TCP and UDP)

Guidelines

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

Assignment Overview

For this project, you will write a server program that will serve as a key value store. It will be set up to allow a single client to communicate with the server and perform three basic operations:

1) PUT (key, value)
2) GET (key)
3) DELETE (key)

A Hash Map could be used for setting up Key value stores. See
For this project you will set up your server to be single-threaded and it only has to respond to a single request at a time (e.g. it need not be multi-threaded – that will be part of Project #2). You must also use two distinct L4 communication protocols: UDP and TCP. What this means is that your client and server programs must use sockets (no RPC….yet, that's project #2) and be configurable such that you can dictate that client and server communicate using UDP for a given test run, but also be able to accomplish the same task using TCP. If you choose, you could have two completely separate sets of applications, one that uses UDP and one that uses TCP or you may combine them.

Your implementation may be written in Java. Your source code should be well-factored and well-commented. That means you should comment your code and appropriately split the project into multiple functions and/or classes; for example, you might have a helper function/class to encode/decode UDP packets for your protocol, or you might share some logic between this part of the assignment and the TCP client/server in the next part.

The client must fulfill the following requirements:

- The client must take the following command line arguments, in the order listed:
  - The hostname or IP address of the server (it must accept either).
  - The port number of the server.
- The client should be robust to server failure by using a timeout mechanism to deal with an unresponsive server; if it does not receive a response to a particular request, you should note it in a client log and send the remaining requests.
- You will have to design a simple protocol to communicate packet contents for the three request types along with data passed along as part of the requests (e.g. keys, values, etc.) The client must be robust to malformed or unrequested datagram packets. If it receives such a datagram packet, it should report it in a human-readable way (e.g., "received unsolicited response acknowledging

unknown PUT/GET/DELETE with an invalid KEY" - something to that effect to denote an receiving an erroneous request) to your server log.

- Every line the client prints to the client log should be time-stamped with the current system time. You may format the time any way you like as long as your output maintains millisecond precision.
- You must have **two instances** of your client (or two separate clients):
    - One that communicates with the server over TCP
    - One that communicates with the server over UDP

The server must fulfill the following requirements:

- The server must take the following command line arguments, in the order listed:
- The port number it is to listen for datagram packets on.
- The server should run forever (until forcibly killed by an external signal, such as a Control-C, a kill, or pressing the "Stop" button in Eclipse).
- The server must display the requests received, and its responses, both in a human readable fashion; that is, it must explicitly print to the server log that it received a query from a particular `InetAddress` and port number for a specific word, and then print to the log its response to that query.
- The server must be robust to malformed datagram packets. If it receives a malformed datagram packet, it should report it in a human-readable way (e.g., "received malformed request of length n from <address>:<port>") to the server log. Do not attempt to just print malformed datagram packets to standard error verbatim; you won't like the results.
- Every line the server prints to standard output or standard error must be time-stamped with the current system time (i.e., `System.currentTimeMillis()`). You may format the time any way you like as long as your output maintains millisecond precision.
- You must have **two instances** of your server (or two separate servers):
    - One that communicates with the server over TCP
    - One that communicates with the server over UDP

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.

Evaluation

Your single-threaded Key-Value Store client and server will be evaluated on how well they interoperate with each other and with the sample client and server provided, as well as their conformance to the requirements above.

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. Provide a use case (application)

where you would apply this in practice. 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.

Evaluation

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).

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 source code files implementing the two client and two server programs, i.e., plus any additional support code
2. Your executive summary.