**ENGR 516: Assignment 1 Report**

**Date-** 9/20/2020

**Design specification:**

**server.py:** This module contains the code required for running the server.

**Memcache Server:**

Considering generality, I implemented the server attributes and functions in a class. This is could help me create more servers with required attributes later in the assignments.

I create a server by instantiating the Memached class. While creating the object I pass the required parameters- HEADER (which specifies the initial length of the message that a server can accept), SERVER\_IP, PORT, FORMAT (which specifies the compatible text-format).

After creating the object, I run the server, and listen to connections from clients. I have used the threading module in python to handle concurrent requests from clients.

I have segregated each operation of the server into different methods in the class- The send method handles the operations for sending a message to the client. Similarly, the command handler method parses the message from the client.

**memcacheAPI.py:**

I have created a separate module to handle the memcache operations. The module has two methods- set and get. The set method validates the user input and stores the key, value, and bytes as a space delimited line. Writing to the file operation is Atomic; since this operation is not a thread safe, I have used the threading.Lock method to ensure it is performed by one thread at a time. This would help in maintaining consistency. To test this- you can create concurrent sessions and try setting the value of the same key.

I append the data to the file if the key is valid and is not present in the text file. But, if the key is present, I update the value and bytes.

**client.py:** This module contains the code required for creating and running a client. In the future assignments, I plan to implement the client attributes and functions inside a class. Right now, this module contains the driver code to run a client session and to send, receive, or parse message from a client. To create multiple clients manually, you can run this module in multiple terminal windows.

**db.txt:** This file contains the key value pairs. Initially, it would be empty.

**clientTest.py:** This file contains the test case to create multiple client sessions and send requests to the server. I created a test case to send 500 set requests to the server, and check the keys are getting stored.

**Experimental Evaluation:**

I tested on 10-12 concurrent sessions both in my local machine and in the burrow linux server provided by Luddy. I did not see any delay in performance as the number of sessions I created was nominal.

One drawback in my design is that I read the all lines in the text file and then make the required update for each set request from a user. This could be problematic if we have a large text file. One way to speed up the performance is to maintain a data structure like hash-map to handle updates and reads while the server is running- I could read the text file only once while starting the server and store the data in a hash map. I could the ensure atomic updates to the data-structure, and then finally write the data to the file before closing the server.

This would improve the speed of read and operations, but it could lead to a memory overhead if we have large amount of data.

The number of concurrent requests which my server could handle would depend on the system capacity.

For efficient handling of large sized keys and values, first I send the message length to the server, and then ask the server to listen to the stream of bytes equivalent to the message length. So, the max size of keys and values would mostly depend on the system’s capacity. Also, the server can accept space delimited values.

**Limitations:**

1. Cannot accept multi-line values. The server can accept space delimited values, but the performance will not be consistent for multi-line values.
2. Low-Performance on large data. The performance would be poor on large datasets as I read and write the entire data for each command operation(get/set).
3. No user-authentication.
4. Lack of modularity in client design. Although the client functions are segregated into separate methods, the design can be improved by maintaining the client attributes in a class.