ECE419 Design Document M1

Bohao Wang 1002993442
Jingwei Zhang 1002896690
Yanyi Zhang 1003327517

Design of System

General description: Currently, KVClient instance is used to listen on users input. KVStore instance is established per KVClient instance in the current situation to connect with the ClientConnection instance on KVServer for message transmission. The single KVServer can establish multiple ClientConnection instances simultaneously. KV Storage is the well encapsulated single instance that incorporates cache to communicate with the file system. The design diagram is shown in Figure 1.

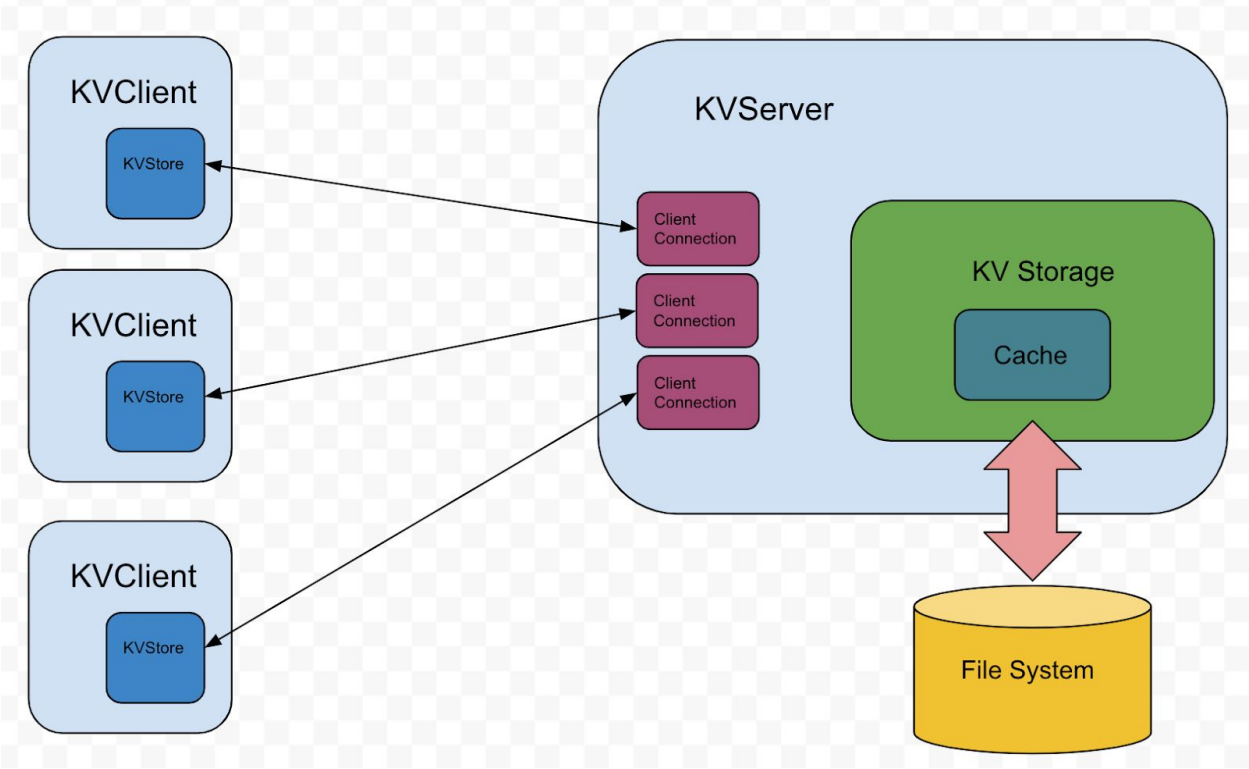

Figure 1. The client and server model for M1

Communication Protocol

Based on the given KVMessage interface, we built a KVMessageImple class adopting the Gson library from Google in the shared folder. We decided to use this library because Json is the state of the art message format through network communication with clear key-value pairs. With Gson library, fields in class objects can be conveniently parsed into string and then be converted into TextMessage format to transfer through the network and vice versa.

Our message transformation is shown in the Figure 2: (e.g, PUT key1 value1).

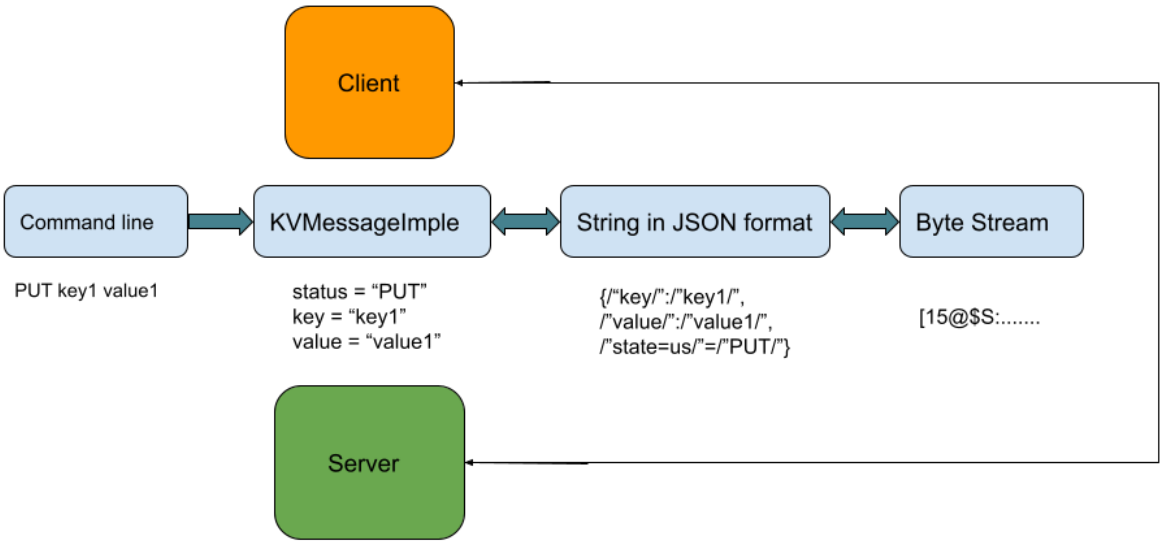

Figure 2. The message transformation between client and server.

Persistence Storage:

We use a json file to store all the key-value pairs in our storage. Whenever the server receives a request to put a key-value pair on the storage, the program will parse the data in the json file, update the values accordingly and then write the data back in the json. Both put and get operations are done in O(n) time since it needs to parse all the key-value pairs in the json file.

Cache:

Currently we have implemented LRU and FIFO caches to accelerate our storage system. The cache is initialized with a max capacity, whenever the cache exceeds the capacity it evicts entries according to the strategies specified.

Performance Evaluation:

Figure 3 plots the average latency of 5000 random requests at different put/get ratios. It is easy to observe that the latency gets larger as the ratio of PUT operations increases. This is because a higher ratio of PUT operations generates more entries in the database, thus drops the performance.

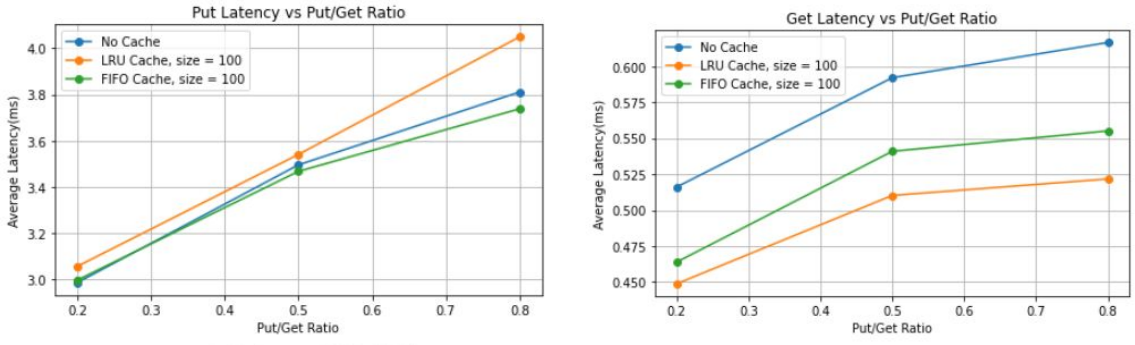

Figure 3. Put Latency (on the left)/Get Latency (on the right) vs. Different Get/Put Ratio.

In the following figures, we tested the performance by varying cache sizes and total number of requests. In figure 4, we can see that a bigger cache size improves the performance of the GET operation but does not affect the PUT operation much. This is because our storage system still needs to update the disk file with O(n) complexity on PUT operations. In figure 5, we can see how increasing the number of requests can increase the latencies of the operations. Note that we keep the max number of entries at 1000 for all of our performance tests.

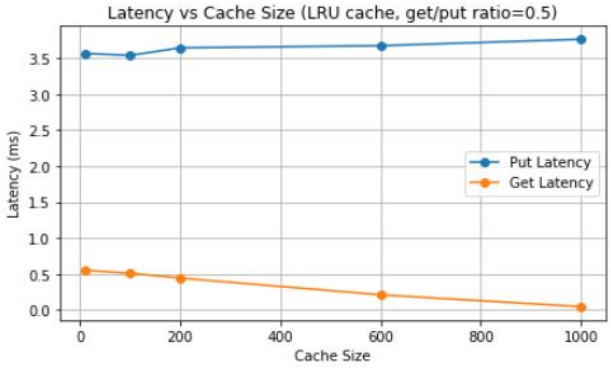

Figure 4. Latency vs. Different Cache Sizes

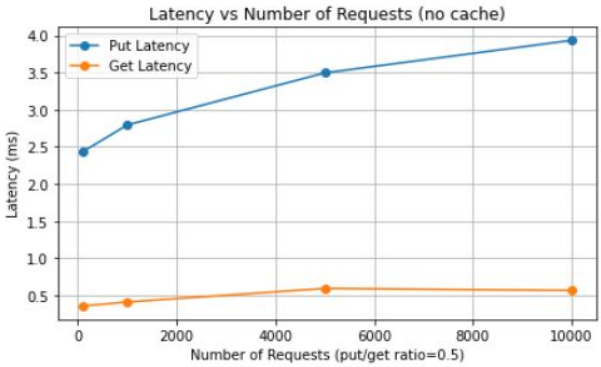

Figure 5. Latency vs. Number of Requests

Appendix: Additional Tests:

We have added additional 7 unit tests additional to the test cases that have been provided. Detailed descriptions of the 7 unit tests are listed below.

Name	Description
testSuccessfulEncode	Test the message transformation from String (Json format) into the KVMessage object.

Name	Description
testKVStorageBasic	Tests the put, get and update functionalities on KV Storage.

Name	Description
testKVStorageClear	Tests if storage clear function clears all the entries on KV Storage.

Name	Description
testKVStorageMore	Tests the delete functionality on KV Storage. Also tests if the storage handles the key does not exist error correctly

Name	Description
testFIFOCache	Tests put, get, delete functions on FIFO cache. Also checks if the cache evicts the oldest item when it is at its maximum capacity.

Name	Description
testLRUCache	Tests put, get, delete functions on LRU cache. Also checks if the cache evicts the least recently used item when it is at its maximum capacity.

Name	Description
testDeleteError	Test the delete function if the key to be deleted does not exist