ece419-design milestone1 documentation

January 2021

1 Overview

Our milestone1 is a system consisting of single thread clients and one multi-threading server. There're three main components of our project: communication layer, storage layer and client side application

2 Responsibility and Setup

see **README.md**

3 Functionality of each component

Communication and receiver logic
Communication module e.g: CommunicationTextMessageHandler, it's a module **in the middle of client and server** shared by ooth of them for communication and wrapping corresponding operation protocols. The translation of bytes to receiver object (KVMessage) and from object to bytes is handled by our Communication module and KVStore in client, KVClientConnection on server side. It'll first convert the KVMessage to raw bytes to send to the other end, and will parse the byte data into kVMMessage object

Server structure
Our server is a multithreaded server, Each KVClientConnection will help handle subsequent get and put request of a client in a seperate thread after a client establishes connection with the server. And after we parse KVMessage and its related status, the later storage layer will process the information. Our storage layer consists of cache and persistence layer.

Cache layer
Our cache latyer is implemented mainly by three types, FIFOCache, LFUCache and LRUCacahe. They'll cache values with different mechanisms and help the server gets the value in a faster pattern.

Persistence layer We implement our persistent layer with the help of JSON serialization. After our cache becomes full, it'll trigger respective eviction policy and then evict it to the file system. We'll first save it to file with help of org.json, and also scan the whole file and get it from json when getting the corresponding key. We'll optimize our file system in the later milestones like using "bulk eviction" and metadata to boost its query and write speed.

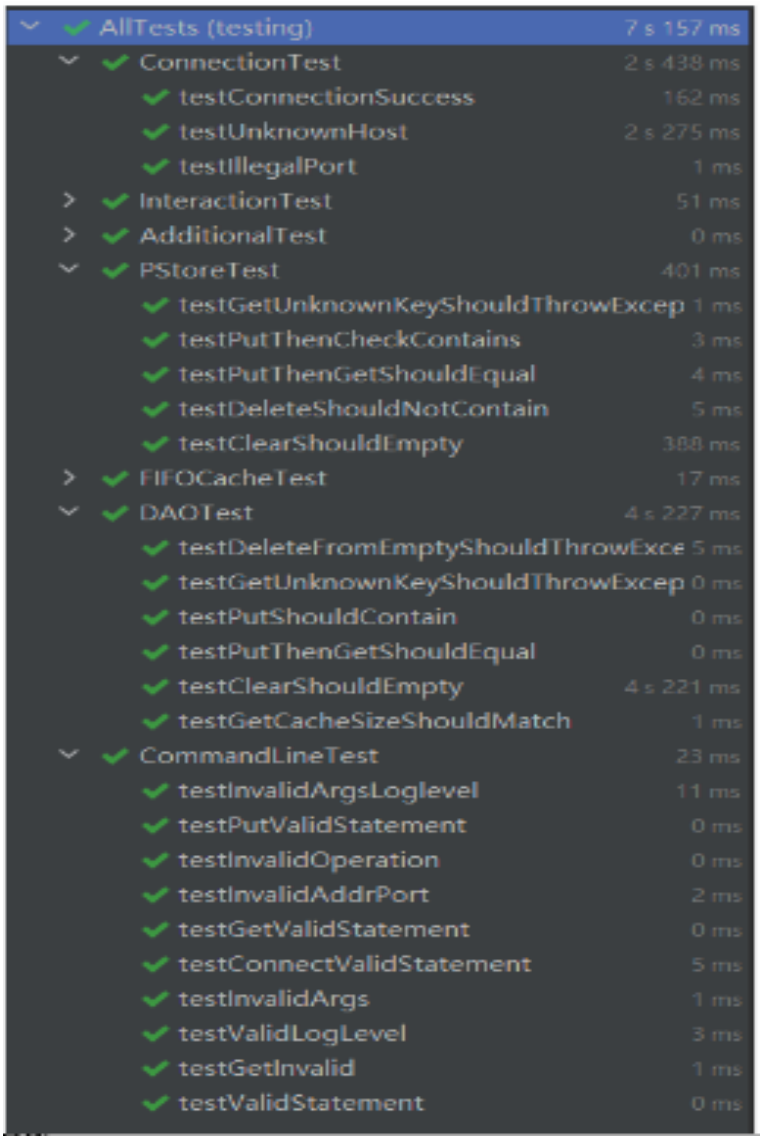
4 Testing components

We use **Junit** to test our project, we've implemented a varity of test cases, such as the command line input test, persistence layer test and cache strategy test. Besides, tests about exception handling, **Eviction ordering**, and interaction tests (get, put) are also included **All tests are passed** (appendix 1)

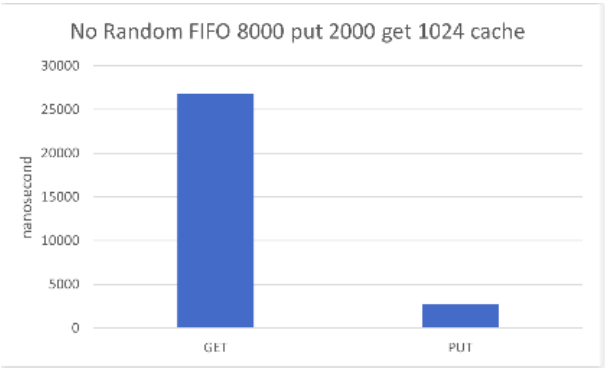
5 Performance Evaluation

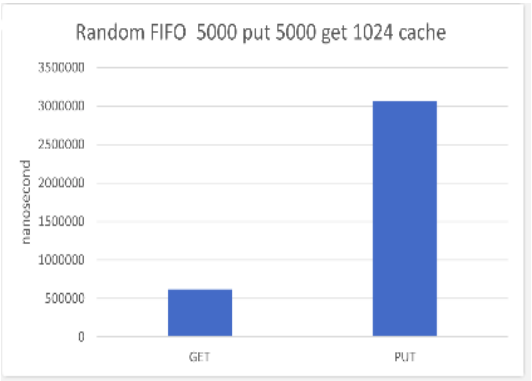
(See Appendix 2) We tested average time under different conditions, including whether key value pairs are randomly generated, diffferent number of put and get ratios 1. put/get (8000:2000) 0.8 2. put/get (5000:5000) 0.5 3. put/get (2000:8000) 0.2. with cache size 1024. the performance is increasing over all as get/put ratio increases, our implementation is more beneficial in conditions where there're more queries

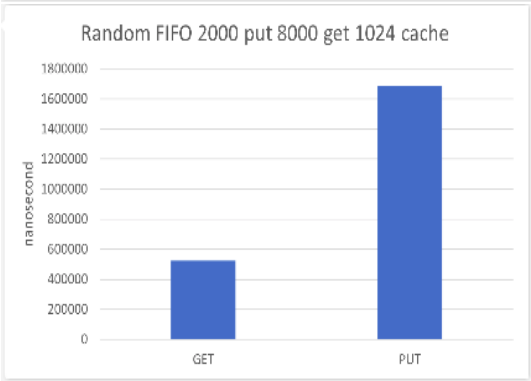
6 Appendix 1

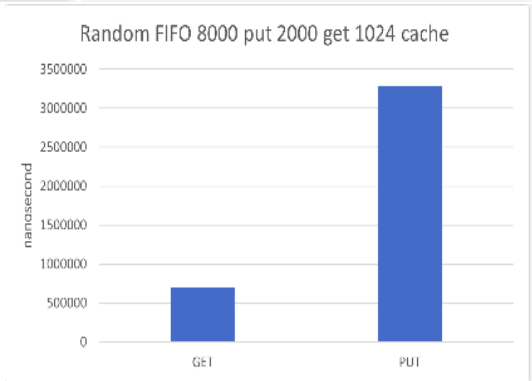

AllTests (testing) 7 s 157 ms
ConnectionTest 2 s 438 ms
testConnectionSuccess 162 ms
testUnknownHost 2 s 275 ms
testIllegalPort 1 ms
InteractionTest 51 ms
AdditionalTest 0 ms
PStoreTest 401 ms
testGetUnknownKeyShouldThrowExcep 1 ms
testPutThenCheckContains 3 ms
testPutThenGetShouldEqual 4 ms
testDeleteShouldNotContain 5 ms
testClearShouldEmpty 388 ms
FIFOCacheTest 17 ms
DAOTest 4 s 227 ms
testDeleteFromEmptyShouldThrowExce 5 ms
testGetUnknownKeyShouldThrowExcep 0 ms
testPutShouldContain 0 ms
testPutThenGetShouldEqual 0 ms
testClearShouldEmpty 4 s 221 ms
testGetCacheSizeShouldMatch 1 ms
CommandLineTest 23 ms
testInvalidArgsLoglevel 11 ms
testPutValidStatement 0 ms
testInvalidOperation 0 ms
testInvalidAddrPort 2 ms
testGetValidStatement 0 ms
testConnectValidStatement 5 ms
testInvalidArgs 1 ms
testValidLogLevel 3 ms
testGetInvalid 1 ms
testValidStatement 0 ms

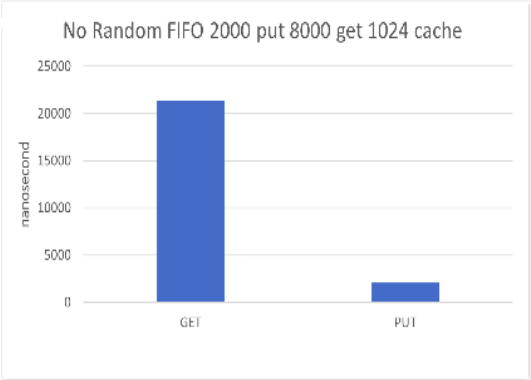
7 Appendix 2

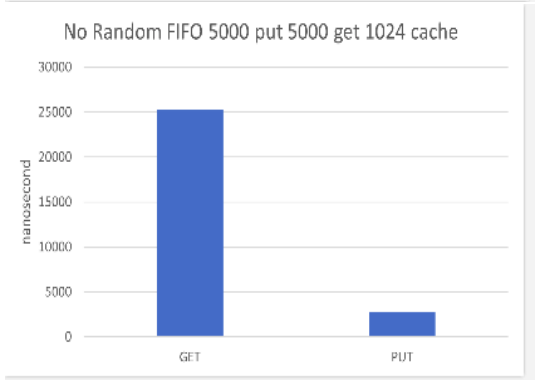

No Random FIFO 8000 put 2000 get 1024 cache
nanosecond
30000
25000
20000
15000
10000
5000
0
GET
PUT


Random FIFO 5000 put 5000 get 1024 cache
nanosecond
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
GET
PUT


Random FIFO 2000 put 8000 get 1024 cache
nanosecond
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
GET
PUT


Random FIFO 8000 put 2000 get 1024 cache
nanosecond
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
GET
PUT


No Random FIFO 2000 put 8000 get 1024 cache
nanosecond
25000
20000
15000
10000
5000
0
GET
PUT


No Random FIFO 5000 put 5000 get 1024 cache
nanosecond
30000
25000
20000
15000
10000
5000
0
GET
PUT