DECS PA4: Key-Value Store

Team Rocket
Roll no. : 203050015, Ashish Aggarwal
Roll no. : 203059009, Ajaykumar Shyamsunder
Roll no. : 203050039, Keshav Agarwal

Tuesday 10th November, 2020

Contents

1 KV store coding

1.1 KV server

1.1.1 Design

- Reading the configuration from server.cnf file and starting the server.
- Creating initial thread equal to no mentioned in the server.cnf file. context for each thread is created as soon as thread is added to thread pool.
- Each thread can handle some no of active clients mentioned in server.cnf and threads can be reused if old clients closes connections. Client will be assigned to worker thread in round robin fashion.
- worker threads are waiting for the first client connection and will wait on epoll wait once connection is received.
- If client closes its connection its context will be deleted from epoll context. and same thread can accept other client.
- Server send 200 response code on successful and send back "success" response message or key-value(in case of get). and 240 if unsuccessful with appropriate error message.

1.2 KV cache

1.2.1 Design LRU

- A Hash Table of cache size is used to have a O(1) search. Key is passed through a hash function and the node is inserted into the hash table with the derived hash value. A linked list is maintained to handle hash collisions.
- A Doubly Linked list is maintained to implement LRU. On every node access, the node is moved to the head. The last node is the least recently used node and is evicted.
- A free list is also maintained which contains the free nodes. Any node deleted from cache is inserted in this list. To add a key value pair, a node is popped from this list.
- Reader Writer locks are maintained at node level.

1.2.2 Design LFU

- A Hash Table of cache size is used to have a O(1) search. Key is passed through a hash function and the node is inserted into the hash table with the derived hash value. A linked list is maintained to handle hash collisions.
- A Doubly Linked list is maintained to implement LFU. On every node access, the frequency is incremented. The least frequency node is evicted.
- A free list is also maintained which contains the free nodes. Any node deleted from cache is inserted in this list. To add a key value pair, a node is popped from this list.
- Reader Writer locks are maintained at node level.

1.3 KV client

1.3.1 Design

- Multiple clients are created by forking and creating required no. Of process(client).
- Parent then goes on to wait until all client are done executing.
- Each client is given their input file descriptors corresponding to their .csv input file which contain get, put, delete requests.
- Each forked process goes on to make connection with the server and start sending requests present in csv.
- Each request (opcode,key,value) is read from their .csv and encoded in the message format and sent. Response from server is read and the message is decoded.

1.4 KV File System

1.4.1 Design

- Number of different files to store key values are fixed as specified in server.cnf.
- **Write** we will use cpp std hash function and generate a no b/w (0 to number of files-1) and store the key value in the file if request comes to FS if key does not present already otherwise just update existing keys.
- **Read** we will use cpp std hash function and generate a no b/w (0 to number of files-1) and search the file based on hash index and return the value of key. If file or key not present, appropriate error message will be sent back to client.
- **Delete** we will use cpp std hash function and generate a no b/w (0 to number of files-1) and search the key in file based on hash index and mark the key as deleted. If file or key not present, appropriate error message will be sent back to client.

2 KV store performance

please find attached the following graphs for performance analysis.

2.1 LFU performance

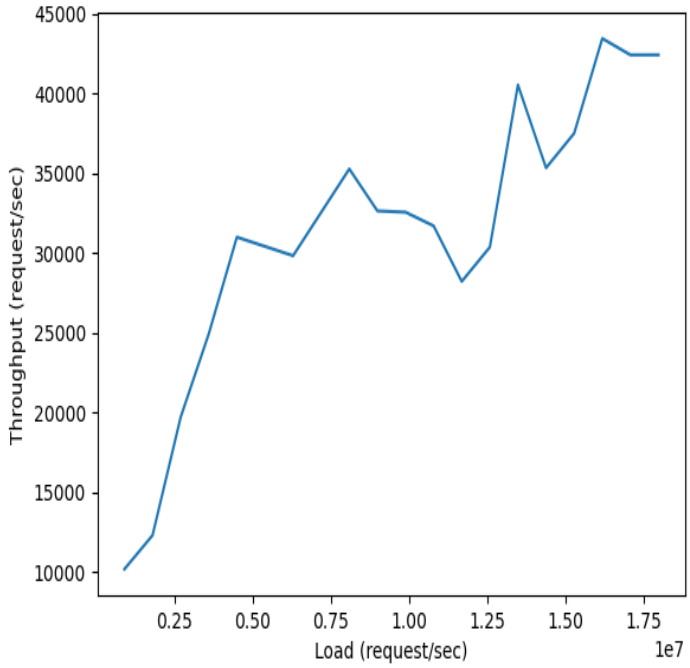

Figure 1: Throughput Vs load

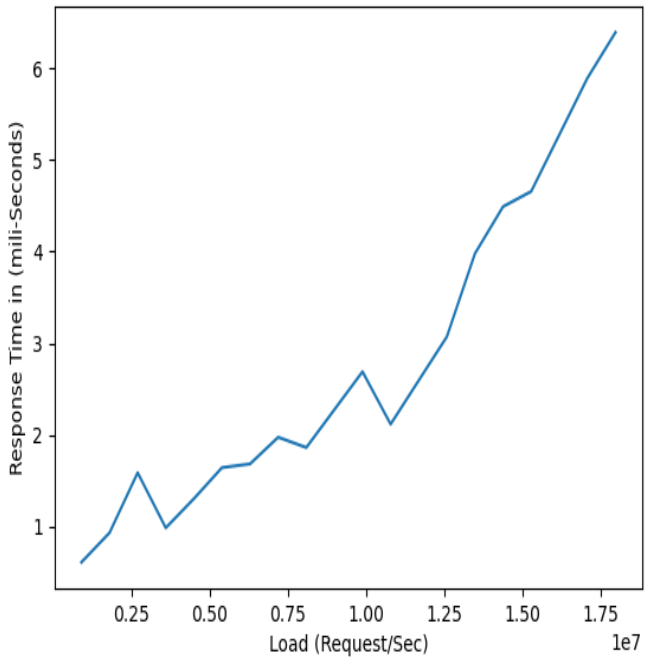

Figure 2: Response Time (as seen from the client) vs Load (in requests per second).

2.2 LRU performance

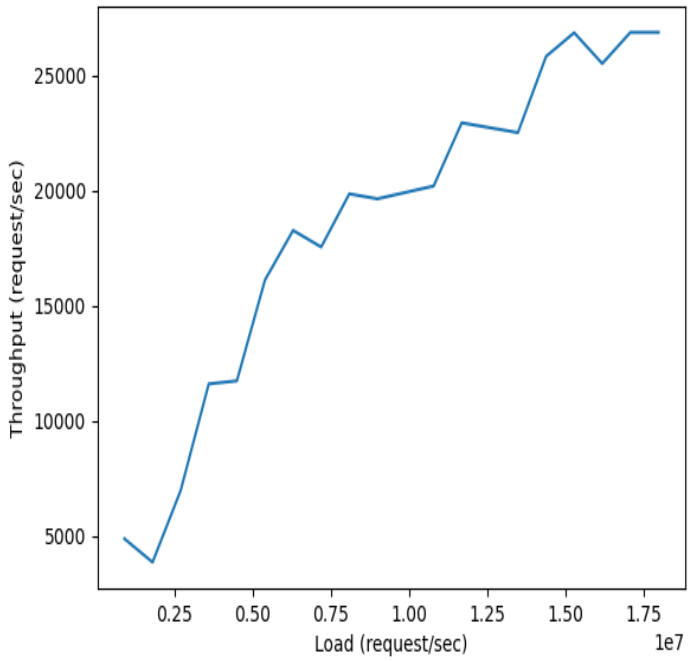

Figure 3: Throughput Vs load

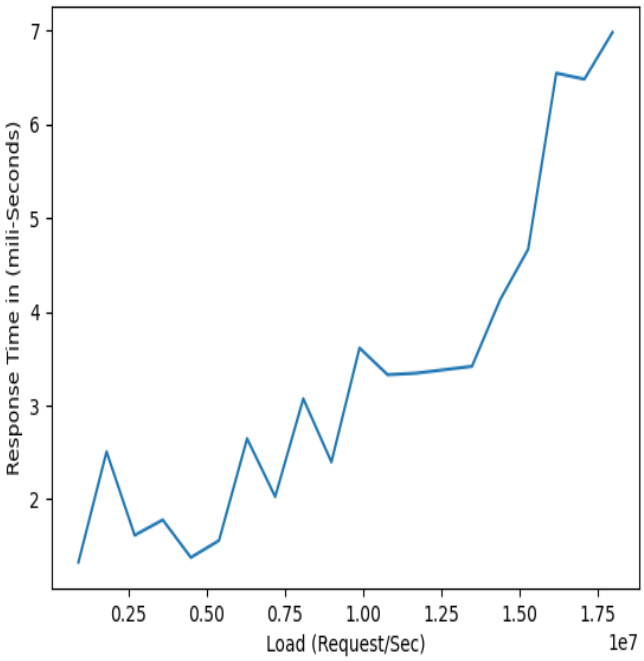

Figure 4: Response Time (as seen from the client) vs Load (in requests per second).