Le Hung Thinh

January 23, 2020

1 Thought process

Initially, I have thought about storing all data (keys and values) in a big text file but if the data getting big, that file cannot be fit on RAM, or finding a key and update its value or reading value of the key I have to open the big file and store the value of the key (value can be big) and the complexity is O(N), or if I delete a key, I have to make a copy of that text file and loop through every single file of the file.

So I have thought about splitting the big file into smaller files. By storing value of a key in a text file and file's name is the key. If I want to search for a key, OS handles it for me, or if I need to update a key, the key can be found and overwritten or appended new data and I do not need to worry about size of the value of the key. I also classify value data structure by saving file on difference directories. Key and modify time is stored in RAM (I got this idea by learning how redis store data).

I choose Django Python instead of Golang because in Golang, I only know how to write API, send request throught server by using curl but you need to implement CLI so I choose Django because Django could handle CLI for me and I only take care of implementing logic. If you require me to write API or build server, I could write code in Golang that could handle concurrency, automation testing. In Django, I test my code by manual testing

2 Design and Implementation

My design is like this:

- Create **storage** directory and **metadata** directory.
- In **storage** directory, there is 3 directories: **string, set, list** corresponding to the given data structures. **metadata** directory is storing **snapshot**
- Every file in each data structure directories has name is key and content of the file is the value of the key
- When client sends a POST request to server, server chooses corresponding directory to write, update, or read data and displays result to client

- RAM keeps track of working history by creating a hashtable with key is name of the file and value is time modified the content of file. Expire time is like this except value of hashtable is expire time set by client

My code works like this:
When we first start server, system checks if all directories above exist or not, create them if not exist.
keyTime and **keyExpire** is global hashtable that keep track working history
When client sends request to server, server reprocesses input and checks if the method required is existed in predefined list of methods.
Because this website works like CLI so I split input by space. First parameter after splitting is method user required and the rest parameters are inputs of the method.

With operations on data structures, there are 3 operations:

- read data
- write data
- update data

By reading data, client may use method of this data structure on other data structure (like LPOP on a key contains set value). To avoid this, I just create folder containing appropriate data (set folder contains only set value,...). When client reads data, first I check if key given by client exists in RAM then check data structure corresponding to the method and then check if the file exists in the folder corresponding to the data structure. When all checks pass, I just read data and do some transformation steps and response to client (like SINTER read all contents of files and convert these contents into python set data structure and use set method). Like GET, LLEN, etc

By writting data, this means we write new data or overwrite exist data. By writting new data, we just need to take care of method and write data to corresponding folder. While overwritting exist data is like update data. Like SET

Updating data is the same as reading data, client may use method of this data structure on other data structure. We also do check like reading data to find right file. After all checks pass, we load data, convert data into corresponding data structures and use methods of those data structures to transform data and overwrite exist file (like LPOP, SADD, etc) or append to the end of file (like RPUSH)
Example: client sends GET key to server, server first prepocessing request from user to get method and parameter (in this case is GET and key), then check if GET method exists, then server checks if the key exists on RAM, if yes then check if file key.txt exists in string folder because GET is method of string so it

check key existence in string folder but not in other folder (to avoid GET value of a list or set). If all tests pass, then server open key.txt file, load file content and render HTML file with the content and response rendered HTML to client

Data Expiration operations work quite different, they do not need to read file's content. KEYS method query all key of keyTime global variable (on RAM). DEL method still need to check if key exists then keep searching the file's name until name found and delete the file without open the file. FLUSHDB delete all files in data structure directories one by one without open the files. EXPIRE is working on keyTime, keyExpire variable (on RAM) and do not need to work with files. TTL only works with keyExpire variable.

SAVE makes deep copy of keyExpire and keyTime (so client could send SAVE request as many as they like), convert datetime value to string value and save as json file
RESTORE reads saved files from SAVE method and load saved data to RAM (as hashtable)

3 Challenges

When deploying on Heroku, RESTORE could not be used when you stop web app because Disk storage on Heroku is ephemeral. That means I cannot create text file by python code even I execute linux command from python by os.system(command) but when I execute by python shell line by line of code of the file I have executed, it works. I could solve this problem by intergrating with S3 and I could store all file to S3. The website work very unstable so in order to test perfectly, please run on local machine.
I haven't tried any other clould like AWS or Google cloud or digitalocean because they require Credit Card so I will test on other cloud services later.
I also design reading data from file and writting data to file methods that handle with file local machine. So when you integrating with S3 or database, you just change those methods. As well as delete