M111 - Big Data - Spring 2021 - Programming Assignment by Kristian Mitrofan , DS2.19.0001

This assignment was coded in Python and was run and tested on a Python environment using the Visual Code IDE. The delivered code fully implements the project as described in the pdf of the assignment (the last 10% part as well), so lets go through how each part was done.

1. Data Creation (20%) - createData.py

I noticed that the format we want our data in is really similar to a json format which incidentally is the same format a python dictionary is using to store its data. So I started from there and after creating the main key (e.g. person1) I am calling the function generate_value() which recursively creates the whole value of the main key.

I made a few assumptions on this part. **The major one is that I added a new variable type called "set"** (e.g. address, contact in my file) which translates to a variable where the type is one or more pairs of key : value. (e.g. contact : {phone : "1234"). This was done because it looks more natural to have specific variables like address and contact to have nested values in them instead of random variables like height. We could ignore this assumption by easily implementing code to randomly ignore a variable type and give it the same functionality as "set" but I felt that by giving some specific variables the "set" type the data seems more natural. Some additional assumptions I made are that the max integer value is 1000000 and the max float value is 1000 along with the fact that the vocabulary of all string based variables consists of only lower-cased letters and numbers. The above can be easily changed in the script using the constant variables at the beginning.

The rest of the script works as follows. For each line 'i' that needs to be created we generate a main key person + 'i ' and a main value that has a random maximum depth between 0 (e.g. "person1" : {}) and the maximum given depth. This is due to the fact that we randomize the number of key : value pairs for each "set" variable. In each recursive call of the generate_value() the script randomly picks a variable from the keyFile.txt and assigns it a value according to its type and then it calls itself with a depth one lower that before if the type is a "set". The function terminates at 0 depth when it returns the whole main_value and writes the main_key : main_value in the dataToIndex.txt.

KV Broker (20%) - kvBroker.py

The kvBroker should only be run when all the servers in the serverFile are already running and listening for connections. I made this assumption as there were no instructions on how to handle this. When run the kvBroker will connect to all the servers and add that connection to a dictionary (key is host+"-"+port) for future use. After it has connected to all of them it will start sending PUT commands using the data from dataToIndex.txt to k of them each time. So now each row of the file has been imported to k number of servers resulting in a k replication factor. Then the kvBroker will start receiving input commands from the user.

Along with the required "PUT","GET","QUERY" and "DELETE" commands **one more was added called "TERMINATE"**. When give this command as is the kvBroker will send a message to all the remaining servers to … terminate and will exit itself. When given with a proper argument (e.g. 127.0.0.1-60001) it will send the message to that server only and the kvBroker will keep running. This was implemented to test that the k replication factor works properly. We can see below kvBroker running after connecting to three kvServers and inserting 1000 rows. The commands are followed by a key that can be between " or not except the PUT command which needs to follow a strict format as described in the given pdf (e.g. PUT "person17" : {"number": "819856"; "height": "359.65"; "age": "903707"})

Three servers running with k = 2 replication factor

Last PUT insert

Missing } at the end

Terminate one server

Can no longer delete entries

Terminate another server

Can no longer search entries either

```
729"}, "age": "494604"}, "street": "jo"} was inserted!
OK: person998 : {"nationality": "pm5"} was inserted!
OK: person999 : {"height": "562.57", "nationality": "1i", "contact": {"number": "
944385", "street": "5"}} was inserted!
INPUT A COMMAND: GET person998
OK: person998 : {"nationality": "pm5"}
INPUT A COMMAND: GET person1000
NOT FOUND: person1000 could not be found!
INPUT A COMMAND: PUT person1000 : {"Hello" : "World"
ERROR: data is in the wrong format!
INPUT A COMMAND: PUT person1000 : {"Hello" : "World"}
OK: person1000 : {"Hello" : "World"} was inserted!
INPUT A COMMAND: GET person1000
OK: person1000 : {"Hello": "World"}
INPUT A COMMAND: QUERY person999
OK: person999 : {"height": "562.57"; "nationality": "1i"; "contact": {"number": "
944385"; "street": "5"}}
INPUT A COMMAND: QUERY person999.height
OK: person999.height : 562.57
INPUT A COMMAND: QUERY person999.contact.number
OK: person999.contact.number : 944385
INPUT A COMMAND: DELETE person998
OK: person998 was deleted!
INPUT A COMMAND: GET person998
NOT FOUND: person998 could not be found!
INPUT A COMMAND: TERMINATE 127.0.0.1-60003
OK: Server with ip: 127.0.0.1 and port: 60003 has terminated!
INPUT A COMMAND: DELETE person1000
ERROR: at least one server is down, entry cannot be deleted!
INPUT A COMMAND: GET person1000
OK: person1000 : {"Hello": "World"}
INPUT A COMMAND: TERMINATE 127.0.0.1-60002
OK: Server with ip: 127.0.0.1 and port: 60002 has terminated!
INPUT A COMMAND: GET person1000
ERROR: k or more servers are down so it is not safe to search the entry!
INPUT A COMMAND: TERMINATE
OK: Server with ip: 127.0.0.1 and port: 60001 has terminated!
All connections have been terminated, broker closing too...
PS C:\Users\chris\BigData\key-value-db>
```

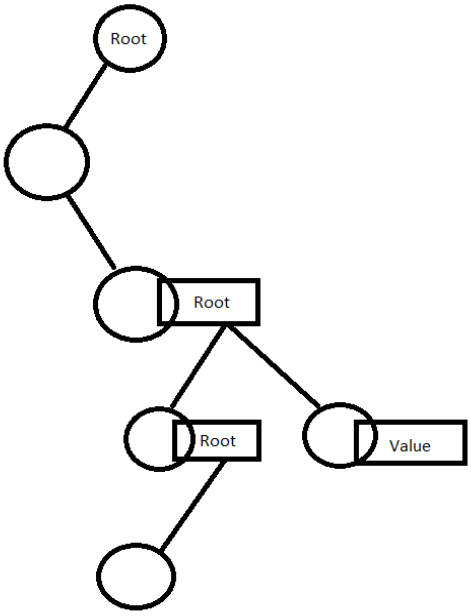
KV Server (50%) - kvServer.py

When run, the server will create an instance of an empty kvTrie and after it is connected by the kvBroker it will start receiving commands from it and execute them accordingly on the Trie. After executing each command it will send an appropriate response to the kvBroker (OK,ERROR,NOT FOUND) along with any data required.

Using Only a Trie (10%) - kvTrie

I initially implemented a simple Trie with the only caveat being that it has an additional element in each node called trie which can potentially store additional Tries in it. I then implemented search and insert for the basic trie which follow the basic logic of traversing the trie through the children of each node until we find the end of the key that we need to search or insert.

I then implemented a more complex version of the above class called kvTrie which by using the Trie functions along with some recursive functions of its own (rec_insert,rec_search) can traverse a Trie of Tries which looks something like this:



Each nested key of the same depth that belongs to the same parent key inside the main value will be inserted in a nested trie (e.g. person1 : { “name” : “Mary” ; “address” : { “street” : “Panepistimiou” ; “number” : 12 } in this case for depth = 1 there is name and address while for depth 2 and parent_key address there is street and number). So only the Trie data structure is used for searching and inserting data in this project.