M111 - Big Data

# Programming Assignment

Marios Papamichalopoulos - CS3190006

## Introduction

All the requirements have been implemented except using only a trie. The assignment can always be found at this url after the deadline https://github.com/PapamichMarios/KeyValueStoreServer. If you want to access it before the deadline send me an email to add you as a contributor.

## Tools Used

- Python 3.8
- PyCharm
- Windows 10

## Files

### Python

- **./data_creation/createData.py**: creates input data for the key-value broker and stores them in *dataToIndex.txt*
- **kvBroker.py**: key-value store broker
- **kvServer.py**: key-value store server
- **utils.py** : utility functions
- **logging.py**: logging functions
- **properties.py**: strings and other properties
- **errors.py**: error functions
- **trie.py**: class for the trie data structure

### Text

- **dataToIndex.txt**: contains data for kvBroker.py indexing
- **keyFile.txt**: contains data for *createData.py*
  The data types available are: [string, float, int, set].
  **set** symbolizes a nested key-value, e.g. location set -> “location” : { “key” : “value” }
- **serverFile.txt**: contains ips and ports for the kvBroker.py

## How To Run

From root directory:

- Create data:

  *python ./data_creation/createData.py -k ./data_creation/keyFile.txt -n 1000 -d 3 -l 4 -m 5*

  Or
  *python ./data_creation/createData.py -k ./data_creation/keyFile.txt*

- Run Key Value Servers
  *python kvServer.py -a 127.0.0.1 -p 65432*
  *python kvServer.py -a 127.0.0.1 -p 65433*
  *python kvServer.py -a 127.0.0.1 -p 65434*

  Or

  *python kvServer.py -p 65432*
  *python kvServer.py -p 65433*
  *python kvServer.py -p 65434*

- Run Key Value Brokers
  *python kvBroker.py -s serverFile.txt -i ./data_creation/dataToIndex.txt -k 2*

## Execution

The user generates data using the *createData.py* program, giving as input a *keyFile.txt* containing keys and their types. Bear in mind that the key types must be one of [string, float, int, set], where *set* is a nested key-value, e.g. location set -> “location”: { “key” : “value” }. The resulting *dataToIndex.txt*, is fed to the *kvBroker.py* for the indexing phase.

The *kvBroker.py* connects to all the servers using the *serverFile.txt*, reads line by line the *dataToIndex.txt* and sends **PUT** requests to *k* replicas like so:

*PUT “key_0”: {“age”: 71; “name”: “dasd”}*

In case some records were duplicate, the corresponding server returns an **ALREADY_EXISTS** error to the broker. All the servers log in console the requests received and their responses, which is useful for debugging. The top level keys are all of format: *`key_${i}`*, where i is the line counter of the file. The requests can be of arbitrary size and you can tweak the *BUFFER_SIZE* in the *properties.py* file if you want.

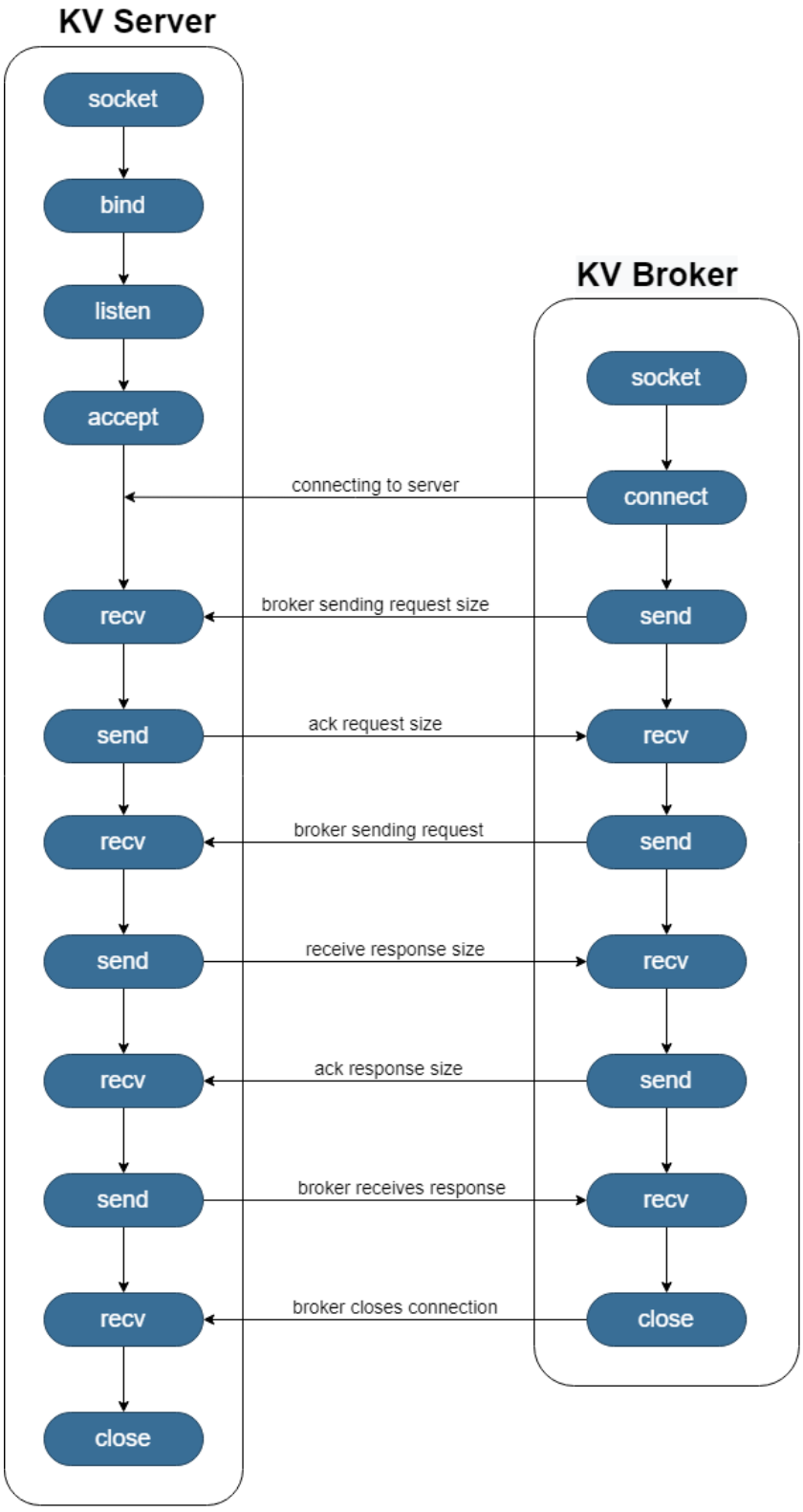
After the indexing phase ends the kvBroker.py awaits the commands **GET**, **DELETE** and **QUERY** by the user. The default behavior is not giving quotes to the keys but the server responses come with quotes (look at *Execution* section). In order not to lose request messages, before sending them both the broker and the server send a message containing the request size. The *kvServer.py* uses a trie data structure to answer each query respectively.

The final message is always in json format and consists of an object with two key-values: *success* and *data*. E.g. *{"success": true; "data": {"key_0": {"street": "dkbfx"; "address": {"level": 33}; "height": 91.21095965769372; "level": 41}}}*.

*KvBroker.py* is responsible for taking the final message in json format and printing the data key, which in case of success=True prints the resulting value and in case of success=False prints the appropriate error to the user.

In case a server is down the program continues its execution.
If a server is down the user is not allowed to **DELETE** a key, an error pops up.
If **k** servers are down the user is allowed to **GET** and **QUERY** but a warning pops up.

Following is a figure regarding the communication between a server and a broker:


KV Server
socket
bind
listen
accept
recv
send
recv
send
recv
send
recv
send
close
KV Broker
socket
connect
send
recv
send
recv
send
recv
close
connecting to server
broker sending request size
ack request size
broker sending request
receive response size
ack response size
broker receives response
broker closes connection

## Sample Execution

Let's say that we have 3 kvServer.py instances running in 3 different ports and two of them contain:

*"key_0" : {"street": "dkbfx" ; "address": {"level": 33} ; "height": 91.21095965769372 ; "level": 41}*
*"key_1" : {"location": {"location": {"height": 5.104534969166997}} ; "level": 28}*

Following are some sample executions given at **kvBroker.py**. In the first line is the user input and the line after that the response returned from the **kvServer.py**:

kv_broker$: **asdsa**

*ERROR: BAD_SYNTAX*

kv_broker$: **GET key_0**

*"key_0": {"street": "dkbfx", "address": {"level": 33}, "height": 91.21095965769372, "level": 41}*

kv_broker$: **QUERY key_0**

*"key_0": {"street": "dkbfx", "address": {"level": 33}, "height": 91.21095965769372, "level": 41}*

kv_broker$: **QUERY key_0.address.level**

*"key_0.address.level": 33*

kv_broker$: **GET kEsda**

*ERROR: NOT_FOUND*

kv_broker$: **DELETE key_0**

*OK*

kv_broker$: **GET key_0**

*ERROR: NOT_FOUND*

Let's say one server dies unexpectedly:

kv_broker$: **DELETE key_1**

*ERROR - REPLICA_IS_DOWN: CANNOT_EXECUTE*

Now another one dies (with k=2)

kv_broker$: **GET key_1**

*WARNING - REPLICA_IS_DOWN: CANNOT_GUARANTEE_CORRECT_OUTPUT*