M111 - Big Data - Spring 2021 - Programming Assignment

George Katsogiannis-Meimarakis - ds1200011@di.uoa.gr
MSc Data Science & Information Technologies
Dept. of Informatics & Telecommunications
National & Kapodistrian University of Athens

Abstract

We have created a key-value store architecture using a data structure called the Trie. Two separate programs have been developed for accessing, distributing and storing the data: the KV Broker and the KV Server, along with a third program for generating test data. In this report, we describe the programs that have been developed and give an explanation of the overall architecture and any specific decisions taken for our implementation.

1 Introduction

The architecture of the key-value store and its main features will be described in this section. The store operates in two phases: the set-up phase and the command phase.

During the set-up phase, the KV Broker reads the data from a file and sends each key-value pair to the servers for storing. The servers store the received data in a trie structure. In order to store the data effectively, the KV Broker distributes each high-level key-value pair to k different KV Servers, where k is the replication factor. This replication means that even if $k-1$ become unavailable, we can still query the data without any problem. When all the data has been distributed, the broker informs the servers accordingly and we enter the command phase.

During the command phase, the data has been distributed and the user is free to execute any of the three available commands: GET, QUERY and DELETE. Note that DELETE can only be performed if all the servers are available and the other two commands can be performed if no more than $k-1$ servers have become unavailable. When the user is finished, the STOP command can terminate all the servers and the broker.

The code of this assignment was developed using the Python programming language. Git and Github were used for version control.

1.1 File Structure

This project is comprised of the following files:

1. `createData.py`: A script for creating data that can be used for this project

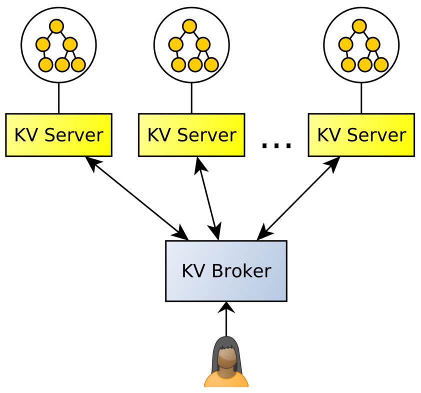

Figure 1: An overview of the overall key-value store architecture

```
"key0" : { "street" : "n" }
"key1" : { "age" : 90 }
"key2" : { "name" : "taal" ; "street" : "omh" ; "height" : 83.46 }
"key3" : { "street" : "g" ; "level" : 672 ; "name" : "njx" }
"key5" : { "level" : 14 ; "street" : "vq" ; "height" : 57.67 }
"key4" : { "name" : "r" ; "level" : 915 }
"key6" : { "street" : "zin" ; "level" : 815 }
"key7" : { "street" : "whkq" ; "level" : 291 }
"key8" : { "age" : 82 }
"key9" : { "street" : { "level" : 40 ; "height" : 43.65 ; "age" : 450 } }
```

Figure 2: An example of the data produced by the data creation script

2. `kvBroker.py`: The key-value broker first distributes the data to the servers and then allows the user to execute commands by sending various commands to the servers and gathering their responses.
3. `kvServer.py`: The key-value server first receives data to store from the broker and the waits for commands, to which it responds accordingly.
4. `trie.py`: Contains the implementation of the trie data structure which is used by the key-value servers for storing the data.

2 Data Creation

The data creation script can randomly generate data in a format that is compatible with the KV Broker and Server. An example of generated data can be seen in Figure 2.

The parameters of the script are the following:

- n: The number of lines (or top-level key-value pairs) to generate
- d: The possible level of nesting, e.g. $d = 0$ means no nesting
- m: The maximum number of keys in each level of the value
- l: The maximum length of a string value
- k: A file containing a list of key names and their data types

The generated top-level keys will take names in the form of "key$i$" where $i = 0, \dots, n-1$. The names of the keys appearing inside the values are taken from the key file, that must be provided.

Concerning nesting, when generating a value there is a pre-set probability that the value is going to be a nested value, given that the nesting level is not exceeded. This probability is originally set to 0.1. In practice, this means that even with $d > 0$ it is possible to generate a key-value pair without nesting and not every generated pair is guaranteed to have the same level of nesting. This approach was chosen to add more diversity to the data.

Finally, it must be noted that the parameter m defines the maximum number of keys that can appear in each **level** and not in the entire value including all levels. This means that each line can have up to $d \times m$ keys.

3 Trie

A trie[1] is a tree-type data structure that is usefull for searching keys from a set. In this project we use tries to store the key-value pairs generated from the script discussed in the previous section. Nested values can be stored as tries themselves. In Figure 3, we can observe a visual representation of a trie storing the following data:

{ name : "George" ; age : 24 ; address : { street : "Nikis" ; number: 22 ; city : "Athens" } }

[1]https://en.wikipedia.org/wiki/Trie

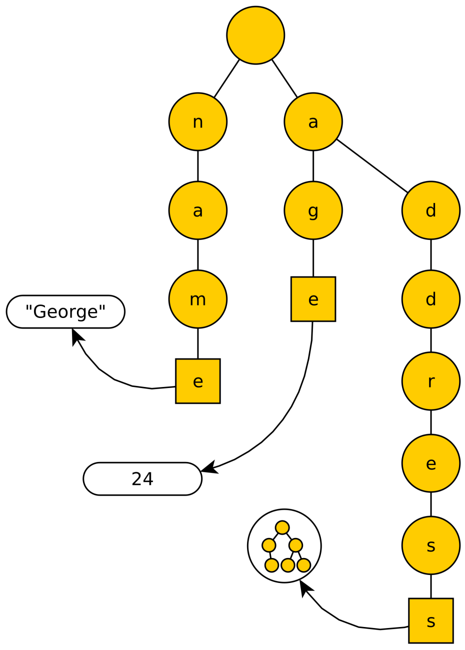

Figure 3: An example of a trie that can contain another trie as a value. Nodes with values are shown as squares, while nodes without a value are shown as circles.

For the means of this project, the trie data structure was implemented from scratch. In the following sections we will describe the implemented methods for the trie.

3.1 Insert

Given a key and a value, the insert method will traverse the trie recursively, it will create any necessary key nodes and will insert the given value at the last key node. If a value is inserted with a key that already has a value assigned, the existing value will be replaced. If a dictionary is given as a value (e.g. { street : ”Nikis” ; number: 22 ; city : ”Athens” }), then a new trie will be created with the keys and values of the dictionary and will be stored as a value.

3.2 Find

Given a key, the find method will traverse the trie recursively and will return the key’s value, if the key is found, or None if the key is not found.

3.3 Delete

Given a key, the delete method will traverse the trie recursively and will return True if the key was found and its value was deleted, or False if the key was not found. When a value is deleted, the key nodes will remain and the value will be set to None.

3.4 Convert to string

An additional method was implemented for converting the trie into a printable string format. This was necessary for sending query results back to the KV Broker. The way this method is implemented is by recursively traversing the trie, similarly to the previous methods.

4 KV Broker

The KV Broker is responsible for reading all the data from a given file and distributing it to the KV Servers, with a replication factor k. After the data has been distributed, the KV Broker allows the user to perform three different types of queries (GET, QUERY, DELETE), by communicating with the servers and gathering their responses.

More specifically, during the set-up phase, the broker reads the data file line by line and randomly selects k servers to which it sends the line. After receiving confirmation that the server stored the data, it moves on to the next line. When all data has been distributed, the broker sends the STOP command to all servers, signalling that the set-up phase is complete and they must move on to the command phase.

During the command phase, the user is able to execute queries through the command line. There are three types of commands: (i) GET, (ii) QUERY, (iii) DELETE. We will go into further detail concerning the commands in the next session. When the user gives the STOP command, the broker send a STOP command to all servers, signalling them to terminate and then terminates itself.

The parameters of the KV Broker are the following:

- s: A file containing the addresses and ports of the servers
- i: A file containing the data (produced with the data creation script)
- k: The replication factor, i.e. to how many servers each line of data must be sent

A note on replication: The goal of sending the same data to multiple servers is being able to perform queries even if some of the servers become unavailable. Besides the DELETE command, which can only be executed if all servers are available, the GET and QUERY commands can be reliably executed as long as less than k servers have become unavailable.

5 KV Server

The KV Server take the following two parameters:

- a: The IP address of the server
- p: The port of the server

The server starts listening on the given address and port for a connection from the broker. The server expects key-value pairs from the broker and stores each pair it receives in its main trie. The main trie contains all the top-level keys, each of which point to a different trie in which its value is stored. The KV Server **only uses tries** for storing the data.

We can see an example of how this is achieved in Figure 4. In this example the KV server is storing five different top-level keys: ”key0”, ”key1”, ”key2”, ”key10” and ”key11”. The terminal token of each key has a value that is pointing to a different trie which is created for storing the values of the key.

After the server receives the STOP command from the broker, it enters the command phase. During the command phase, the server accepts the three aforementioned commands and responds accordingly:

- GET <key>: The server searches for <key> in its main trie. If it finds the requested key in the top level, it responds with <key>:<value>, otherwise it responds with NOT FOUND
- QUERY <key1>.<key2>...<keyN>: The server searches for <key1> in its main trie. If this key points to a different trie, it searches for <key2> in the found trie and so on. If any of the keys before <keyN> do not point to a trie or if <keyN> does not point to a value in the appropriate trie, the server returns NOT FOUND. Otherwise it returns the value it found.
- DELETE <key>: The server searches for <key> in its main trie. If it finds the requested key in the top level, it deletes it and responds with OK, otherwise it responds with NOT FOUND.

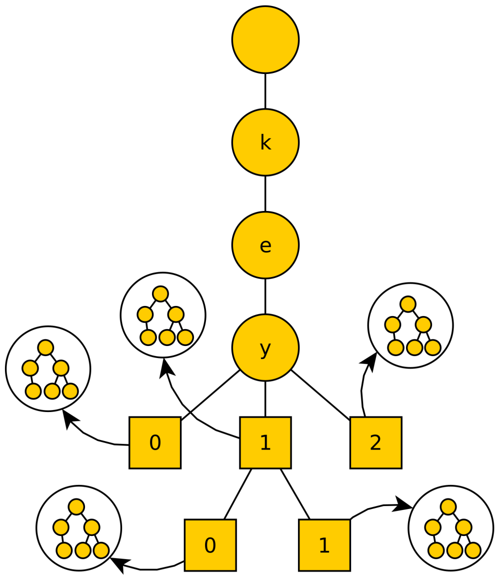

Figure 4: An example of the KV Server's main trie, containing sub-tries for each entry received by the KV Broker. Nodes with values are shown as squares, while nodes without a value are shown as circles.