PROGRAMMING ASSIGNMENT 3 – CS553

This assignment has been developed by:

The authors are:

Anushka Dhar – A20322969

Pablo Cortes – A20339462
Enrique Escribano – A20339698

The contributions of each team member are the following:

Anushka: file generator, HTML code for website (client part)

Pablo and Enrique: Server part, Benchmark and Performance

Just in the beginning we crerated an account in Google App Engine to be able to access at this platform. After that we just followed step by step the Python tutorial on GAE to get familiarized with it. Starting with the "helloworld" app executing it on our local machine until create our own application that is the one which has been uploaded in the url: ***http://aepcs553.appspot.com***

Main.py

This file contains the code for both client and server parts.

Main class:

It contains the index page of the app deployed on GAE.
It is defined as a get method that show html code on the web browser.
It contains html code for every section: insert, check, remove, find and list.

The client part of the assignment is implemented in this class using self.response.out.write("html code").

Insert class:

Defined as a post method, we select a file to upload from our computer and enter a file key value for this file.

We are using BlobstoreUploadHandler to upload the files in an easier way.

Depending on its size (if its smaller or bigger than 100KB), the file will be saved into Memcache or Google Cloud Storage (GCS).

Note.- We manage a flag to enable or not the use of Memcache, because for the benchmarking we are asked to do some experiments without using Memcache.

In case the file is saved into Memcache, we only need to do: memcache.add(filekey, file).

In our case we get the file uploaded doing self.get_uploads('name of the input field of the HTML form') and taking the first element in the array (in case we upload more than one file at the same time using the website, we only be able to save the first file uploaded).

In case the file is saved to GCS, we need to use the API files.gs.create(path) in our bucket in GCS. After that, using blobstore.fetch_data() we write it. Then finalize the file in order to let it to be readable in GCS.

Check Class:

The idea is filtering all the file keys stored in the system to find one with the file key given. If its return count is different from 0, the file with the file key given already exists. If its zero, it has not been inserted.

Find Class:

Similar to Check , we filter all the files stored and check if the given file key is there. If it is not, there is an error. If it is already there, we return its content (reading it in a buffer and then writing it using self.response.out.write(buffer)) if it is in GCS or we download (using send_blob()) it if it is in Memcache.

List Class:

The idea is very simple. We get all the files stored, iterate all of them and write each of them using self.response.out.write("filekey").

Remove Class:

Similar idea too. First we get all the files and get the file key we want to remove. Check if it exists; db-get(filekey). If it doesnt exist, return error. If it exists, then we need to check if it is stored in memcache or GCS for using the correspondent API. For memcache we use memcache.delete(filekey) and forGCS we use files.delete(pathtofilekey).

InsertURL Class

It is only used for inserting files during the runtest from command line. It invokes insert and generate the url of the upload of the file.

Benchmark.py:

We have created this class to be able to execute all tests you require in this assignment. To carry it out, we just call some methods that have been designed before in the main class main.py. Those methods allow you to “insert”, “find” and “remove” files in our application. All of this is coded with multithreading so we can enhance the performance just using more threads to treat different files at the same time, we have just been aware of locking the threads to manage some variables to achieve our goal. The file that we want to work with in each method is mapped thanks to the “key” that we send to each method we use.

Benchmark.py prints in the shell all the results that we need to do the performance of this assignment such as the total time this code needs to insert all 411 files.

Explanation of file generation code :

In GenerateFile.java , the function random_string_create() generates a string of random characters by taking characters from CHARACTER_LIST. The function random_create_number() generates random integers by taking the integers from CHARACTER_LIST. Then in the main function, we generate the name of the file folder and files with random alphanumeric characters. For this we make the use of switch case where we create a folder by making use of mkdirs() function and we apply a for loop ,where we the number of times we want to run the for loop is equivalent to number of times we want to create the file. The name of the folder is equivalent to the size of the file. To generate the random name of the file we call the function random_string_create() and pass 10 as parameter to generate a file name of 10 characters.Then we make use of new File() function to create a new file and make use of FileWriter and BufferWriter to write into the file. With the help of for loop we will write into the file by calling the function bufferwriter.write(gf.random_string_create()) function. The cases in switch will be of 1KB, 10KB, 100KB, 1MB, 10MB and 100MB. And same steps would be repeated for generating files depending upon the size of the file.