**Programming Assignment #3**

**Google App Engine**

Done by

Chaitanya Kumar Anichetty Murali

Shivakumar Vinayagam

Vinodh Kandaswami

**INTRODUCTION:**

Google App Engine (often referred to as GAE or simply App Engine) is a platform as a service (PaaS) cloud computing platform for developing and hosting web applications in Google-managed data centers. Applications are sandboxed and run across multiple servers. App Engine offers automatic scaling for web applications—as the number of requests increases for an application, App Engine automatically allocates more resources for the web application to handle the additional demand. Google App Engine is free up to a certain level of consumed resources. Fees are charged for additional storage, bandwidth, or instance hours required by the application. It was first released as a preview version in April 2008 and came out of preview in September 2011.

**Deployed System in GAE:**

The GAE server is uploaded at URL:

http://cloudstoragepa04.appspot.com

**Design:**

We have designed a browser interface using python which displays the various options for the various operations given in the assignment

**Insert** (Upload file into GCS and if size is less than 100 Kb upload into Memcache also)

gcs.open() used to insert into GCS.

memcache.add() used to insert into Memcache for file size less than 100 KB

**Check Key** (if file exists)

Memcache.get() to check the file existence in memcache

Gcs.open() to check in file existence in gcs

**Find Key** (Download File)

Memcache.get(<filename>) to find file in memcache and download it

gcs.open(<filename>) to open to file and download it from gcs

**Remove Key** (Delete File)

Gcs.delete() to remove the single file in gcs

**Key listing** (List all files)

files.listdir() to list the file in the Bucket

**Find key in GCS** (if file exits in GCS)

gcs.open(<filename>) to open to file and download it from gcs

**Find key in Memcache** ( if File exits in Memcache)

Memcache.get(<filename>) to find file in memcache and download it

**Remove all Cache**(Removes files from Memcache alone)

Memcache.flush\_all() to remove all files in memcache

**Remove all**(Removes files from both GCS and memcache)

Files.delete() to remove all files in bucket

**Cache Size**( Mem cache size)

Memcache.get\_stats() to retrieve the status of memcache and use “bytes” key to get the size of memcache

**Cache elements** (Memcache element list)

Memcache.get\_stats() to retrieve the status of memcache and use “items” key to get the list files of memcache

**Storage Size** (GCS Storage Size)

Gcs.listbucket() to find the files in GCS and we incrementally add the file sizes of each file to get the total size of the bucket

**Storage Elements** (GCS Element list)

Files.listdir() to find the files in bucket and hence the no of files

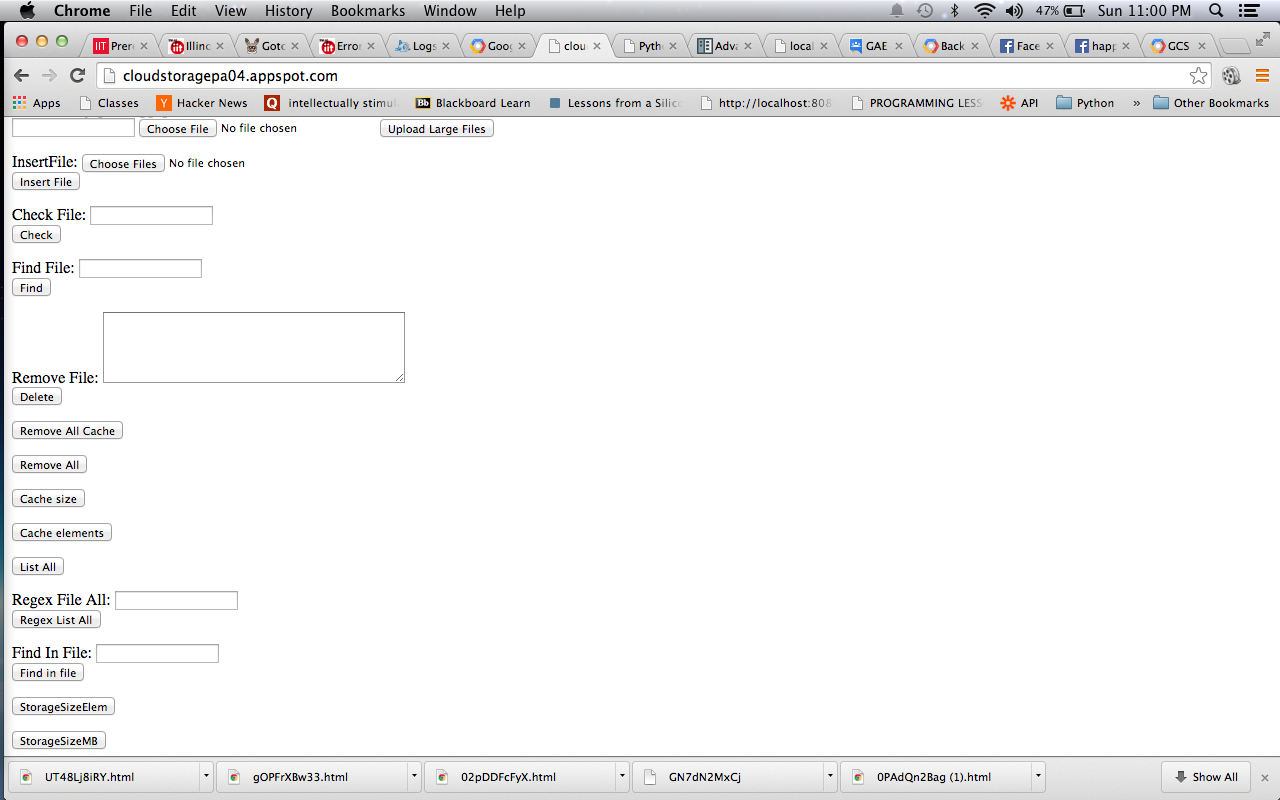
**Find in File** (Find Expression in File)

Gcs.listbucket() to get files in bucket and use re module in python to match the regex pattern with single file

**List Filename Based on Partial String**(Find all the file names which contain the string regular expression)

Gcs.listbucket() to get files in bucket and use re module in python to match the regex pattern with list of files.

The brower Interface looks like this



**Randomized File Creation:**

We create the files which are given random alpha-numerical names of 10 characters long. The contents of the files are random strings of 100 bytes long per line. Hence a 1KB file contains 10 line of 100 characters each. This randomized file was created using inbuilt Java API function which returns Alphanumeric String of varied size containing alphabets small and capital and numbers.

It has been included as a runnable jar called filecreate.jar and the class name is create without a package

Hence the file can be executed using terminal and using the class name create in filecreate,jar.

**Manual:**

1. **Download GCS client library using > pip install GoogleAppEngineCloudStorageClient -t <your\_app\_directory>**
2. **Use <path to google\_appengine>/appcfg.py update <path to your application> to upload the cloud storage application to google app engine.**
3. **Use the application url unique to your application to upload and download files to your cloud storage.**
4. **Use the default bucket ( in this case cloudstoragepa04.appspot.com) to store files.**

**Steps to bench mark the cloud storage application using a multithreaded client program.**

1. **Run the script ./bm\_threading .py to bench mark using 4 threads.**
2. **Run the script ./bm\_nothread.py to bench mark for no threads.**
3. **Edit both of the files to change the target of the thread to either upload or search or delete to benchmark the corresponding application functionality.**

**Performance:**

The performance evaluation was done using a Client Python Program which calls on the Python server deployed on the Google app engine and gives it posts and gets requests in order to upload, find and delete files in the Google cloud storage. Hence we use the client program on a local machine in order to implement client multithreading and benchmarking of the server upload, find and delete requests. Thus it mostly depends on the internet plan which is used on the client side where the local system is situated. And the performance measures will change according to the internet speed in usage.

|  |  |  |
| --- | --- | --- |
| **Throughput (MB/sec)** | **single thread** | **multithread** |
| **insert with memcache** | **0.11** | **0.287** |
| **insert without memcache** | **0.1042** | **0.281** |
| **find with memcache** | **0.252** | **0.704** |
| **find without memcache** | **0.244** | **0.683** |
| **delete with memcache** | **0.809** | **1.8** |
| **delete without memcache** | **0.782** | **1.77** |

**THROUGHPUT (MB/sec) GRAPH**

**x-axis :no of threads which is 1 and 4 threads**

**y-axis :Throughput (MB/sec)**

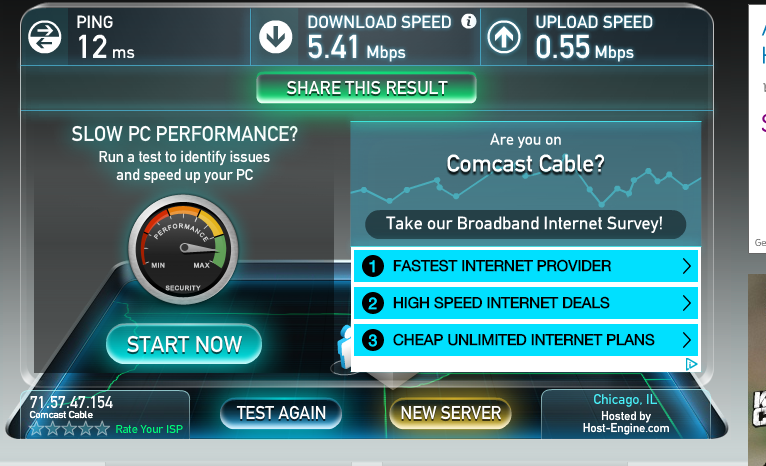
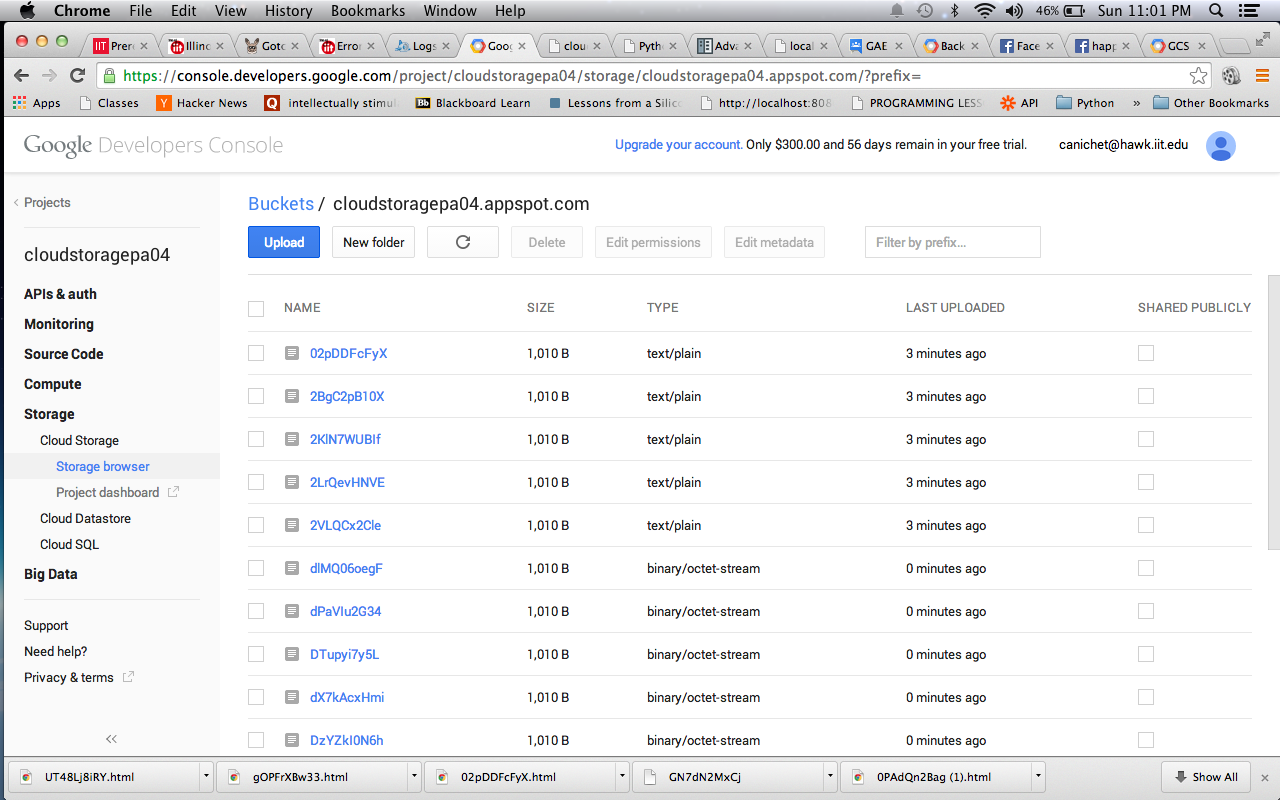
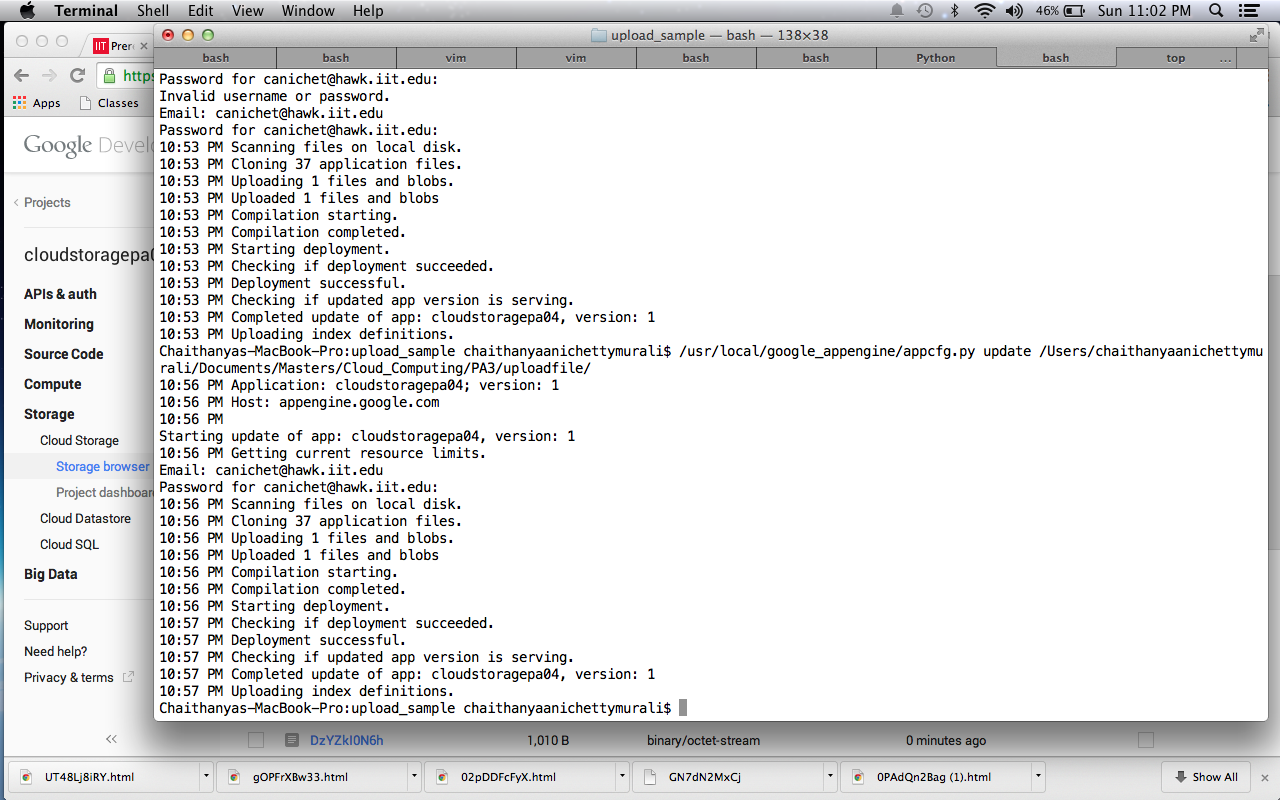
Hence it can be deduced from the graph that the insert, find and delete operation depend on the Network bandwidth in use and it varies from client to client. But we can see that the insert(upload) is much slower that the find operation (which is download) and in turn which is much slower than the delete which depends on the I/O speed in the disk in the cloud storage.

Thus the Delete is much faster than both insert and find. Also find which is download is much faster than the insert.

There isn’t much difference between the throughput for using memcache and without memcache as the network bandwidth drones out the difference. But it can be noticed from the time taken that the find using memcache is much faster than the find without using memcache.

The **average file access latency** for the 411 files upload which is write and download which is read is 12.6 micro seconds.

**SCREENSHOTS:**



**Comparison to Amazon S3:**

Google cloud storage costs were compared with the amazon s3 storage costs and it can be deduced that the **Google cloud storage costs are cheaper** for the given scenario. The Scenario given is workload increased to 1 million, hence dataset is now 411 million files that stores 311 TB. The 1 million users need to transfer 311 TB of data to the cloud, and then read 622 TB of data from the cloud. Since the workload is spread out over 1 month time, the total cost is broken up into 3 categories - transfer the data in (311TB of data), store the data (311TB of data), and retrieve the data (622TB of data). The Google cloud storage cost was calculated to reach about $13,391.44for one month including the I/O throughputs for the put and get requests from clients. Similarly the Amazon s3 cloud storage costs were calculated to be about $ 61767.62 for the same scenario. Hence it is clear from the storage costs and the I/O costs alone that the Google cloud storage is cheaper for these conditions of above. This does not include the compute costs, server running costs as it was not requested. In conclusion we can say that **Google cloud storage has much cheaper storage and I/O costs compared to Amazon S3.**