

Distributed Storage Using Google App Engine

CS553 Fall 2014

Summary

This report represents a detailed analysis on how to implement distributed file storage using google app engine into GCS and Memcache. The web application performs different functions like upload, find, remove and list of files present in the storage system. Performance of one and four threads are compared and performance plot is generated.

Introduction:

Platform as a service (PaaS): PaaS is a category of cloud computing services that provides a computing platform and a solution stack as a service. Along with software as a service (SaaS) and infrastructure as a service (IaaS), it is a service model of cloud computing. In this model, the consumer creates an application or service using tools and/or libraries from the provider. The consumer also controls software deployment and configuration settings. The provider provides the networks, servers, storage, and other services that are required to host the consumer's application.

PaaS offerings facilitate the deployment of applications or services without the cost and complexity of buying and managing the underlying hardware and software and provisioning hosting capabilities. All "as-a-service" offerings are characterized as providing low initial cost, incremental cost as your service usage grows, self-service, best practices built-in, resource sharing, automated deployment, management services, lifecycle management, reuse. PaaS provides these capabilities for application and service development.

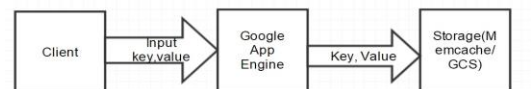
Google App Engine: Google App Engine is a PaaS offering that lets you build and run applications on Google's infrastructure. App Engine applications are easy to build, easy to maintain, and easy to scale as your traffic and data storage needs change. With App Engine, there are no

servers for you to maintain. You simply upload your application and it's ready to go.

1. Design

The application implements **HTML/CSS** and **JavaScript** for the front-end and **Python** in the back-end for the development of the web application in accordance with APIs provided by google. The primary storage location for the files that are uploaded by the user is GCS bucket: **/cloudstore-storage/** and if the file size is less than 100Kb the file is saved In both Memcache and GCS.

The system works based on the Client-Server model, where the client webpage is rendered using **JINJA2** and the request is processed using **Webapp2 Request Handler**. The main purpose of this application is to provide file storage on cloud where the user can upload and download files from the cloud along with other basic functionalities.



The user can perform the following functionalities:

- (i) Insert files
- (ii) Check files
- (iii) Find (download) files
- (iv) Remove
- (v) List the contents

The server responds to the client request and acts accordingly.

Distributed Storage Using Google App Engine

CS553 Fall 2014

Design Trade-offs:

The application has made use of webapp2 which is allowing us to handle HTTP request and respond to the requests.

We are making use of google.appengine.api which is allowing us to access Memcache and GCS API.

We have tried to use one page application model to maximize the ease of user's access.

In order to upload files greater than 20 MB we are splitting the file into chunks and recursively uploading the files into GAE where the files are merged and stored.

The data stored in the Memcache is restricted for an hour in order to reduce storage costs.

Future Improvements:

- Asynchronous requests can be given instead of form submit.
- File compression can be implemented in order to reduce the payloads.
- If the file is not present in Memcache, the keys should be fetched from the GCS and stored in the Memcache.
- To make the application multi-browser compatible.

2. Manual

Requirements:

- Python 2.7.x
- Google App Engine SDK
- Google Chrome

The entire project consists of three sections

(i) File generator

The program is used to generate dataset containing random data with the specified file sizes, considering 1KB = 1024Bytes and 1MB = 1024KB. The program is developed using python

where the file containing 100 Bytes per line and the file name restricted to 10 characters.

Execution steps:

\$ python filegen.py

(ii) Single Threaded model

The program is designed to perform six operations.

(a) Insert

HTML form is used to get the input file(s) from the user. Form type = "file" tag is used. The request is processed using webapp2, where the parameters are read and the file size is computed and stored in both Memcache and GCS if the size is less than or equal to 100KB and file sizes greater than 100KB are stored in GCS. The file is stored as a key,value pair where the key is the name of the file and the value represents the contents of the file.

```
cloudstorage.open(filename, mode='w', content_type=None, options=None, read_buffer_size=storage_api.ReadBuffer.DEFAULT_BUFFER_SIZE, retry_params=None)
```

```
memcache.add(key, value, time=0, min_compress_len=0, namespace=None)
```

(b) Check

The check operation obtains the key from the user and checks whether the key is present in the Memcache or GCS.

```
cloudstorage.stat(filename, retry_params=None)
```

```
memcache.get(key, namespace=None, for_cas=False)
```

(c) Find

The find operation retrieves the file from the Memcache or GCS upon the user's request. The key is entered by the user.

Distributed Storage Using Google App Engine

CS553 Fall 2014

```
cloudstorage.open(filename, mode='r', content_type=None, options=None, read_buffer_size=storage_api.ReadBuffer.DEFAULT_BUFFER_SIZE, retry_params=None)
```

```
memcache.get(key, namespace=None, for_cache=False)
```

(d) Remove

The remove function prompts the user to enter the key of the record to be removed from the Memcache and GCS.

```
cloudstorage.delete (filename, retry_params=None)
```

```
memcache.delete(key, seconds=0, namespace=None)
```

(e) List

The function will list all the items present in Memcache and GCS and it will display as a list to the user. There is an API for GCS to list the contents of the file but in case of Memcache, we have made a master file which consists of the keys that are uploaded and displayed them.

(f) Find All

It will download all the files present in the storage according to the keys present in the master file and the bucket list.

Execution steps:

- Create an application in GAE console with a google account
- Go to console.developer.google.com And enable billing
- Under API , enable google cloud storage and under storage section go to storage browser
- Add a bucket

The deployment descriptor app.yaml must be updated with

- Application Identifier
- Version
- Handler (URL Mapping)
- Libraries(Third Party Extensions)

Open main.py and mainthread.py and replace the bucket_name with the storage bucket created.

Deploy the python code in Google App Engine using the command given below

```
$ ./appcfg.py update (Path to workspace)
```

- Open Google Chrome

URL : <http://6.gae-cloudstore.appspot.com/>

(iii) Multi-threaded model

The same functionalities are performed as Single thread model, but using four threads. The threads are created and initialized using

```
class myThread (threading.Thread):
```

```
def __init__(self, threadID, name, counter):
```

and are triggered using **thread.start()**, which in turn calls the **run(self)** method which performs threading. The application uses 4 threads to process multiple requests.

Execution Steps

- Application Identifier
- Version
- Handler (URL Mapping)
- Libraries(Third Party Extensions)

Deploy the python code in Google App Engine

```
$ ./appcfg.py update (Path to workspace)
```

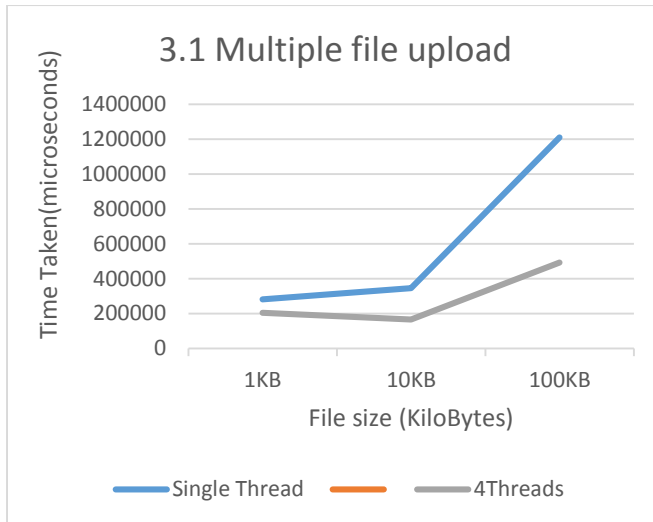
- User is redirected to the web page

URL : <http://9.gae-cloudstore.appspot.com/>

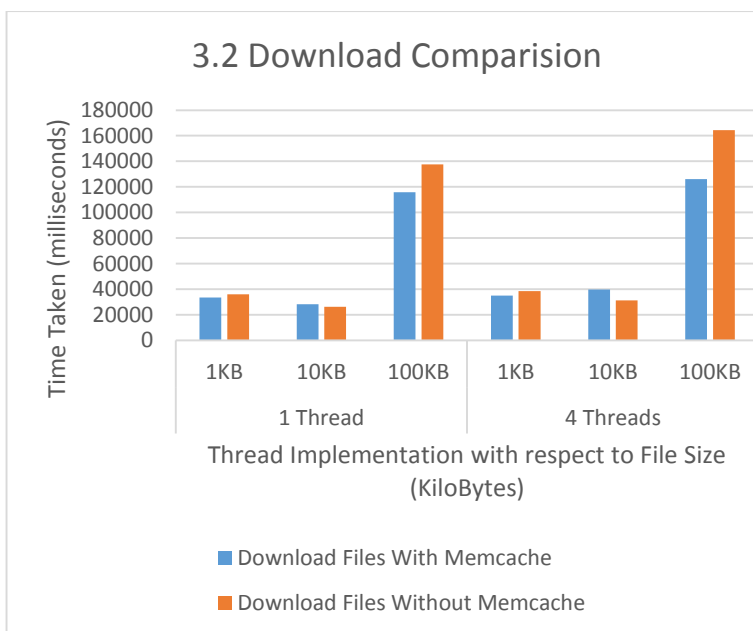
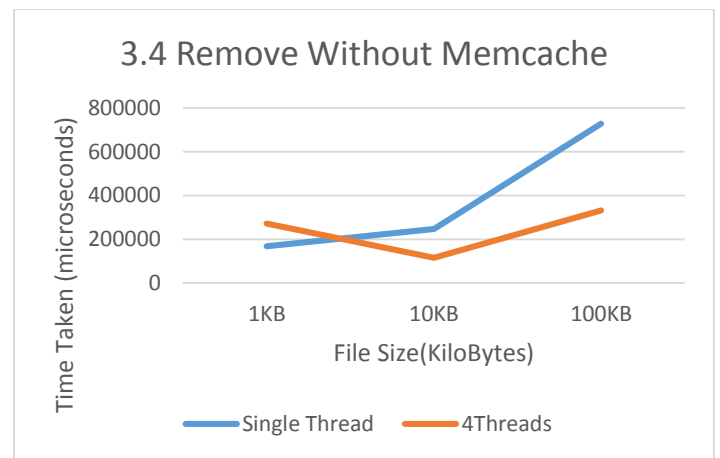
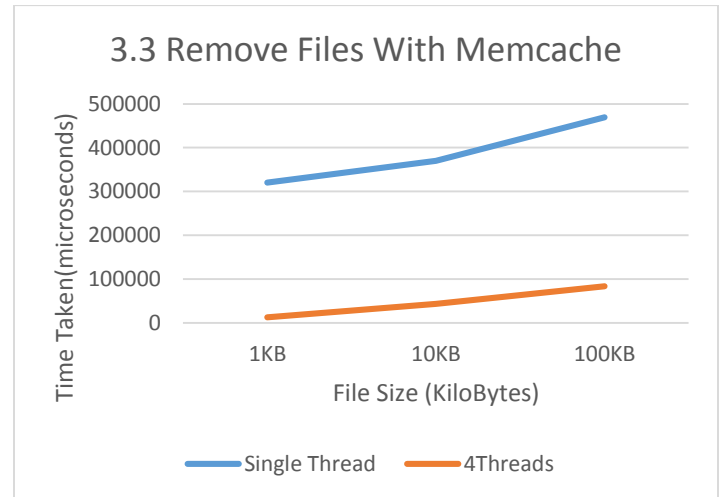
Distributed Storage Using Google App Engine

CS553 Fall 2014

3. Performance



From the above graphs we can observe that upload of large amount of files can be optimized using multiple threads as in 3.1. Memcache is a server side memory storage used for faster retrieval of smaller file sizes. We can see a considerable increase performance when downloading using memcache as in figure 3.2.



Removing multiple files using Memcache causes the file to be removed from both Memcache and Google Cloud Storage, hence takes more time(as in Fig. 3.3 & Fig. 3.4).

Throughput & Latency:

Throughput when measured for gae-cloudstore.appspot.com averages to **7.700 MB/s** across various upload operations and the measured latency for downloading of one file is **59 ms**. The roundtrip time measured through ping utility averages to **22ms**

Distributed Storage Using Google App Engine

CS553 Fall 2014

```
C:\windows\system
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Rahul>ping www.6.gae-clodustore.appspot.com

Pinging appspot.1.google.com [74.125.201.141] with 32 bytes of data:
Reply from 74.125.201.141: bytes=32 time=25ms TTL=47
Reply from 74.125.201.141: bytes=32 time=20ms TTL=47
Reply from 74.125.201.141: bytes=32 time=22ms TTL=47
Reply from 74.125.201.141: bytes=32 time=23ms TTL=47

Ping statistics for 74.125.201.141:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 20ms, Maximum = 25ms, Average = 22ms

C:\Users\Rahul>ping www.9.gae-clodustore.appspot.com

Pinging appspot.1.google.com [74.125.201.141] with 32 bytes of data:
Reply from 74.125.201.141: bytes=32 time=19ms TTL=47
Reply from 74.125.201.141: bytes=32 time=22ms TTL=47
Reply from 74.125.201.141: bytes=32 time=24ms TTL=47
Reply from 74.125.201.141: bytes=32 time=23ms TTL=47

Ping statistics for 74.125.201.141:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 19ms, Maximum = 24ms, Average = 22ms

C:\Users\Rahul>
```

4. Comparison to Amazon S3

The cost of storage was calculated using simple monthly calculator provided by Amazon Web services and Google pricing calculator.

	S3	GCS
Transfer 311 TB	\$22047.33	N/A
Store 311 TB	\$10187.51	\$8280.06
Retrieve 622TB	\$39085	\$38215

The table above shows that Google storage is cheaper compared to S3 in case of storing 311TB data as well as retrieving 622 TB of data.

Constraints:

- We tried uploading 10*10MB files and 1 100MB file using form submit but due to insufficient browser memory we got chrome crash report.

- All the above calculations are inclusive of JavaScript operations on file data. Which adds few overheads on uploading and downloading multiple files.
- The throughput depends on the consistency of the internet connection.
- GCS will be unavailable if any of the instance is flushed due to inconsistency in Appengine services.
- Values may be evicted from the cache on addition of new values if running low on memory.
- Application availability depends upon various google services <https://code.google.com/status/appengine>

Conclusion:

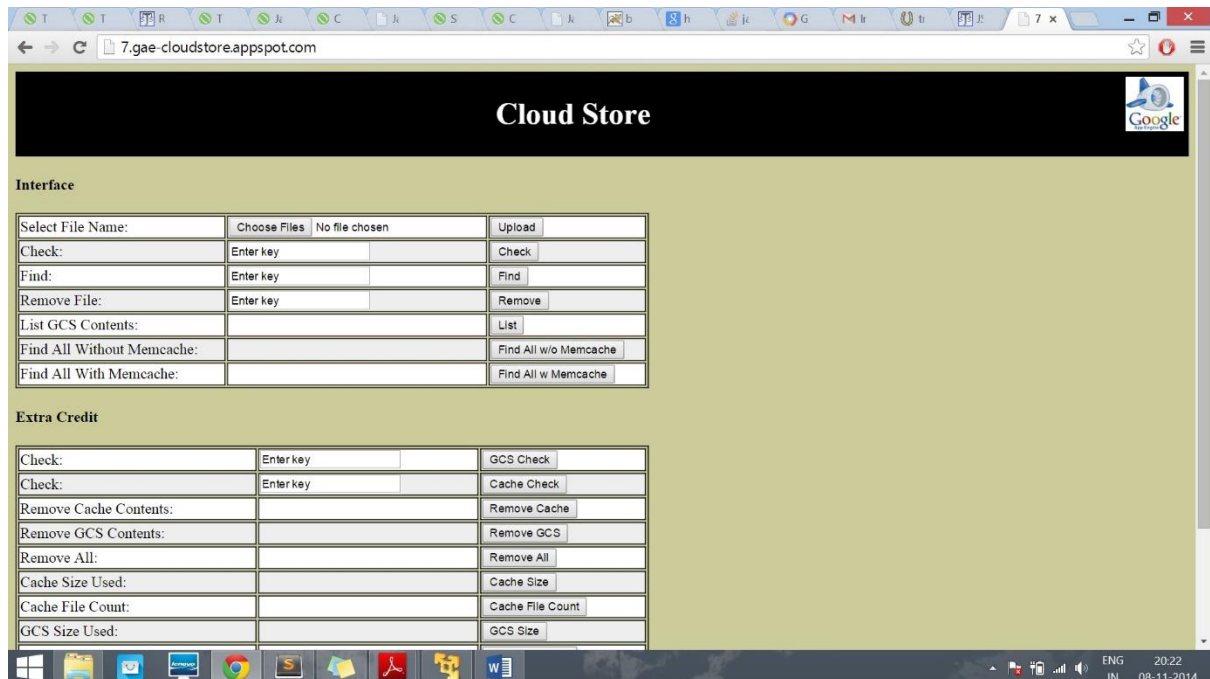
In this project we tried to analyze the various distributed storage services provided by Google AppEngine Service and compares its performance with Amazon S3. We found that using Google AppEngine as web application development environment with Google Cloud Storage Service gives an optimal cost.

<http://www.cloudberrylab.com/blog/amazon-s3-azure-and-google-cloud-prices-compare/>

This link compares the recent price drops in both the services indicating Google AppEngine is in Lead.

SCREENSHOTS:

1.Web Application



2. Appengine Logs

The screenshot shows the Google Developers Console interface for the 'gae-cloudstore' project. The left sidebar contains navigation links for Projects, CloudStore, APIs & auth, Monitoring, Source Code, and Compute. The 'Monitoring' section is expanded, showing 'Logs'. The main content area displays a list of logs for the 'App Engine' service, filtered by 'default module' and '7'. The logs are dated '2014-11-08' and scanned between '18:57:27' and '20:23:35'. The log entries show various requests, including 'wp4.png' and 'wp4.png', with status codes, response sizes, and response times. The bottom of the screen shows a Windows taskbar with various application icons and a system clock indicating '20:27' on '08-11-2014'.

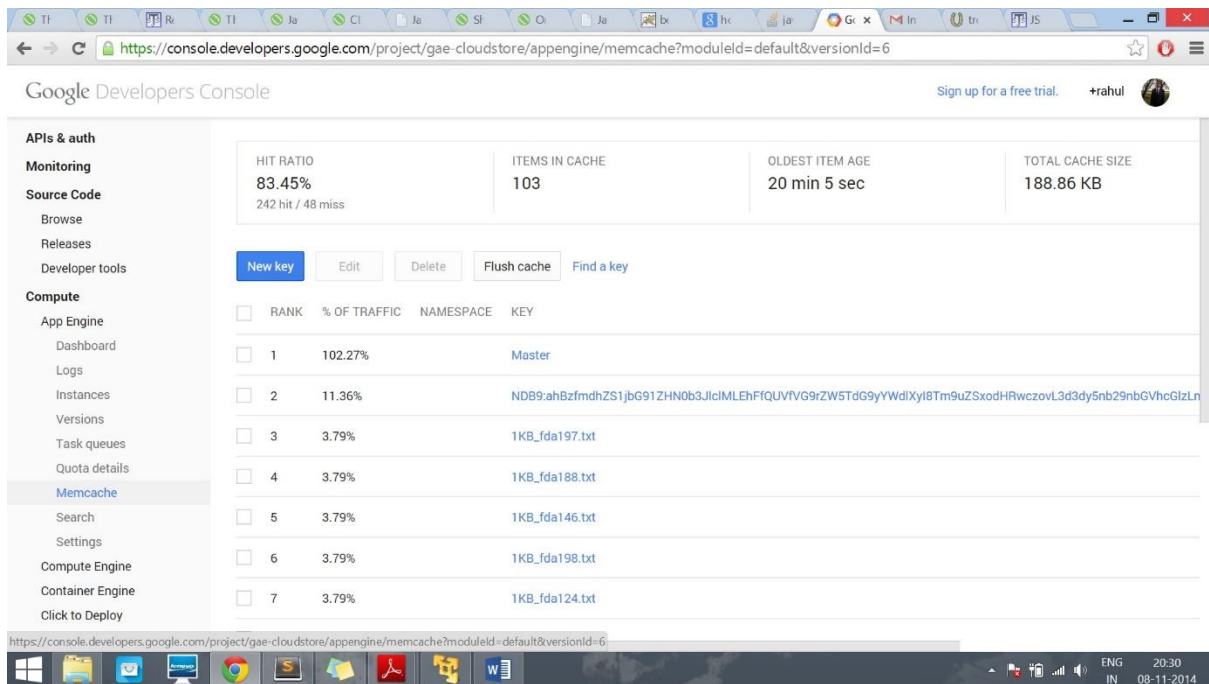
Time	Status	Size	Response Time	URL
20:21:22.097	200	2.93	10ms	/
20:21:22.460	404	188 B	9ms	/wp4.png
20:21:23.670	200	2.93	10ms	/
20:21:24.519	404	188 B	8ms	/wp4.png
20:21:26.546	200	2.93	1278ms	/
20:21:26.722	200	2.93	10ms	/
20:21:26.929	404	188 B	9ms	/wp4.png
20:21:28.719	200	2.93	11ms	/
20:21:28.969	404	188 B	9ms	/wp4.png
20:21:31.825	200	2.93	58ms	/
20:21:33.939	404	188 B	59ms	/wp4.png
20:23:05.182	200	123 B	316ms	/
20:23:07.717	200	2.93	11ms	/
20:23:34.852	200	2.93	11ms	/
20:23:35.071	404	188 B	8ms	/wp4.png

3. Google Cloud Storage

The screenshot shows the Google Developers Console interface for the 'gae-cloudstore' project, specifically the 'Cloud Storage' section. The left sidebar contains navigation links for Developer tools, Compute, and Storage. The 'Storage' section is expanded, showing 'Cloud Storage'. The main content area displays a list of buckets and files. The 'Buckets / cloudstore-storage' section shows a list of files, including '1KB_fda100.txt', '1KB_fda101.txt', '1KB_fda102.txt', '1KB_fda103.txt', '1KB_fda104.txt', '1KB_fda105.txt', '1KB_fda106.txt', '1KB_fda107.txt', '1KB_fda108.txt', and '1KB_fda109.txt'. Each file is 1 KB in size, has a 'text/plain' MIME type, and was uploaded '0 minutes ago'. The bottom of the screen shows a Windows taskbar with various application icons and a system clock indicating '20:29' on '08-11-2014'.

File Name	Size	MIME Type	Uploaded
1KB_fda100.txt	1 KB	text/plain	0 minutes ago
1KB_fda101.txt	1 KB	text/plain	0 minutes ago
1KB_fda102.txt	1 KB	text/plain	0 minutes ago
1KB_fda103.txt	1 KB	text/plain	0 minutes ago
1KB_fda104.txt	1 KB	text/plain	0 minutes ago
1KB_fda105.txt	1 KB	text/plain	0 minutes ago
1KB_fda106.txt	1 KB	text/plain	0 minutes ago
1KB_fda107.txt	1 KB	text/plain	0 minutes ago
1KB_fda108.txt	1 KB	text/plain	0 minutes ago
1KB_fda109.txt	1 KB	text/plain	0 minutes ago

4. Memcache



Google Developers Console

Sign up for a free trial. +rahul

APIs & auth

Monitoring

Source Code

Browse

Releases

Developer tools

Compute

App Engine

Dashboard

Logs

Instances

Versions

Task queues

Quota details

Memcache

Search

Settings

Compute Engine

Container Engine

Click to Deploy

HIT RATIO
83.45%
242 hit / 48 miss

ITEMS IN CACHE
103

OLDEST ITEM AGE
20 min 5 sec

TOTAL CACHE SIZE
188.86 KB

New key Edit Delete Flush cache Find a key

<input type="checkbox"/>	RANK	% OF TRAFFIC	NAMESPACE	KEY
<input type="checkbox"/>	1	102.27%		Master
<input type="checkbox"/>	2	11.36%		NDB9:ahBzfmdhZS1jbG91ZHN0b3JlclMLEhFQUVVG9rZW5TdG9yYWdlXyl8Tm9uZSxodHRwc2ovL3d3dy5nb29nbGVhcGlzLn
<input type="checkbox"/>	3	3.79%		1KB_fda197.txt
<input type="checkbox"/>	4	3.79%		1KB_fda188.txt
<input type="checkbox"/>	5	3.79%		1KB_fda146.txt
<input type="checkbox"/>	6	3.79%		1KB_fda198.txt
<input type="checkbox"/>	7	3.79%		1KB_fda124.txt

https://console.developers.google.com/project/gae-cloudstore/appengine/memcache?moduleId=default&versionId=6

ENG 20:30 08-11-2014

References:

<https://cloud.google.com/appengine/docs/python/gettingstartedpython27/introduction>

<https://cloud.google.com/appengine/docs/python/tools/webapp2>

<https://cloud.google.com/appengine/docs/python/memcache/>

<https://cloud.google.com/appengine/docs/python/memcache/functions>

<https://cloud.google.com/appengine/docs/python/googlecloudstorageclient/>

http://www.tutorialspoint.com/python/python_multithreading.htm

<http://www.w3schools.com/js/default.asp>

http://www.w3schools.com/html/html_layout.asp

<https://developer.mozilla.org/en-US/docs/Web/API/Blob>

<http://webapp-improved.appspot.com/guide/request.html#guide-request>

http://www.tutorialspoint.com/python/python_cgi_programming.htm

<https://groups.google.com/forum/#!forum/google-appengine>

CREDITS

1. Arihant Raj Nagarajan(A20334121)
2. Rahul Krishnamurthy(A20330185)
3. Shashank Sharma(A20330372)