Distributed File System with Google App Engine

Vivek Pabani

Illinois Institute of Technology
CWID A20332117
vpabani@hawk.iit.edu

Abstract

The distributed storage system built on top of Google App Engine is a part of and assignment under course CS553 - Cloud Computing. The storage system supports basic and advanced functions of a general file system. It uses the google cloud storage library to store the files in cloud storage and Memcache to improve the performance.

I. Introduction

Google App Engine is a Platform as a Service (PaaS) offering that lets you build and run applications on Google infrastructure. I have built a distributed storage system on top of Google App Engine, using Google Cloud Storage Client Library. This storage system supports storing arbitrary files using a unique key, where the key is a unique filename. The storage system provides basic functions such as search, download and remove files from the cloud storage. It also provides additional functionalities such as finding number of files in storage system, or finding a string in given file. I have also implemented memcache for small size files to improve performance. The benchmarking was done on a dataset of 411 files for uploading and downloading all the files with 1 and 4 threads.

II. Basic Functions

I. insert(key, value)

Insert function inserts a file with key as filename and value as file-data. Google App Engine restricts the file size to be 32 MB per request to be handled by the application. To over come this restriction, I have implemented insert function in two parts. User can upload multiple files with total size less than 32 MB, which will be processed by the application, and will be inserted to gcs bucket. For large files, user can upload single file of any size which will be redirected to gcs bucket by POST method. If the file size less than 100 KB, the file-data is also stored to memcache as key-value pair.

II. check(key)

Check function searches for given file in the system. It first checks the memcache; If file is not found in memcache, it searches in the cloud storage.

III. find(key)

Find functions is used to get the file with given key. It first invokes check function; If file exists, it checks the memcache and cloud storage, and returns the file.

IV. remove(key)

Remove function is used to delete the file from memcache and cloud storage. This function first checks if file exists in system, and delete the file if it is found.

V. listing()

Listing functions returns the list of files stored in the cloud storage. The return type of this function is an array of filenames.

III. Additional Functions

I. checkStorage(key)

CheckStorage function checks for the given file only in cloud storage.

II. checkCache(key)

CheckCache function is used to check if given key exists in the memcache.

III. removeAllCache()

RemoveAllCache removes all the data from memcache. It uses flushAll functionality of class memcache.

IV. removeAll()

RemoveAll function removes all the files and data from cloud storage and memcache. It invokes the listing function, and performs remove function on each file.

V. cacheSizeMB()

CacheSizeMB function returns the size of data stored in memcache. It uses getStats functiuon of class memcache.

VI. cacheSizeElem()

CacheSizeElem function returns the number of elements stored in memcache. It uses getStats function of class memcache.

VII. storageSizeMB()

StorageSizeMB function calculates the total size of files stored in cloud storage. It first invokes the listing function, and uses stats function to get size of each file.

VIII. storageSizeElem()

StorageSizeElem function counts the number of files stored in cloud storage. It invokes the listing function and returns the size of list.

IX. findInFile(key, string)

FindInFile function accepts one key as filename and a string as value. It searches the file for the given string and returns true if found.

X. listingRegEx(string)

ListingRegEx function takes a string as argument, and searches the cloud storage for filename matching with the string. This function invokes listing function, and searches for matching filename.

IV. Data-set Generation

As per program definition, I have created 411 files with size ranging from 1 KB to 100 MB for dataset. The total size of dataset is 311 MB. All the files have 100 characters per line. The filenames and file-data are generated using choice method of class random from python library.

V. Workload Generation

There are two types of workload generation for this assignment. For upload task, I divided the dataset into small equal size groups of files. These files were uplaoded to the cloud storage and memcache with help of insert function described in basic functions. Google App Engine restricts the response time of one request to 60 seconds. For downloading all the files from cloud storage, I divided the files in groups to over come the response time limit. The work load logic is written on server side, which reads the files and write the response to one output file.

VI. Benchmark

For the upload task, all 411 files were divided into groups and uploaded to the cloud storage using insert function. This task was done with 1 thread and 4 threads. Upload was performed with and without memcache for benchmarking. The difference between perfomance with 1 and 4 thread was significant, while memcache did not affect the performance. The latency for insert with 1 thread was 5.4 micro seconds. For 4 threads, the latency was 4.3 micro seconds.

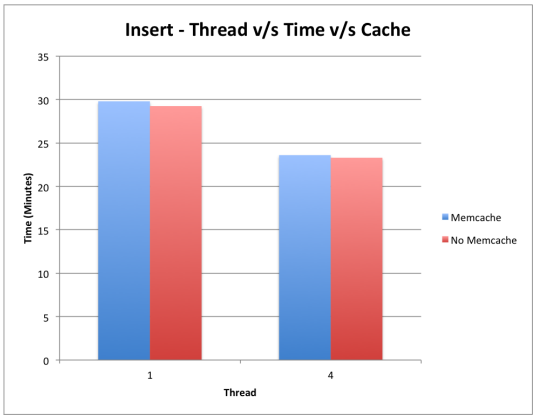

Figure 1: *Insert function performed on 411 files.*

The download task was done for all 411 files twice with random distribution using find function. The task was divided into groups to over come the response time and size limit imposed by GAE. The performance with 4 threads was better than performance with 1 thread. Memcache improved the performance by small margin. This was mainly due to the fact that big size files dominated the download task. The

latency for find with 1 thread was 19.3 micro seconds. For 4 threads, the latency was 17.3 micro seconds.

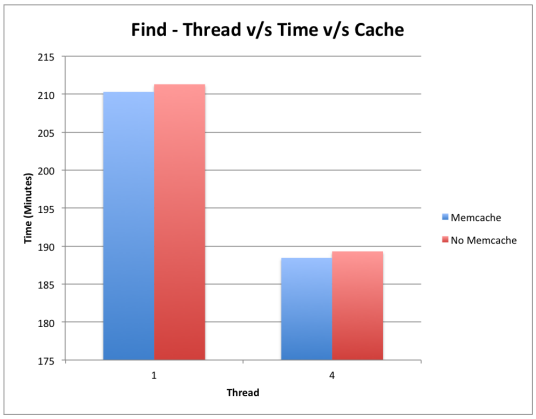

Figure 2: *Find function performed on 822 files.*

The remove task was done on all the files with 1 and 4 threads. The time to remove all files with 1 thread was 1.3 minutes. For 4 threads, the time was 0.9 minutes.

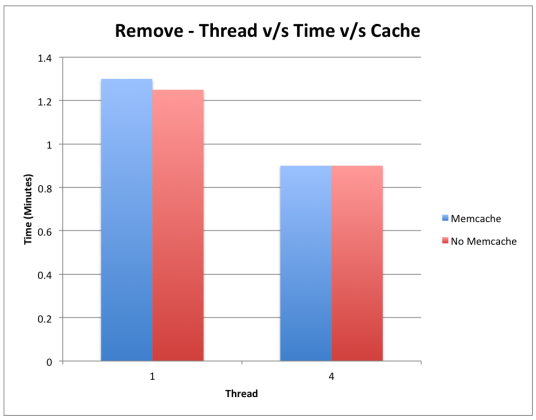

Figure 3: *Remove function performed on all 411 files*

VII. Comparison

If the workload is increased by 1 million times then we need to store 311 TB data with network transfer of 622 TB. As per rates mentioned on amazon website, the total cost over one month for such a workload will be approximately USD 38500 while GAE charges approximately USD 58000 for given workload. So clearly, amazon S3 will be prefered over Google App Engine

VIII. Conclusion

The distributed file system uses only google cloud storage to store the files as per program definition. Current system supports all the required functionalities with acceptable performance. The performance can be improved with use of Blobstore which supports multiple uploads in one request. The performance can also be improved by allowing the memcache to store files less than 1 MB instead of 100 KB.