CS 553 Cloud Computing

Programming Assignment 3

Boyang Li (A20314367)
Xingtan Hu (A20304622)
Zichen Zhang (A20307812)

Responding Link

http://li-hu-zhang.appspot.com

Design

1. Overall Program Design

This program covers implementation of a non-trivial distributed system by using Google App engine. Then making the comparison between GAE and Amazon S3 on cost effective.
We build a distributed storage system on top of Google App Engine, by using Google Cloud Storage Client Library. We use Memcache to store small files (which <= 100KB), and Google Cloud Storage to store medium and large files.
Our storage system can be accessed by using a web browser. We use webapp2 technology. Through this web page, multiple operation can be achieved.

1) **Insert(Key, Value)**: add a file named key with the contents stored in value to the distributed storage system. And based on requirement, big file stores in Google Cloud Storage, small files stores in Memcache;
2) **Check(Key)**: verify if a file named key exists in the Memcache or Google Cloud storage;
3) **Find(key)**: retrieves the contents of a file named key from the Memcache or Google Cloud storage;
4) **Remove(key)**: remove the file named key from the Memcache or Google Cloud storage;
5) **Listing()**: retrieve a list of all file names as an array. Besides that, we also add the length of the array to this operation, which you can see how many files in the whole storage system;
6) **Check cloud storage(key)**: verify if a file named key exists in the Google Storage System;
7) **CheckCache(key)**: verify if a file named key exists in the Memcache of the storage system;
8) **RemoveAllCache()**: Remove all the files in Memcache;
9) **RemoveAll()**: Remove all the files in Memcache and Google Storage System;
10) **CacheSizeElem()**: retrieve the total number of files in the Memcache of distributed storage system;
11) **StorageSizeElem()**: retrieve the total number of files in the Google Cloud storage;
12) **FindInFile(key, string)**: searches for a regular expression in file key;

13) **Listing(string)**: retrieve a list of all file names as an array, whose names match the regular expression string.

2. Design Tradeoffs

At the beginning of doing this assignment, we try to use Java + JSP to achieve the goal of this assignment, but when we try to upload files to the Google App Engine, we find that in the request structure, what we got actually is the empty information. So we decided to change our method, which using Python to complete this assignment. However, by using Python, we lose some program's executing efficiency which compared to Java.

3. Possible Improvement

Because all the benchmark and performance testing is based on the communication between client and server, which makes network speed a key factor. We did a lot of benchmark and performance testing in school's network. Unfortunately, the quality of network environment is very poor these days, sometimes it's even unstable. When we did the testing part, the results of same testing can be various due to the network speed.

4. Possible Extensions

Due to the time restriction, we didn't achieve more extra credit and some great extension which we really want to make it happen.
In our Listing() operation, users can only see the array list of the file, and how many files exist in Memcache or Google Cloud Storage. Actually, download and delete operation should also be added to the Listing() operation.
For example, when users at the Listing web page of the storage system, it will be better if he can directly download or delete arbitrary file(s) through the web page.

Manual

1. Client Instruction

At client side, what will show to you is a web page. When you open a browser and type into the application address provided before, a HTML page will be displayed. The page is consisted by two parts: Main operation and Extra credit. You could select the operations listed below each parts. For "insert file" operation, you may select one or multiple files at one time. If you just select one file, you can choose whether to input the File key or not. The default setting of File key for the file is file name. Don't input File key when you select multiple files. After you already selected files, click the submit button and the files will be automatically transferred to Google Cloud Storage or Memcache.

For "Find" operation, you can input the file key and it will retrieves the contents of the file key from the cache or the distributed storage system.

For "Check" operation, you can input the file key and it will verify if a file named key exists in the cache or distributed storage system. If exists, return the result(boolean) to you.

For "Remove" operation, you can input the file key and it will remove the file named key from the cache and the distributed storage system.

For "List file" operation, this will retrieve a list of all file names back to you.

Below is for "Extra Credit" part.

For "Check in Storage" operation, this will verify if a file named key exists in the distributed storage system. If exists, return the result(boolean) to you.

For "Check in Cache" operation, this will verify if a file named key exists in the cache of the distributed storage system. If exists, return the result(boolean) to you.

For "Remove All Cache" operation, this will remove all files from the cache.

For "Remove All" operation, this will remove all files from the cache and the distributed storage system.

For "Cache Size Element" operation, this will retrieve the total number of files in the cache of distributed storage system.

For "Storage Size Element" operation, this will retrieve the total number of files in the distributed storage system.

For "Find in File" operation, input the File key and what you want to search in the file(eg. An expression), the application will do the matching in the server side. If it finds the input expression in the file, the application will return the file to you. Otherwise, nothing will be returned.

For "Listing" operation, this will retrieve a list of all file names as an array, whose names match the regular expression string.

2. Server Instruction

At the server side, which is Google App Engine, there are totally 12 operations running on it. All these operations will be automatically mapped to a special handler to implement according to the labels within the html.

First, we are gonna briefly talk about five operations in the standard part. These five operations are Insert, Check, Find, Remove and Listing.

Insert: The Insert operation is one of the most important operations in this system. It should be invoked like insert(key, value), in which key is the filename by default and the value is the file content stored as an object type.
The interface for this operation on the web page is a upload button, controlled by UploadHandler. When the client has finished choosing the file and inputed the key and clicked the button. This operation will be called to upload all files for client.
During this procedure, two things will be checked at the server side. If the client doesn't specify a key for the file, then the key will be set as the filename by default. At the meantime, if the size of the file is no larger than 100KB, the file will be stored in Memcache instead of Google Cloud Storage in order to improve the performance of the system. To upload the file into Google Cloud Storage other than Memcache, we applied blob_store APIs to implement our job.

Check: This operation is used to find out whether a file exists in either Memcache or Google Cloud Storage. It is controlled by checkHandler at the server side.
The interface for check operation on the web page is a check button. How this works is that client gives a key as the input and this check operation will find the certain file stored with this key no matter in Memcache or in Google Cloud Storage. If no file be found, then prompts to client that there is no such file. Otherwise, prompts to client that the file exists and where it stored in.

Find: This operation is used to retrieve the contents of a certain file. In another word, it is the download operation. It is controlled by DownloadHandler at the server side.
To invoke this operation, the client only needs to give a special key as the input and click the download button on the web page.
Firstly, it will locate the file with the key, which is input, to decide whether the file is stored in Memcache or in Google Cloud Storage. If the file does not exist, this operation will give a message to the client that it does not exist. Otherwise, the web page will send a request to the server that asks to retrieve the data from either Memcache or Google Cloud Storage. Finally, the contents of the file will be write onto the web page so that the client can review the content easily without leaving the web page.

Remove: This operation is very much similar with Check operation. Although it is controlled by another handler called RemoveHandler. The first part of remove is exactly the same as check.
After the client invokes this operation from the web page by giving an input as key and clicking the remove button. This operation will delete the file no matter it is stored in Memcache or in Google Cloud Storage. Again, if file does not exist, client will be prompted with a message instead of crash the web page.

Listing: This is a different operation compared to the above operations. It is controlled by a handler called ListHandler and the implementation is quite straight forward.

There is no input needed for this operation, and when client invokes this operation, only one thing need to be done at the server side. When we upload the file, all the keys will actually be stored. So it just needs to get the number of keys and return it back to the web page to the client.

Next part contains some operations listed in the extra credit part. There are totally ten operations but we only implement 7 of them.

CheckCache: This operation is some kind like a easy version of the check operation listed in the standard part, because this operation only need to check Memcache to find out whether a file exists or not at the server side. And as always, prompts the results to the client on the web page. It is invoked as CheckCache(key), which means an input argument from client is needed for this operation.

CheckStorage: Like CheckCache, this operation receive an input key as argument and return to the client that whether this file exists or not. The difference between this operation and CheckCache operation is just they go through the files stored in different places.

RemoveAllCache: This operation gives client an easy way to delete all the files in the Memcache. Instead of deleting a certain file by matching key, this operation is pretty straight forward and less of work. At the server side, the handler only need to go through all the files in the Memcache and delete every single of them.

RemoveAll(): Very much like RemoveAllCache, no input argument needed and the handler at the server side just need to go through all the files stored in both Memcache and Google Cloud Storage and then delete every single of them.

CacheSizeElem: This operation is meant to give the number of files stored in Memcache to client. The same as RemoveAllCache, the handler only need to go through all files in the Memcache and simply calculate the number of files.

FindInFile: This operation could be the most complex one in this assignment. It will be invoked with the format like FindInFile(key, string).
At the server side, it can be divide into three major part. The first part is to locate the file in Memcache or Google Cloud Storage, we have implemented this in Check operation in the standard part. The second part is to retrieve the contents of the file, it also has been implemented in Find operation. The last part is to match the string, which is one of the inputs, this can be done with regular expressions at the server side.
The output of this operation is the a line of string that contains the given string if the file exists.

Listing: This operation is also very much alike the Listing operation in the standard part. The only different is matching the file name using blur searching rules. This operation will return a list of filename if they are exist.

3. File generator

In our zip file, there's a python file named "gen_file.py", which can create a dataset comprising of 411 files that is given random alpha-numerical names of 10 characters long. The contents of the files is random strings of 100 bytes long per line. There are 100 1KB files, 100 10KB files, 100 100KB files, 100 1MB files, 10 10MB files and 1 100MB file. The generated 411 files is already uploaded to the storage system(you can check it in Listing web page). After you run this python file, 411 files will be generated automatically under a folder named "gen_file" in the same directory with this python file.

4. Run Benchmark

In our zip file, there's a python file named "Benchmark.py", which is used to run the benchmark. To run this benchmark, you have to fill two argument. The first one is number of thread (1 or 4), the second one is the operation you want to do a benchmark (Insert, Find/Download and Remove). For example, Benchmark.py 1 insert (means run benchmark for insert operation in 1 thread).

Runtime Environment Settings

Python 2.7.6
Google App Engine
Google Cloud Storage

Performance

(PS: for specific timing data, you can find log in our log folder in zip file)

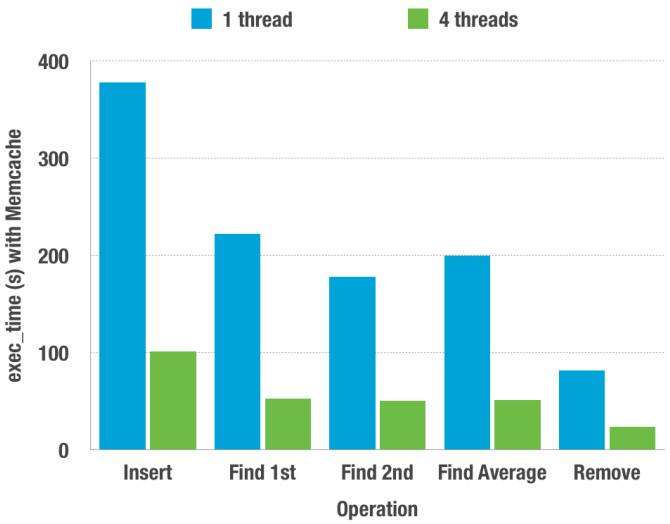
1. Benchmark and Experiment Overview

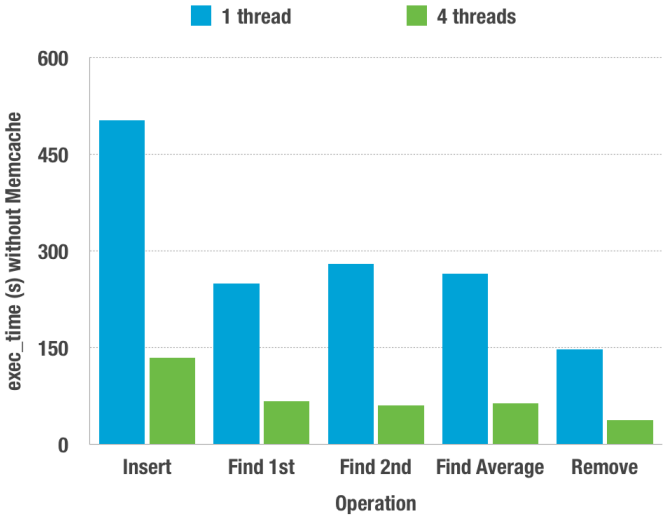
Operation Benchmark with Memcache (unit: s)

	Insert	Find 1st	Find 2nd	Find Average	Remove
1	377.832581	222.157964	178.005524	200.081744	81.452727
4	100.934682	52.565265	50.59674	51.5810025	24.007882

Operation Benchmark without Memcache (unit: s)

Thread	Insert	Find 1st	Find 2nd	Find Average	Remove
1	502.990164	250.18921	280.183976	265.186593	147.622272
4	134.216562	67.21813	60.929535	64.0738325	37.498019





As you can see in the figures above, 4 threads takes less time than 1 thread in each operation. It is because that each thread can deal with operation parallel, it takes less execute time.

For each condition of thread, operation Remove takes less time to execute. Remove is handle inside the server, it doesn't need much communication between client and server.

With the same amount of threads, each operation with Memcache execute much faster than the operation without Memcache. As we known, Memcache is fast, and the files which are smaller than 100KB in our dataset is stored in Memcache. So the operations with Memcache have a better performance.

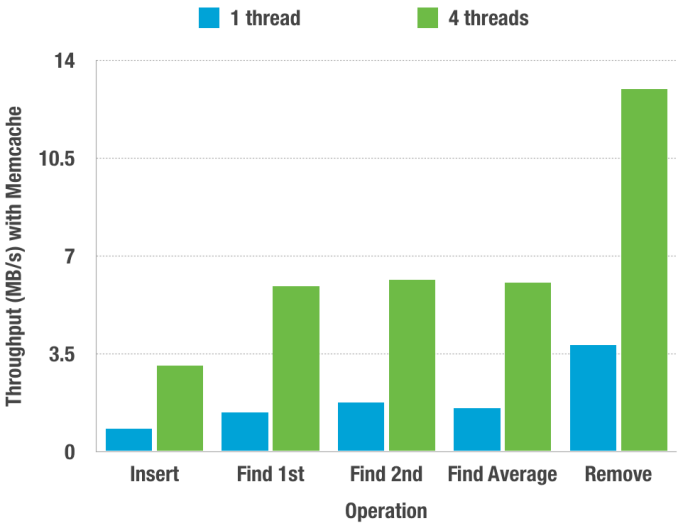
2. Overall Throughput

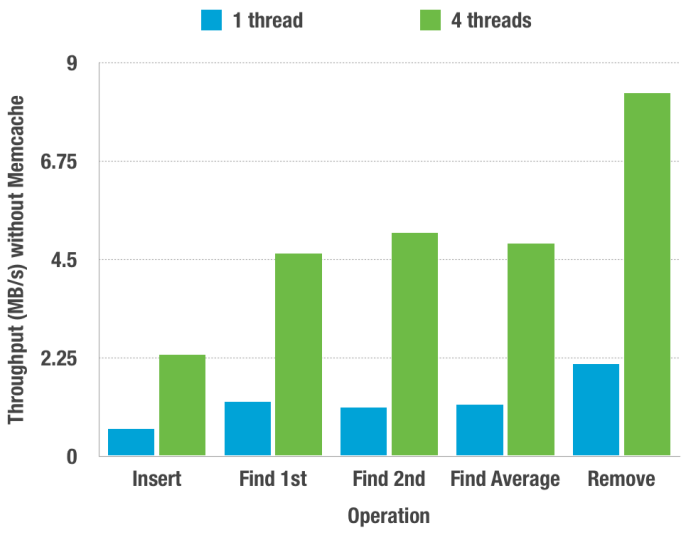
Operation Overall Throughput with Memcache (unit: MB/s)

	Insert	Find 1st	Find 2nd	Find Average	Remove
1	0.823115	1.399904	1.747136	1.554364	3.818165
4	3.081201	5.916454	6.146641	6.029351	12.954079

Operation Overall throughput without Memcache (unit: MB/s)

	Insert	Find 1st	Find 2nd	Find Average	Remove
1	0.618302	1.243059	1.109984	1.172759	2.106728
4	2.317151	4.626728	5.104257	4.853775	8.293771





Throughput = Total Size of Dataset / Execute Time

In our result, throughput is calculated by the formula showed above.
Based on the figures above, 4 threads have much bigger throughput than 1 thread in each operation. It is because that each thread can deal with operation parallel, it can handle much more throughput.
For each condition of thread, operation Remove has the biggest throughput. It is because that operation Remove is handle inside the server, it doesn't need much communication between client and server.
With the same amount of threads, each operation with Memcache has a bigger throughput than the operation without Memcache. As we known, Memcache is fast, and the files which are smaller than 100KB in our dataset is stored in Memcache. And operations with Memcache execute much faster than the operations without Memcache. Because they have the same size of the data size, less execute time means bigger throughput.

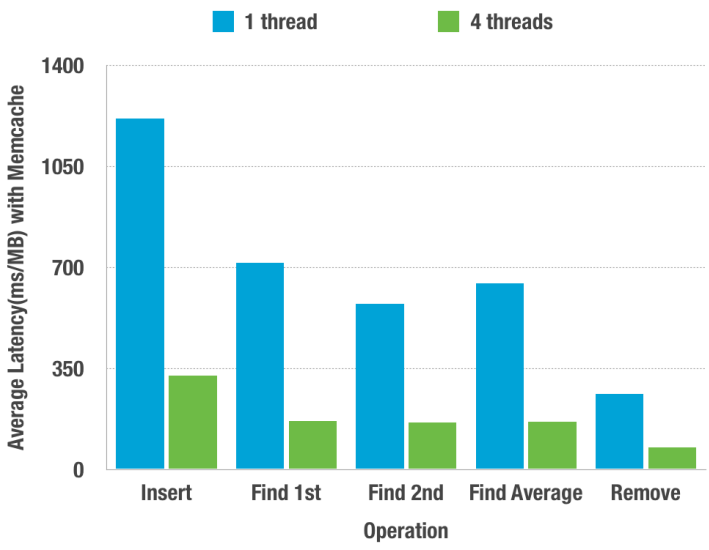
3. Average File Access Latency

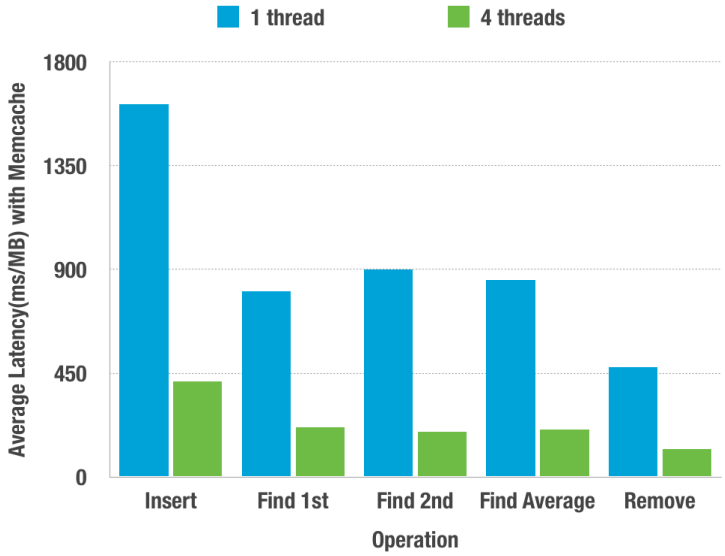
Operation Average Latency with Memcache (unit: ms/MB)

	Insert	Find 1st	Find 2nd	Find Average	Remove
1	1214.89707	714.3347	572.36529	643.34995	261.90592
4	324.54877	169.02016	162.69048	165.85533	77.19576

Operation Average Latency without Memcache (unit: ms/MB)

	Insert	Find 1st	Find 2nd	Find Average	Remove
1	1617.33263	804.46704	900.91389	852.69011	474.66972
4	413.56445	216.13546	195.9149	206.02521	120.57242

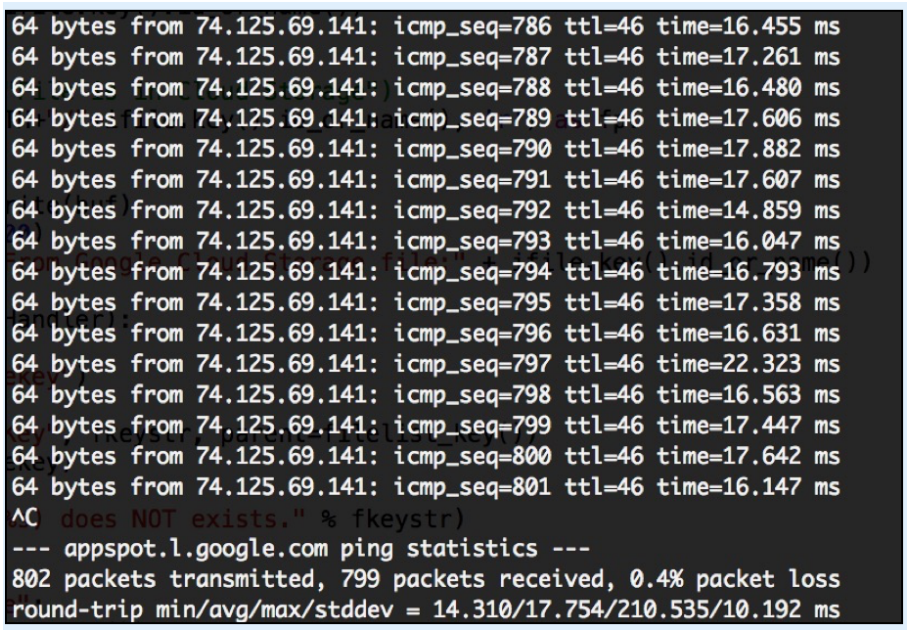




Latency = Execute Time / Total Size of Dataset

In our result, latency is calculated by the formula showed above.
Based on the figures above, 4 threads have smaller latency than 1 thread in each operation. It is because that each thread can deal with operation parallel, it can handle much more throughput, which also mean it has a smaller latency.
For each condition of thread, operation Remove has the biggest throughput. It is because that operation Remove is handle inside the server, it doesn't need much communication between client and server. For example, remove operation doesn't have the same situation likes the insert operation that when you upload every single file, client has to submit a request to server.
With the same amount of threads, as we mention above, each operation with Memcache has a bigger throughput than the operation without Memcache, which also means Memcache has a small latency.

4. Ping Utility



The minimum latency is 14.310ms per 64 bytes. The reason why the smallest latency(77.19576 ms/MB) we got is smaller than the minimum latency which is achieved by ping command between client local machine and google cloud storage is that we have a bunch of files in Memcache. As we all known, Memcache is much faster than google cloud storage. So our latency is much smaller than the minimum latency we got from the ping command.

Comparison to Amazon S3

Amazon S3

Data Storage: 311TB $10187.57/month

Data Transfer:
- Inter-Region Data Transfer Out: 622TB $13930.26/month
- Data Transfer Out: 622TB $39085.15/month
- Data Transfer In: free

Google App Engine

Data Storage: 311TB $8280.064/month

Data Transfer:

Out: 622TB $50954.24/month

In: free

Based on our analysis, we can safely draw the conclusion:

1. For data storage, Google App Engine is more cost effective
2. For data transfer out, Amazon S3 is more cost effective
3. For data transfer in, Amazon S3 and Google App Engine has the same cost effective.

Conclusion

Through one week's work, we get used to use the Google App Engine. We make our own small apps on Google Cloud Storage by python. By comparing the cost between Amazon S3 and Google App Engine, we knew which is more cost effective.