Design

Python: 2.7.4

Google App Engine: 1.9.15

Part a: overall program design

For the overall design, by using the Google Cloud storage client library, we implement a distributed storage system on the top of the Google App Engine.

Because it should be a key/value store, so that we import the db model from Google App engine to be the structures. We design every file key should have properties such as key.location and key.filekey, and it should has key.size to record the size of the file.

When insert the files, we use the method get_uploads('file) the know the information, if the file size is less than <=100KB, then the file should be store into the both the memcache and the GCS. In this case, we implement the memecache by imported the memcache from google.appengine.api , using like memcache.add(mykey, info) to add the value and memcache.delete(filekey.key().id_or_name()) to delete the memcahche. Also when file size >100KB, the file will be only stored into the GCS, like this : write_path = files.gs.create(BUCKET_PATH+"/"+filekey.key().id_or_name(), mime_type='text/plain', acl='public-read').

For function check(key), to implement it, we just need to check the filekey exist in the db or not, like this: filekeys.filter('__key__ =', db.Key.from_path("FileKey", fkeystr, parent=filelist_key())), by using the filter.

For function Find(key), we need to retrieve the content from the file, so that if the file is also stored in memcache, we can get the content like this: info = memcache.get(ifile.key().id_or_name()), if the file is in GCS, we can get the file like this: with files.open(BUCKET_PATH+"/"+ifile.key().id_or_name(), 'r') as fp.

For function Remove (key) , we get the key from the parameter, if the file exist in the memcache, remove like this: memcache.delete(f.key().id_or_name()), if the file exist in the GCS, remove like this: files.delete(BUCKET_PATH+"/"+f.key().id_or_name()).

For function listing(), we get all the file keys from the db and retrieve the file names like this: for filekey in filekeys: self.response.out.write(filekey.key().id_or_name())

Part b: design tradeoffs

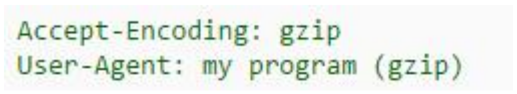
As for the design tradeoffs, in order to get better performance, I have designed some logics to improve it. For example, when call the *Find(key) function, it will first check the file whether exist in the memcache or not. If the file exist in the memcache, then access the file from the memcache. If the file not exist in the memcahce, then check it whether exist in the GCS. I set this logic because* accessing small files from memory should be significantly faster than accessing such small files from persistent storage.

What’s more, the system should handle the case when the file size bigger than the Quotas. In this case, in other words, it means no more capacity. When happen this case, the system should stop upload the file and return a Boolean value.

Part c: possible improvements and extensions

As for the possible improvement, if we want to improve the performance, I think we can do the following two things:
(1): An easy and convenient way to reduce the bandwidth needed for each request is to enable gzip compression. Although this requires additional CPU time to uncompress the results, the trade-off with network costs usually makes it very worthwhile.

In order to receive a gzip-encoded response you must do two things: Set an Accept-Encoding header, and modify your user agent to contain the string gzip. Here is an example of properly formed HTTP headers for enabling gzip compression:



```
Accept-Encoding: gzip
User-Agent: my program (gzip)
```

(2) Another way to improve the performance of your API calls is by requesting only the portion of the data that you're interested in. This lets your application avoid transferring, parsing, and storing unneeded fields, so it can use resources including network, CPU, and memory more efficiently.

By default, the server sends back the full representation of a resource after processing requests. For better performance, you can ask the server to send only the fields you really need and get a *partial response* instead. To request a partial response, use the fields request parameter to specify the fields you want returned. You can use this parameter with any request that returns response data.

Request for a partial response: The following request for this same resource uses the `fields` parameter to significantly reduce the amount of data returned.

```
https://www.googleapis.com/demo/v1?key=YOUR-API-KEY&fields=kind,items(title,characteristics/length)
```

Partial response: In response to the request above, the server sends back a response that contains only the kind information along with a pared-down items array that includes only HTML title and length characteristic information in each item.

```
200 OK

{
  "kind": "demo",
  "items": [
  {
    "title": "First title",
    "characteristics": {
      "length": "short"
    }
  },
  {
    "title": "Second title",
    "characteristics": {
      "length": "long"
    }
  },
  ...
  ]
```