IO/Data Technologies – GridKA 2012

Exercises: Andreas-Joachim Peters

Email Andreas.Joachim.Peters@cern.ch

Exercises Data Technologies

3 A Simple Cloud Storage Implementation

Preparation

We are going to write our own Cloud Storage System.

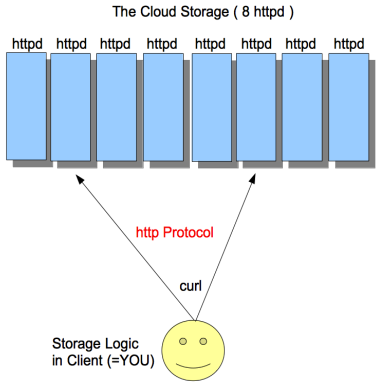
The final goal is to store 256 files located in `/data/dm/cloudstore/` in a distributed storage system and to be able to retrieve, delete and list them!

As storage backend we will use 8 HTTP server which allow you to upload, download and list files. These nodes run a simple Apache **httpd** web server which provides to every client machine an individual storage area to avoid interference between you.

We will implement all the logic on client side in shell functions and we will run without any security mechanisms and error recovery.

The web server URLs to use are:

`http://gks131/`
`http://gks132/`
`http://gks133/`
`http://gks134/`
`http://gks135/`
`http://gks136/`
`http://gks137/`
`http://gks138/`



A basic concept usable for cloud storage is to store files internally not with their logical file name but with their checksum (or hash value) as file name.

A hash with a good uniqueness for this purpose is the SHA1 hash. You calculate the SHA1 hash of a file using the '`sha1sum`' command. The output is the hexadecimal representation of the 160bit hash value.

Inside the storage system one can use the hexadecimal hash value as file name. To get back the original file names you have to maintain a catalog to map your real file name to the SHA1 value of a stored file. In any hash implementation you have to deal with hash collisions.

We will **simplify** our system avoiding a catalog by storing files not with the SHA1 value of the file contents but using the SHA1 value of the file name itself. We don't deal with (very improbable) hash collisions.

We will use four basic commands provided by the `cloud.sh` library to implement our Cloud storage system. These commands use curl and lynx to issue HTTP requests:

- **upload**
- **download**
- **delete**
- **list**

As preparation copy the cloud shell library to your home directory:

```
cp /data/dm/cloudlibrary/cloud.sh $HOME
```

'source' the library to get all functions available in your current shell:

```
source $HOME/cloud.sh
```

Reminder: whenever you edit `cloud.sh` you have to source it again to get the updated functions in your current shell.

3.1 Simple Hash Table and File Distribution in a Cloud-like Storage system

To make our Cloud system distributed and not centralized we want to determine automatically the location of files. This allows every client to find a file without asking any database or catalog.

For this purpose we use a DHT (distributed hash table) and assign hash space to each HTTP server.

Imagine we have a hash function which provides hash values from 1 to 8, we can distribute the hash space equally over our 8 machines like this:

Hash Key	Hash Value	Web Server
0,1	1	gks131
2,3,	2	gks132
4,5	3	gks133
6,7	4	gks134
8,9	5	gks135
10,11	6	gks136
12,13	7	gks137
14,15	8	gks138

For our storage we use the first letter of the SHA1 filename hash as `hashkey`. This is a hexadecimal number (0-f). Therefore we need to map the 16 possible values to our eight machines (`hashvalue 1-8`)
In the exercises we call the 80 bytes hexadecimal string `hash`.

For simplification the library contains a function, which maps hex values 0-f to decimal values 1-8 as shown in the key-value hashtable above.

```
function h8d {...}
```

Preparation Exercises

3.1.1 Compute the SHA1 value of the string `/etc/passwd` ***using the sha1string function provided by the library.*** *Try to understand how the command is implemented.*

3.1.2 What is the formula used to convert 4-bit hex values to the hash table [1,2,3,4,5,6,7,8] ?
Read the code or try it out!

3.1.3 Assign the sha1 value of the string `/etc/passwd` ***to a variable named*** **`hash`**.

HINT: you can assign the output of a command in the shell like this

```
value=`<cmd>`
```

3.1.4 Extract the first letter of the variable 'hash' from 3.1.3 and store it in a variable `hashkey` .
HINT: you can output a string variable like (substitute the <> fields with real numbers!)

```
echo ${hash:<offset>:<length>}
```

3.1.5 Get the `hashvalue` ***using the h8d function for the variable*** `hashkey` ***from 3.1 4?***

3.1.6 Use the `upload` ***function to upload the file*** `/etc/passwd` ***according to the computed*** `hashvalue` ***and use the hexadecimal SHA1 filename*** `hash` ***as target name for the upload. To which server name is your file uploaded?***

3.1.7 Use the `download` ***function to download the previous uploaded file.***

3.1.8 Use the **`list`** ***function to list all files on the server where you uploaded the file in 3.1.6.***
Remark: you can also see the server contents using a browser!

3.1.9 Use the `delete` ***function to remove the uploaded file from 3.1.6.***

In the following we are going to implement now high-level functions to store, retrieve, list and delete files in our Cloud Storage system.

`cloud_upload`

`cloud_download`

`cloud_rm`

`cloud_ls`

These functions are already defined in the cloud library. On top of each function there is an exact description what they are supposed to do and which arguments to provide. Your task is mainly to prepare the arguments to call the corresponding low-level functions (`upload`,`download`,`rm`)

3.1.10 **Implement the `cloud_upload` function which uploads a local file to the DHT computed location.**
Using this function distribute the 256 files from `/data/dm/cloudstore/` to the eight web server using the SHA1 value of the full file path.

HINT: you get the file list easily using
```
find /data/dm/cloudstore/ -type f
```

HINT: you can loop over a list in bash like:
```
for name in `<cmd creating list>`; do echo $name; done
```

3***.1.11*** ***Implement the `cloud_download` function which downloads the DHT distributed file to the local file system. As a test download the file `/data/dm/cloudstore/etc/hosts` to `/tmp/hosts` and compare with the original file.***

3.1.12 ***Now count the number of files per web server. How is the balancing of the distribution?***

3.1.13 ***Implement the function `cloud_rm` and remove the file*** `/data/dm/cloudstore/etc/hosts`

3***.1.14*** ***Add 'bucket' functionality to the the `cloud_upload` function e.g. every file name you upload is added into the file*** `~/.bashcloud.bucket`

HINT: you should avoid to add the same file name twice if you repeat an upload. Useful commands are
```
cat <infile> |sort|uniq > <outfile>
```
to filter names which appear more than once.
`<infile>` and `<outfile>` have to be different.
You can rename the result files using 'mv' or '`rename`'.
Repeat the upload of 256 files from 3.1.10 twice and check that you have each file name only once.

3.1.15 ***Implement the `cloud_ls` function just listing your bucket file!***

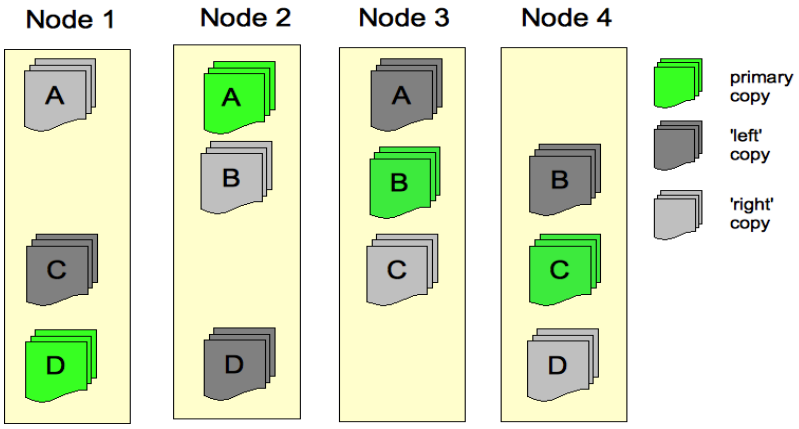
3.1.16 ***Update the `cloud_rm` function to remove deleted files from the bucket.***
HINT: you can remove a name using
```
cat <infile> | grep -v <name> > <outfile>'  # infile>!
=<outfile>!
```
Remove all 256 files from your storage!

Remark: Instead of having the bucket locally one should also upload the bucket to the storage system and store it with the hash of the bucket name e.g. sha1('/') or sha1('mybucket') as storage file name. Otherwise the bucket contents is not shared between clients!

3.2 Introducing redundancy in the Cloud Storage System

Our storage system has a problem. If we shutdown the HTTP server on one node you possibly cannot read anymore about 1/8 of your files. We need to introduce redundancy in our storage system.

Therefore you need to extend `cloud_upload` to store additionally another replica of each file on the 'hash neighbour'. E.g. if our file has a hash value of 3, we store another copy on the machines defined as <hash>-1 and <hash>+1 = 2 and 4. Write down a hash table indicating the three locations for files with hash values 1-8.



Hash Value	Primary Server	Left Server	Right Server
1	gks131		
2	gks132		
3	gks133		
4	gks134		
5	gks135		
6	gks136		
7	gks137		
8	gks138		

3.2.1 What are the hash neighbours of hash value 1 ?

3.2.2 Integrate the file distribution of replicas updating your `cloud_upload` *function according to the extended replica table. You can use the library function* `leftvalue` *and* `rightvalue` *to get the neighbor values for a given hashvalue.* (Don't forget to add the hash range to the calls leftvalue,rightvalue calls – see documentation)
Rerun the upload of the 256 files.

3.3 Testing redundancy in the Cloud Storage System

3.3.1 Delete the primary hash (file copy) location of the file `/data/dm/cloudstore/etc/hosts`*.*

3.3.2 Implement a more robust download function `cloud_download` *which can fall back to alternative replica's.*
HINT: the download function returns !0 if the download fails and you can just chain commands like
<download1> || <download2> || <download3>

3.3.3 Is it better to prefer always the primary location or choose a random replica in a storage system? What would be the best algorithm to choose a replica in your opinion?

3.4 Resizing of Cloud Storage System

The last 4 nodes in your hash table are put into maintenance.

3.4.1 Rewrite the hash table with the remaining four nodes and implement the function `h4d` *representing the new hash function.*

3.4.2 We need a tool to redistribute files according to their changed hash table. Write down the logic of the program to do the redistribution. Which percentage of files has to be moved?

3.5 Block based Storage Systems (advanced)

To get a uniform space distribution filesystems like HDFS split files into 16k blocks. Write a tool which allows you to upload and downloads files which are split into equal blocks.

The layout of blocks (which blocks belong to which file) has to be stored in a seperate file `<filename>.layout` and the layout file should also be stored in the storage system.

Now implement the functions:

`cloud_block_upload`

`cloud_block_download`

HINT: You can use the `split` command to create blocks

```
split -d -b 16384 <filename> /tmp/cloud_block.
```

then upload the pieces with SHA1 hash <filename>.01 ... <filename>.<N>

add the name of each piece to a file `/tmp/cloud_block.layout` and upload this file with SHA1 hash <filename>.layout

```
# for name in `ls /tmp/cloud_block.*`; suffix=`echo $name | cut -d '.' -f2`; do .... ; done
```

rm /tmp/cloud_block.*

HINT: You can use the `cat` command to reassemble blocks

download the layout file <filename>.layout

download all pieces in the layout file

```
# for name in `cat /tmp/cloud_block.layout`;  do ... ; done
```

You can use the file `/data/dm/cloudstore/etc/openldap/certs/cert8.db` to test (it has 64k and should be split into 4 pieces).