Programming Assignment #1

COEN241 Cloud Computing
Department of Computer Engineering
Santa Clara University

Dr. Ming-Hwa Wang — Winter Quarter 2021
Phone: (408) 805-4175 — Email address: m1wang@scu.edu
Course website: — http://www.cse.scu.edu/~mwang2/cloud/
Office Hours: — Tuesday & Thursday 9:00-9:30pm

Due date: midnight January 24, 2021

Simplified OpenStack Object Storage, Part I (200 points)

Please implement a simplified OpenStack object storage using C, C++, Java, or Go. An OpenStack object storage (e.g., the Swift object storage project) stores and retrieves objects (i.e., files) on a distributed file system with scale and provides software-defined-storage (SDS), which is scalable, durable, available, manageable, flexible, adaptable, open, economic, and eventually consistent.

There are the following processes need to be implemented:

a) proxy process communicates with external clients, and accepts download, list, upload, (stat,) and delete operations
b) routing process finds storage location using consistent hashing
c) storage process stores objects and metadata
d) consistency process ensures data integrity and availability by finding failed drives or corrupted data, and replicating objects to a preset number of copies, by default, 2

The first part of this implementation includes a) and b) above, and the second part contains c) and d).

The proxy process is implemented using client-server network programming, to simplify your work, both server and client run on the Linux machines. You should run the server process first using partition power and list of available disks as command-line arguments, the server will find an available port automatically and display the port number, then the client can run using either the server's IP address or the server's machine name and the port number displayed by the server as command-line argument. E.g.,

```
$ server 16 129.210.16.80 129.210.16.81 129.210.16.83 129.210.16.86
Port number = 9999
$ client 129.210.16.80 9999
```

The routing process assumes any object consists of a unique user name and non-unique object name in the form of "user/object" and hashed by md5sum – <<< "<string>" or echo "<string>". A fixed partition power is given and therefore total number of partition is $2^{\text{partition power}}$. The algorithm always store object to the least-used drive. A device lookup table contains number-of-replica rows and number-of-partition columns, each entry is the drive to store the given object. To simply your work, no authentication/authorization is needed.

The commands should be supported including the following with the output expected:

- download <user/object> - display the context of <user/object>
- list <user> - display the <user>'s objects/files in "ls –lrt" format
- upload <user/object> - display which disks the <user/object> is saved
- delete <user/object> - display a confirmation or error message
- add <disk> - display new partitions with all <user/object> within
- remove <disk>- display where old partitions went to

The md5 hashing takes the <user/object> input as a string, and generates a 128-bit hashed value in hex. You divide the hashed value into $2^{\text{partition power}}$ partitions and assign partitions roughly equally to all disks you have. Assume that each machine only has one disk drive, so we use the machine name for the disk. We also assume all objects are text files.

Student Name:

ID:

Score:
Correctness and boundary condition (55%):
Compiling without warning/error (5%):
Error Handling (5%):
Automatic available port finding (2.5%):
Support both host name and IP address (2.5%):
Display output on both server and client windows (5%):
Modular design, file/directory organizing, showing input, documentation, coding standards, sympathy/typing point with README (20%):
Automation for client and server side (5%):

Subtotal:
Late penalty (20% per day):
Special service penalty (5%):

Total score: