SaaS between Peers within Cloud Instances

Abhishek A Vasanth Kumar
Anmol Vijaywargiya
Jagdish Bhanushali
Priyansh Jain
Yash Bansal
Computer Science and Engineering
Santa Clara University

Abhishek Gupta
Computer Science and Engineering
Santa Clara University
Santa Clara, California
agupta3@scu.edu

Abstract — This report presents a methodology to share cloud's storage as a service between instances of cloud users. It addresses a few difficulties in the present cloud storage model, those are, the need to buy extra storage for some small work if you have used up all of your allotted storage and, the absence of any platform/infrastructure which facilitates in sharing of storage within cloud vendors and different cloud vendors. Fundamentally, the procedure consists of registering users' information on a cloud centralised machine. This is followed by the uploading of files by the users through the centralised platform to a central server. The server will store the uploaded files on the cloud machines and manage the file storage according to the space available. The platform will be a community cloud catalysing the users in the community to share storage resource thereby helping each other to solve the hurdle of limited space and also utilizing the resources provided by cloud in a better way.

I. Introduction

Cloud computing is an internet-based computing that provides shared computing resources for processing and storing data to computers and other devices on demand. It is a model for empowering far-reaching, on-demand access to a shared pool of configurable computing resources that can be rapidly provisioned and deployed with least management effort. In plain terms, enterprises and users can choose and acquire a broad collection of data processing and data storage solutions, quickly and easily with minimum effort. According to National Institute of Standards and Technology, Cloud Computing essentially has five fundamental characteristics – on-demand self-service, broad network access, resource pooling, rapid elasticity and measure service [1].

Cloud computing vendors/providers offer various service models of cloud, of which the three standard models are Infrastructure as a Service (Iaas), Platform as a Service (Paas) and Software as a Service (Saas). Since they offer abstraction, the service models portray as layers in a stack. Software as a Service (Saas) – the user uses the cloud computing vendor/provider's application on a cloud infrastructure. The application can be accessed from various devices through a web browser or a program interface. Platform as Service (Paas) offers the users, the ability to deploy custom-created or acquired applications built using any programming languages/environments onto cloud infrastructure without the need to manage or control the underlying cloud infrastructure. Infrastructure as a Service (Iaas) – It is a standardized and automated offering where computing, storage and networking resources and capabilities are offered by vendors/providers to customers/users on-demand. They can self-provision this infrastructure using a web-based graphical user interface that operates as a management console for the environment. Apart from the standard models, cloud computing offers various services such as Mobile "backend" as a service (MBaas), Serverless Computing, Database as a Service (DBaas), Backup as a Service (Baas) and Storage as a Service (Saas).

Storage as Service is a business model in which a cloud computing enterprise rents space in their storage infrastructure to a smaller enterprise or user/customer as a service. Storage as a service model is becoming more and more popular among not only Internet users but among users of scientific and other computing infrastructures.

Amazon Elastic Compute Cloud (EC2) forms a fundamental part of Amazon's cloud computing platform, Amazon Web Services (AWS). It allows users to rent virtual computers to run their own computer applications. EC2 encourages scalable deployment of applications by providing web service through which a user can boot an Amazon Machine Image (AMI) to configure a virtual machine, "instance", containing any software desired. A user can create, launch, and terminate instances as needed. According to a study by Cloudyn (Cloud Cost Management Company), the utilization of resources over 400,000 monitored instances was only 17% [5]. Furthermore, the company found the common issue was allocating larger than required instances. Larger the instance, the worse the utilization. Currently, the basic free plan of Amazon Web Services EC2 instance (t2.micro) comes with 8GB of storage space.

There are two problems associated with this allocation. Firstly, such a large storage space is of no use to users who create an EC2 instance for small tasks as the remaining space goes unused. Secondly, users, a student for example, creates a free AWS EC2 instance to implement some project work. What will happen if the project work spans out of the allocated storage space (insufficient resources) and needs more resources? The user will have to spend money and buy resources. AWS charges around $1GB/month for extra storage services. In this paper, we propose to implement a platform, which overcomes these problems. A platform on which users can share their unused storage space and can request and obtain storage space from their peers, free of cost; which facilitates effective utilization of storage space and minimize the amount of unused storage space.

Features proposed for implementing Storage as a Service within cloud instances include:

- A graphical user interface based platform to allow users to store and share files on cloud.

Source Code: https://github.com/jagdishbhanushali/CloudFi

- Registration module for users (also contributors) to register their AWS EC2 instance to contribute storage space.
- A centralized management console (orchestrator) to handle file storage among the available EC2 nodes.

Currently, the basic plan of Amazon Web Services comes with 8GB of storage space and amazon charges around $1 GB/month for its extra storage services if needed. This cost can be reduced to zero if we use the free storage available with the other peers of the community.

II. PROBLEM ANALYSIS

Space requirement of machines are unpredictable and keeps on fluctuating. Sometimes there is a necessity of more space and sometimes they are laying waste on the cloud. It's not always feasible to buy extra storage space for temporary work.

Transferring files between user's local system and cloud instance is also tedious work. In current system user has to download ftp software in their system and transfer files. It is not possible to always download and install ftp software; on public computer user might not be willing to share their private key and do not want to take a risk by installing software, which saves personal information.

III. IMPLEMENTATION

The basic idea of the implementation is there is a platform, where the users can register themselves, register their instances and can upload or download the files. First, these files are transferred to the central platform or the orchestrator, which will decide to which available instance the file has to be transferred and then manage the entry of the respective node to the database.

To implement the idea, Amazon Web Services' instances are used. The website is hosted on the instance provided by the Amazon Web Services. To transfer files and to check the status of available free space of the EC2 instance Linux scripts are developed. After registering and logging, the user will have to download this script and run it on his/her instance.

The user can share files with to the platform by uploading files to the website. Once files are received on the server, it starts searching for space where that file is to be accommodated. To accomplish the task of hosting the transferred files, Worst Fit Algorithm is implemented, which is addressed in detail in the later part of the paper.

User's account detail is stored in MySQL database, along with the information of all the available instances that are registered with the service and the information of the host to which files have been moved is maintained; to recover files in case of any errors. MySQL database also stores the information about the file, its owner and the information related to its distribution. When a user will transfer files to the server, the server will make two copies of the files and then transfers them to other user's machine.

The structural diagram of the project is:

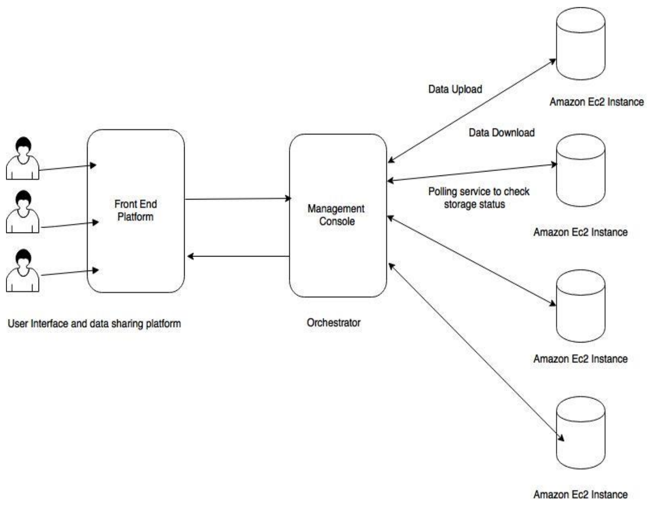


Figure 1 - System Architecture

The following are the screenshots of the platform.

1. Login Screen

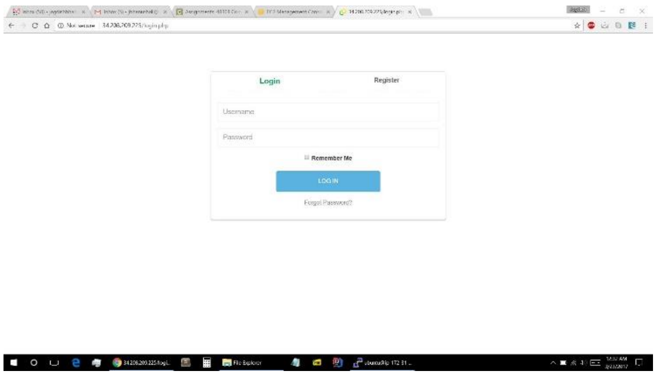


Figure 2

2. Registration Screen

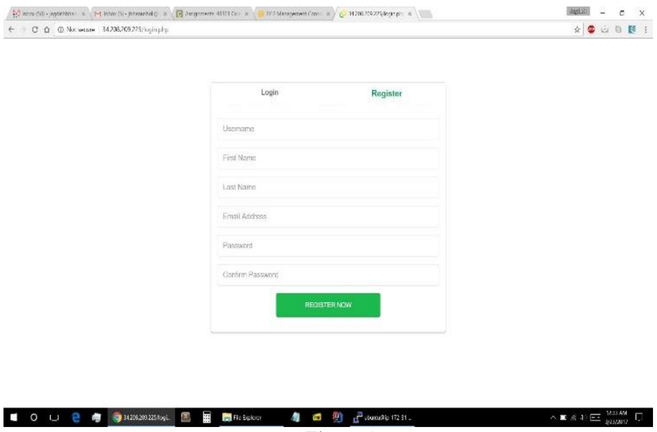
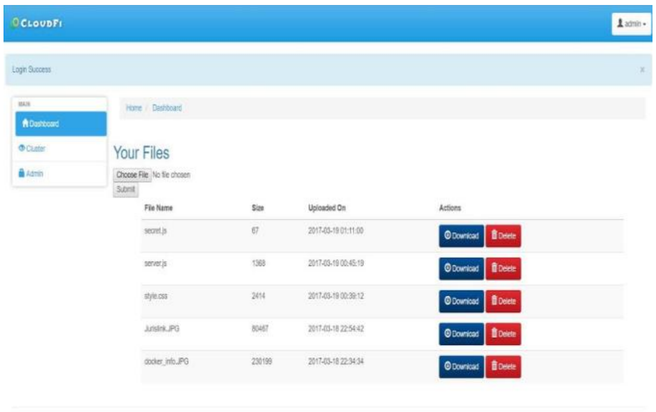


Figure 3

3. User's File Uploaded Page



CloudFi

Login Success

Home / Dashboard

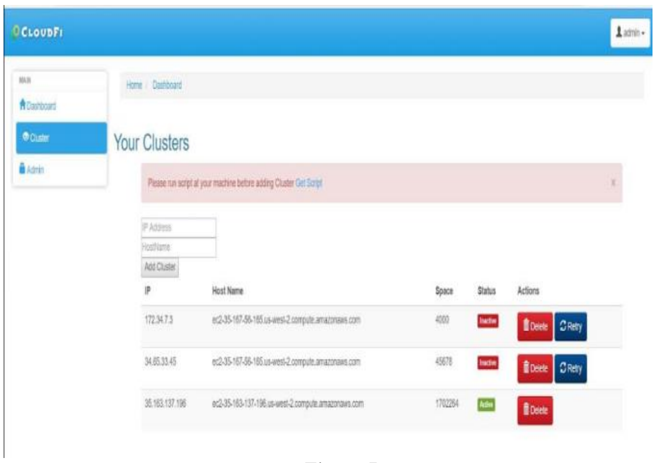
Your Files

Choose File No file chosen
Submit

File Name	Size	Uploaded On	Actions
secret.js	67	2017-03-19 01:11:00	Download Delete
server.js	1368	2017-03-19 00:45:19	Download Delete
style.css	2414	2017-03-19 00:39:12	Download Delete
Jurislink.JPG	80467	2017-03-18 22:54:42	Download Delete
docker_info.JPG	230199	2017-03-18 22:34:34	Download Delete

Figure 4

4. Registered Instances



CloudFi

Home / Dashboard

Your Clusters

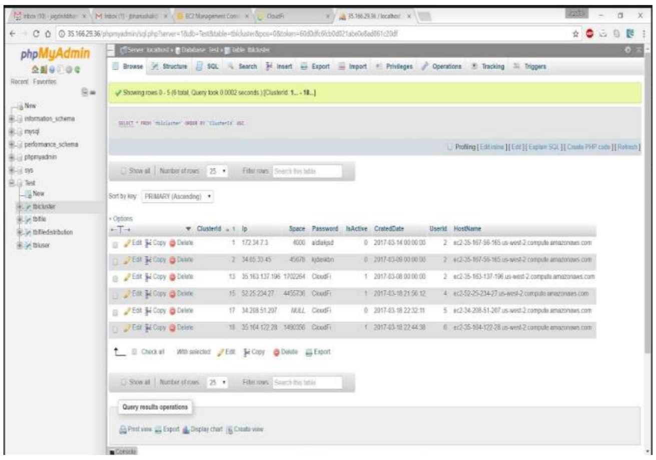
Please run script at your machine before adding Cluster Get Script

IP Address
HostName
Add Cluster

IP	Host Name	Space	Status	Actions
172.34.7.3	ec2-35-167-56-165.us-west-2.compute.amazonaws.com	4000	Inactive	Delete Retry
34.65.33.45	ec2-35-167-56-165.us-west-2.compute.amazonaws.com	45678	Inactive	Delete Retry
35.163.137.196	ec2-35-163-137-196.us-west-2.compute.amazonaws.com	1702264	Active	Delete

Figure 5

5. Internal Database

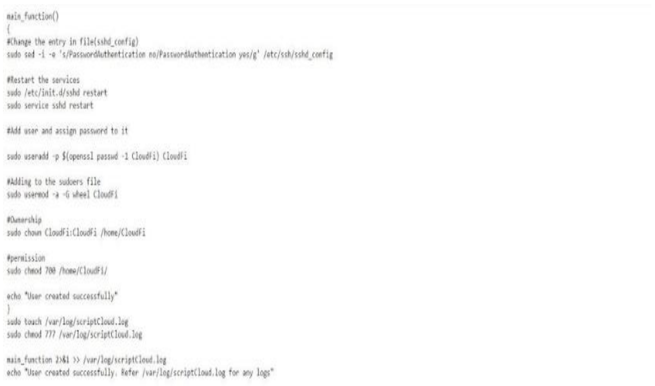


```
SELECT * FROM `tblcluster` ORDER BY `ClusterId` ASC
```

	ClusterId	Ip	Space	Password	IsActive	CratedDate	UserId	HostName
Edit Copy Delete	1	172.34.7.3	4000	aldlakjsd	0	2017-03-14 00:00:00	2	ec2-35-167-56-165.us-west-2.compute.amazonaws.com
Edit Copy Delete	2	34.65.33.45	45678	kjdasldn	0	2017-03-09 00:00:00	2	ec2-35-167-56-165.us-west-2.compute.amazonaws.com
Edit Copy Delete	13	35.163.137.196	1702264	CloudFi	1	2017-03-08 00:00:00	2	ec2-35-163-137-196.us-west-2.compute.amazonaws.com
Edit Copy Delete	15	52.25.234.27	4455736	CloudFi	1	2017-03-18 21:56:12	4	ec2-52-25-234-27.us-west-2.compute.amazonaws.com
Edit Copy Delete	17	34.208.51.207	NULL	CloudFi	0	2017-03-18 22:32:11	5	ec2-34-208-51-207.us-west-2.compute.amazonaws.com
Edit Copy Delete	18	35.164.122.28	1490356	CloudFi	1	2017-03-18 22:44:38	6	ec2-35-164-122-28.us-west-2.compute.amazonaws.com

Figure 6

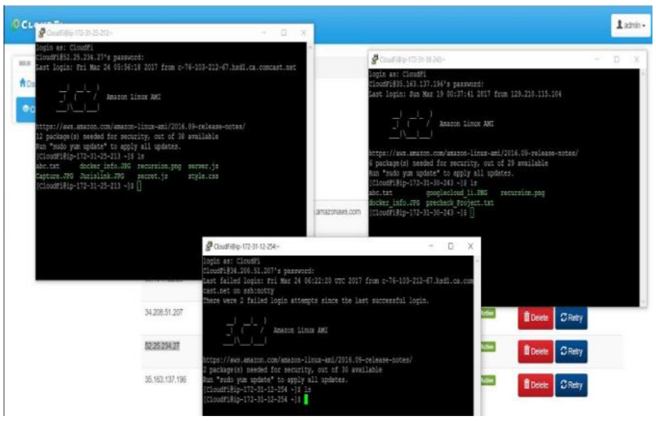
6. Script to create a local user



```
main_function()
{
#Change the entry in file(sshd_config)
sudo sed -i -e 's/PasswordAuthentication no/PasswordAuthentication yes/g' /etc/ssh/sshd_config

#Restart the services
sudo /etc/init.d/sshd restart
sudo service sshd restart

#Add user and assign password to it

sudo useradd -p $(openssl passwd -1 CloudFi) CloudFi

#Adding to the sudoers file
sudo usermod -a -G wheel CloudFi

#Ownership
sudo chown CloudFi:CloudFi /home/CloudFi

#permission
sudo chmod 700 /home/CloudFi/

echo "User created successfully"
}
sudo touch /var/log/scriptCloud.log
sudo chmod 777 /var/log/scriptCloud.log

main_function 2>&1 >> /var/log/scriptCloud.log
echo "User created successfully. Refer /var/log/scriptCloud.log for any logs"
```

Figure 7

7. Before transfer of files



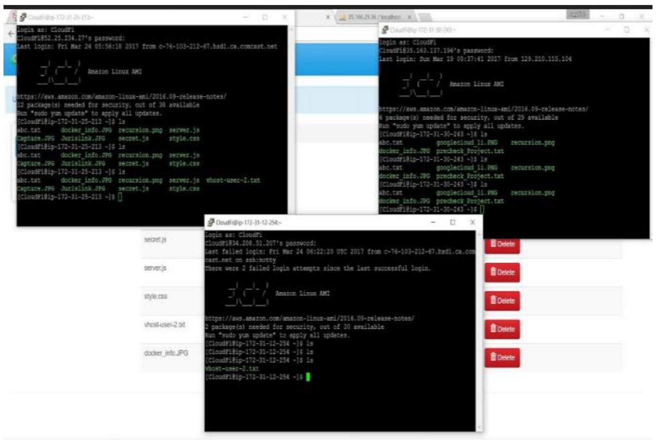
```
login as: CloudFi
CloudFi@52.25.234.27's password:
Last login: Fri Mar 24 05:56:18 2017 from c-76-103-212-67.hsd1.ca.comcast.net

                    Amazon Linux AMI

https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/
12 package(s) needed for security, out of 38 available
Run "sudo yum update" to apply all updates.
[CloudFi@ip-172-31-25-213 ~]$ ls
abc.txt      docker_info.JPG  recursion.png  server.js
Capture.JPG  Jurislink.JPG    secret.js      style.css
[CloudFi@ip-172-31-25-213 ~]$
```

```
login as: CloudFi
CloudFi@35.163.137.196's password:
Last login: Sun Mar 19 00:37:41 2017 from 129.210.115.104

                    Amazon Linux AMI

https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/
6 package(s) needed for security, out of 29 available
Run "sudo yum update" to apply all updates.
[CloudFi@ip-172-31-30-243 ~]$ ls
abc.txt          googlecloud_11.PNG   recursion.png
docker_info.JPG  precheck_Project.txt
[CloudFi@ip-172-31-30-243 ~]$
```

```
login as: CloudFi
CloudFi@34.208.51.207's password:
Last failed login: Fri Mar 24 06:22:20 UTC 2017 from c-76-103-212-67.hsd1.ca.comcast.net on ssh:notty
There were 2 failed login attempts since the last successful login.

                    Amazon Linux AMI

https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/
2 package(s) needed for security, out of 30 available
Run "sudo yum update" to apply all updates.
[CloudFi@ip-172-31-12-254 ~]$ ls
[CloudFi@ip-172-31-12-254 ~]$
```

Figure 8

8. Replication and Transfer Confirmation



```
login as: CloudFi
CloudFi@52.25.234.27's password:
Last login: Fri Mar 24 05:56:18 2017 from c-76-103-212-67.hsd1.ca.comcast.net

                    Amazon Linux AMI

https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/
12 package(s) needed for security, out of 38 available
Run "sudo yum update" to apply all updates.
[CloudFi@ip-172-31-25-213 ~]$ ls
abc.txt      docker_info.JPG  recursion.png  server.js
Capture.JPG  Jurislink.JPG    secret.js      style.css
[CloudFi@ip-172-31-25-213 ~]$ ls
abc.txt      docker_info.JPG  recursion.png  server.js
Capture.JPG  Jurislink.JPG    secret.js      style.css
[CloudFi@ip-172-31-25-213 ~]$ ls
abc.txt      docker_info.JPG  recursion.png  server.js  vhost-user-2.txt
Capture.JPG  Jurislink.JPG    secret.js      style.css
[CloudFi@ip-172-31-25-213 ~]$
```

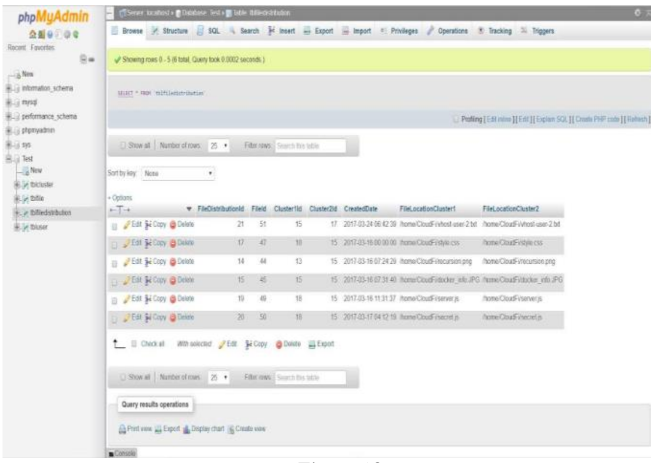
```
login as: CloudFi
CloudFi@35.163.137.196's password:
Last login: Sun Mar 19 00:37:41 2017 from 129.210.115.104

                    Amazon Linux AMI

https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/
6 package(s) needed for security, out of 29 available
Run "sudo yum update" to apply all updates.
[CloudFi@ip-172-31-30-243 ~]$ ls
abc.txt          googlecloud_11.PNG   recursion.png
docker_info.JPG  precheck_Project.txt
[CloudFi@ip-172-31-30-243 ~]$ ls
abc.txt          googlecloud_11.PNG   recursion.png
docker_info.JPG  precheck_Project.txt
[CloudFi@ip-172-31-30-243 ~]$ ls
abc.txt          googlecloud_11.PNG   recursion.png
docker_info.JPG  precheck_Project.txt
[CloudFi@ip-172-31-30-243 ~]$
```

```
login as: CloudFi
CloudFi@34.208.51.207's password:
Last failed login: Fri Mar 24 06:22:20 UTC 2017 from c-76-103-212-67.hsd1.ca.comcast.net on ssh:notty
There were 2 failed login attempts since the last successful login.

                    Amazon Linux AMI

https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/
2 package(s) needed for security, out of 30 available
Run "sudo yum update" to apply all updates.
[CloudFi@ip-172-31-12-254 ~]$ ls
[CloudFi@ip-172-31-12-254 ~]$ ls
[CloudFi@ip-172-31-12-254 ~]$ ls
vhost-user-2.txt
[CloudFi@ip-172-31-12-254 ~]$
```

Figure 9

Source Code: https://github.com/jagdishbhanushali/CloudFi

9. Database Status



Showing rows 0 - 5 (6 total, Query took 0.0002 seconds.)

```
SELECT * FROM `tblfiledistribution`
```

	FileDistributionId	FileId	Cluster1Id	Cluster2Id	CreatedDate	FileLocationCluster1	FileLocationCluster2
Edit Copy Delete	21	51	15	17	2017-03-24 06:42:39	/home/CloudFi/vhost-user-2.txt	/home/CloudFi/vhost-user-2.txt
Edit Copy Delete	17	47	18	15	2017-03-16 00:00:00	/home/CloudFi/style.css	/home/CloudFi/style.css
Edit Copy Delete	14	44	13	15	2017-03-16 07:24:29	/home/CloudFi/recursion.png	/home/CloudFi/recursion.png
Edit Copy Delete	15	45	15	15	2017-03-16 07:31:40	/home/CloudFi/docker_info.JPG	/home/CloudFi/docker_info.JPG
Edit Copy Delete	19	49	18	15	2017-03-16 11:31:37	/home/CloudFi/server.js	/home/CloudFi/server.js
Edit Copy Delete	20	50	18	15	2017-03-17 04:12:19	/home/CloudFi/secret.js	/home/CloudFi/secret.js

Figure 10

The technologies used to build the platform is divided into two categories: Front-end and Backend.

HTML 5, CSS, Bootstrap and JavaScript are the technologies used in designing and developing the front-end of the platform. The goal was to create a simple, self-explanatory yet appealing user interface aimed to provide fluid and effortless interaction between the platform and the user.

The back-end of the platform comprises of PHP7, MySQL, Amazon Web Services and Google Cloud. PHP can be easily deployed and is optimized for making web applications quickly. MySQL database can be scaled on-demand, offers complete work flow control and has high performance. Hence the reason for using the above mentioned technologies to build the platform.

Flow-charts depicting the sequence of actions involved in the working of the platform.

1. Adding a cluster

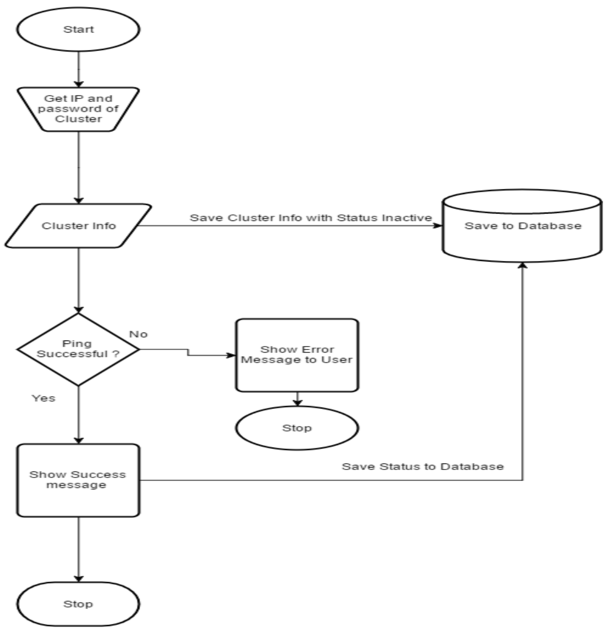


Figure 11

2. Uploading a file

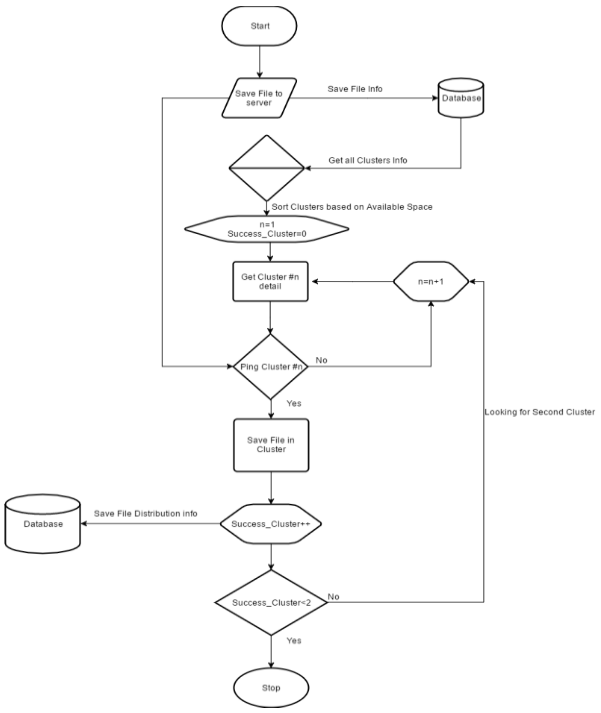


Figure 12

3. Downloading a file

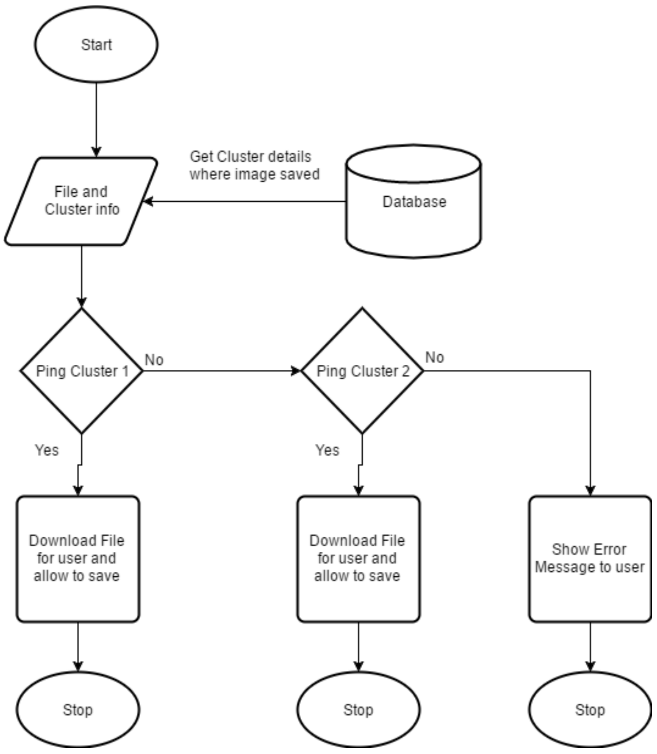

Start
Get Cluster details where image saved
File and Cluster info
Database
Ping Cluster 1
No
Ping Cluster 2
No
Yes
Yes
Download File for user and allow to save
Download File for user and allow to save
Show Error Message to user
Stop
Stop
Stop

Figure 13

IV. ALGORITHMS

After the user transfers the files to the orchestrator, the issue is where this file should be moved and to which registered instance this file should be transferred. This problem is similar to the Memory Allocation handled by the Memory Management Unit in case of Operating System.

There are multiple Algorithms devised to tackle such problems:

First Fit [8]: The first fit approach is to allocate the file in the first free space (bin) large enough to accommodate it. The first fit algorithm provides a fast but often non-optimal solution, involving placing each item into the first bin in which it will fit. It requires Θ (n log n) time, where n is the number of available storage spaces to be packed. The algorithm can be made much more effective by first sorting the list of storage spaces into decreasing order (sometimes known as the first-fit decreasing algorithm), although this still does not guarantee an optimal solution, and for longer lists may increase the running time of the algorithm. It is known, however, that there always exists at least one ordering of items that allows first-fit to produce an optimal solution [9].

Best Fit [7]: The best fit algorithm deals with allocating the smallest free storage space, which meets the requirement of the requesting file. This algorithm first searches the entire list of free storage spaces and considers the smallest space that is adequate. It then tries to find a space, which is close to actual file size needed. Space utilization is much better than first fit as this algorithm searches the smallest free storage space first available but it is very slow.

Next Fit [7]: It is a modified version of First Fit. It begins as first fit to find a free space but when called next time it starts searching from where it left off, not from the beginning.

Worst Fit [9]: Worst Fit approach is to find the largest available space and allocate the file on that space. The algorithm places a file in the largest block of unallocated storage space available.

Every algorithm has some trade-offs and in our case we found that worst fit will serve our purpose.

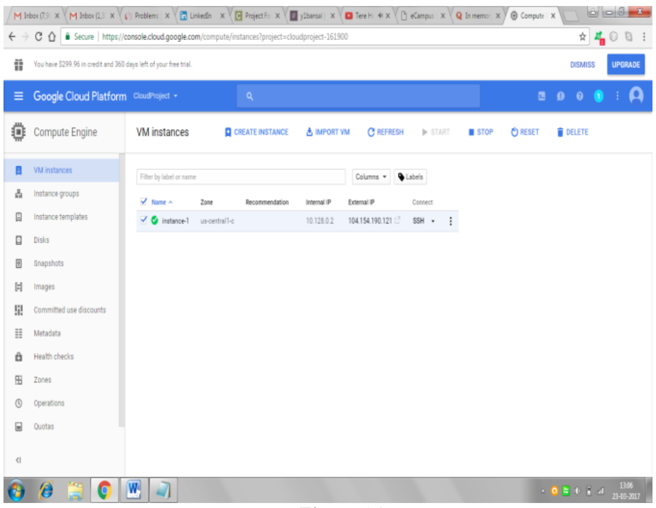
As in worst fit algorithm after storing the file in the maximum space, there will be bigger storage space left as compared to the other algorithms and this space can be used to accommodate other files as well. The idea is that this placement will create the largest hold after the allocations, thus increasing the possibility that compared to best fit; another file can use the remaining space.

V. CROSS PLATFORM

To test if the idea works on cross platforms, an account on Google Cloud is created along with an instance. Registered the instance with the platform and followed the steps as mentioned under the implementation section of this paper.

After completing all the prerequisites, it is found that the transfer of files from Amazon EC2 machine to the Google Machine was a success.

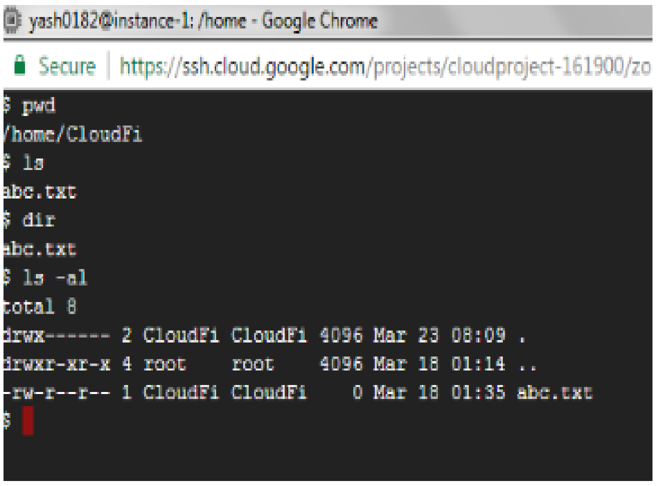
1. Creating an instance



Name	Zone	Recommendation	Internal IP	External IP	Connect
instance-1	us-central1-c		10.128.0.2	104.154.190.121	SSH

Figure 14

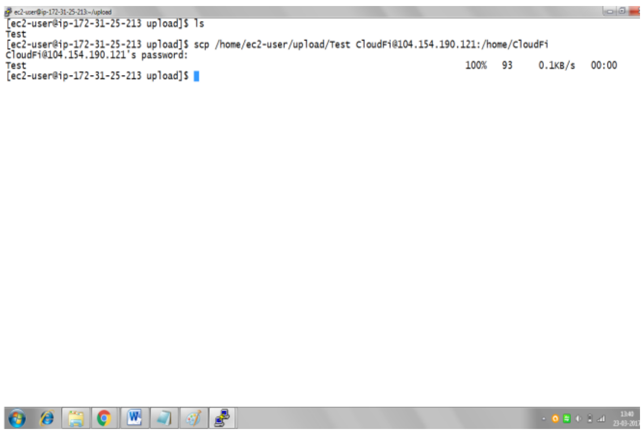
2. Google Machine before transfer process


yash0182@instance-1: /home - Google Chrome
Secure | https://ssh.cloud.google.com/projects/cloudproject-161900/zo

```
$ pwd
/home/CloudFi
$ ls
abc.txt
$ dir
abc.txt
$ ls -al
total 8
drwx------ 2 CloudFi CloudFi 4096 Mar 23 08:09 .
drwxr-xr-x 4 root    root    4096 Mar 18 01:14 ..
-rw-r--r-- 1 CloudFi CloudFi    0 Mar 18 01:35 abc.txt
$
```

Figure 15

3. Transferring file from Amazon EC2 instance

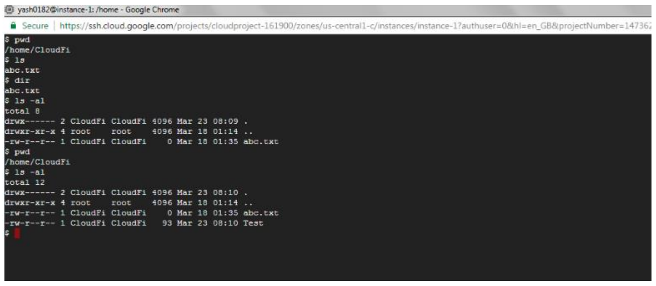


```
[ec2-user@ip-172-31-25-213 upload]$ ls
Test
[ec2-user@ip-172-31-25-213 upload]$ scp /home/ec2-user/upload/Test CloudFi@104.154.190.121:/home/CloudFi
CloudFi@104.154.190.121's password:
Test                                                                    100%   93     0.1KB/s   00:00
[ec2-user@ip-172-31-25-213 upload]$
```

Figure 16

Source Code: https://github.com/jagdishbhanushali/CloudFi

4. Confirmation of successful transfer of file


yash0182@instance-1: /home - Google Chrome

Secure | https://ssh.cloud.google.com/projects/cloudproject-161900/zones/us-central1-c/instances/instance-1?authuser=0&hl=en_GB&projectNumber=147362

```
$ pwd
/home/CloudFi
$ ls
abc.txt
$ dir
abc.txt
$ ls -al
total 8
drwx------ 2 CloudFi CloudFi 4096 Mar 23 08:09 .
drwxr-xr-x 4 root    root    4096 Mar 18 01:14 ..
-rw-r--r-- 1 CloudFi CloudFi    0 Mar 18 01:35 abc.txt
$ pwd
/home/CloudFi
$ ls -al
total 12
drwx------ 2 CloudFi CloudFi 4096 Mar 23 08:10 .
drwxr-xr-x 4 root    root    4096 Mar 18 01:14 ..
-rw-r--r-- 1 CloudFi CloudFi    0 Mar 18 01:35 abc.txt
-rw-r--r-- 1 CloudFi CloudFi   93 Mar 23 08:10 Test
$
```

Figure 17

VI. RELATED WORK, SIMILARITIES, DIFFERENCES AND DISCUSSION

A. Grid Computing

The idea of this implementation is inspired by the concept of Grid computing. Grid computing is the utilization of unused computer resources at the nodes of a network by pooling them together, to solve a common problem. For example, in a grid computing environment deployed over a network where desktop machines comprise the nodes of the system, the virtual grid may divide a single task into several independent task and use the idle CPU cycles on the node desktop machines to perform the computation and send back the results (An example of such a system would be the Berkeley Open Infrastructure for Network Computing).

In our case, we are utilizing Amazon Web Services EC2 t2.micro free tier instances and are pooling them together to build a storage cloud.

B. OpenStack Swift Object Store

The OpenStack Object Store project, known as Swift, offers cloud storage software to store and retrieve huge data with a simple API. It is built for scale and optimized for durability, availability, and concurrency across the entire data set. Swift is ideal for storing unstructured data that can grow without bound. The components of Swift are Proxy Server, Ring, and Storage Nodes.

Swift Components

The Proxy Server is responsible for tying together the rest of the Swift architecture. For each request, it will look up the location of the account, container, or object in the ring and route the request accordingly.

The ring represents a mapping between the names of entities stored on disk and their physical location. There are separate rings for accounts, containers, and one object ring per storage policy. When other components need to perform any operation on an object, container, or account, they need to interact with the appropriate ring to determine its location in the cluster.

Taking idea from above components of swift we have developed a main console similar to Proxy server of swift which is responsible to interact with users and management of their files between storage nodes. The functionalities of the ring like management of storage nodes is done by the main console only which includes addition/deletion/replication of data between the storage nodes.

C. Peer to Peer Cloud Computing

The concept of creating a cloud-computing environment by connecting individual commodity machines over the internet is called Peer to Peer Cloud Computing. In P2P cloud, the primary challenge is to build reliable network of machines since the nodes are distributed across the internet. However, in our approach we are leveraging existing reliable network and machines provided by the cloud vendors.

VII. FUTURE ENHANCEMENTS

- Implementing the platform's management console using OpenStack swift and developing APIs, which allows users to interact with cloud storage similar to the functionalities offered by Swift.
- Encryption of data stored on the users' node to improve security and privacy.
- Division of large files into small files. Storing of small chunks on different storage nodes to manage the storage space effectively.
- Developing an agent that will run on storage nodes and sends the storage statistics and configuration details such as host address or IP address to the management console; this agent will replace a service currently running in a particular interval of time, which checks the storage status of node.

VIII. CHALLENGES FACED

We faced challenges to trigger the Linux processes from the PHP script. The PHP version 5 was not compatible with the SSH2 library used to trigger shell scripts. To overcome this the PHP was upgraded to version 7 but PhpMyAdmin was now not compatible with the combination of PHP7 and HTTPD webserver.

Finally, we used Linux Ubuntu version in which the PhpMyAdmin was compatible with Apache2 and PHP7.

IX. LIMITATIONS

A. Storage Nodes are Unreliable

The proposed scheme utilizes the user's t2.micro EC2 instances. The availability of this nodes are dependent upon the contributors (who is sharing their EC2 instance storage space) will, contributor can shut down his EC2 instance, which will result in node failure in the community cloud.

B. Security and Privacy

The data stored in the storage is exposed to the contributor, as he is super user of the EC2 instance. The storage space available in the cloud is directly proportional to number of contributors.

X. REFERENCES

[1] Cloud Storage Architecture IEEE paper G.Kulkarni,R,wagmare http://ieeexplore.ieee.org/abstract/document/6366026/?reload=true

[2] Cloud Storage as Infrastructure of Cloud Computing,Jiyi Wu,Lingdi ping.http://ieeexplore.ieee.org/abstract/document/5565955/

[3] http://amzn.to/2jojJfa

[4] Peer to Peer Cloud Computing by Ozalp Babaoglu and Moreno Morzella:

http://www.cs.unibo.it/babaoglu/papers/pdf/P2Pclouds.pdf

[5] http://blogs.forrester.com/james_staten/12-05-24-cloud_inefficiency_bad_habits_are_hard_to_break

[6] https://www.multcloud.com/

[7] https://www.quora.com/What-are-the-first-fit-next-fit-and-best-fit-algorithms-for-memory-management

[8] https://en.wikipedia.org/wiki/Bin_packing_problem#CITEREFLewis2009

[9] http://courses.cs.vt.edu/csonline/OS/Lessons/MemoryAllocation

Source Code: https://github.com/jagdishbhanushali/CloudFi