Encrypted Distribution File System

A PROJECT REPORT

Submitted by:-

Aileni Rohan Reddy-19BCE2086

Suyasha-19BCE0321

Srushti Jagtap-19BCE0325

Course Code:

CSE4001

Course Title:

Parallel and Distributed Computing

Under the guidance of

Dr. M. Narayanamoorthi

Associate Professor, SCOPE,

VIT , Vellore.

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

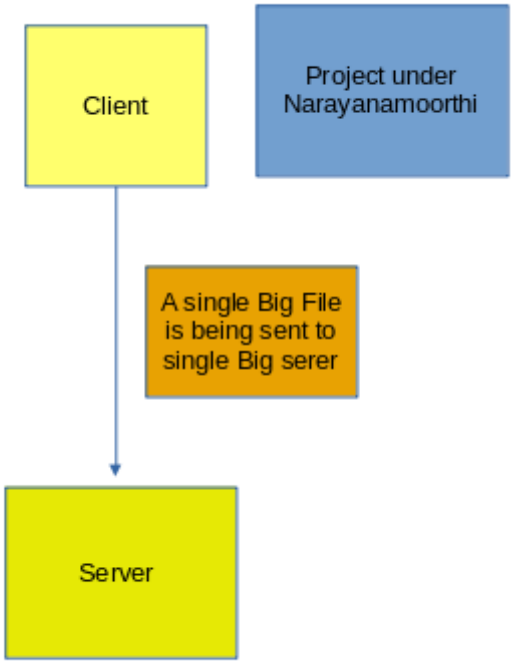
April, 2022

1. Abstract

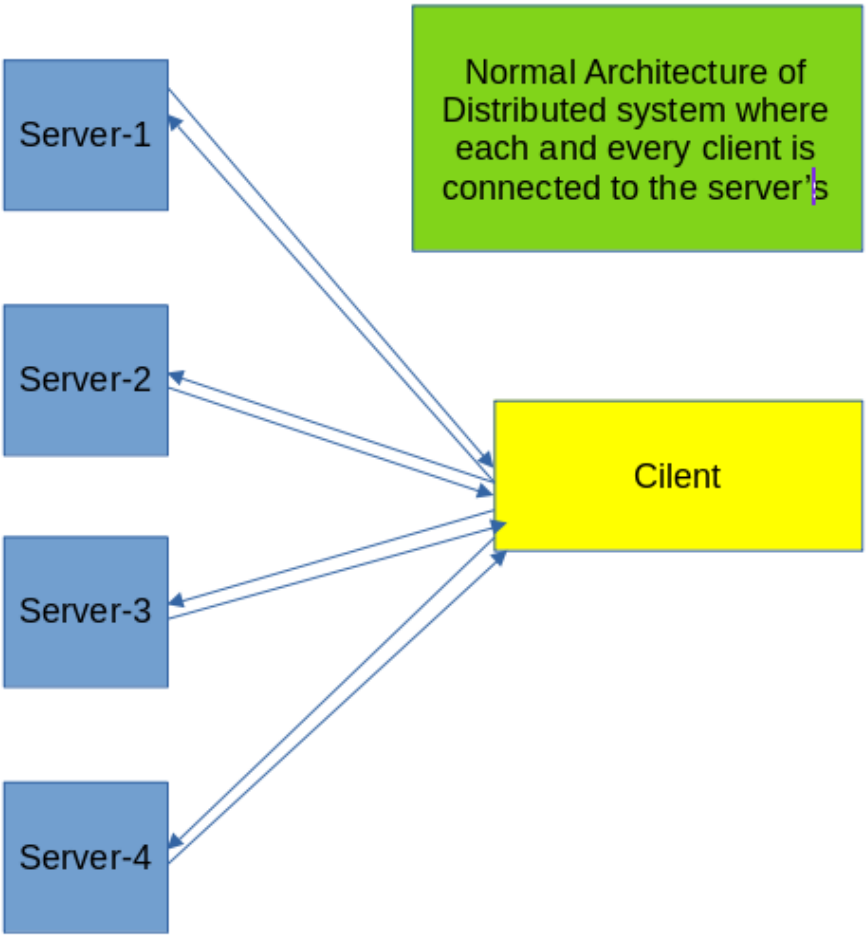
Encryption Algorithms are necessary part of the data sharing today as every bit of data need encryption so that it could be transferred from one place to another without the fear of data leaking. Every lost bit of data puts the system to danger. Encryption algorithms compose of necessary steps involving a key that helps converting plain text to cipher text. All the steps must be performed carefully to ensure that
the data encrypted is found back. Thus, many algorithms are made to run sequentially and in serial.This makes the algorithm to take much computation time.
We aim to reduce the total computation time needed to complete the process by parallelizing the algorithm using the concept of multi-threading. The complexity, data structures and the overall procedure of the algorithm would remain constant.

Normal Single Server Architecture:-

This Type of Architecture is normally used in almost all cloud providers where a big file is uploaded to single big server



This creates lot of latency in the network and may decrease the speed for other processes to Increase the what is we can increase the number of server and divide the file into multiple chunks and send it into different servers from client side itself event load on server decreases ultimatly increasing the speed of the whole system



Now when client wants to upload file it divides the files into smaller chunks and distributes the file into the server it will create a 2-way connection between the client and server so which ultimatly decreases the computation time and decreases latency

2. Introduction

Cloud storage is new way of storage in web2.0 where all the storage is more decentralized and more secure .But uploading a large file is lot costly as for big companies getting files at a lower latency is quite beificial

DISADVANTAGES OF CLOUD STORAGE

Vulnerability to attacks: Security vulnerability is another downside of dealing with cloud computing providers. Any confidential information about the business can be exchanged with a third-party cloud computing service provider. This knowledge could be exploited by hackers.

Downtime: That's because your cloud provider may face power failure, poor access to the internet, maintenance of services, etc.

Vendor Agreement: A organization may face some severe challenges due to the discrepancies between provider solutions as it wants to switch from one cloud platform to another.

Limited control: Cloud clients can be faced with minimal influence over their implementations. On remote servers that are entirely operated and managed by service providers, cloud services run

Platform dependencies: Another of the drawbacks of cloud storage is tacit dependence, also known as' provider lock-in'. Often, deep-rooted discrepancies between provider platforms will make it difficult to switch from one to another cloud platform.

Variation in Costs: Cloud hosting is an inexpensive choice, but it can be expensive if you remember the implementation of the applications.

Internet connectivity: In cloud computing, good Internet access is a must. Without an internet connection, you can't get cloud connectivity.

Lack of support: Cloud Storage providers struggle to provide clients with sufficient assistance. In addition, they tend to focus on FAQs or online assistance from their customers, which can be a boring task for non-technical individuals.

Varied performance: Any selfish action or DDOS attack on your tenant could impact your shared resource efficiency.

Technical issues: Cloud infrastructure is often vulnerable to instability and other technological problems. Even in terms of retaining high maintenance standards, the best cloud service provider companies can face this kind of challenge.

There are many more disadvantages for heavy storage in cloud computing So, we came up with a way of distributing a single file into different servers such that even if there s a fialure of multiple servers files can still be fetched more efficient way with less latency than cloud storage

3. Overview of the work:-

3.a. Objectives of the work:-

The objective of the Distributed file system is that to use in build python socker library where it connects to the servers which is sitting on the same localhost sockets like 10001,10002,10003,10004 and client ping to it.When a client wants to upload a file it divides files into equal chunks as we are sending in sockets the file it converts into byte stream when the server gets the file then the server converts the byte stream into file stream which is readable.

3.b. Software Requirments:-

3.b.i. Python3.7

3.b.ii. Docker

3.b.iii. Redis

3.b.iv. Free ports At 10001,10002,10003,10004

3.b.v. GCC 7.4 and above

3.c. Hardware Requirements:-

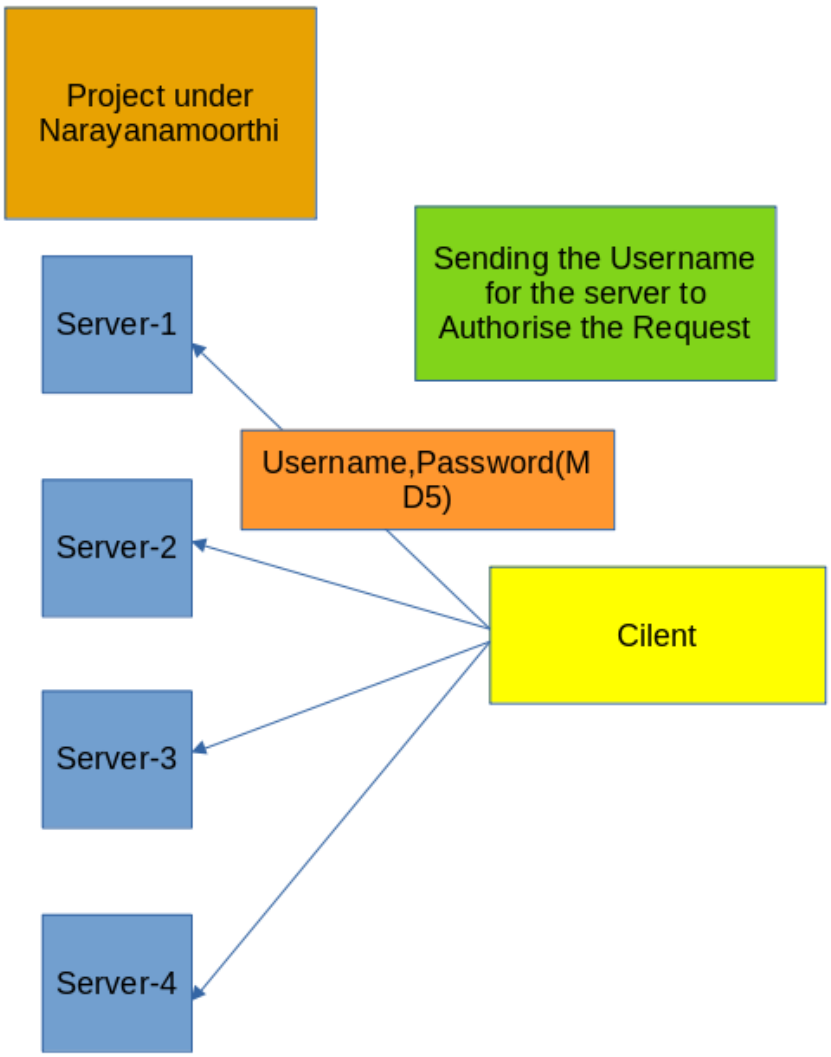
The processor can be multiple core or a single core. The processor should not be single threaded. The performance of all the programs have been tested on Intel i5 8 th Generation processors.The Preffered Os is Linux for enabling faster port and socket programming

4. System Design :-

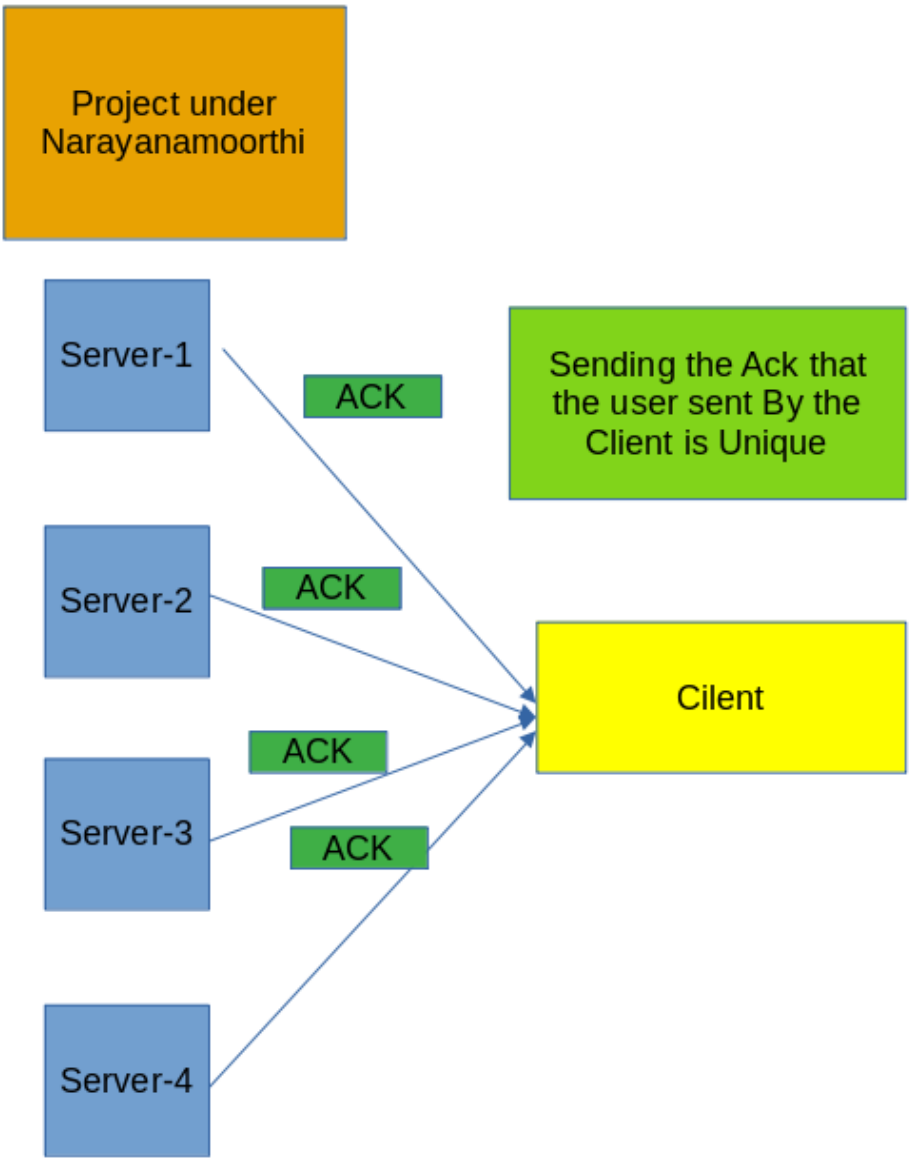
Authorization Based Client server Architecture:-

In this process when the client connect to the server it prompts for username and passwrod when it authorized in the client side there should be a way to authorize in backend (server) processs also so it send username and password in MD5 encrypted format as the data flows through sockets aren't that secure and cant be vulunrable to backdoor attacks so when the sever also gets the auth details it authicate and sends back the ACK (Acknolgment) request such that the user is authorized to send request .

Block Diagram for Auth Mechanism :-



The server accepting the Image:-



code for auth Client and Server:-

```python
def client_auth(auth_dict, username, password):
    ct = 0
    auth_status=''
    for key, value in auth_dict.items():
        ct += 1
        if auth_status != '':
            pass
        else:
            if ct < len(auth_dict):

                if username == key:
                    print('Correct username.')

                    if password == value:
                        print('Correct password.')

                        auth_status='Authorization Granted.\n'
                        print(auth_status)
                        conn.send(auth_status.encode())
                        pass
                    else:
                        print('Incorrect password.')
                        auth_status = 'Authorization Denied.\n'
                        print(auth_status)
                        conn.send(auth_status.encode())
                        sys.exit()
                else:
                    continue

            else:
                if username == key:
                    print('Correct username.')

                    if password == value:
                        print('Correct password.')

                        auth_status='Authorization Granted.\n'
                        print(auth_status)
                        conn.send(auth_status.encode())
                        pass
                    else:
                        print('Incorrect password.')
                        auth_status = 'Authorization Denied.\n'
                        print(auth_status)
                        conn.send(auth_status.encode())
                        sys.exit()
                else:
                    print('Incorrect username.')
                    auth_status = 'Authorization Denied.\n'
                    print(auth_status)
                    conn.send(auth_status.encode())
                    sys.exit()
```

Where the passwords are the stored in dnf.conf file through which server and client

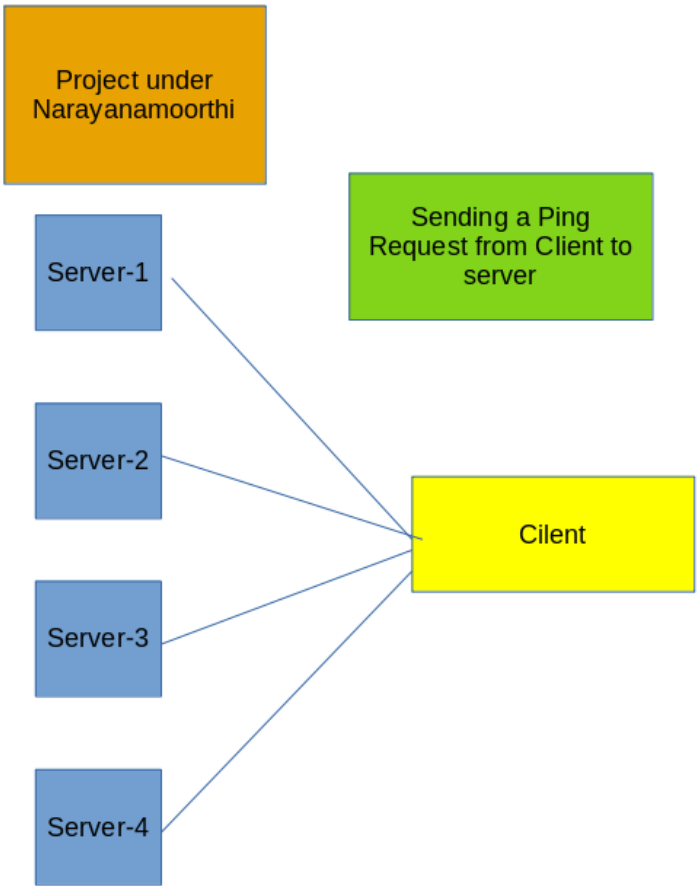
dfs.conf:-

```
dfc.conf
socket > PythonDFS > DFC > dfc.conf
Server DFS1 127.0.0.1:10001
Server DFS2 127.0.0.1:10002
Server DFS3 127.0.0.1:10003
Server DFS3 127.0.0.1:10004
Username: rohan
Password: c916d142f0dc7f9389653a164f1d4e9d
Username: suyasha
Password: 4fa6e053e6cdead4e4a56c82befc4eb8
Username: srusti
Password: e7741308a786f5522a681aa8fe343363
```

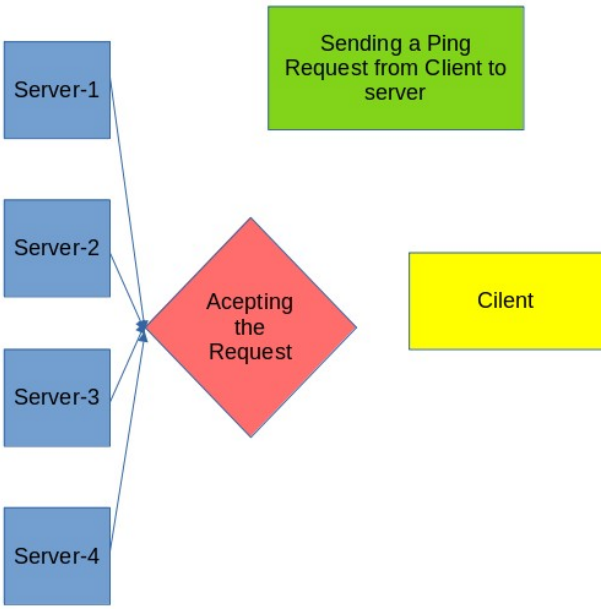
This file contains information of users and location of servers

Before Auth the server sends a ping on all the server so that it can comes to conculsion that on to which server It can send the Auth request

Block Diagram for Initial Ping:-



After server Accepts the Request:-



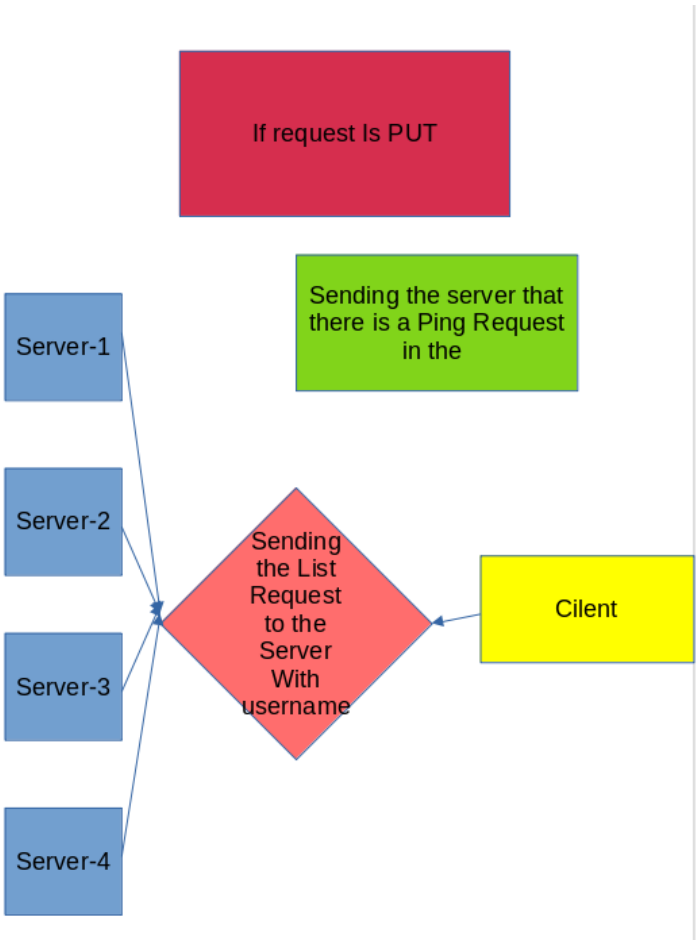
Type of request that a Client can make to The server are:-

- PUT
- GET
- LIST

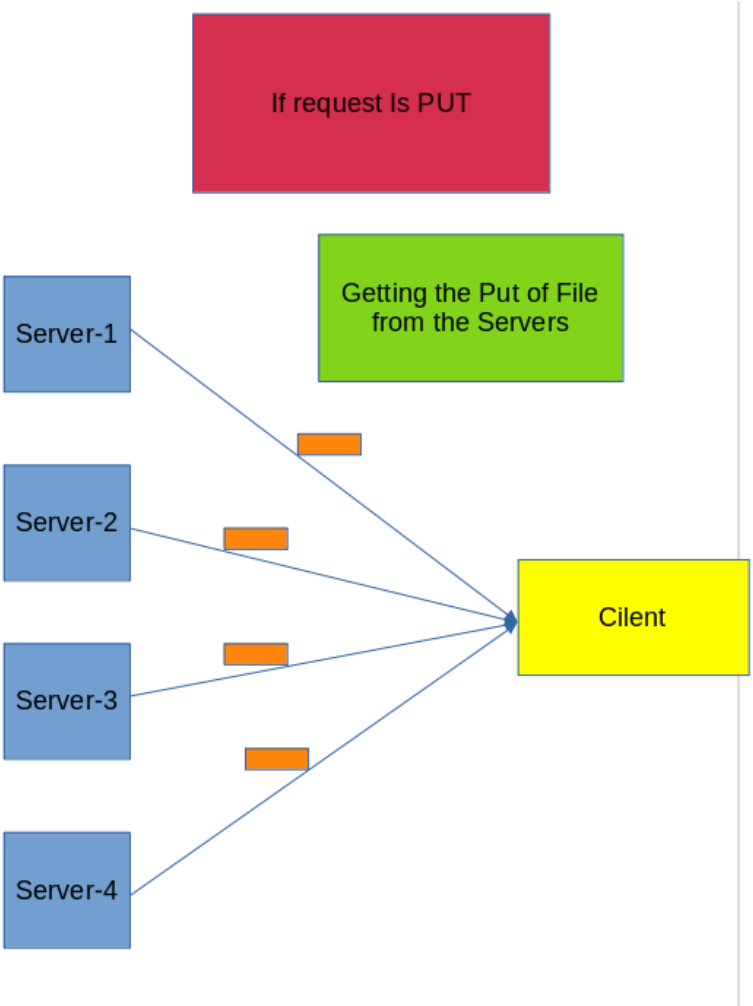
PUT:-

When this command is used to a Put command is send to all the servers where these server acknowwledge with accpetance then the client sent then the files are divided into chunks of parts and being sent to the server

Block Diagram for Put Request:-



Reciving The acknowldgement from the server:-



Fitting Strategy with Hashing:-

The Fitting Stategy in the files with Hashing

[PUT] method:
PUT sends any text files located within the DFC folder into the DFS folders for distributed storage.

PUT splits files into 4 chunks, stores pairs of chunks into each server after hashing file and taking the modulus of the hash to ensure fair distribution, according to the table below. The duplication of files ensures reliability if 1 server is down.

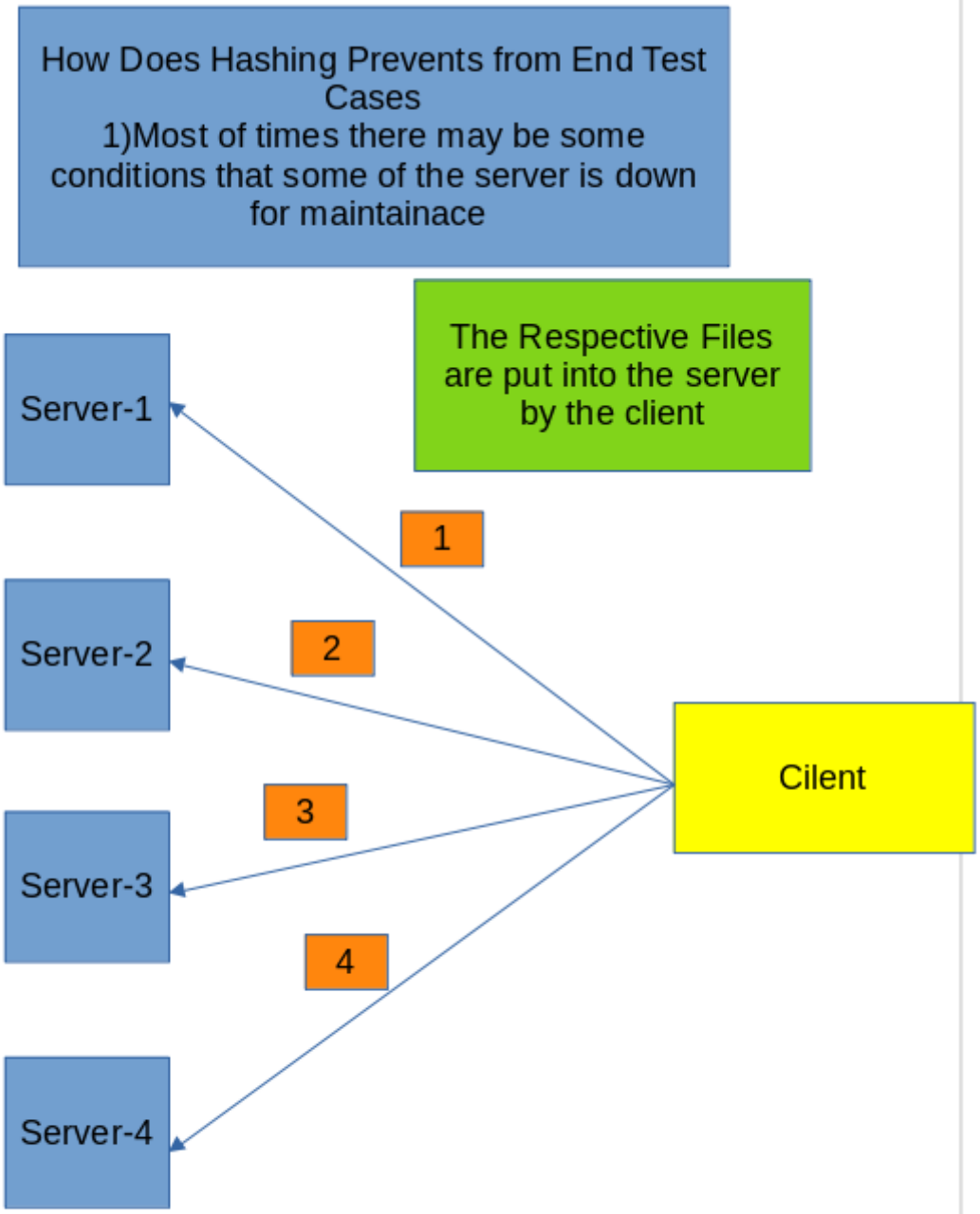
PUT also lists files within the DFC folder which are available for transfer.

hash mod	DFS1	DFS2	DFS3	DFS4
0	(1,2)	(2,3)	(3,4)	(4,1)
1	(4,1)	(1,2)	(2,3)	(3,4)
2	(3,4)	(4,1)	(1,2)	(2,3)
3	(2,3)	(3,4)	(4,1)	(1,2)

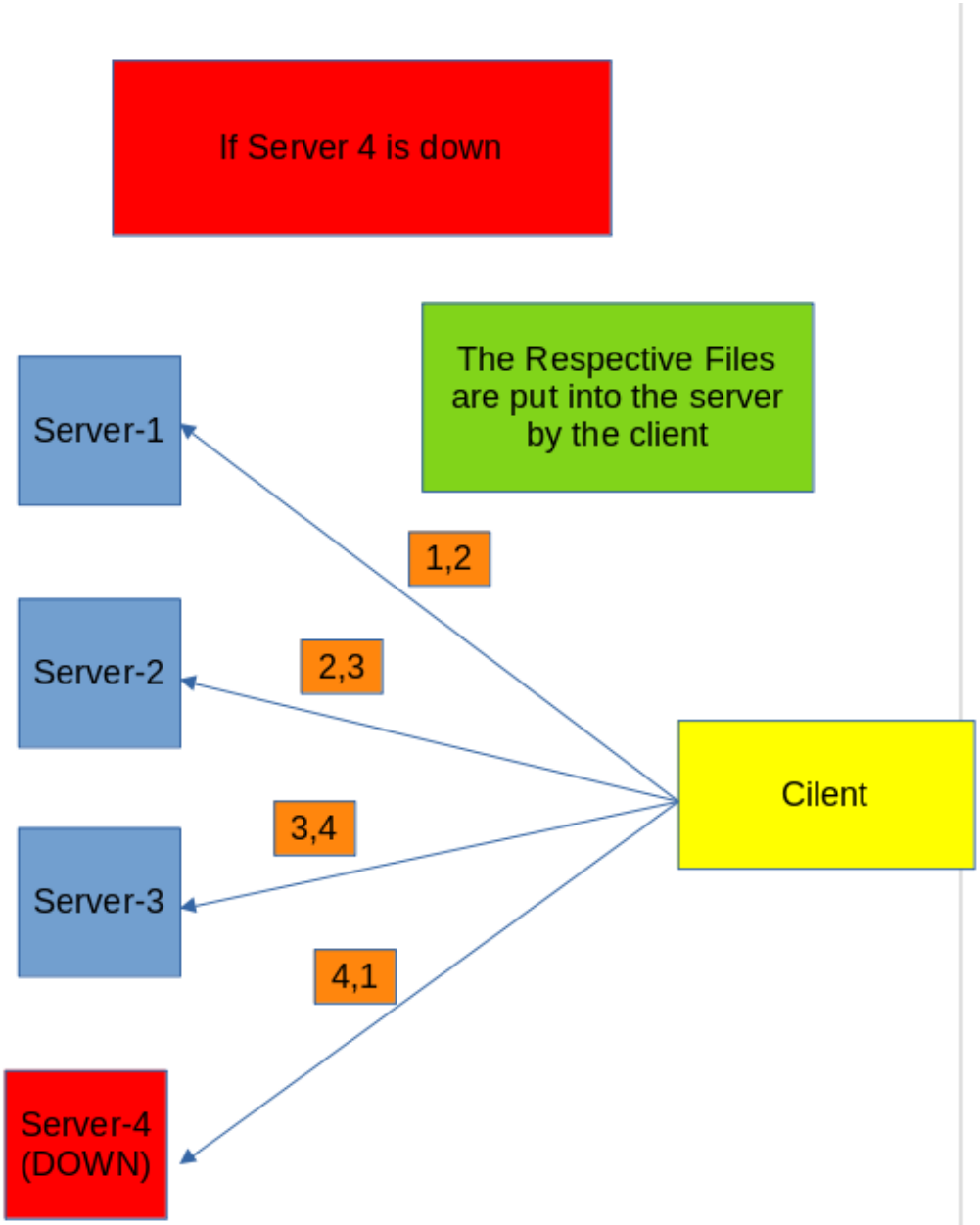
This hashing mechanism make sure that the if a any one server is down then the file Reveied may not be distrubed

Block for Representing Distribution:-

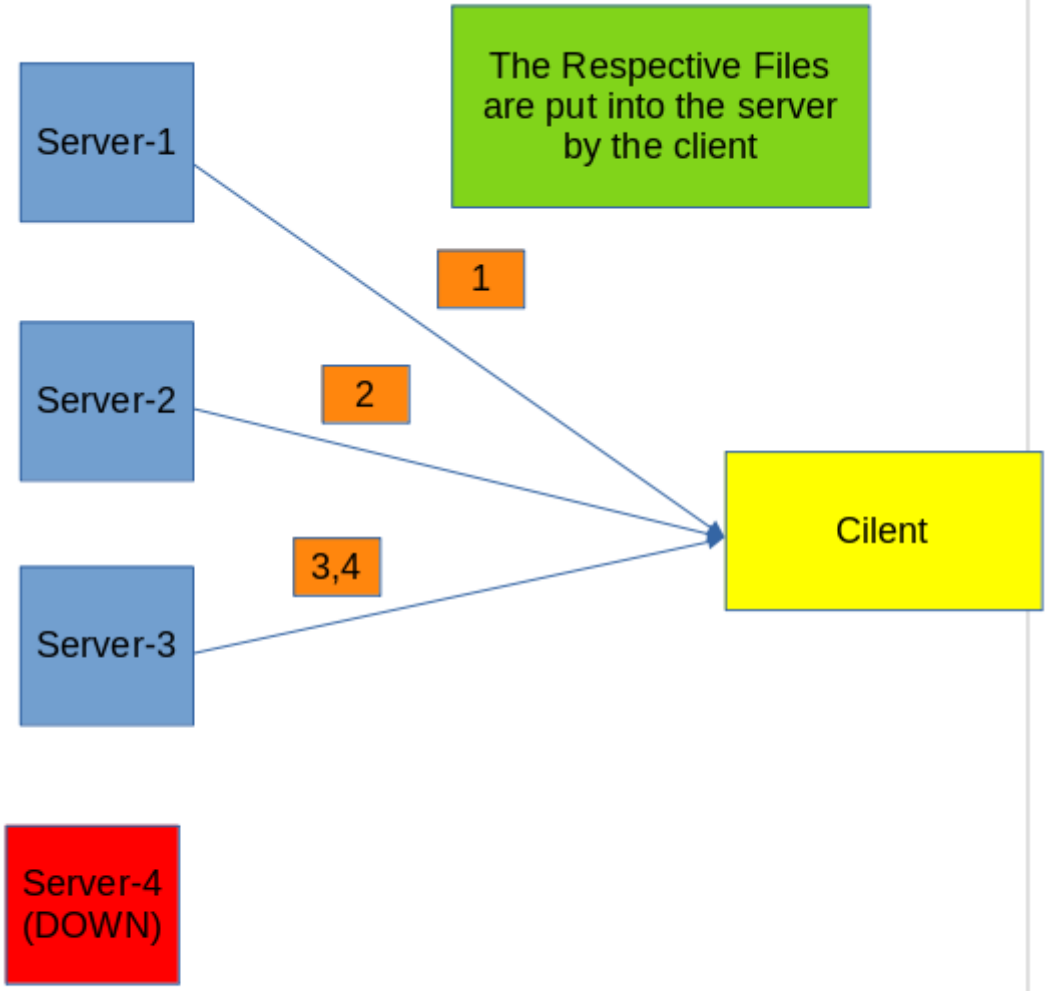
END test cases:-



If any particular server is down the files are distribured according to the hash table:-



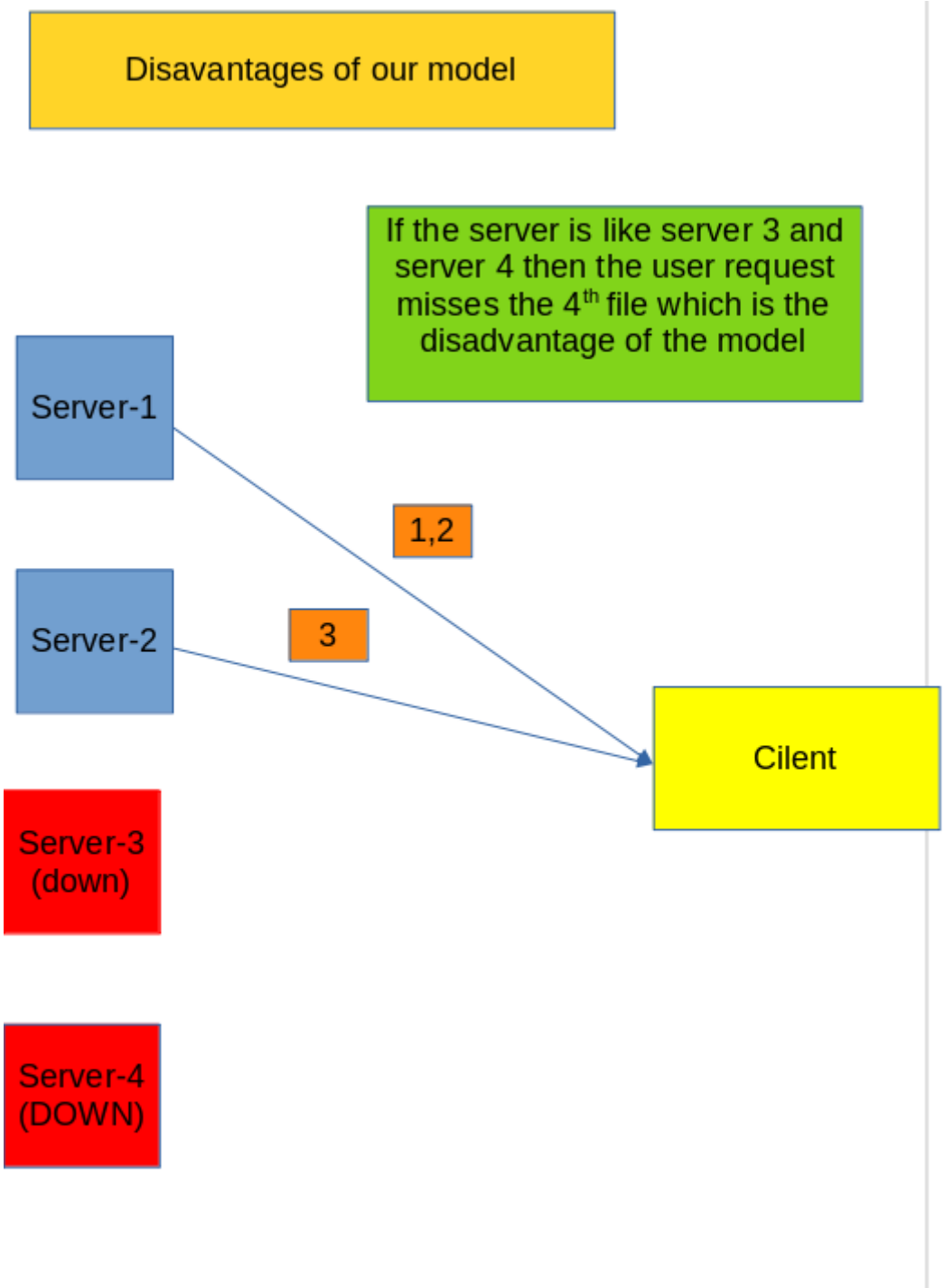
IF the client wants the file back then



Then the client can get the file back from the server 3 with hasing table

Disadvantages of this Hasing Mechanism:-

What is the 2 servers are down then the Hasing table may not help to retrive the file:-

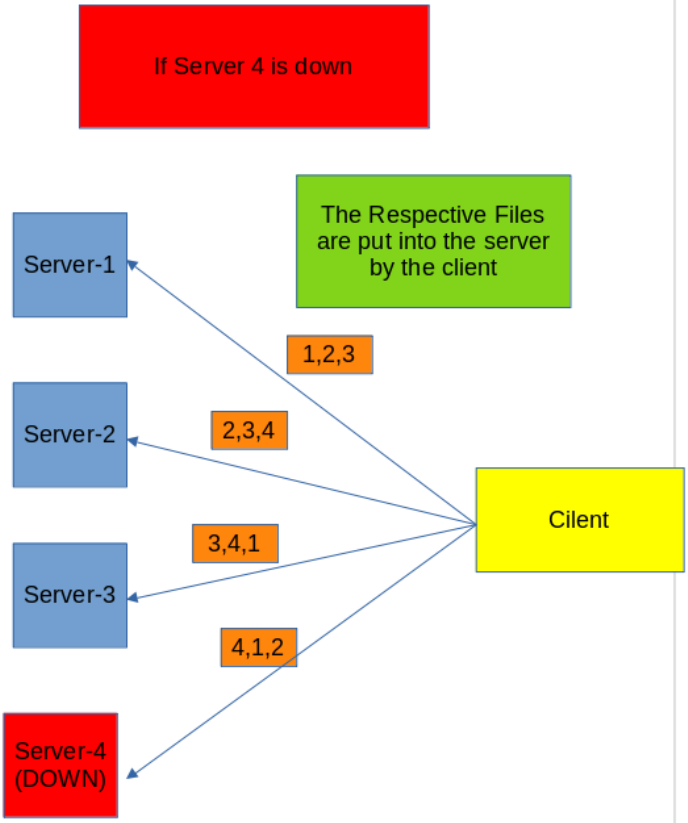


In this case the client may lose the file 4 in the result of

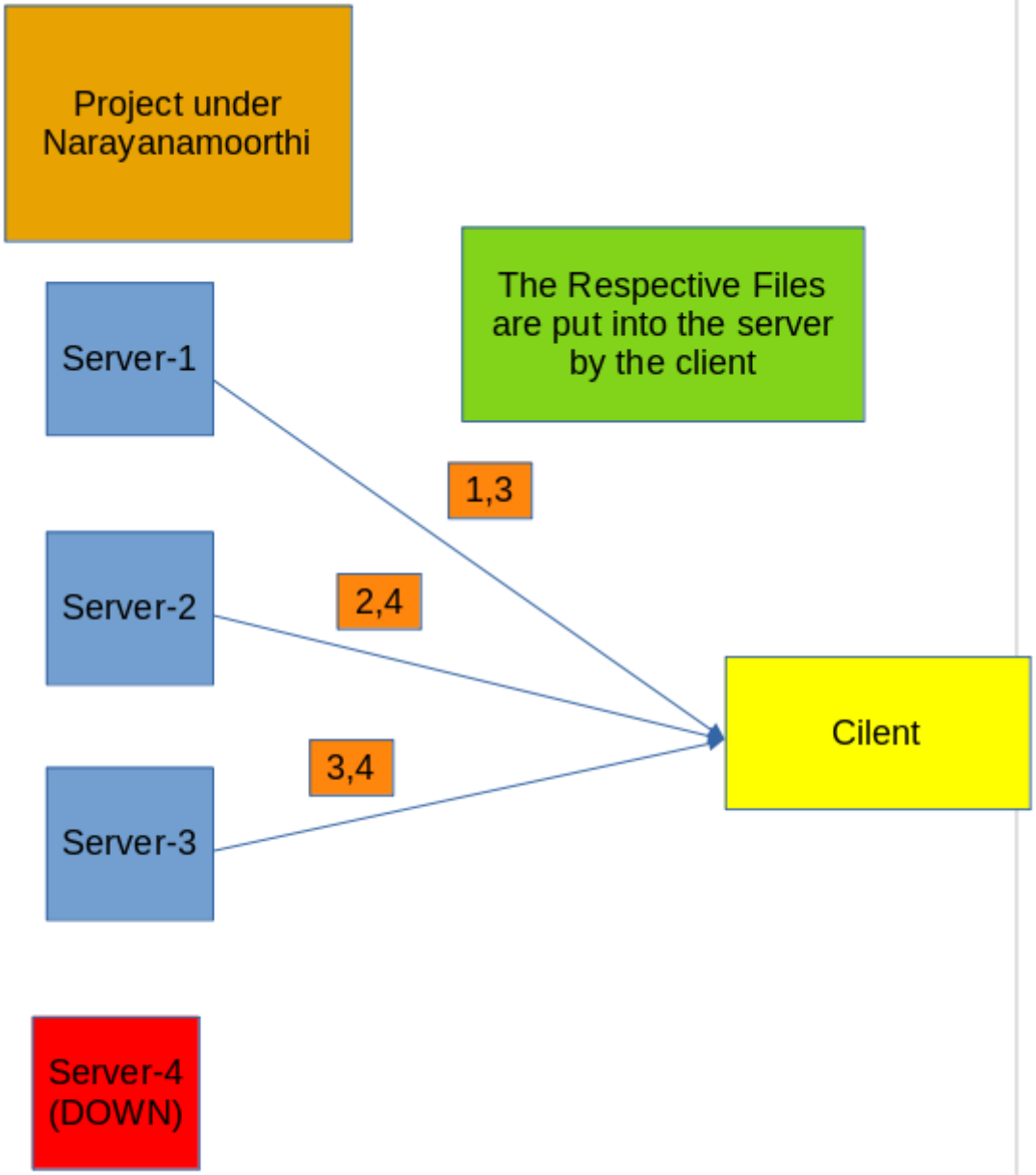
Solution for above Problem ;-

Is to increase the Hasing tables

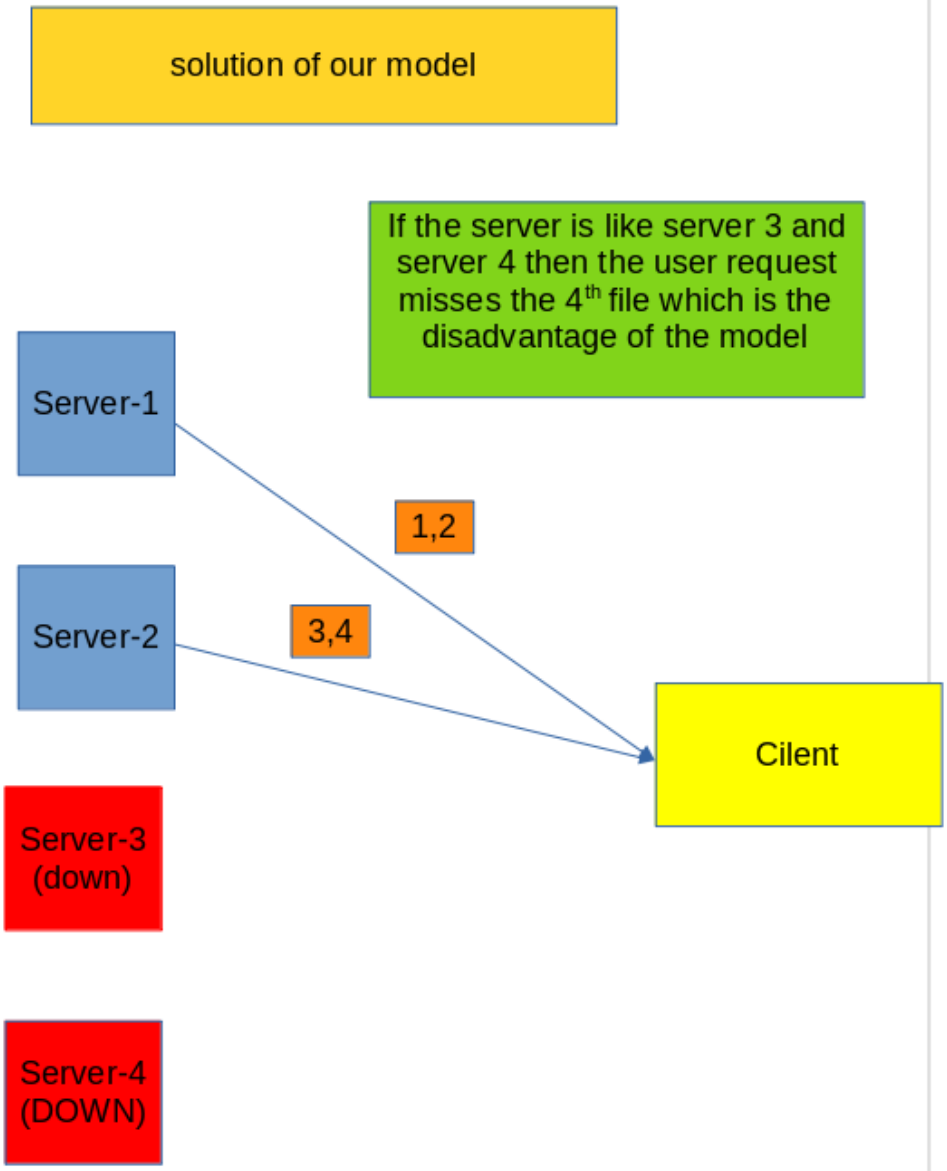
hash mod	DFS1	DFS2	DFS3	DFS4
0	(1,2,3)	(2,3,4)	(3,4,1)	(4,1,2)
1	(4,1,2)	(1,2,3)	(2,3,4)	(3,4,1)
2	(3,4,1)	(4,1,2)	(1,2,3)	(2,3,4)
3	(2,3,4)	(3,4,1)	(4,1,2)	(1,2,3)



Now If 2 servers are Down:-



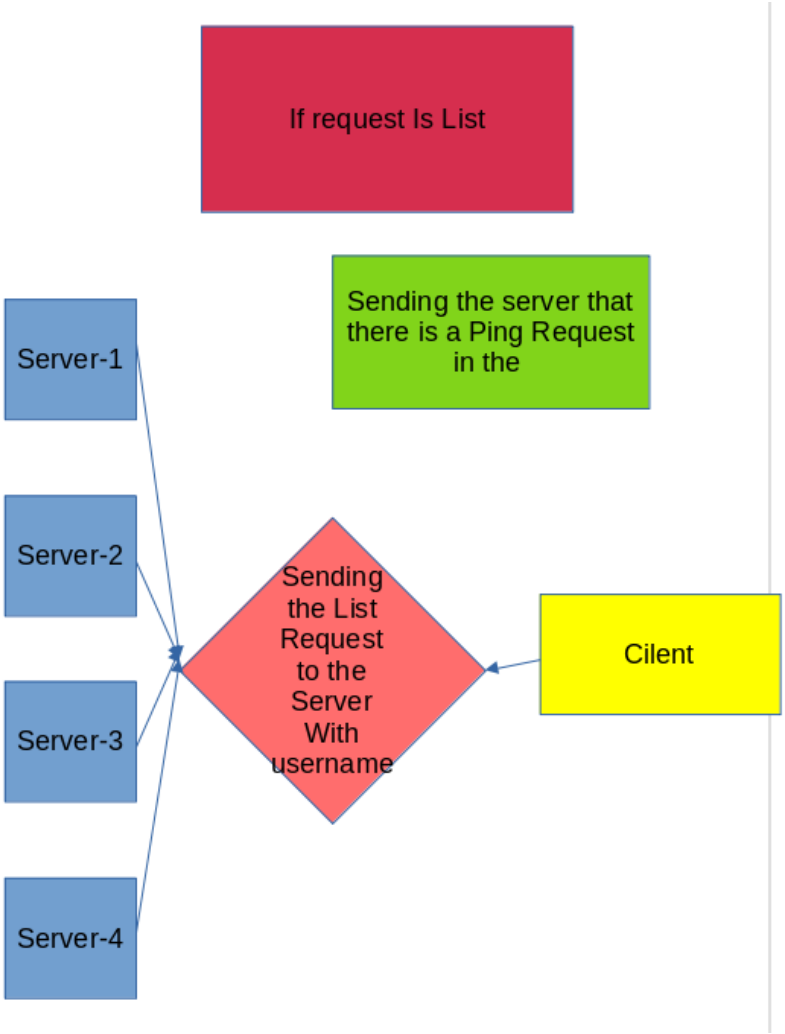
so server 4 and server 3 are down:-

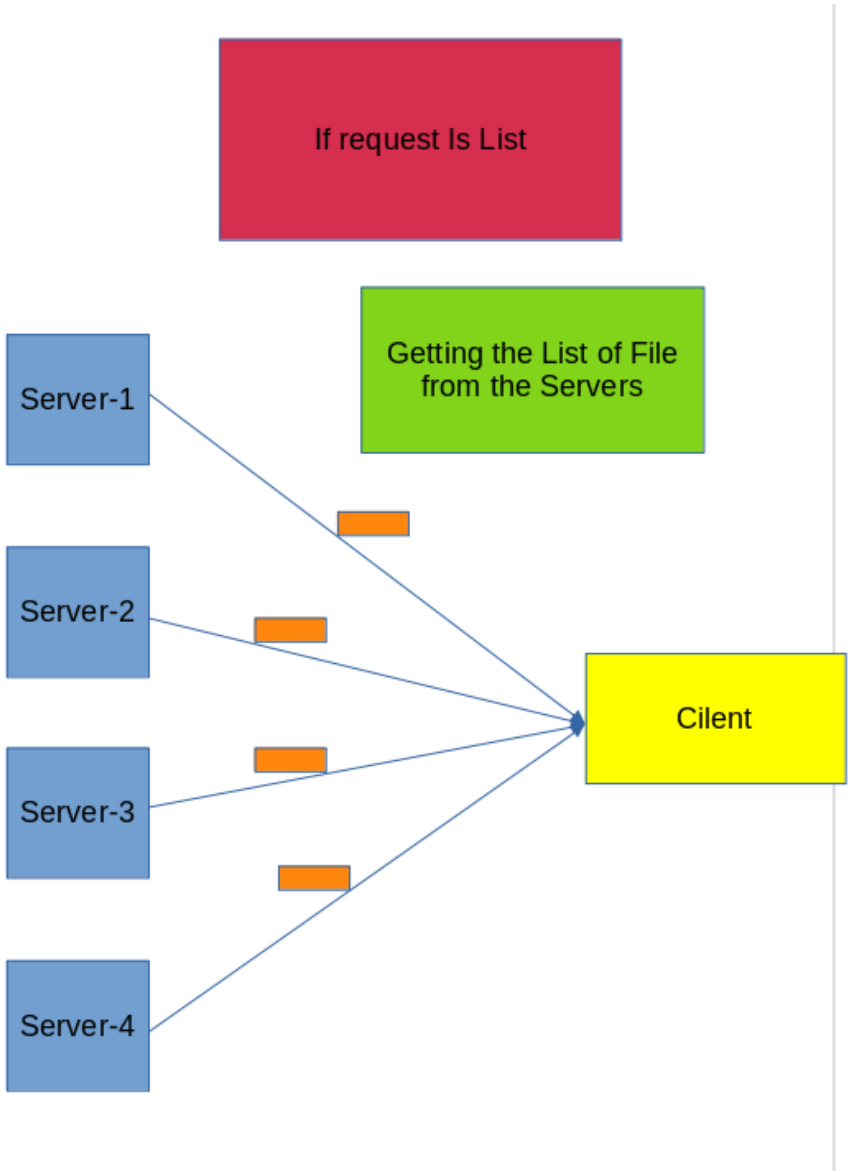


LIST command:-

This command is used by the client when they request for the files that they have already in the servers which the server only provides a list of available files

Block Diagrams:-

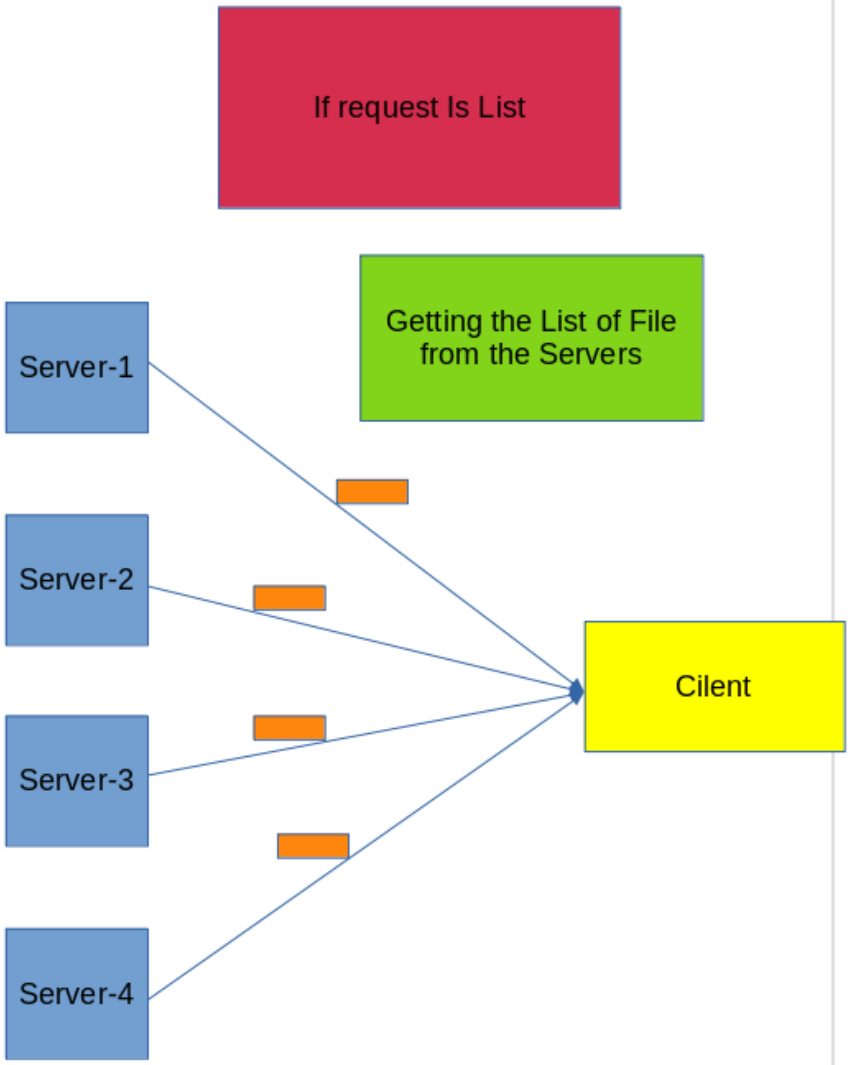




GET command:-

This is similar way of list command but in this case the files are requested instead of the File names

<u>Non Blocking Disadvantage of the Model:-</u>



5. Output and Performance Analysis:-

5.a. Execution Screeenshots:-

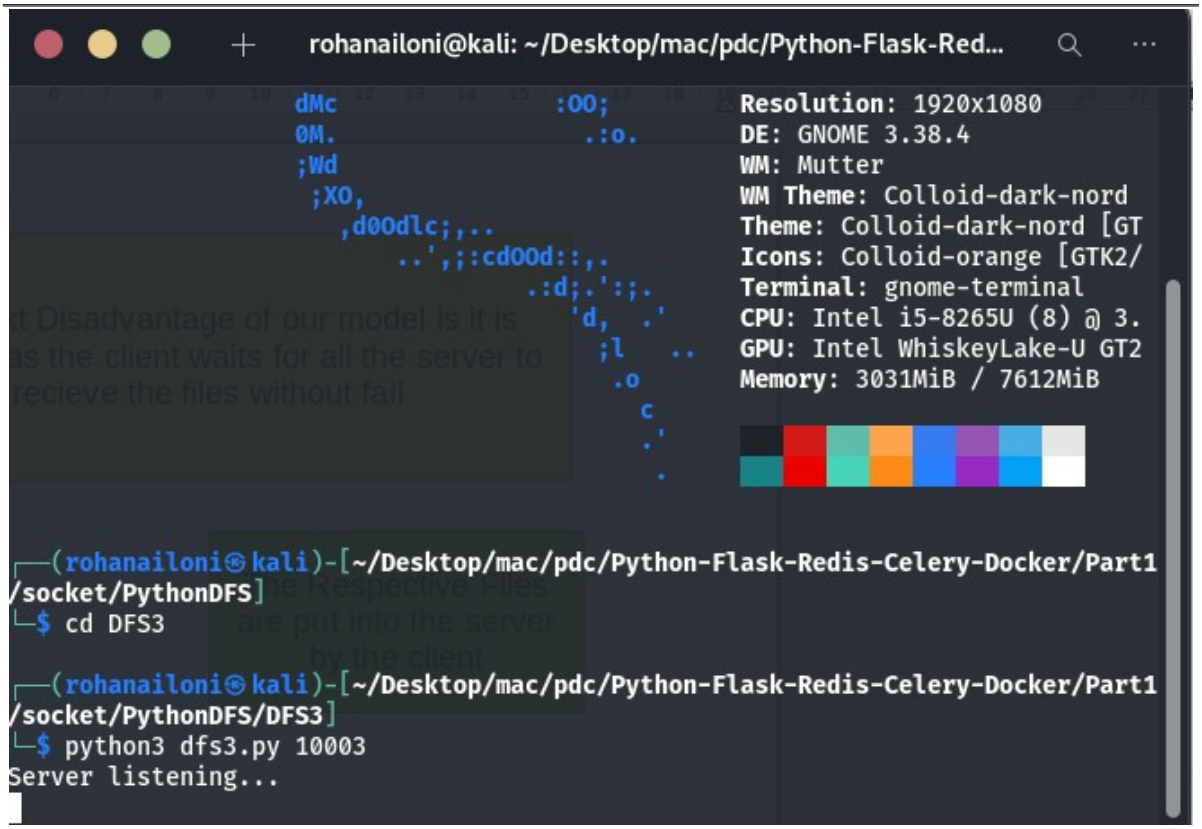
5.a.i.1. Running server 1

```
rohanailoni@kali: ~/Desktop/mac/pdc/Python-Flask-Red...
                dMc                 :OO;          Resolution: 1920x1080
                0M.                   .:o.        DE: GNOME 3.38.4
                ;Wd                               WM: Mutter
                 ;XO,                             WM Theme: Colloid-dark-nord
                   ,d0Odlc;,..                    Theme: Colloid-dark-nord [GT
                       ..',;:cdOOd::,.            Icons: Colloid-orange [GTK2/
                                .:d;.':;.         Terminal: gnome-terminal
                                   'd,  .'        CPU: Intel i5-8265U (8) @ 3.
                                     ;l   ..      GPU: Intel WhiskeyLake-U GT2
                                      .o          Memory: 3056MiB / 7612MiB
                                        c
                                        .'
                                         .

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS]
└─$ cd DFS1

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS/DFS1]
└─$ python3 dfs1.py 10001
Server listening...
```

5.a.i.2. running server 2

```
rohanailoni@kali: ~/Desktop/mac/pdc/Python-Flask-Red...

Resolution: 1920x1080
DE: GNOME 3.38.4
WM: Mutter
WM Theme: Colloid-dark-nord
Theme: Colloid-dark-nord [GT
Icons: Colloid-orange [GTK2/
Terminal: gnome-terminal
CPU: Intel i5-8265U (8) @ 3.
GPU: Intel WhiskeyLake-U GT2
Memory: 3028MiB / 7612MiB

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS]
└─$ cd DFS2

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS/DFS2]
└─$ python3 dfs2.py 10002
Server listening...
```

5.a.i.3. Running server 3



5.a.i.4. Running the client with Authorizartion:-

```
rohanailoni@kali: ~/Desktop/mac/pdc/Python-Flask-Red...
└─$ ls -a
.  ..  DFC  DFS1  DFS2  DFS3  DFS4

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS]
└─$ cd DFC

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS/DFC]
└─$ python3 dfc.py dfc.conf
username: rohan
password: rohan
Authorization Granted.
Connected to server DFS1
Connected to server DFS2
Connected to server DFS3
Could not connect to server DFS4
From DFS1: Authorization Granted.

From DFS2: Authorization Granted.

From DFS3: Authorization Granted.

Please specify a command [get, list, put]:
```

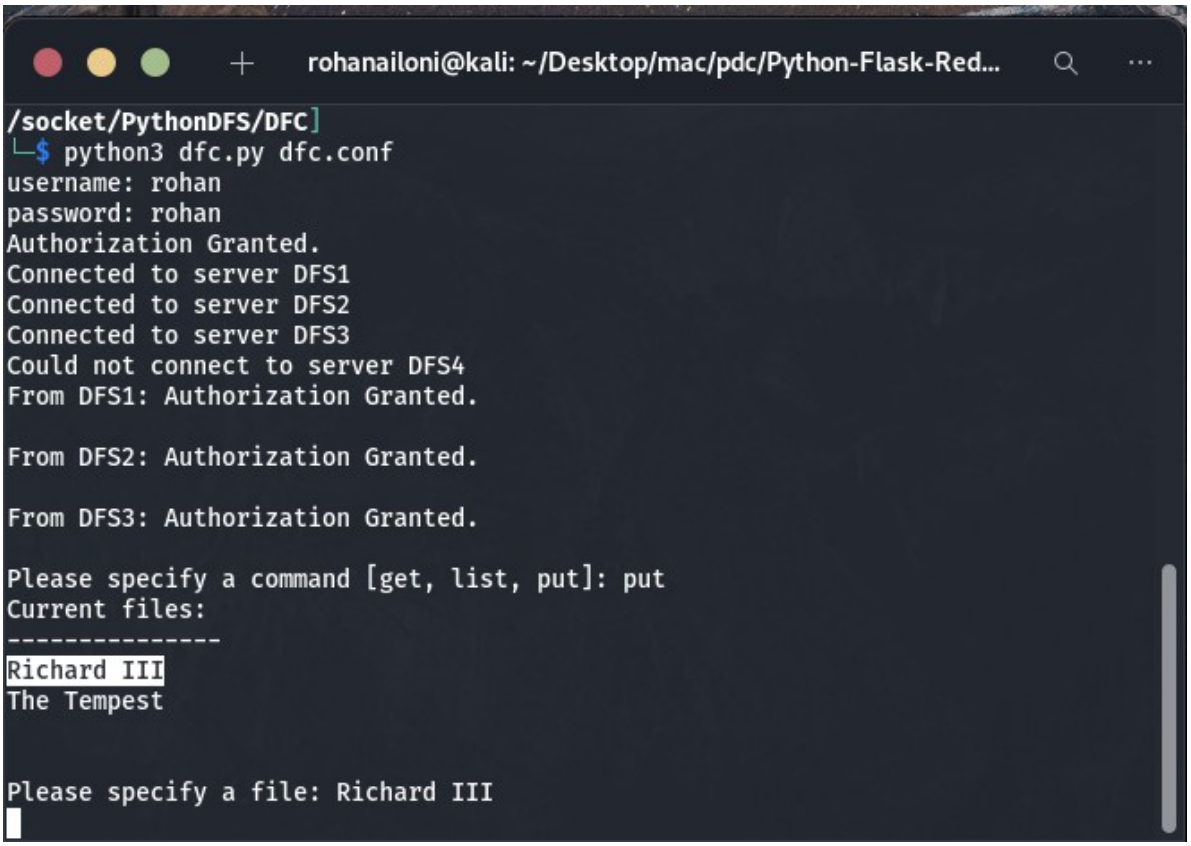
5.a.i.5. Server Accepting (Authenticating) the user:-

```
rohanailoni@kali: ~/Desktop/mac/pdc/Python-Flask-Red...
CPU: Intel i5-8265U (8) @ 3.
GPU: Intel WhiskeyLake-U GT2
Memory: 3056MiB / 7612MiB

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS]
└─$ cd DFS1

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS/DFS1]
└─$ python3 dfs1.py 10001
Server listening...
Connected to Client.
received username
received password
Correct username.
Correct password.
Authorization Granted.
```

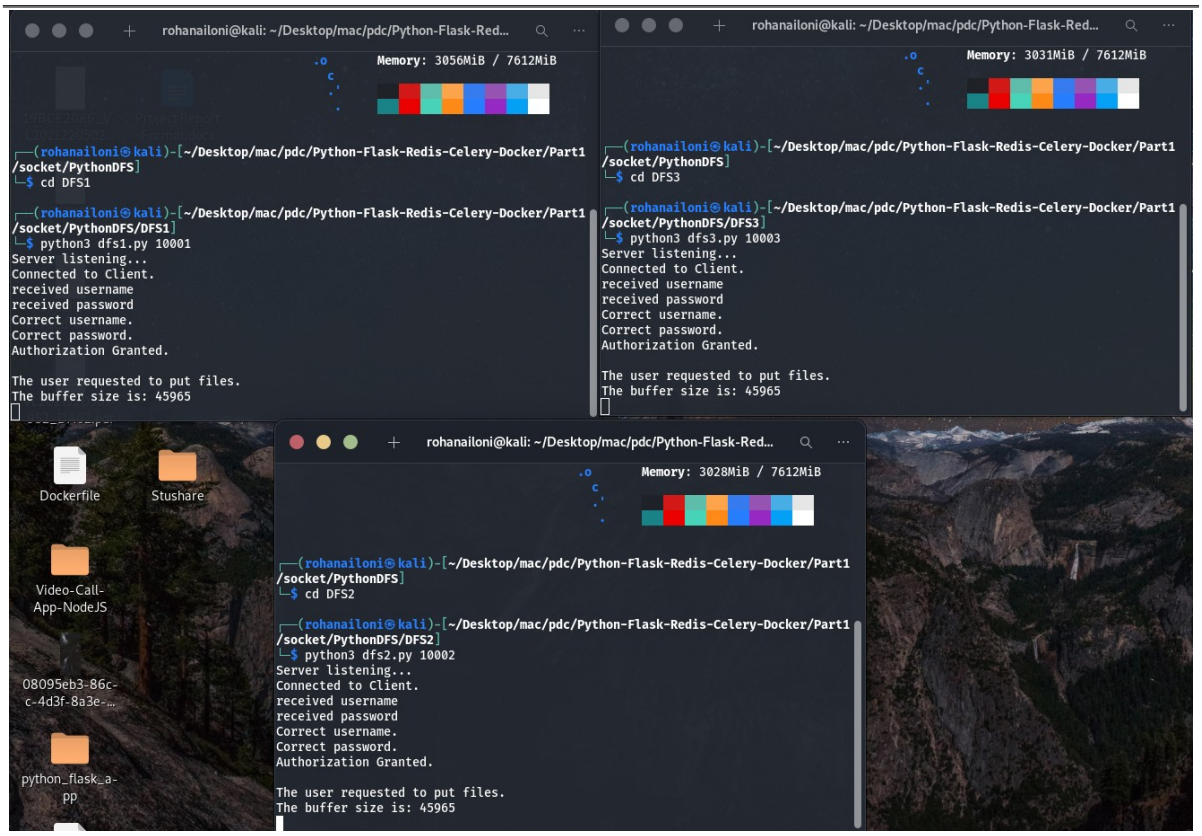
```
rohanailoni@kali: ~/Desktop/mac/pdc/Python-Flask-Red...
CPU: Intel i5-8265U (8) @ 3.
GPU: Intel WhiskeyLake-U GT2
Memory: 3031MiB / 7612MiB

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS]
└─$ cd DFS3

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS/DFS3]
└─$ python3 dfs3.py 10003
Server listening...
Connected to Client.
received username
received password
Correct username.
Correct password.
Authorization Granted.
```

```
rohanailoni@kali: ~/Desktop/mac/pdc/Python-Flask-Red...
CPU: Intel i5-8265U (8) @ 3.
GPU: Intel WhiskeyLake-U GT2
Memory: 3028MiB / 7612MiB

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS]
└─$ cd DFS2

┌──(rohanailoni㉿kali)-[~/Desktop/mac/pdc/Python-Flask-Redis-Celery-Docker/Part1
/socket/PythonDFS/DFS2]
└─$ python3 dfs2.py 10002
Server listening...
Connected to Client.
received username
received password
Correct username.
Correct password.
Authorization Granted.
```

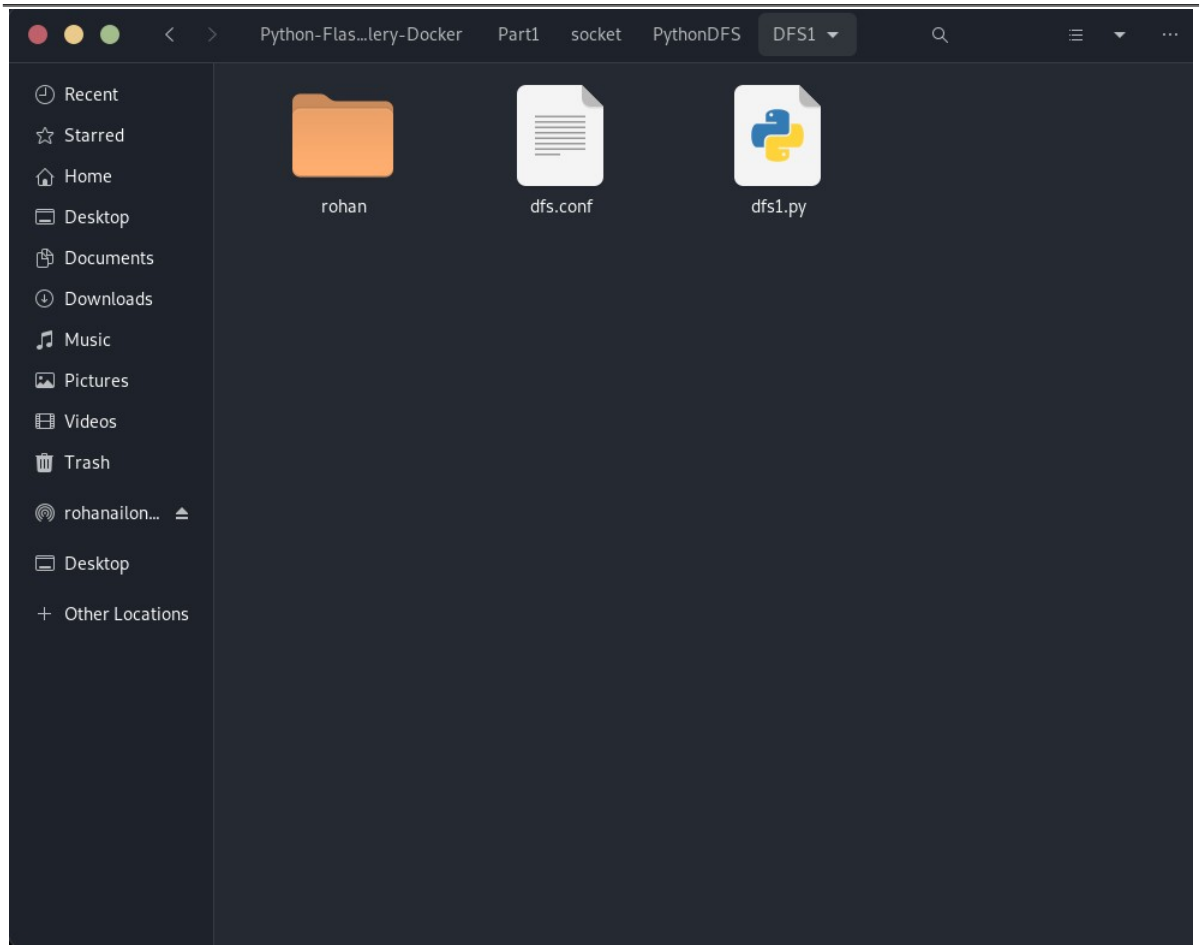
5.a.i.6.PUTTING a file into the Servers:-



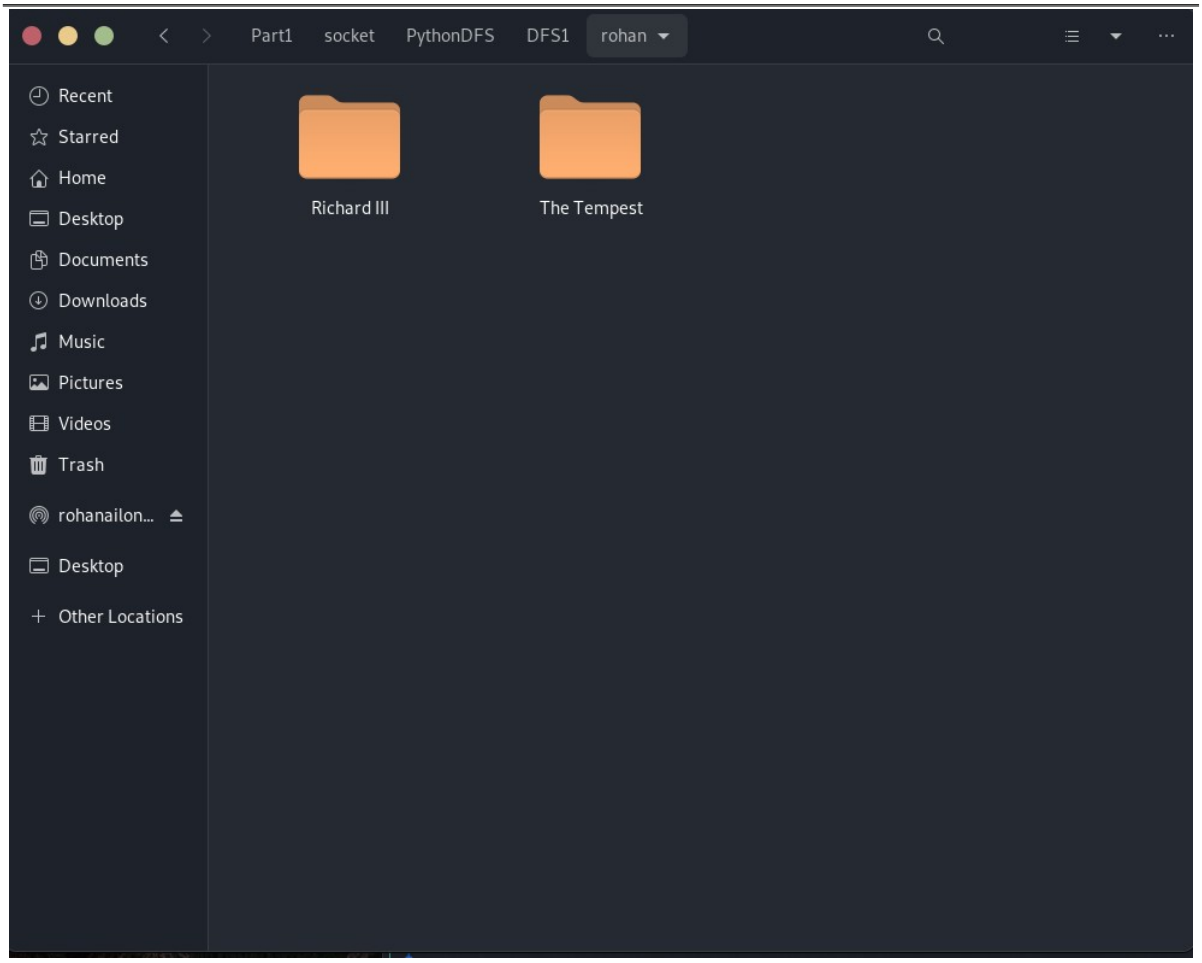
5.a.i.7.Servers Recieving the chunks of files :-



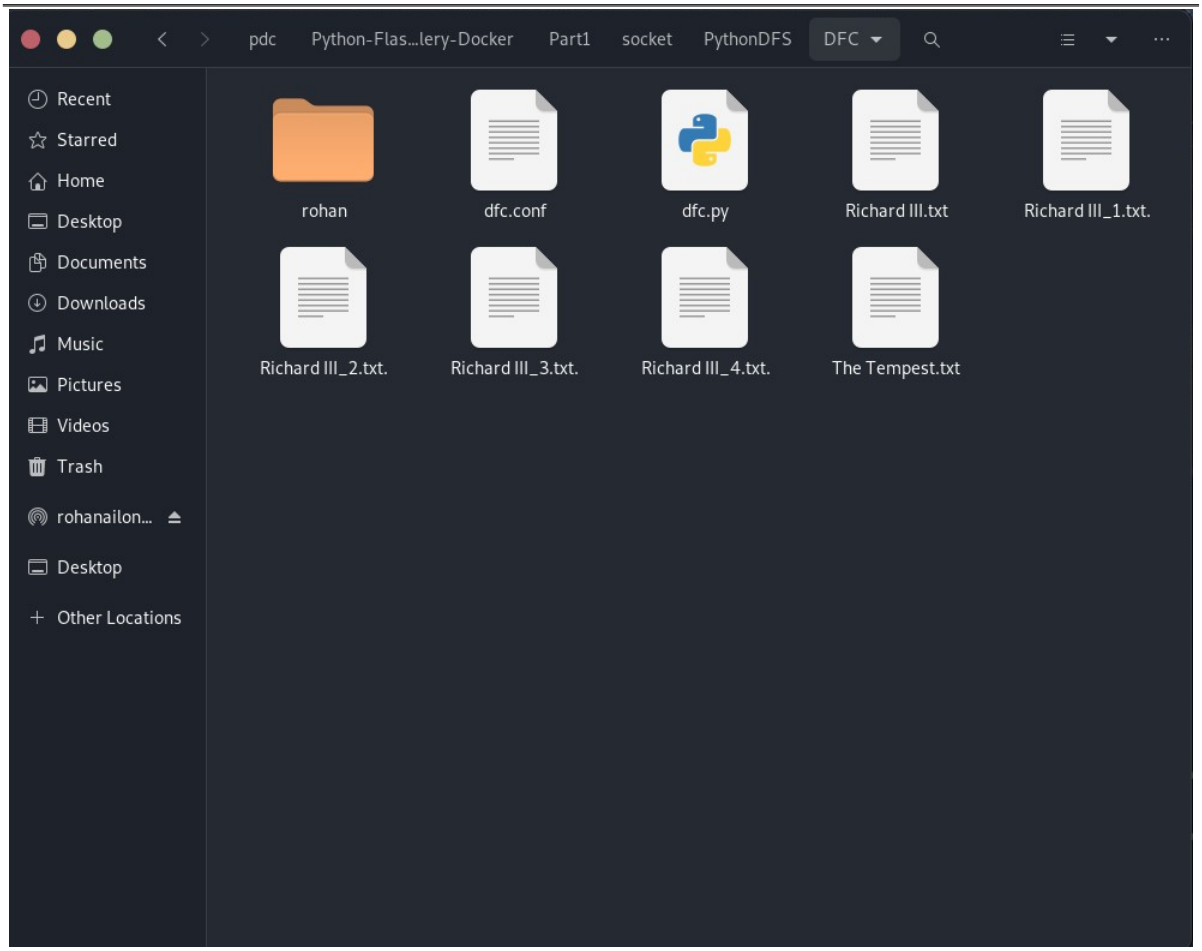
5.a.i.8.New Folder is created in Server side folders



5.a.i.9. And a Folder for File is Created:-



5.a.i.10. Chunks of files are created in the folders ;-



1.1. Source Code:-

1.2. Github Link for code:-

https://github.com/rohanailoni/pdc_project/tree/main/socket/PythonDFS

code for DFC.py

```
#! /usr/bin/env python3


import os
import re
import sys
import time
import socket
import glob
import pickle
```

```
def check_args():

if len(sys.argv) != 2:
print("ERROR: Must supply port number \nUSAGE: py dfs1.py 10001")
sys.exit()

else:
try:
if int(sys.argv[1]) != 10001:
print("ERROR: Port number must be 10001")
sys.exit()
else:
return int(sys.argv[1])
except ValueError:
print("ERROR: Port number must be a number.")
sys.exit()

check_args()
def auth_params():
config_file='dfs.conf'
fh=open(config_file, mode='r', encoding='cp1252')
users=re.findall(r'Username: .*', fh.read())
usernames=list()
for i in range(0, len(users)):
usernames.append(str(users[i]).split()[1])
fh.close()
fh=open(config_file, mode='r', encoding='cp1252')
passes=re.findall(r'Password: .*', fh.read())
passwords=list()
for i in range(0, len(passes)):
passwords.append(str(passes[i]).split()[1])
fh.close()
global auth_dict
auth_dict = {}
for i in range(0, len(users)):
entry={usernames[i]:passwords[i]}
auth_dict.update(entry)

return auth_dict

def client_auth(auth_dict, username, password):
ct = 0
auth_status=''
for key, value in auth_dict.items():
ct += 1
if auth_status != '':
pass
```

```
else:
# check all users up to last
if ct < len(auth_dict):
if username == key:
print('Correct username.')
if password == value:
print('Correct password.')
auth_status='Authorization Granted.\n'
print(auth_status)
conn.send(auth_status.encode())
pass
else:
print('Incorrect password.')
auth_status = 'Authorization Denied.\n'
print(auth_status)
conn.send(auth_status.encode())
sys.exit()
else:
continue
# check last user
else:
if username == key:
print('Correct username.')
if password == value:
print('Correct password.')
auth_status='Authorization Granted.\n'
print(auth_status)
conn.send(auth_status.encode())
pass
else:
print('Incorrect password.')
auth_status = 'Authorization Denied.\n'
print(auth_status)
conn.send(auth_status.encode())
sys.exit()
else:
print('Incorrect username.')
auth_status = 'Authorization Denied.\n'
print(auth_status)
conn.send(auth_status.encode())
sys.exit()

def put(new_dir_path):
try:
buffersize = int(conn.recv(2048).decode())
print('The buffer size is: ' +str(buffersize))
except ValueError:
print('The buffer size is not a number. \nExiting now...')
```

```
sys.exit()
name1 = conn.recv(1024).decode()
chunk1 = conn.recv(buffersize).decode()
print('Receiving ' +name1 +'...\n')
file_folder = name1.split('_')[0]
new_folder_path = os.getcwd() +'/' +username +'/' +file_folder
if os.path.isdir(new_folder_path) == False:
try:
os.mkdir(new_folder_path)
print ("Successfully created the folder %s " % new_folder_path)
pass
except OSError:
print ("Creation of the folder %s failed" % new_folder_path)
else:
pass
fh=open(os.path.join(new_folder_path, name1), 'w')
fh.write(chunk1)
fh.close()

exists = new_folder_path +'/' +name1
if os.path.isfile(exists) == True:
response = 'Chunk 1 successfully transferred.\n'
print(response)
conn.send(response.encode())
else:
response = 'Chunk 1 transfer incomplete.\n'
print(response)
conn.send(response.encode())
name2 = conn.recv(1024).decode()
chunk2 = conn.recv(buffersize).decode()
print('Receiving ' +name2 +'...\n')

fh=open(os.path.join(new_folder_path, name2), 'w')
fh.write(chunk2)
fh.close()
exists = new_folder_path +'/' +name2
if os.path.isfile(exists) == True:
response = 'Chunk 2 successfully transferred.\n'
print(response)
conn.send(response.encode())
else:
response = 'Chunk 2 transfer incomplete.\n'
print(response)
conn.send(response.encode())
print('Exiting now...')
sys.exit()
```

```
def new_dir(username):

global new_dir_path
new_dir_path = os.getcwd() +'/' +username

if os.path.isdir(new_dir_path) == False:
try:
os.mkdir(new_dir_path)
print ("Successfully created the directory %s " % new_dir_path)
return new_dir_path
except OSError:
print ("Creation of the directory %s failed" % new_dir_path)
else:
return new_dir_path
pass


def list_files(username):
user_dir = os.getcwd() +'/' +username
file_dir_list = next(os.walk(user_dir))[1]
if file_dir_list == []:
response='There are no files yet.'
print(response)
conn.send(response.encode())
else:
file_list = []
for i in range(0, len(file_dir_list)):
file_dir = file_dir_list[i]
file_list.append(os.listdir(user_dir +"/" +file_dir))
if file_list == [[]]:
response='There are no files yet.'
print(response)
conn.send(response.encode())
else:
with open('filenames.txt', 'w') as fh:
for list in file_list:
for file in range(0, len(list)):
fh.write('%s\n' % list[file])
file_names=open('filenames.txt', 'rb').read()
conn.send(file_names)
print('\nSending file names...\n')

os.remove('filenames.txt')
def get(username):

filename = conn.recv(1024).decode()
print('User ' +username +' requested: ' +filename)
```

```
user_dir = os.getcwd() +'/' +username
file_dir = os.path.join(user_dir, filename)
user_dir_filelist = next(os.walk(user_dir))[1]
if user_dir_filelist == []:
response='Your directory has no files yet.\nExiting now...'
print('User directory has no file folders.\nExiting now...')
conn.send(response.encode())
sys.exit()
else:
file_dir_chunklist = next(os.walk(file_dir))[2]

if file_dir_chunklist == []:
response='You do not have any files in the folder yet.\nExiting now...'
print('File folder empty.\nExiting now...')
conn.send(response.encode())
sys.exit()
else:
ct = 0
for chunk in user_dir_filelist:
ct += 1
if ct < len(user_dir_filelist):
if filename == chunk:
response='Server is preparing file transfer...'
print('File found.')
conn.send(response.encode())
time.sleep(1)
break
else:
continue
# if ct == length of list
else:
if filename == chunk:
response='Server is preparing file transfer...'
print('File found.')
conn.send(response.encode())
time.sleep(1)
pass
else:
response='No such file exists.\nExiting now...'
print(response)
conn.send(response.encode())
sys.exit()
name1, name2 = os.listdir(file_dir)
chunk1 = username +'/' +chunk +'/' +name1
chunk2 = username +'/' +chunk +'/' +name2
statinfo=os.stat(chunk1)
buffersize=round(float(statinfo.st_size)) +4
conn.send(str(buffersize).encode())
```

```
time.sleep(1)
chunk1_num = name1.split('_')[1]
chunk2_num = name2.split('_')[1]
if chunk1_num == '1.txt' and chunk2_num == '4.txt':
conn.send(name2.encode())
time.sleep(0.5)
chunk2=open(chunk2,'rb').read()
conn.send(chunk2)
print('Sending chunk 1: ' +name2)
else:
conn.send(name1.encode())
time.sleep(0.5)
chunk1=open(chunk1,'rb').read()
conn.send(chunk1)
print('Sending chunk 1: ' +name1)
FINACK = conn.recv(1024).decode()
if FINACK == 'Transfer incomplete':
if chunk1_num == '1.txt' and chunk2_num == '4.txt':
conn.send(name1.encode())
time.sleep(0.5)
chunk1=open(chunk1,'rb').read()
conn.send(chunk1)
print('Sending chunk 2: ' +name1)
else:
conn.send(name2.encode())
time.sleep(0.5)
chunk2=open(chunk2,'rb').read()
conn.send(chunk2)
print('Sending chunk 2: ' +name2)

FIN = conn.recv(1024).decode()
print(FIN)
# iff FIN, exit (FINACK == 'Transfer successful.')
else:
# print and exit
print(FINACK +'\nExiting now...')
sys.exit()

server_name = '127.0.0.1'
server_port = int(sys.argv[1])
# define socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_name, server_port))
server_socket.listen(5)
print('Server listening...')

while True:
conn, client_address = server_socket.accept()
```

```
print('Connected to Client.')
username = conn.recv(2048)
username = username.decode()
print('received username')
password = conn.recv(2048)
password = password.decode()
print('received password')
auth_params()
client_auth(auth_dict, username, password)
new_dir(username)
command = conn.recv(1024).decode()
print('The user requested to ' +command + ' files.')
# PUT
if command == 'put':
put(new_dir_path)
# LIST
elif command == 'list':
list_files(username)
answer = conn.recv(1024).decode()
print('The user now requests to ' +answer +' files.')

if answer == 'put':
print('Receiving files...')
put(new_dir_path)

elif answer == 'get':
get(username)
else:
print('Exiting now...')
sys.exit()
elif command == 'get':
get(username)
else:
print('Command does not exist.\nExiting now...')
sys.exit()

conn.close()
```

Code for DFS.py:-

```
#! /usr/bin/env python3


import re
import os
```

```
import sys
import glob
import time
import pickle
import socket
import hashlib

def check_args():
if len(sys.argv) != 2:
print("ERROR: Must supply an argument \nUSAGE: py dfc.py dfc.conf")
sys.exit()

elif sys.argv[1].lower() != 'dfc.conf':
print("ERROR: Must supply a valid argument \nUSAGE: py dfc.py dfc.conf")
sys.exit()
elif os.path.isfile(sys.argv[1]) != True:
print("ERROR: dfc.conf not found.")
sys.exit()
else:
return sys.argv[1]
def user_auth():
fh = open('dfc.conf', mode='r', encoding='cp1252')
users=re.findall(r'Username: .*', fh.read())
usernames=list()
for i in range(0, len(users)):
usernames.append(str(users[i]).split()[1])
fh.close()
fh = open('dfc.conf', mode='r', encoding='cp1252')
passes=re.findall(r'Password: .*', fh.read())
passwords=list()
for i in range(0, len(passes)):
passwords.append(str(passes[i]).split()[1])
fh.close()
global auth_dict
auth_dict = {}
for i in range(0, len(users)):
entry={usernames[i]:passwords[i]}
auth_dict.update(entry)
return auth_dict

def authenticate():
user_auth()
auth_status = ''
for i in range(0, 4):
if auth_status == 'Valid username.':
pass
else:
username = input('username: ')
```

```
username_auth = []
ct = 0
for key, value in auth_dict.items():
ct += 1
if username == key:
username_auth.append(ct)
else:
username_auth.append(0)
if i < 2:
if sum(username_auth) > 0:
auth_status = 'Valid username.'
continue
else:
print('Username does not exist. You have ' +str(3-i) + ' attempts left.')
continue
elif i == 2:
if sum(username_auth) > 0:
auth_status = 'Valid username.'
continue
else:
print('Username does not exist. You have ' +str(3-i) + ' attempt left.')
continue
else:
if sum(username_auth) > 0:
auth_status = 'Valid username.'
continue
else:
print('Username does not exist. You have no more attempts.\nExiting now....')
sys.exit()
user_index = sum(username_auth)

auth_status = ''
for i in range(0, 4):
if auth_status == 'Valid password.':
pass
else:
password = input('password: ')
hash=hashlib.md5()
hash.update(password.encode())
password = hash.hexdigest()
password_auth = []
ct = 0
for key, value in auth_dict.items():
ct += 1
if password == value:
password_auth.append(ct)
else:
password_auth.append(0)
```

```
if i < 2:
if sum(password_auth) > 0:

if user_index == sum(password_auth):
auth_status = 'Valid password.'
continue
else:
print('Wrong password. You have ' +str(3-i) + ' attempts left.')
continue
else:
print('Wrong password. You have ' +str(3-i) + ' attempts left.')
continue
elif i == 2:
if sum(password_auth) > 0:
if user_index == sum(password_auth):
auth_status = 'Valid password.'
continue
else:
print('Wrong password. You have ' +str(3-i) + ' attempt left.')
continue
else:
print('Wrong password. You have ' +str(3-i) + ' attempt left.')
continue
else:
if user_index == sum(password_auth):
auth_status = 'Valid password.'
continue
else:
print('Wrong password. You have no more attempts.\nExiting now....')
sys.exit()
print('Authorization Granted.')
global final_authorization
final_authorization = (username, password)
return final_authorization

def server_conf():
fh = open('dfc.conf', mode='r', encoding='cp1252')
params = re.findall(r'DFS.*', fh.read())

s_names = list()
for i in range(0, len(params)):
s_names.append(str(params[i]).split()[1].split(":")[0])
s_ports = list()
for i in range(0, len(params)):
s_ports.append(str(params[i]).split()[1].split(":")[1])

s_names_dict = {}
```

```
for i in range(0, len(params)):
entry={'server' +str(i+1):s_names[i]}
s_names_dict.update(entry)
s_ports_dict = {}
for i in range(0, len(params)):
entry={'server' +str(i+1):s_ports[i]}
s_ports_dict.update(entry)
global server_list
server_list = list()
ct = 0
for i in range(0, len(params)):
ct += 1
server_list.append((s_names_dict['server' +str(ct)],\
int(s_ports_dict['server' + str(ct)])))
return server_list

def split_files(filename, chunksize):

with open(filename + '.txt', 'rb') as bytefile:
content = bytearray(os.path.getsize(filename + '.txt'))
bytefile.readinto(content)
for count, i in enumerate(range(0, len(content), chunksize)):
with open(filename + '_' + str(count+1) + '.txt.', 'wb') as fh:
fh.write(content[i: i + chunksize])

def chunk_pairs(filename):
# group chunks in paired lists # per table:
pair1 = [filename +'_1.txt', filename +'_2.txt'] # 1,2
pair2 = [filename +'_2.txt', filename +'_3.txt'] # 2,3
pair3 = [filename +'_3.txt', filename +'_4.txt'] # 3,4
pair4 = [filename +'_4.txt', filename +'_1.txt'] # 4,1


hash=hashlib.md5()
with open(filename +'.txt', 'rb') as fh:
buffer = fh.read()
hash.update(buffer)
storeval = int(hash.hexdigest(), 16) % 4

if storeval == 0:
dfs1 = pair1
dfs2 = pair2
dfs3 = pair3
dfs4 = pair4
elif storeval == 1:
dfs1 = pair4
dfs2 = pair1
```

```
dfs3 = pair2
dfs4 = pair3
elif storeval == 2:
dfs1 = pair3
dfs2 = pair4
dfs3 = pair1
dfs4 = pair2
else:
dfs1 = pair2
dfs2 = pair3
dfs3 = pair4
dfs4 = pair1
return dfs1, dfs2, dfs3, dfs4

def get_command():

global command
command = ''
for i in range(0, 4):
if command != '':
return command
break
else:
comm = input('Please specify a command [get, list, put]: ')
if i < 2:
if comm.lower() == 'get':
command = 'get'
continue
elif comm.lower() == 'list':
command = 'list'
continue
elif comm.lower() == 'put':
command = 'put'
continue
else:
print('There is no such command. You have ' +str(3-i) + ' attempts left.')
continue
elif i == 2:
if comm.lower() == 'get':
command = 'get'
continue
elif comm.lower() == 'list':
command = 'list'
continue
elif comm.lower() == 'put':
command = 'put'
continue
else:
```

```
print('There is no such command. You have ' +str(3-i) + ' attempt left.')
continue
else:
print('There is no such command. You have no more attempts.\nExiting now....')
sys.exit()
def get_filename():
for i in range(0, 2):
if i == 0:
txtfiles = []
print('Current files: ')
print('-' * 15)
for file in glob.glob("*.txt"):
txtfiles.append(file)
print(file.split(".")[0])
print('\n')
filename = input('Please specify a file: ')
try:
statinfo = os.stat(filename + '.txt')
break
except FileNotFoundError:
print('There is no such file in the directory.\nPlease try again.\n')
continue
else:
txtfiles = []
print('Current files: ')
print('-' * 15)
for file in glob.glob("*.txt"):
txtfiles.append(file)
print(file.split(".")[0])
print('\n')
filename = input('Please specify a file: ')
try:
statinfo = os.stat(filename + '.txt')
except FileNotFoundError:
print('There is no such file in the directory.\nExiting now...')
sys.exit()
global filename_statinfo
filename_statinfo = (filename, statinfo)
return filename_statinfo


def client():
authenticate()
username = final_authorization[0]
password = final_authorization[1]
server_conf()

try:
```

```
client_socket1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket1.connect(server_list[0])
status1 = ('Connected to server', 'DFS1')
print(status1[0], status1[1])
time.sleep(1)
except ConnectionRefusedError:
status1 = ('Could not connect to server', 'DFS1')
print(status1[0], status1[1])
try:
client_socket2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket2.connect(server_list[1])
status2 = ('Connected to server', 'DFS2')
print(status2[0], status2[1])
time.sleep(1)
except ConnectionRefusedError:
status2 = ('Could not connect to server', 'DFS2')
print(status2[0], status2[1])
try:
client_socket3 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket3.connect(server_list[2])
status3 = ('Connected to server', 'DFS3')
print(status3[0], status3[1])
time.sleep(1)
except ConnectionRefusedError:
status3 = ('Could not connect to server', 'DFS3')
print(status3[0], status3[1])
try:
client_socket4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket4.connect(server_list[3])
status4 = ('Connected to server', 'DFS4')
print(status4[0], status4[1])
time.sleep(1)
except ConnectionRefusedError:
status4 = ('Could not connect to server', 'DFS4')
print(status4[0], status4[1])


if status1[0] == 'Could not connect to server' and status2[0] == 'Could not connect
to server' \
and status3[0] == 'Could not connect to server' and status4[0] == 'Could not
connect to server':
print('All servers are down.\nExiting now...')
sys.exit()
else:
pass
conns = (client_socket1, client_socket2, client_socket3, client_socket4)
DFSS = ('DFS1', 'DFS2', 'DFS3', 'DFS4')
```

```
for i in range(0,4):
try:
conns[i].send(username.encode())
time.sleep(1)
except OSError:
pass
for i in range(0,4):
try:
conns[i].send(password.encode())
except OSError:
pass
for i in range(0,4):
try:
response = conns[i].recv(1024)
print('From ' +DFSS[i] +': ' +response.decode())
except OSError:
pass

get_command()
if command.lower() == 'put':
for i in range(0,4):
try:
conns[i].send(command.encode())
except OSError:
pass

get_filename()
filename = filename_statinfo[0]
statinfo = filename_statinfo[1]
filesize = statinfo.st_size
buffersize = round(float(filesize)/4) +4
split_files(filename, buffersize)
dfs1, dfs2, dfs3, dfs4 = chunk_pairs(filename)
dfss = (dfs1, dfs2, dfs3, dfs4)
for i in range(0,4):
try:
conns[i].send(str(buffersize).encode())
except OSError:
pass
for i in range(0,4):
try:
conns[i].send(dfss[i][0].encode())
time.sleep(0.5)
chunk1=open(dfss[i][0], 'rb').read()
conns[i].send(chunk1)
print('\nSending ' +str(dfss[i][0]) +'...\n')
except OSError:
```

```
pass
for i in range(0,4):
try:
response=conns[i].recv(1024).decode()
if response == 'Chunk 1 successfully transferred.\n':
print(DFSS[i] +' Chunk 1 transfer complete.')
else:
print(DFSS[i] +' Chunk 1 transfer failed.')
except OSError:
pass
for i in range(0,4):
try:
conns[i].send(dfss[i][1].encode())
time.sleep(0.5)
chunk2=open(dfss[i][1], 'rb').read()
conns[i].send(chunk2)
print('\nSending ' +str(dfss[i][1]) +'...\n')
except OSError:
pass
for i in range(0,4):
try:
response=conns[i].recv(1024).decode()
if response == 'Chunk 2 successfully transferred.\n':
print(DFSS[i] +' Chunk 2 transfer complete.')
else:
print(DFSS[i] +' Chunk 2 transfer incomplete.')
except OSError:
pass
os.remove(str(dfs1[0]))
os.remove(str(dfs1[1]))
os.remove(str(dfs3[0]))
os.remove(str(dfs3[1]))
print('\nExiting now...')
sys.exit()
# LIST
elif command.lower() == 'list':
for i in range(0,4):
try:
conns[i].send(command.encode())
except OSError:
pass
for i in range(0,4):
try:
file_names=conns[i].recv(4096).decode()
print('\nCurrent ' +DFSS[i] +'\%s files:' %username)
print('-' * 27)
print(file_names)
except OSError:
```

```
pass
print('\nWould you like to get files, put files, or exit?')
answer = input('[get, put, exit]: ')
for i in range(0,4):
try:
conns[i].send(answer.encode())
except OSError:
pass
if answer.lower() == 'put':

get_filename()
filename = filename_statinfo[0]
statinfo = filename_statinfo[1]
filesize = statinfo.st_size
buffersize = round(float(filesize)/4) +4
split_files(filename, buffersize)
dfs1, dfs2, dfs3, dfs4 = chunk_pairs(filename)
dfss = (dfs1, dfs2, dfs3, dfs4)
for i in range(0,4):
try:
conns[i].send(str(buffersize).encode())
except OSError:
pass
for i in range(0,4):
try:
conns[i].send(dfss[i][0].encode())
time.sleep(0.5)
chunk1=open(dfss[i][0], 'rb').read()
conns[i].send(chunk1)
print('\nSending ' +str(dfss[i][0]) +'...\n')
except OSError:
pass
for i in range(0,4):
try:
response=conns[i].recv(1024).decode()
if response == 'Chunk 1 successfully transferred.\n':
print(DFSS[i] +' Chunk 1 transfer complete.')
else:
print(DFSS[i] +' Chunk 1 transfer failed.')
except OSError:
pass
for i in range(0,4):
try:
conns[i].send(dfss[i][1].encode())
time.sleep(0.5)
chunk2=open(dfss[i][1], 'rb').read()
conns[i].send(chunk2)
print('\nSending ' +str(dfss[i][1]) +'...\n')
```

```
except OSError:
pass
for i in range(0,4):
try:
response=conns[i].recv(1024).decode()
if response == 'Chunk 2 successfully transferred.\n':
print(DFSS[i] +' Chunk 2 transfer complete.')
else:
print(DFSS[i] +' Chunk 2 transfer incomplete.')
except OSError:
pass
os.remove(str(dfs1[0]))
os.remove(str(dfs1[1]))
os.remove(str(dfs3[0]))
os.remove(str(dfs3[1]))
print('\nExiting now...')
sys.exit()

elif answer.lower() == 'get':
new_dir_path = os.getcwd() +'/' +username

if os.path.isdir(new_dir_path) == False:
try:
os.mkdir(new_dir_path)
print ("Successfully created the directory %s " % new_dir_path)
pass
except OSError:
print ("Creation of the directory %s failed" % new_dir_path)
else:
pass
filename = input('Please specify a file: ')
for i in range(0,4):
try:
conns[i].send(filename.encode())
except OSError:
pass
for i in range(0,4):
try:
answer=conns[i].recv(1024).decode()
except OSError:
pass
for i in range(0,4):
if answer == 'Server is preparing file transfer...':
try:
buffersize=int(conns[i].recv(1024).decode())
except OSError:
pass
else:
```

```
try:
print(answer)
sys.exit()
except OSError:
pass
chunk_list = []
for i in range(0,4):
try:
name=conns[i].recv(1024).decode()
chunk_list.append(name)
except OSError:
pass
for i in range(0,len(chunk_list)):
try:
chunk1=conns[i].recv(buffersize).decode()
with open(os.path.join(new_dir_path, chunk_list[i]), 'w') as fh:
fh.write(chunk1)
print('File chunks successfully transferred.')
except OSError:
pass
arrived = chunk_list
num_chunks = len(arrived)
if num_chunks < 4:

NACK = 'Transfer incomplete'
print(NACK +'\nOnly ' +str(num_chunks) +' out of 4 chunks arrived.')
for i in range(0,4):
try:
conns[i].send(NACK.encode())
except OSError:
pass

chunk2_list = []
for i in range(0,4):
try:
name2=conns[i].recv(1024).decode()
chunk2_list.append(name2)
except OSError:
pass
print('Receiving second batch...')
for i in range(0,len(chunk2_list)):
try:
chunk2=conns[i].recv(buffersize).decode()
with open(os.path.join(new_dir_path, chunk2_list[i]), 'w') as fh:
fh.write(chunk2)
print('File chunks successfully transferred.')
except OSError:
pass
```

```
arrived2 = os.listdir(new_dir_path)

arrived2_clean = []
for i in range(0, len(arrived2)):
if arrived2[i].split('_')[0] == filename:
arrived2_clean.append(arrived2[i])
else:
pass
arrived2_intlist = []
for i in range(0,len(arrived2_clean)):
arrived2_intlist.append(int(arrived2_clean[i].split('_')[1].split('.')[0]))

if arrived2_intlist == [1,2,3,4]:
print('Chunks 1 through 4 are present.')
FIN = 'Transfer successful.'
for i in range(0,4):
try:
conns[i].send(FIN.encode())
except OSError:
pass
final_filename = arrived2_clean[0].split('_')[0] +'.txt'
with open(username +'/' +final_filename, 'wb') as outfile:
for chunk_name in arrived2_clean:
with open(username +'/' +chunk_name, 'rb') as infile:
outfile.write(infile.read())
print('File successfully reconstructed.')
for i in range(0,len(arrived2_clean)):
try:
os.remove(str(username +'/' +arrived2_clean[i]))
except IndexError:
pass
print('Exiting now...')
sys.exit()
else:

FIN = 'Transfer failed.\nExiting now...'
for i in range(0,4):
try:
conns[i].send(FIN.encode())
except OSError:
pass
print(FIN)
sys.exit()
else:
print('A total of ' +str(num_chunks) +' chunks arrived.')
arrived_ordered = []
for i in range(0,4):
arrived_ordered.append(int(arrived[i].split('_')[1].split('.')[0]))
```

```
arrived_ordered.sort()
if arrived_ordered == [1,2,3,4]:
print('All four chunks are present.')
FIN = 'Transfer successful.'
for i in range(0,4):
try:
conns[i].send(FIN.encode())
except OSError:
pass
chunk_list.sort()
final_filename = chunk_list[0].split('_')[0] +'.txt'
with open(username +'/' +final_filename, 'wb') as outfile:
for chunk_name in chunk_list:
with open(username +'/' +chunk_name, 'rb') as infile:
outfile.write(infile.read())
print('File successfully reconstructed.')
for i in range(0,4):
try:
os.remove(str(username +'/' +chunk_list[i]))
except IndexError:
pass
sys.exit()
else:
# if the ordered list is not [1,2,3,4]
FIN = 'Transfer failed.\Exiting now...'
for i in range(0,4):
try:
conns[i].send(FIN.encode())
except OSError:
pass
print(FIN)
sys.exit()
elif answer.lower() == 'exit':
print('Exiting now...')
sys.exit()
else:
print('This method does not exist.\nExiting now...')
sys.exit()

else:
for i in range(0,4):
try:
conns[i].send(command.encode())
except OSError:
pass
new_dir_path = os.getcwd() +'/' +username

if os.path.isdir(new_dir_path) == False:
```

```
try:
os.mkdir(new_dir_path)
print ("Successfully created the directory %s " % new_dir_path)
pass
except OSError:
print ("Creation of the directory %s failed" % new_dir_path)
else:
pass
filename = input('Please specify a file: ')
for i in range(0,4):
try:
conns[i].send(filename.encode())
except OSError:
pass
for i in range(0,4):
try:
answer=conns[i].recv(1024).decode()
except OSError:
pass
for i in range(0,4):
if answer == 'Server is preparing file transfer...':
try:
buffersize=int(conns[i].recv(1024).decode())
print(answer)
except OSError:
pass

else:
try:
print(answer)
sys.exit()
except OSError:
pass
chunk_list = []
for i in range(0,4):
try:
name=conns[i].recv(1024).decode()
chunk_list.append(name)
except OSError:
pass
for i in range(0,len(chunk_list)):
try:
chunk1=conns[i].recv(buffersize).decode()
with open(os.path.join(new_dir_path, chunk_list[i]), 'w') as fh:
fh.write(chunk1)
print('File chunks successfully transferred.')
except OSError:
pass
```

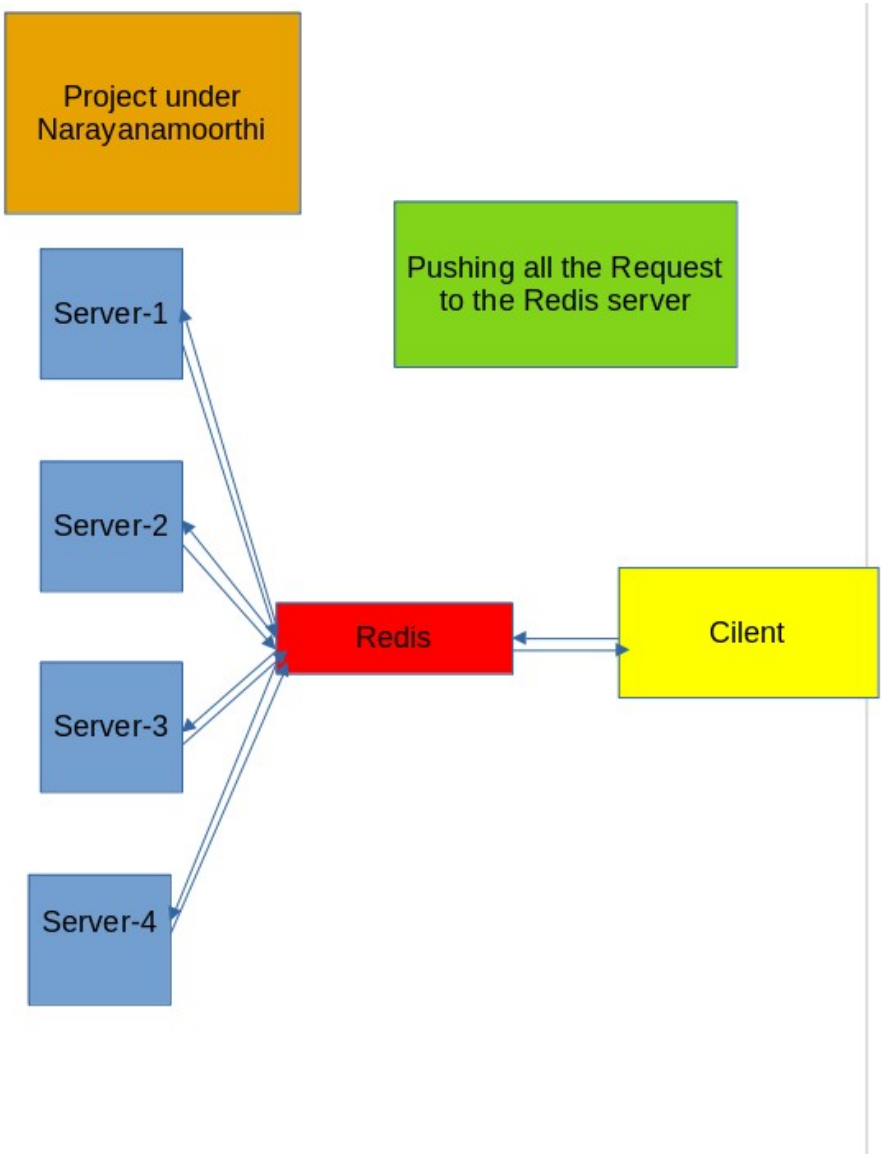
```
arrived = chunk_list
num_chunks = len(arrived)
if num_chunks < 4:

NACK = 'Transfer incomplete'
print(NACK +'\nOnly ' +str(num_chunks) +' out of 4 chunks arrived.')
for i in range(0,4):
try:
conns[i].send(NACK.encode())
except OSError:
pass

chunk2_list = []
for i in range(0,4):
try:
name2=conns[i].recv(1024).decode()
chunk2_list.append(name2)
except OSError:
pass
print('Receiving second batch...')
for i in range(0,len(chunk2_list)):
try:
chunk2=conns[i].recv(buffersize).decode()
with open(os.path.join(new_dir_path, chunk2_list[i]), 'w') as fh:
fh.write(chunk2)
print('File chunks successfully transferred.')
except OSError:
pass
arrived2 = os.listdir(new_dir_path)

arrived2_clean = []
for i in range(0, len(arrived2)):
if arrived2[i].split('_')[0] == filename:
arrived2_clean.append(arrived2[i])
else:
pass
arrived2_intlist = []
for i in range(0,len(arrived2_clean)):
arrived2_intlist.append(int(arrived2_clean[i].split('_')[1].split('.')[0]))

if arrived2_intlist == [1,2,3,4]:
print('Chunks 1 through 4 are present.')
FIN = 'Transfer successful.'
for i in range(0,4):
try:
conns[i].send(FIN.encode())
```

```
except OSError:
pass
final_filename = arrived2_clean[0].split('_')[0] +'.txt'
with open(username +'/' +final_filename, 'wb') as outfile:
for chunk_name in arrived2_clean:
with open(username +'/' +chunk_name, 'rb') as infile:
outfile.write(infile.read())
print('File successfully reconstructed.')
for i in range(0,len(arrived2_clean)):
try:
os.remove(str(username +'/' +arrived2_clean[i]))
except IndexError:
pass
print('Exiting now...')
sys.exit()
else:

FIN = 'Transfer failed.\nExiting now...'
for i in range(0,4):
try:
conns[i].send(FIN.encode())
except OSError:
pass
print(FIN)
sys.exit()
else:
print('A total of ' +str(num_chunks) +' chunks arrived.')
arrived_ordered = []
for i in range(0,4):
arrived_ordered.append(int(arrived[i].split('_')[1].split('.')[0]))
arrived_ordered.sort()
if arrived_ordered == [1,2,3,4]:
print('All four chunks are present.')
FIN = 'Transfer successful.'
for i in range(0,4):
try:
conns[i].send(FIN.encode())
except OSError:
pass
chunk_list.sort()
final_filename = chunk_list[0].split('_')[0] +'.txt'
with open(username +'/' +final_filename, 'wb') as outfile:
for chunk_name in chunk_list:
with open(username +'/' +chunk_name, 'rb') as infile:
outfile.write(infile.read())
print('File successfully reconstructed.')
for i in range(0,4):
try:
```

```
os.remove(str(username +'/' +chunk_list[i]))
except IndexError:
pass
print('Exiting now...')
sys.exit()
else:
FIN = 'Transfer failed.\Exiting now...'
for i in range(0,4):
try:
conns[i].send(FIN.encode())
except OSError:
pass
print(FIN)
sys.exit()
if __name__ =='__main__':
check_args()
client()
```

6. Conculsion:-

As Disscussed above the above Architecture works in a Blocking way which for large files may delay the user of resources.Which can be imporvised by adding a Queing Channels Like Redis or Rabbit MQ where the client puhsed the chunk of file into the redis server and does some other work but in this case of the server to it will accpet the request when ever it is free to execute the process request from reedis;-



7. References:-

Amalraj, A. J., & Jose, J. J. R. (2016). A survey paper on cryptography techniques.International Journal of Computer Science and Mobile Computing, 5(8), 55-59.

7.a.i. Mahajan, P., & Sachdeva, A. (2013). A study of Distributed File System, MD5 for security. Global Journal of Computer Science and Technology.

7.a.ii. Venkat Prasad K., & Magesh S. (2015). A SURVEY ON ENCRYPTION ALGORITHMS USING MODERN TECHNIQUES. International Journal of Pure and Applied Mathematics

7.a.iii. Alemami, Y., Mohamed, M. A., & Atiewi, S. Research on Various Cryptography Techniques. Yang, C. C., Chang, T. S., & Jen, C. W. (1998). A new RSA cryptosystem hardware design based on Montgomery's algorithm. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 45(7), 908-913.

• Huang, X., & Wang, W. (2015). A novel and efficient design for an Distributed System Ecosystem with a very Big servers. IEEE Transactions on Circuits and Systems II: Express Briefs, 62(10), 972-976.

7.a.iv. Chang, C. C., & Hwang, M. S. (1996). Parallel computation of the generating keys for RSA cryptosystems. Electronics Letters, 32(15), 1365-1366.

7.a.v. Rami Aldahdooh. Parallel Implementation and Analysis of Distributed File System

7.a.vi. Pachori, V., Ansari, G., & Chaudhary, N. (2012). Improved performance of advance encryption standard using parallel computing. International Journal of Engine ATTAR, N., DELDARI, H., & KALANTARI, M. (2017). AES ENCRYPTION ALGORITHM PARALLELIZATION IN ORDER TO USE BIG DATA CLOUDx