实验二 Hadoop 并行编程

一、Hadoop DFS常用指令

1. Hadoop实验方法

```
2021214316@thumm01:~$ hadoop fs
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] <localsrc> ... <dst>]
```

```
2021214316@thumm01:/home/dsjxtjc$ hadoop fs -ls /
Found 2 items
drwxr-xr-x   - root  supergroup          0 2021-10-05 12:42 /dsjxtjc
drwxrwxrwx   - jtliu supergroup          0 2020-12-21 23:25 /tmp
```

```
2021214316@thumm01:~$ hadoop fs -cat /dsjxtjc/2021214316/test.txt
2021-10-19 20:19:41,407 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
Hello Hadoop
```

```
2021214316@thumm01:~$ hadoop fs -help copyFromLocal
-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] <localsrc> ... <dst> :
  Copy files from the local file system into fs. Copying fails if the file already
  exists, unless the -f flag is given.
  Flags:

  -p                  Preserves access and modification times, ownership and the
                      mode.
  -f                  Overwrites the destination if it already exists.
  -t <thread count>  Number of threads to be used, default is 1.
  -l                  Allow DataNode to lazily persist the file to disk. Forces
                      replication factor of 1. This flag will result in reduced
                      durability. Use with care.
  -d                  Skip creation of temporary file(<dst>._COPYING_).
```

2. 通过Web 查看Hadoop 运行情况

```
✘ xueyuan@XueY ~ ssh 2021214316@10.103.9.11 -L 9870:192.168.0.101:9870
2021214316@10.103.9.11's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-210-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

70 个可升级软件包。
2 个安全更新。

New release '18.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


*** 需要重启系统 ***
Last login: Tue Oct 19 19:54:07 2021 from 10.97.135.16
```

Overview 'thumm01:9000' (active)

Started:	Mon Oct 11 19:55:35 +0800 2021
Version:	3.2.1, rb3cbbb467e22ea829b3808f4b7b01d07e0bf3842
Compiled:	Tue Sep 10 23:56:00 +0800 2019 by rohithsharmaks from branch-3.2.1
Cluster ID:	CID-918a6ae8-dd65-4451-9fb8-ddfd885bdbde
Block Pool ID:	BP-495645297-192.168.0.101-1604038182293

二、分布式文件系统

1. copyFromLocal

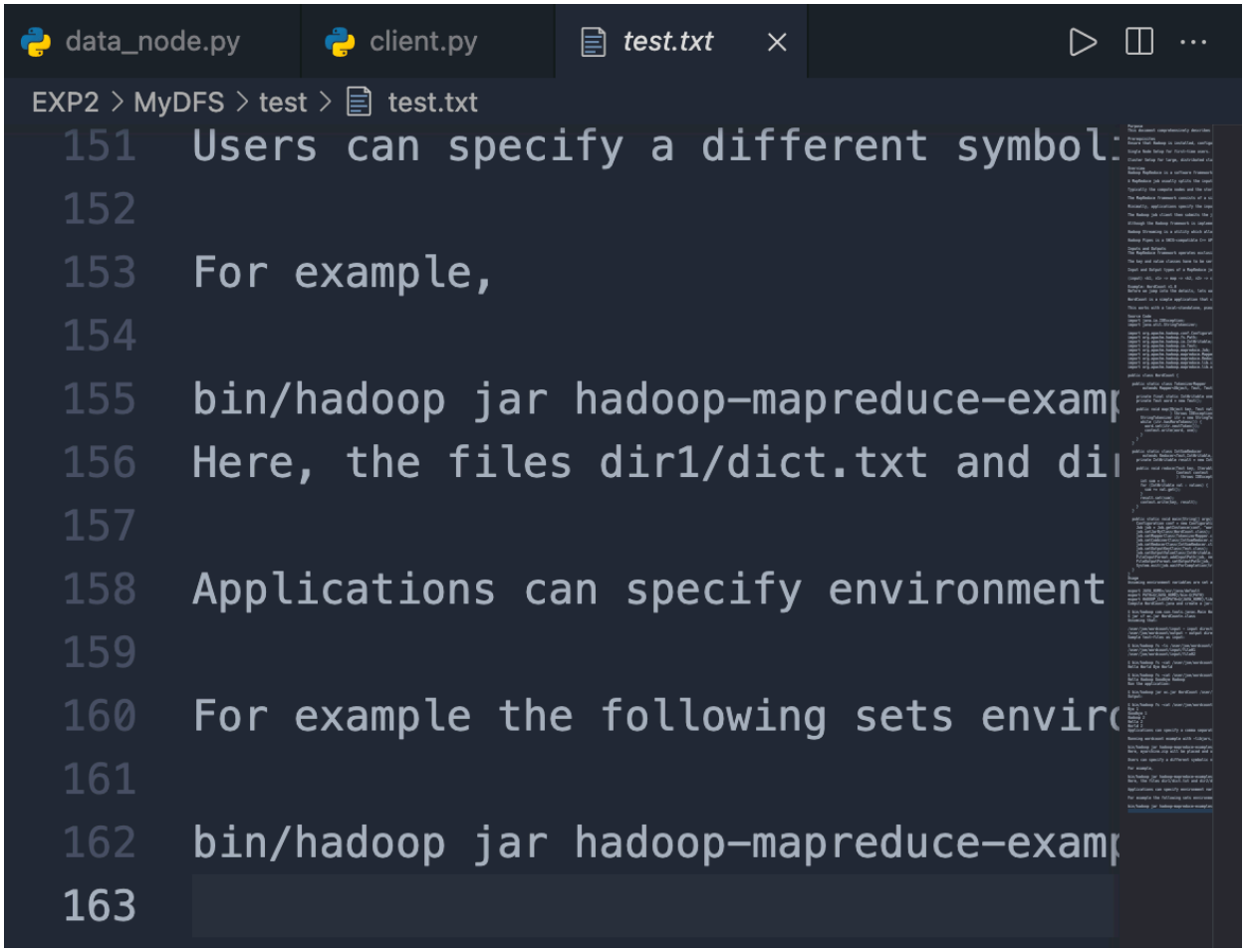
文档给出范例，略。

2. copyToLocal

```
    def copyToLocal(self, dfs_path, local_path):
        request = "get_fat_item {}".format(dfs_path)
        print("Request: {}".format(request))
        # TODO: 从NameNode获取一张FAT表；打印FAT表；根据FAT表逐个从目标DataNode请求数据块，写入到本地文件中
```

```
        self.name_node_sock.send(bytes(request, encoding='utf-8'))
        fat_pd = self.name_node_sock.recv(BUF_SIZE)

        # 打印FAT表，并使用pandas读取
        fat_pd = str(fat_pd, encoding='utf-8')
        print("Fat: \n{}".format(fat_pd))
        fat = pd.read_csv(StringIO(fat_pd))

        # 根据FAT表逐个向目标DataNode发送数据块
        fp = open(local_path,'w')
        for idx, row in fat.iterrows():
            data_node_sock = socket.socket()
            data_node_sock.connect((row['host_name'], data_node_port))

            blk_path = dfs_path + ".blk{}".format(row['blk_no'])
            request = "load {}".format(blk_path)
            data_node_sock.send(bytes(request, encoding='utf-8'))
            time.sleep(0.2)
            # 两次传输需要间隔一段时间，避免粘包
            data = data_node_sock.recv(BUF_SIZE)
            data = str(data, encoding='utf-8')
            fp.write(data)
            data_node_sock.close()
        fp.close()
```

```
2021214316@thumm01:~/EXP2/MyDFS/test$ ls -all
total 20
drwxr-xr-x 2 2021214316 dsjxtjc 4096 10月  19 22:02 .
drwxr-xr-x 6 2021214316 dsjxtjc 4096 10月  19 22:25 ..
-rw-r--r-- 1 2021214316 dsjxtjc 8411 10月  20 13:52 test.txt
2021214316@thumm01:~/EXP2/MyDFS/test$
```



3. ls

```
    def ls(self, dfs_path):
        try:
            # TODO: 向NameNode发送请求，查看dfs_path下文件或者文件夹信息
            request = "ls {}".format(dfs_path)
            self.name_node_sock.send(bytes(request, encoding='utf-8'))
            response = self.name_node_sock.recv(BUF_SIZE)
            print(response)
        except Exception as e:
            print(e)
```

```
2021214316@thumm01:~/EXP2/MyDFS$ python3 client.py -ls /test/test.txt
b'blk_no,host_name,blk_size\n0,localhost,4096\n1,localhost,4096\n2,localhost,219
\n'
2021214316@thumm01:~/EXP2/MyDFS$
```

```
2021214316@thumm01:~/EXP2/MyDFS$ python3 name_node.py
Name node started
connected by ('127.0.0.1', 35474)
Request: ['ls', '/test/test.txt']
Response: blk_no,host_name,blk_size
0,localhost,4096
1,localhost,4096
2,localhost,219
```

4. rm

```
    def rm(self, dfs_path):
        request = "rm_fat_item {}".format(dfs_path)
        print("Request: {}".format(request))

        # 从NameNode获取改文件的FAT表，获取后删除
        self.name_node_sock.send(bytes(request, encoding='utf-8'))
        fat_pd = self.name_node_sock.recv(BUF_SIZE)

        # 打印FAT表，并使用pandas读取
        fat_pd = str(fat_pd, encoding='utf-8')
        print("Fat: \n{}".format(fat_pd))
        fat = pd.read_csv(StringIO(fat_pd))

        # 根据FAT表逐个向目标DataNode发送要删的数据块
        for idx, row in fat.iterrows():
            data_node_sock = socket.socket()
            data_node_sock.connect((row['host_name'], data_node_port))
            blk_path = dfs_path + ".blk{}".format(row['blk_no'])

            request = "rm {}".format(blk_path)
            data_node_sock.send(bytes(request, encoding='utf-8'))
            time.sleep(0.2)  # 两次传输需要间隔一s段时间，避免粘包
            response = data_node_sock.recv(BUF_SIZE)
            print(response)
            data_node_sock.close()
```

```
2021214316@thumm01:~/EXP2/MyDFS$ python3 client.py -rm /test/test.txt
Request: rm_fat_item /test/test.txt
Fat:
blk_no,host_name,blk_size
0,localhost,4096
1,localhost,4096
2,localhost,219

blk_no,host_name,blk_size
0,localhost,4096
1,localhost,4096
2,localhost,219

b'Remove chunk ./dfs/data/test/test.txt.blk0 successfully~'
b'Remove chunk ./dfs/data/test/test.txt.blk1 successfully~'
b'Remove chunk ./dfs/data/test/test.txt.blk2 successfully~'
```

5. data replication

name_node.py

```
    def new_fat_item(self, dfs_path, file_size):
        nb_blks = int(math.ceil(file_size / dfs_blk_size))
        print(file_size, nb_blks)

        # todo 如果dfs_replication为复数时可以新增host_name的数目
        data_pd = pd.DataFrame(columns=['blk_no', 'host_name', 'blk_size'])
        num_row = 0
        for i in range(nb_blks):
            blk_no = i
            host_name = np.random.choice(host_list, size=dfs_replication,
replace=False)
            blk_size = min(dfs_blk_size, file_size - i * dfs_blk_size)
            for j in host_name:
                data_pd.loc[num_row] = [blk_no, j, blk_size]
                num_row += 1
        # 获取本地路径
        local_path = name_node_dir + dfs_path
        # 若目录不存在则创建新目录
        os.system("mkdir -p {}".format(os.path.dirname(local_path)))
        # 保存FAT表为CSV文件
        data_pd.to_csv(local_path, index=False)
        # 同时返回CSV内容到请求节点
        return data_pd.to_csv(index=False)
```

client.py

```
    def copyFromLocal(self, local_path, dfs_path):
```

```
        file_size = os.path.getsize(local_path)
        print("File size: {}".format(file_size))

        request = "new_fat_item {} {}".format(dfs_path, file_size)
        print("Request: {}".format(request))

        # 从NameNode获取一张FAT表
        self.name_node_sock.send(bytes(request, encoding='utf-8'))
        fat_pd = self.name_node_sock.recv(BUF_SIZE)

        # 打印FAT表，并使用pandas读取
        fat_pd = str(fat_pd, encoding='utf-8')
        print("Fat: \n{}".format(fat_pd))
        fat = pd.read_csv(StringIO(fat_pd))


        # 根据FAT表逐个向目标DataNode发送数据块
        fp = open(local_path)
        blk_no = 0
        data = None
        for idx, row in fat.iterrows():
            if idx == 0 or row['blk_no'] != blk_no:
                data = fp.read(int(row['blk_size']))
                blk_no = row['blk_no']

            data_node_sock = socket.socket()
            data_node_sock.connect((row['host_name'], data_node_port))
            blk_path = dfs_path + ".blk{}".format(row['blk_no'])

            request = "store {}".format(blk_path)
            data_node_sock.send(bytes(request, encoding='utf-8'))
            time.sleep(0.2)  # 两次传输需要间隔一s段时间，避免粘包
            data_node_sock.send(bytes(data, encoding='utf-8'))
            data_node_sock.close()
        fp.close()
```

取host_list为五个节点，replication设为3

thumm01-client

```
2021214316@thumm01:~/EXP2/MyDFS$ python3 client.py -copyFromLocal test/test.txt /test.txt
File size: 8411
Request: new_fat_item /test.txt 8411
Fat:
blk_no,host_name,blk_size
0,thumm03,4096
0,thumm04,4096
0,thumm02,4096
1,thumm02,4096
1,thumm04,4096
1,thumm05,4096
2,thumm01,219
2,thumm04,219
2,thumm05,219
```

thumm01-name node

```
2021214316@thumm01:~/EXP2/MyDFS$ ls
client.py  common.py  data_node.py  dfs  name_node.py  __pycache__  test
2021214316@thumm01:~/EXP2/MyDFS$ python3 name_node.py
Name node started
connected by ('127.0.0.1', 43150)
Request: ['new_fat_item', '/test.txt', '8411']
8411 3
Response: blk_no,host_name,blk_size
0,thumm03,4096
0,thumm04,4096
0,thumm02,4096
1,thumm02,4096
1,thumm04,4096
1,thumm05,4096
2,thumm01,219
2,thumm04,219
2,thumm05,219
```

thumm01-data node

```
^C2021214316@thumm01:~/EXP2/MyDFS$ python3 data_node.py
clear
Received request from ('192.168.0.101', 38982)
['store', '/test.txt.blk2']
store
Store chunk ./dfs/data/test.txt.blk2 successfully~
```

thumm02-data node

```
2021214316@thumm02:~/EXP2/MyDFS$ python3 data_node.py
Received request from ('192.168.0.101', 53322)
['store', '/test.txt.blk0']
store
Store chunk ./dfs/data/test.txt.blk0 successfully~
Received request from ('192.168.0.101', 53324)
['store', '/test.txt.blk1']
store
Store chunk ./dfs/data/test.txt.blk1 successfully~
```

thumm03-data node

```
2021214316@thumm03:~/EXP2/MyDFS$ python3 data_node.py
Received request from ('192.168.0.101', 40616)
['store', '/test.txt.blk0']
store
Store chunk ./dfs/data/test.txt.blk0 successfully~
```

thumm04-data node

```
2021214316@thumm04:~/EXP2/MyDFS$ python3 data_node.py
Received request from ('192.168.0.101', 49596)
['store', '/test.txt.blk0']
store
Store chunk ./dfs/data/test.txt.blk0 successfully~
Received request from ('192.168.0.101', 49602)
['store', '/test.txt.blk1']
store
Store chunk ./dfs/data/test.txt.blk1 successfully~
Received request from ('192.168.0.101', 49608)
['store', '/test.txt.blk2']
store
Store chunk ./dfs/data/test.txt.blk2 successfully~
```

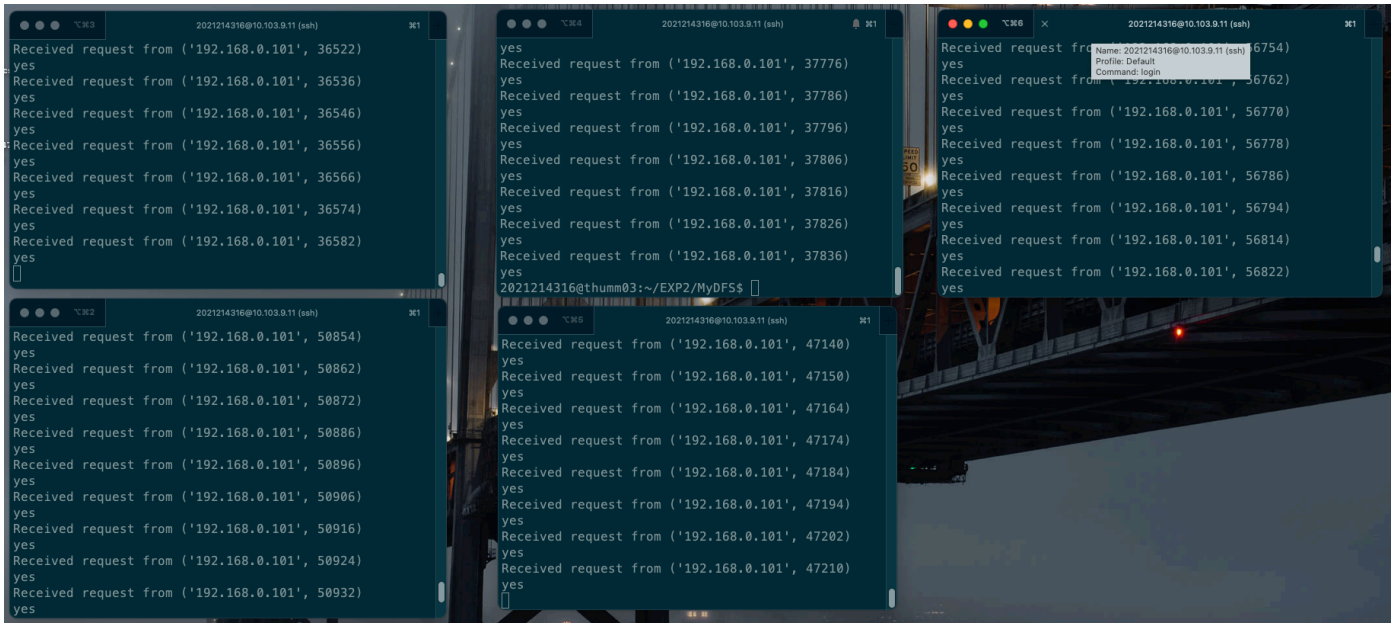
thumm05-data node

```
2021214316@thumm05:~/EXP2/MyDFS$ python3 data_node.py
Received request from ('192.168.0.101', 59284)
['store', '/test.txt.blk1']
store
Store chunk ./dfs/data/test.txt.blk1 successfully~
Received request from ('192.168.0.101', 59290)
['store', '/test.txt.blk2']
store
Store chunk ./dfs/data/test.txt.blk2 successfully~
```

6. HeartBeat

定期交流：namenode主动向所有host的datanode发送信息，如果收到回应，则说明datanode运行正常，如果socket连接失败，说明datanode挂了。

```
2021214316@thumm01:~/EXP2/MyDFS$ python3 name_node.py
Name node started
b'yes'
b'yes'
b'yes'
b'yes'
b'yes'
b'yes'
```



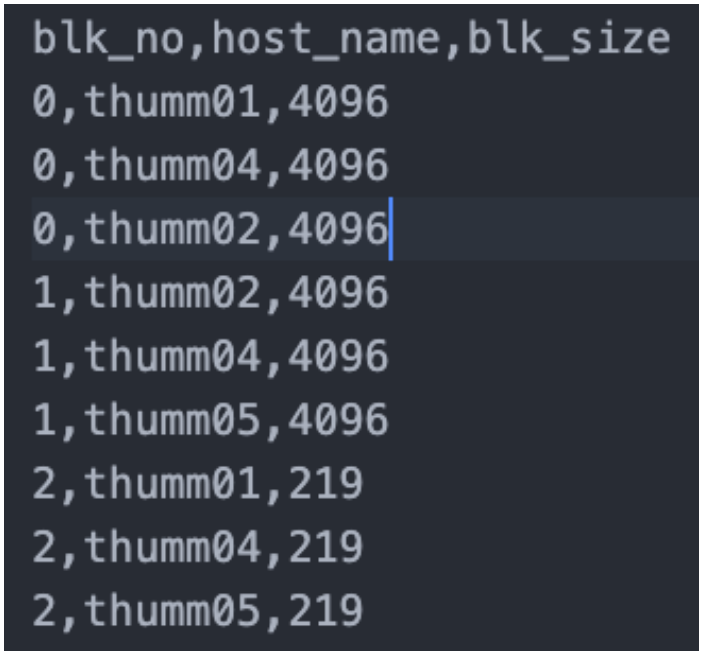
手动结束thumm03节点后(ctrl+c)，显示如下

```
b'yes'
b'yes'
b'yes'
b'yes'
b'yes'
b'yes'
[Errno 111] Connection refused
```

接着对挂掉的节点，查本地的FAT 表，从有数据块的节点down该块，并选择一个活着的节点存储该数据块，实现备份功能。根据下图所示FAT表，thumm03仅有0号数据块，在其挂掉后被备份在thumm01内（随机选择的），并修改FAT表。

```
blk_no,host_name,blk_size
0,thumm03,4096
0,thumm04,4096
0,thumm02,4096
1,thumm02,4096
1,thumm04,4096
1,thumm05,4096
2,thumm01,219
2,thumm04,219
2,thumm05,219
```

```
test.txt.blk0                                 100% 4100     4.0KB/s   00:00
test.txt.blk0                                 100% 4100     4.0KB/s   00:00
test.txt.blk0                                 100% 4100     4.0KB/s   00:00
thumm03 is dead. Data are backed up in ['thumm01']
b'yes'
b'yes'
b'yes'
b'yes'
```



namenode代码吗，注意这里使用了Thread进行多线程运行，Thread1用于namenode与client的交互，Thread2用于HeartBeat的实现。

```
'''
......
'''

def name_node_run():
    # 创建NameNode并启动
    name_node = NameNode()
    name_node.run()

def heart_beat():
    dead_host_list=[]
    live_host_list= host_list
    while True:
        for host in live_host_list:
            try:
                sock = socket.socket()
                sock.connect((host, data_node_port))
                request = "Still alive?"
                sock.send(bytes(request, encoding='utf-8'))
                ans = sock.recv(4096)
                print(ans)
                continue
            except Exception as e:  # data_node挂掉后对丢失的文件做备份
                pass
                # print(e)
            fat_path = "~/EXP2/MyDFS/dfs/name/test.txt"
            fat = pd.read_csv(fat_path)
            for idx, row in fat.iterrows():
                if row['host_name'] == host:  # 找到挂掉的host所在行
```

```
                    blk = row['blk_no']  # 挂掉的host存的块号
                    # 再遍历一遍，找到相同的块号且活着的节点，把该块数据down到本地
                    store_blk = [host]  # 代表存blk数据的host有哪些
                    for idx2, row2 in fat.iterrows():
                        if row2['blk_no'] == blk and row2['host_name'] != host:
                            alive_host = row2['host_name']
                            store_blk.append(alive_host)
                            down_cmd = "scp -r
"+str(alive_host)+":~/EXP2/MyDFS/dfs/data/test.txt.blk"+str(blk) \
                                + " ~/EXP2/MyDFS/dfs/data/temp/"
                            os.system(down_cmd)
                    # 把数据发到活着的节点
                    back_up_host = np.random.choice([x for x in live_host_list if x not
in store_blk],
                                    size=1, replace=False)
                    up_cmd = "scp -r
"+"~/EXP2/MyDFS/dfs/data/temp/test.txt.blk"+str(blk)+" "\
                        +str(back_up_host[0])+":~/EXP2/MyDFS/dfs/data/"
                    os.system(up_cmd)
                    # 修改FAT表这一行，并更新本地存储
                    row['host_name'] = back_up_host[0]
                    fat.to_csv("~/EXP2/MyDFS/dfs/name/heartbeat_fat.txt", index=False)
            dead_host_list.append(host)
            print(host,"is dead. Data are backed up in",back_up_host)
        # 把dead的节点从host_list里删除，并开始下一轮HeartBeat
        live_host_list = [x for x in live_host_list if x not in dead_host_list]
        time.sleep(0.2)



thread1 = threading.Thread(target=name_node_run)
thread2 = threading.Thread(target=heart_beat)

thread1.start()
thread2.start()
```

datanode代码

```
class DataNode:
    def run(self):
        # 创建一个监听的socket
        listen_fd = socket.socket()
        try:
            # 监听端口
            listen_fd.bind(("0.0.0.0", data_node_port))
            listen_fd.listen(5)
            while True:
                # 等待连接，连接后返回通信用的套接字
                sock_fd, addr = listen_fd.accept()
```

```
            print("Received request from {}".format(addr))

            try:
                # 获取请求方发送的指令
                request = str(sock_fd.recv(BUF_SIZE), encoding='utf-8')

                # 如果指令是heartbeat
                if request == "Still alive?":
                    response = "yes"
                else:
                    request = request.split()  # 指令之间使用空白符分割
                    print(request)

                    cmd = request[0]  # 指令第一个为指令类型
                    print(cmd)


                    if cmd == "load":  # 加载数据块
                        dfs_path = request[1]  # 指令第二个参数为DFS目标地址
                        response = self.load(dfs_path)
                    elif cmd == "store":  # 存储数据块
                        dfs_path = request[1]  # 指令第二个参数为DFS目标地址
                        response = self.store(sock_fd, dfs_path)
                    elif cmd == "rm":  # 删除数据块
                        dfs_path = request[1]  # 指令第二个参数为DFS目标地址
                        response = self.rm(dfs_path)
                    elif cmd == "format":  # 格式化DFS
                        response = self.format()
                    else:
                        response = "Undefined command: " + " ".join(request)
                print(response)
                sock_fd.send(bytes(response, encoding='utf-8'))
            except KeyboardInterrupt:
                break
            finally:
                sock_fd.close()
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(e)
    finally:
        listen_fd.close()
```

6.1 Bonus

实现功能：保证操作的原子性，例如name_node生成FAT表后，数据块未全部传输完成，此时data_node挂掉，因此原始数据未被完全传完，此时需要返回相应的报错信息，并清除已传输的部分（调用format功能）。client.py的关键代码如下，以copyFromLocal为例。

```
    def copyFromLocal(self, local_path, dfs_path):
        file_size = os.path.getsize(local_path)
        print("File size: {}".format(file_size))

        request = "new_fat_item {} {}".format(dfs_path, file_size)
        print("Request: {}".format(request))

        # 从NameNode获取一张FAT表
        self.name_node_sock.send(bytes(request, encoding='utf-8'))
        fat_pd = self.name_node_sock.recv(BUF_SIZE)

        # 打印FAT表，并使用pandas读取
        fat_pd = str(fat_pd, encoding='utf-8')
        print("Fat: \n{}".format(fat_pd))
        fat = pd.read_csv(StringIO(fat_pd))


        # 根据FAT表逐个向目标DataNode发送数据块
        fp = open(local_path)
        blk_no = 0
        data = None
        for idx, row in fat.iterrows():
            try:
                if idx == 0 or row['blk_no'] != blk_no:
                    data = fp.read(int(row['blk_size']))
                    blk_no = row['blk_no']
                data_node_sock = socket.socket()
                data_node_sock.connect((row['host_name'], data_node_port))
                blk_path = dfs_path + ".blk{}".format(row['blk_no'])

                request = "store {}".format(blk_path)
                data_node_sock.send(bytes(request, encoding='utf-8'))
                time.sleep(0.2)  # 两次传输需要间隔一s段时间，避免粘包
                data_node_sock.send(bytes(data, encoding='utf-8'))
                data_node_sock.close()
                continue
            except Exception as e:
                pass
            self.format()
        fp.close()
```

三、MapReduce

1. 构造数据集。这里构造了10k个整数

```
with open('/Users/xueyuan/Desktop/data.txt', 'a') as f:
    for i in range(10000):
        data = random.randint(1,1000)
        f.write(str(data)+'\n')
```

2. 本地利用pandas库读取，并计算其均值和方差

```
data = pd.read_csv('/Users/xueyuan/Desktop/data.txt', names=['rdm_data'])
# print(data)
print(data['rdm_data'].mean())
print(data['rdm_data'].var())
```

计算结果为

```
xueyuan@XueY ~ /usr/local/bin/python3 /Users/xueyuan/Desktop/draft.py
500.291
84465.51547054705
```

3. 将数据集传到MyDFS，生成FAT表

```
2021214316@thumm01:~/EXP2/MapReduce$ python3 client.py -copyFromLocal ./data.txt /FAT.txt
File size: 38887
Request: new_fat_item /FAT.txt 38887
Fat:
blk_no,host_name,blk_size
0,thumm01,4096
0,thumm05,4096
0,thumm02,4096
1,thumm05,4096
1,thumm04,4096
1,thumm01,4096
2,thumm05,4096
2,thumm03,4096
2,thumm02,4096
3,thumm04,4096
3,thumm01,4096
3,thumm05,4096
4,thumm01,4096
4,thumm04,4096
4,thumm03,4096
5,thumm04,4096
5,thumm02,4096
5,thumm01,4096
6,thumm04,4096
6,thumm03,4096
6,thumm02,4096
7,thumm02,4096
7,thumm03,4096
7,thumm05,4096
8,thumm01,4096
8,thumm05,4096
8,thumm02,4096
9,thumm02,2023
9,thumm04,2023
9,thumm05,2023
```

4. 实现reducer.py

```
import os
import socket
import time
from io import StringIO

import pandas as pd

from common import *

class Reducer:
    def run(self):
        # 创建一个监听的socket
        listen_fd = socket.socket()
        try:
            # 监听端口
            listen_fd.bind(("0.0.0.0", reducer_port))
            listen_fd.listen(5)
            while True:
                # 等待连接，连接后返回通信用的套接字
                sock_fd, addr = listen_fd.accept()
                print("Received request from {}".format(addr))
                try:
                    # 获取请求方发送的指令
                    request = str(sock_fd.recv(BUF_SIZE), encoding='utf-8')
                    request = request.split()  # 指令之间使用空白符分割
                    if len(request) != 2:
                        print("Invalid command: "+ " ".join(request))
                    else:
                        print(request)

                        host_name = request[0]  # 指令第一个为指令类型
                        blk_no = request[1]

                        # 根据从cliet.py收到的host_name和blk_no，去对应host的datanode里
请求数据
                        data_node_sock = socket.socket()
                        data_node_sock.connect((host_name, data_node_port))

                        blk_path = "/FAT.txt.blk{}".format(blk_no)
                        request = "load {}".format(blk_path)
                        data_node_sock.send(bytes(request, encoding='utf-8'))
                        time.sleep(0.2)  # 两次传输需要间隔一段时间，避免粘包
                        # 收到该块数据
                        data = data_node_sock.recv(BUF_SIZE)
                        data = str(data, encoding='utf-8')
                        with open('./temp_data.txt','w') as f:  # 暂存下来再读取，属于
是不会怎么把data读成pd了
                            f.write(data)
                        data = pd.read_csv('./temp_data.txt', names=['rdm_data'])
```

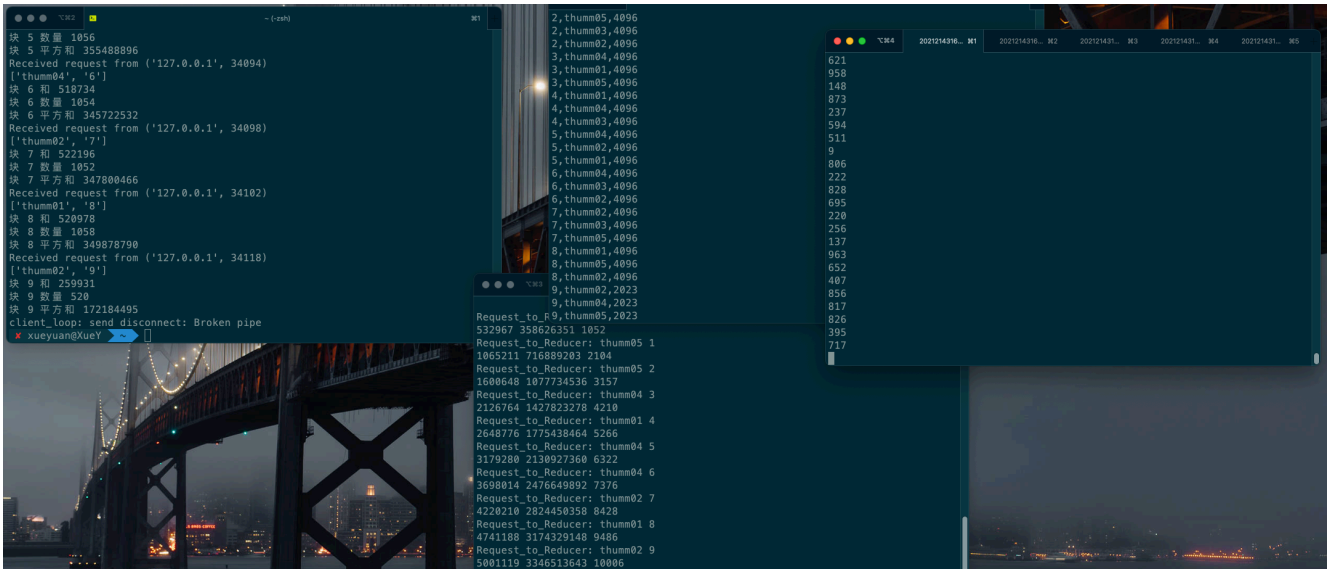
```
                        # 需要返回该块数据的和、平方和、数量
                        print("块",blk_no,"和",data['rdm_data'].sum())
                        print("块",blk_no,"数量",data['rdm_data'].count())
                        # 不会搞平方和，用笨方法了
                        square_sum=0
                        for index, row in data.iterrows():
                            square_sum += row['rdm_data']**2
                        print("块",blk_no,"平方和",square_sum)

                        # 结果返回到client
                        # 以str的形式发list太麻烦了，改为发三个str（数）
                        # response = [data['rdm_data'].sum(),
data['rdm_data'].count(), square_sum]
                        sock_fd.send(bytes(str(data['rdm_data'].sum()),
encoding='utf-8'))
                        time.sleep(0.2)  # 两次传输需要间隔一段时间，避免粘包
                        sock_fd.send(bytes(str(data['rdm_data'].count()),
encoding='utf-8'))
                        time.sleep(0.2)  # 两次传输需要间隔一段时间，避免粘包
                        sock_fd.send(bytes(str(square_sum), encoding='utf-8'))

                except KeyboardInterrupt:
                    break
                finally:
                    sock_fd.close()

        except KeyboardInterrupt:
            pass
        except Exception as e:
            print(e)
        finally:
            listen_fd.close()

# 创建Reducer对象并启动
reducer = Reducer()
reducer.run()
```

5. 修改client.py，增加调用reducer的函数；

```
def mprd(self, dfs_path):
        request = "get_fat_item {}".format(dfs_path)
        print("Request: {}".format(request))
        # 从NameNode获取一张FAT表；打印FAT表
        self.name_node_sock.send(bytes(request, encoding='utf-8'))
        fat_pd = self.name_node_sock.recv(BUF_SIZE)

        # 打印FAT表，并使用pandas读取
        fat_pd = str(fat_pd, encoding='utf-8')
        print("Fat: \n{}".format(fat_pd))
```

```
        fat = pd.read_csv(StringIO(fat_pd))

        # 记录结果
        sum_xi, sum_xi2, sum_cnt = 0,0,0
        # 计数器，因为重复存储repli可以跳过
        tik = 0
        for idx, row in fat.iterrows():  # todo-跳几行访问，一个块处理一次就行了
            # 更新计数器
            if tik % dfs_replication != 0:
                tik += 1
                continue


            host = row['host_name']
            blk = row['blk_no']
            reducer_sock = socket.socket()
            reducer_sock.connect(('localhost', reducer_port))  # reducer和client都跑
在thumm01上
            req = str(host) + " " + str(blk)
            print("Request_to_Reducer: {}".format(req))
            reducer_sock.send(bytes(req, encoding='utf-8'))

            # 接收三个返回结果
            xi = reducer_sock.recv(BUF_SIZE)
            xi = str(xi, encoding='utf-8')
            sum_xi += int(xi)

            cnt = reducer_sock.recv(BUF_SIZE)
            cnt = str(cnt, encoding='utf-8')
            sum_cnt += int(cnt)

            xi2 = reducer_sock.recv(BUF_SIZE)
            xi2 = str(xi2, encoding='utf-8')
            sum_xi2 += int(xi2)

            print(sum_xi, sum_xi2, sum_cnt)

            # 更新计数器
            tik += 1

        # 整合计算最终结果
        mean = sum_xi / sum_cnt
        var = sum_xi2/sum_cnt - mean**2
        print(mean, var)
```

6. 运行命令：python3 client.py -mprd /FAT.txt，左上角为reducer的输出，上中部是namenode取FAT表的输出，右上角是每个节点的datanode取数据块输出；下部是client输出，对于每个数据块，reducer完成计算后，client进行汇总并输出。



7. 由于按块取数据会出现一个数被分割的情况，因此需要修改为按行取数，最后client的结果输出与单机结果相同。