分布式系统大作业

分布式文件系统

姓名	班级	学号
熊明	计科5班	20305055

一、问题描述

设计一个分布式文件系统。该文件系统可以是 client-server 架构，也可以是 P2P 非集中式架构。 要求文件系统具有基本的访问、打开、删除、缓存等功能，同时具有一致性、支持多用户特点。 在设计过程中能够体现在分布式课程中学习到的一些机制或者思想，例如 Paxos 共识、缓存更新机制、访问控制机制、并行扩展等。 实现语言不限， 要求提交代码和实验报告。

1. 编程语言不限，选择自己熟悉的语言，但是推荐用 Python或者 Java 语言实现；
2. 文件系统中不同节点之间的通信方式采用 RPC 模式，可选择 Python 版本的 RPC、gRPC 等；
3. 文件系统具备基本的文件操作模型包括：创建、删除、访问等功能；
4. 作为文件系统的客户端要求具有缓存功能即文件信息首先在本地存储搜索，作为缓存的介质可以是内存也可以是磁盘文件；
5. 为了保证数据的可用性和文件系统性能，数据需要创建多个副本，且在正常情况下，多个副本不在同一物理机器，多个副本之间能够保持一致性（可选择最终一致性即延迟一致性也可以选择瞬时一致性即同时写）；
6. 支持多用户即多个客户端，文件可以并行读写（即包含文件锁）；
7. 对于上述基本功能，可以在本地测试，利用多个进程模拟不同的节点，需要有相应的测试命令或者测试用例，并有截屏或者video 支持；
8. 提交源码和报告，压缩后命名方式为：学号*姓名*班级
9. 实验报告长度不超过 20 页；

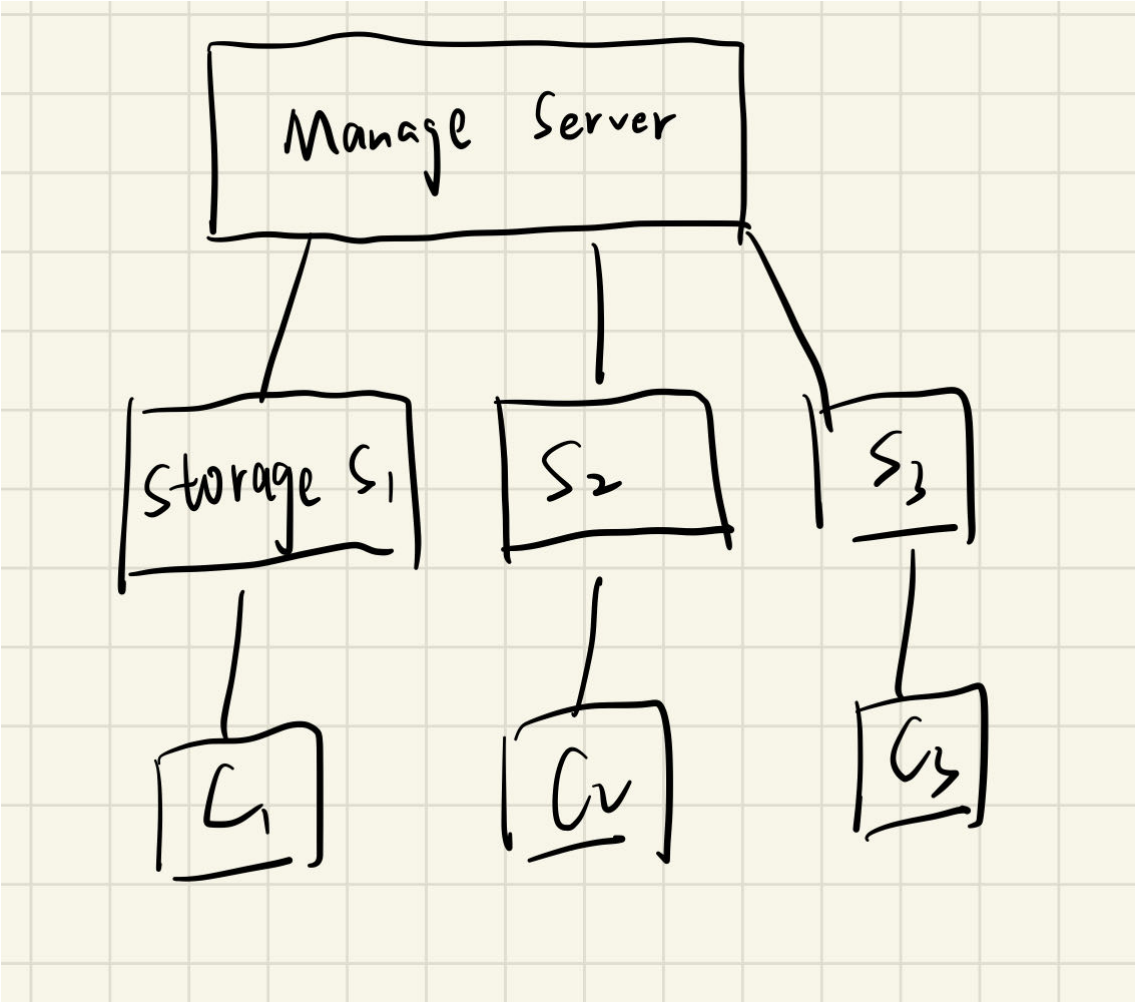
加分项：

1. 加入其它高级功能如缓存更新算法；
2. Paxos 共识方法或者主副本选择算法等；
3. 访问权限控制；
4. 其他高级功能；

二、解决方案

本次实验项目参考了网上的实现结果，但是有本人自己的修改和想法。参考文章：[极简分布式文件系统实现-LiteDFS]

思路如下：



1. 拥有一个主管服务，作用是
    1. 管理存储服务器，维护一个存储服务器表，记录他们的`serverid`
    2. 记录每个存储服务器存储内容，并向外提供查询服务，告知客户所需要的文件地址
    3. 给文件上锁，维护一个文件表来记录上锁情况。保证了同一个可以同时进行读操作和写操作，但是别的用户无法对改文件进行读写
2. 存储服务
    1. 提供存储服务，存储的内容可以是不一样的
    2. 提供下载服务，用户可以找到对应路径下的文件或文件夹，下载到本地读取。
3. 用户，对用户来说，上层服务器是透明的，在用户的视角下，只可以看到一个整体的文件架构

实现：

1. manageServer:
    1. 管理服务器上下线

```
    def offline(self):
        # 当还存在在线数据服务器时，发送警告
        if len(self.serverList):
            print('Warning: There are still online servers. Closing may cause exceptions.')
        print('Management Server is offline')

    def serverOnline(self, request, context):
        # 数据服务器上线，向管理器注册信息
        self.serverList.append(request)
        print('Storage Server:%d is online' % request.id)
```

```
        return ma_pb2.ma_reply(done=1)

    def serverOffline(self, request, context):
        # 数据服务器下线，向管理器注销信息
        remove_id = request.id
        for server in self.serverList:
            if server.id == remove_id:
                remove_server = server
                break
        self.serverList.remove(remove_server)
        print('Storage Server:%d is offline' % request.id)
        return ma_pb2.ma_reply(done=1)
    def getServerList(self, request, context):
        # 获取在线数据服务器信息
        server_list = list()
        for server in self.serverList:
            server_list.append(server)
        return ma_pb2.serverList(list=server_list)
```

在服务器上线下线的过程中，管理服务器端会打印相应信息

2. 记录存储内容，维护存储表

```
    def addFile(self,request,context):
        if(request.filePath in self.fileinfo):

 self.fileinfo[request.filePath].append((request.rootPath,request.serverId))
        else:
            self.fileinfo[request.filePath] = [(request.rootPath,request.serverId)]
        print(self.fileinfo)
        return ma_pb2.ma_reply(done = 1)

    def searchFile(self,request,context):
        print(self.fileinfo)
        if(request.filePath in self.fileinfo):
            random.seed()
            rand = random.randint(0,len(self.fileinfo[request.filePath])-1)
            print(rand)
            rootpath = self.fileinfo[request.filePath][rand][0]
            serverid = self.fileinfo[request.filePath][rand][1]
            return ma_pb2.searchReply(path=rootpath,serverId=serverid,done=1)
        else:
            print("No file found.")
            return ma_pb2.searchReply(path=' ',severId=0,done=0)

    def delectFile(self,request,context):
        if(request.filePath in self.fileinfo):
            del self.fileinfo[request.filePath]
            return ma_pb2.ma_reply(done=1)
        else:
            return ma_pb2.ma_reply(done=0)
```

在用户上传文件或文件夹到存储服务器上时，存储服务器会向管理服务器提供路径和文件名，利于查找

3. 给文件上锁

```
    def lockFile(self, request, context):
        # 给文件上锁
        print('Lock: ' + request.filePath)
        if request.filePath in self.lockList:
            finish = 0
            if request.clientId == self.lockList[request.filePath]:
                reply = 'Alreay lock this file'
            else:
                reply = 'This file is locked by other client'
        else:
            finish = 1
            self.lockList[request.filePath] = request.clientId
            reply = 'Successfully lock the file, do not forget to unlock it after used'
        return ma_pb2.lockReply(done=finish, info=reply)

    def unlockFile(self, request, context):
        # 给文件解锁
        # 给文件上锁
        print('Unlock: ' + request.filePath)
        if request.filePath in self.lockList:
            if request.clientId == self.lockList[request.filePath]:
                finish = 1
                del self.lockList[request.filePath]
                reply = 'Successfully unlock the file'
            else:
                reply = 'This file is locked by other client, you can`t unlock it'
        else:
            finish = 1
            reply = 'This file isn`t locked by any client'
        return ma_pb2.lockReply(done=finish, info=reply)
```

通过维护一个客户id和文件地址的列表，来判断文件是否上锁

2. storageserver

1. 上下线信息告知管理服务器

```
    def online(self):
        # 每一个存储服务器上线要向管理服务器登记注册
        print('connect with the Management Server ...')
        managementChannel = grpc.insecure_channel(parameter._MANAGEMENT_IP + ':' + parameter._MANAGEMENT_PORT)
        self.managementStub = ma_pb2_grpc.managementServerStub(managementChannel)

 self.managementStub.serverOnline(ma_pb2.serverInfo(id=self.id, ip=self.ip, port=self.port))
        print('Storage Server %d is online' % self.id)

    def offline(self):
        # 向管理服务器登记注销

 self.managementStub.serverOffline(ma_pb2.serverId(id=self.id))
        print('Storage Server %d is offline' % self.id)
```

2. 提供存储服务

    1. 上传

```
    def synUpload(self, request, context):
        # 客户端提供文件路径、文件流，服务器更新并同步副本
        try:
            finish = 1
            for iter in request:
                filePath = iter.path
                print('upload: ' + filePath)
                # 二进制打开文件用于写入
                path = os.path.dirname(self.root_path + filePath)
                if not os.path.exists(path):
                    os.mkdir(path)
                with open(self.root_path + filePath, 'wb') as f:
                    f.write(iter.buffer)
            # 把文件广播到其他数据服务器进行同步，保存副本
            # 获取其他服务器信息
            response = self.managementStub.getServerList(ma_pb2.empty(e=1))
            # 遍历其他服务器上传文件
            for server in response.list:
                if server.id != self.id:
                    channel = grpc.insecure_channel(str(server.ip) + ':' + str(server.port))
                    stub = st_pb2_grpc.storageServerStub(channel)
                    stub.upload(self.getBuffer(filePath, self.root_path + filePath))
            print('Successfully uploaded and synchronized the file')
        except Exception as e:
            print(e.args)
```

```
            finish = 0
        return st_pb2.reply(done=finish)

    def upload(self, request, context):
        # 提供文件路径、文件流，服务器更新
        try:
            finish = 1
            for iter in request:
                filePath = iter.path
                print('upload: ' + filePath)
                # 二进制打开文件用于写入
                path = os.path.dirname(self.root_path +
filePath)
                if not os.path.exists(path):
                    os.mkdir(path)
                with open(self.root_path + filePath, 'wb') as
f:
                    f.write(iter.buffer)
            respone =
self.managementStub.addFile(ma_pb2.fileInfo(rootPath=self.root_
path,serverId=self.id,filePath=filePath))
            if respone.done == 1:
                print('Successfully uploaded the file')
            else:
                print('Error add')
        except Exception as e:
            print(e.args)
            finish = 0
        return st_pb2.reply(done=finish)
```

实现了本地服务器上传和广播到其他服务器作为副本存储

2. 下载

```
    def download(self, request, context):
        # 客户端从服务器下载文件
        respone =
self.managementStub.searchFile(ma_pb2.filePath(filePath=request
.path))
        if respone.done==1:
            # 本地存储服务器查找
            filePath = self.root_path + request.path
            print('download: ' + request.path)
            # 检查文件是否存在
            if os.path.exists(filePath):
                with open(filePath, 'rb') as f:
                    buf = f.read(parameter._BUFFER_SIZE)
                    yield st_pb2.fileStream(buffer=buf)
            else:
                # 在其他服务器中查找
                filePath = respone.path+request.path
                with open(filePath, 'rb') as f:
                    buf = f.read(parameter._BUFFER_SIZE)
                    yield st_pb2.fileStream(buffer=buf)
```

首先在本地进行查找，查找不到再向通过manage维护的存储信息表，在其他服务器中查找

3. 向用户提供接口

    1. 获取当前存储服务器文件

```
    def ls(self, request, context):
        # 客户端向服务器查询当前目录
        filePath = self.root_path + request.path
        try:
            dirList = ' '.join(os.listdir(filePath))
        except Exception as e:
            return st_pb2.fileList(list='null')
        return st_pb2.fileList(list=dirList)
```

    获取当前文件列表，如果本地服务器能查询到，就不用与其他服务器交互

    2. 获取全体存储服务器文件列表

```
    def lsall(self,request,context):
        filelist = os.listdir(self.root_path + request.path)
        response = self.managementStub.getServerList(ma_pb2.empty(e=1))
        for server in response.list:
            if server.id != self.id:
                channel = grpc.insecure_channel(str(server.ip) + ':' + str(server.port))
                stub = st_pb2_grpc.storageServerStub(channel)
                resp = stub.ls(st_pb2.file_path(path=request.path))
                # 所有取并集
                if(resp.list!='null'):
                    filelist = list(set(filelist) | set(resp.list.split(' ')))
                else:
                    continue
        filelist.sort()
        filelist = ' '.join(filelist)
        # print(filelist)
        return st_pb2.fileList(list=filelist)
```

    获取当前相对路径下，所有存储服务器的存储的文件列表

    3. 创建文件夹

```
    def mkdir(self, request, context):
        # 客户端要求创建文件夹
        try:
            finish = 1
            filePath = self.root_path + request.path
            # print(filePath)
            respone =
self.managementStub.addFile(ma_pb2.fileInfo(rootPath=filePa
th, serverId=self.id, filePath=request.path))
            if not os.path.exists(filePath):
                os.mkdir(filePath)
        except Exception as e:
            print(e.args)
            finish = 0
        return st_pb2.reply(done=finish)
```

在当前服务器里创建文件夹

4. 删除文件，并应用到全部文件服务器中，删去所有副本

```
    def synDelete(self, request, context):
        # 客户端删除服务器文件并同步
        try:
            finish = 1
            filePath = self.root_path + request.path
            if os.path.exists(filePath):
                # self.delete(request,context)
                try:
                    os.remove(filePath)
                    print('Successfully deleted the file')
                except Exception as e:
                    try:
                        shutil.rmtree(filePath)
                        print('Successfully deleted the
file')
                    except Exception as e:
                        print('delect failed')
                # 把删除命令广播到其他数据服务器进行同步
                # 获取其他服务器信息
            response =
self.managementStub.getServerList(ma_pb2.empty(e=1))
            # 删除管理服务器中的文件表信息
            re =
self.managementStub.delectFile(ma_pb2.filePath(filePath=req
uest.path))
            # 遍历其他服务器删除文件
            for server in response.list:
                if server.id != self.id:
                    channel =
grpc.insecure_channel(str(server.ip) + ':' +
str(server.port))
                    stub =
st_pb2_grpc.storageServerStub(channel)

 stub.delete(st_pb2.file_path(path=request.path))
```

```
            print('Successfully deleted the file and synchronized')
        except Exception as e:
            print(e.args)
            finish = 0
        return st_pb2.reply(done=finish)
```

5. 删除本地存储服务器的文件或文件夹

```
    def delete(self, request, context):
        # 删除服务器文件
        try:
            finish = 1
            filePath = self.root_path + request.path
            if os.path.exists(filePath):
                try:
                    os.remove(filePath)
                    print('Successfully deleted the file')
                except Exception as e:
                    try:
                        shutil.rmtree(filePath)
                        print('Successfully deleted the file')
                    except Exception as e:
                        print('delect failed')
            else:
                print('invild path.')
        except Exception as e:
            print(e.args)
            finish = 0
        return st_pb2.reply(done=finish)
```

3. client

1. 随机选择一个服务器进行连接

```
    def selectStorageServer(self):
        try:
            response = self.maStub.getServerList(ma_pb2.empty(e=1))
            random.seed()
            rand = random.randint(0,len(response.list)-1)
            # 随机给客户一个存储服务器
            server = response.list[rand]
            stChannel = grpc.insecure_channel(str(server.ip)+':'+str(server.port))
            self.stStub = st_pb2_grpc.storageServerStub(stChannel)

            # 这里对用户是透明的，但是为了调试方便打印出来
            self.server_root_path = ROOT_PATH + f'/DATASTORE/storage_%d/'%(server.id)
            print(f'Client {self.id} successfully started and connected with server %d' %(server.id))

```

```
        except Exception as e:
            print(e.args)
            print('Client startup failed.Please try again later.')
```

为了使服务器存储平衡，随机选择存储服务器进行连接

2. 功能模块

    1. ls功能，获取所有存储服务器内，当前文件夹下的全部文件

```
    def ls(self):
        response = self.stStub.lsall(st_pb2.file_path(path=self.cur_path))
        print(response.list)
```

    2. mkdir功能，在当前连接的存储服务器的相对路径下创建文件夹

```
    def mkdir(self, fileName):
        response = self.stStub.mkdir(st_pb2.file_path(path=self.cur_path + fileName))
        if response.done:
            print('Successfully create dir:%s.' % fileName)
        else:
            print('Create failed.')
```

    3. rm功能，删除全部副本

```
    def rm(self, fileName):
        response = self.stStub.synDelete(st_pb2.file_path(path=self.cur_path + fileName))
        if response.done:
            print('Successfully delete dir:%s.' % fileName)
        else:
            print('Delete failed.')
```

    4. 下载文件功能，对外不开放，在读取文件和更新文件时使用，用于本地缓存

```
    def download(self, fileName):
        try:
            response = self.stStub.download(st_pb2.file_path(path=self.cur_path + fileName))
            # 二进制打开文件用于写入
            path = self.root_path + self.cur_path
            # 本地文件不存在就创建
            if not os.path.exists(path):
                os.mkdir(path)
            with open(path + fileName, 'wb') as f:
                for i in response:
                    f.write(i.buffer)
            print('Successfully download the file.')
```

```
        except Exception as e:
            print(e.args)
            print('Download failed.')
```

5. create功能，创建文件

```
    def create(self, fileName):
        self.selectStorageServer()
        path = self.root_path + self.cur_path
        # 本地文件不存在就创建
        if not os.path.exists(path):
            os.mkdir(path)
        with open(path + fileName, 'w') as f:
            try:
                while True:
                    msg = input("Enter ctrl+c to exit: ")
                    f.write(msg)
            except KeyboardInterrupt:
                # 结束写入
                f.close()
        # 上传到连接的服务器
        path = path + fileName  # 本地绝对路径
        rPath = self.cur_path + fileName  # 相对路径
        reply = self.stStub.upload(self.getBuffer(rPath,
path))
        if(reply.done==1):
            print('Successfully create file:' + fileName)
        # 清空缓存
        if(self.cur_path==''):
            os.remove(path)
        else:
            shutil.rmtree(self.root_path+self.cur_path)
```

首先在client用户本地进行缓存，然后创建文件，进行写操作。之后上传到连接的本地服务器中。这里在每次创建文件时，都更新一次该用户连接的服务器，但这对用户是透明的。用户不知道自己是向哪个存储服务器上传了文件，但都可以在连接的窗口中访问。

6. update功能，更新操作

```
    def update(self,filename):
        try:
            response =
self.stStub.ls(st_pb2.file_path(path=self.cur_path))
            if filename in response.list and
os.path.isfile(self.server_root_path + self.cur_path +
filename):
                # 确定文件存在且是文件而非文件夹
                # 对文件进行上锁
                response =
self.maStub.lockFile(ma_pb2.lockInfo(clientId=self.id,
filePath=self.cur_path + filename))
                if response.done == 1:
                    # 成功上锁
```

```
                    self.openFile.append(self.cur_path +
filename)
                    # 下载文件到本地
                    self.download(filename)
                    print('Successfully open: ' + filename)
                    # 输出文件信息
                    print('************FILE
CONTENT************')
                    with open(self.root_path +
self.cur_path + filename, 'r') as f:
                        buf = f.read()
                        print(buf)
                    f.close()
                    print('************UPDATE
FILE**************')
                    with open(self.root_path +
self.cur_path + filename, 'w') as f:
                        try:
                            while True:
                                msg = input("Enter ctrl+c
to exit: ")
                                f.write(msg)
                        except KeyboardInterrupt:
                            # 结束写入
                            f.close()

 print('************************************')
                    # 广播到所有服务器
                    self.upload(filename)
                else:
                    print(response.info)
            elif filename not in response.list:
                resp =
self.maStub.searchFile(ma_pb2.filePath(filePath=self.cur_pa
th + filename))
                if resp.done == 1:
                    prepath = self.server_root_path
                    self.server_root_path = ROOT_PATH +
f'/DATASTORE/storage_%d/' % (resp.serverId)
                    response =
self.maStub.lockFile(ma_pb2.lockInfo(clientId=self.id,
filePath=self.cur_path + filename))
                    if response.done == 1:
                        # 成功上锁
                        self.openFile.append(self.cur_path
+ filename)
                        # 下载文件到本地
                        self.download(filename)
                        print('Successfully open: ' +
filename)
                        # 输出文件信息
                        print('************FILE
CONTENT************')
                        with open(self.root_path +
self.cur_path + filename, 'r') as f:
```

```
                            buf = f.read()
                            print(buf)
                        f.close()
                        print('************UPDATE FILE**************')
                        with open(self.root_path + self.cur_path + filename, 'w') as f:
                            try:
                                while True:
                                    msg = input("Enter ctrl+c to exit: ")
                                    f.write(msg)
                            except KeyboardInterrupt:
                                # 结束写入
                                f.close()

 print('************************************')
                        # 广播到所有服务器
                        self.upload(filename)
                    else:
                        print(response.info)
                self.server_root_path = prepath
            else:
                print('Can`t find this file.')
        except Exception:
            print('no file')
```

写的有些冗余，但大致思路是首先与本地连接的服务器中查找是否有该文件存在，如果存在，则将其展示，并且供用户修改（覆盖修改）。如果不存在，则向manageserver询问该文件是否存在，如果不存在，则输出`no file`，如果存在，则将对应服务器的地址返回，找到该文件后缓存到本地。在更新完文件以后，广播到所有存储服务器内，创建多个副本。

7. 获取文件流，用于与存储服务器交互

```
    def getBuffer(self, rPath, path):
        # 根据文件相对路径和本地绝对路径返回流式数据
        with open(path, 'rb') as f:
            buf = f.read(parameter._BUFFER_SIZE)
            yield st_pb2.upload_file(path=rPath, buffer=buf)
```

8. upload功能，上传该文件到所连接的服务器上，对用户透明

```
    def upload(self, fileName):
        try:
            path = self.root_path + self.cur_path +
fileName  # 本地绝对路径
            rPath = self.cur_path + fileName  # 相对路径
            if os.path.exists(path):
                response =
self.stStub.synUpload(self.getBuffer(rPath, path))
                print('Successfully upload the file.')
            else:
                print('Can`t find this file.')
        except Exception as e:
            print(e.args)
            print('Upload failed.')
```

9. cd功能，移动当前路径

```
    def cd(self, fold):
        response =
self.stStub.ls(st_pb2.file_path(path=self.cur_path))
        path = self.server_root_path + self.cur_path + fold
        try:
            if fold in response.list and
os.path.isdir(path):
                # 成功进入文件夹
                self.cur_path += fold + '/'
            elif fold not in response.list:
                re =
self.maStub.searchFile(ma_pb2.filePath(filePath=self.cur_pa
th+fold))
                if re.done == 1:
                    self.mkdir(fold)
                    self.cur_path += fold + '/'
                else:
                    print('Can`t enter this fold')
            else:
                print('Can`t enter this fold')
        except Exception:
            print("Fold is no exits")
```

首先获取当前连接的存储服务器的文件列表，如果存在需要进入的文件夹，直接将相对路径改为文件夹名即可。如果不存在，则向manageserver确认该文件夹是否存在于其他存储服务器中。如果不存在则返回`Fold is no exits`，如果存在，则直接在当前连接的存储服务器中创建该文件夹

10. cd..功能，返回上一级

```
    def cdBack(self):
        if (self.cur_path != ''):
            self.cur_path =
os.path.dirname(self.cur_path[:-1]) + '/'
            if self.cur_path == '/':
                self.cur_path = ''
        else:
            print('Alreay in root dir.')
```

11. open功能，实现思路与update类似

```
    def open(self, fileName):
        try:
            response =
self.stStub.ls(st_pb2.file_path(path=self.cur_path))
            if fileName in response.list and
os.path.isfile(self.server_root_path + self.cur_path +
fileName):
                # 确定文件存在且是文件而非文件夹
                # 对文件进行上锁
                response =
self.maStub.lockFile(ma_pb2.lockInfo(clientId=self.id,
filePath=self.cur_path + fileName))
                if response.done == 1:
                    # 成功上锁
                    self.openFile.append(self.cur_path +
fileName)
                    # 下载文件到本地
                    self.download(fileName)
                    print('Successfully open: ' + fileName)
                    # 输出文件信息
                    print('************FILE
CONTENT************')
                    with open(self.root_path +
self.cur_path + fileName, 'r') as f:
                        buf = f.read()
                        print(buf)
                    f.close()

 print('************************************')
                else:
                    print(response.info)
            elif fileName not in response.list:
                response =
self.maStub.searchFile(ma_pb2.filePath(filePath=self.cur_pa
th + fileName))
                if response.done==1:
                    prepath = self.server_root_path
                    self.server_root_path = ROOT_PATH +
f'/DATASTORE/storage_%d/' % (response.serverId)
                    if(os.path.isfile(self.server_root_path
+ self.cur_path + fileName)):
```

```
                        response =
self.maStub.lockFile(ma_pb2.lockInfo(clientId=self.id,
filePath=self.cur_path + fileName))
                        if response.done == 1:
                            # 成功上锁

 self.openFile.append(self.cur_path + fileName)
                            # 下载文件到本地
                            self.download(fileName)
                            print('Successfully open: ' +
fileName)
                            # 输出文件信息
                            print('************FILE
CONTENT************')
                            with open(self.root_path +
self.cur_path + fileName, 'r') as f:
                                buf = f.read()
                                print(buf)
                            f.close()

 print('************************************')
                        else:
                            print(response.info)
                        self.server_root_path = prepath
                else:
                    print("no file.")
            except Exception:
                print("no file.")
```

12. close功能，关闭文件，并且解锁

```
    def close(self, fileName):
        path = self.cur_path + fileName
        if path in self.openFile:
            self.openFile.remove(path)
            # 上传文件
            # self.upload(fileName)
            # 解锁文件
            response =
self.maStub.unlockFile(ma_pb2.lockInfo(clientId=self.id,
filePath=self.cur_path + fileName))
            if response.done == 1:
                if(self.cur_path == ''):
                    os.remove(self.root_path + fileName)
                else:

 shutil.rmtree(self.root_path+self.cur_path)
                # os.remove(self.root_path + self.cur_path
+ fileName)
                print('Successfully close: ' + fileName)
            else:
                print(response.info)
        else:
            print('You haven`t open this file.')
```

在open和update中，都会将文件上锁。close功能可以将文件解锁，并且删除本地文件缓存

13. help功能，展示操作详情

```
    def help(self):
        print('Client ID: %d' % (self.id))
        print("-------------------- COMMAND LIST ----------
----------")
        print("ls       : List file directories")
        print("cd       : Change current path")
        print("cd..     : Go back to the previous path")
        print("create   : Create files locally")
        print("open     : Open files")
        print("close    : Close files")
        print("mkdir    : Create a folder")
        print("rm       : Delete files")
        print("update   : Update files")
        print("-----------------------------------------
----------")
```

三、实验结果展示

1. 服务器上线

```
(myDFS) PS D:\myDFS> python .\manageServer\server.py
Management Server is online
Storage Server:1 is online
Storage Server:2 is online
```

创建manageserver，后面两行是创建了storageserver后向manageserver提供的消息

```
(myDFS) PS D:\myDFS> python .\storageServer\server.py 1
connect with the Management Server ...
Storage Server 1 is online
```

1号存储服务器上线

```
(myDFS) PS D:\myDFS> python .\storageServer\server.py 2
connect with the Management Server ...
Storage Server 2 is online
```

2号存储服务器上线

2. 客户端上线

客户id采用随机数1-5，该文件服务器对外是统一的，所以不管客户id是什么，看到的都是相同的，所以直接采用随机数代表id

```
(myDFS) PS D:\myDFS> python .\client\client.py
Connect with the Management Server ...
Client 1 successfully started and connected with server 1
$>
```

上线两个客户端模拟并发执行

```
(myDFS) PS D:\myDFS> python .\client\client.py
Connect with the Management Server ...
Client 5 successfully started and connected with server 2
$>
```

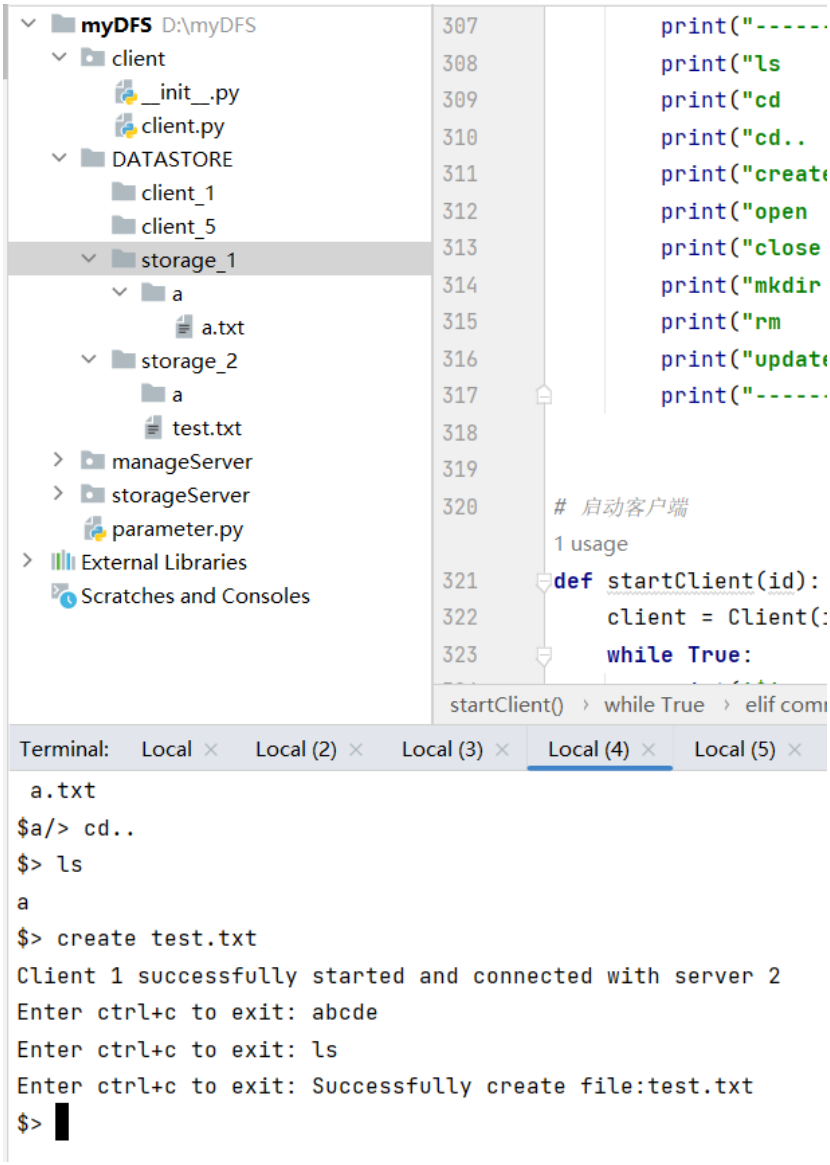
3. 测试在服务器上创建一个a目录，文件夹内有一个a.txt

```
Connect with the Management Server ...
Client 5 successfully started and connected with server 2
$> mkdir a
Successfully create dir:a.
$> cd a
$a/> create a.txt
Client 5 successfully started and connected with server 1
Enter ctrl+c to exit: a
Enter ctrl+c to exit: Successfully create file:a.txt
$a/>
```

可以在另一个进程中，用ls查看

```
(myDFS) PS D:\myDFS> python .\client\client.py
Connect with the Management Server ...
Client 1 successfully started and connected with server 1
$> ls
a
$> cd a
$a/> ls
 a.txt
$a/>
```

4. 测试在连接存储器1，访问2中的文件

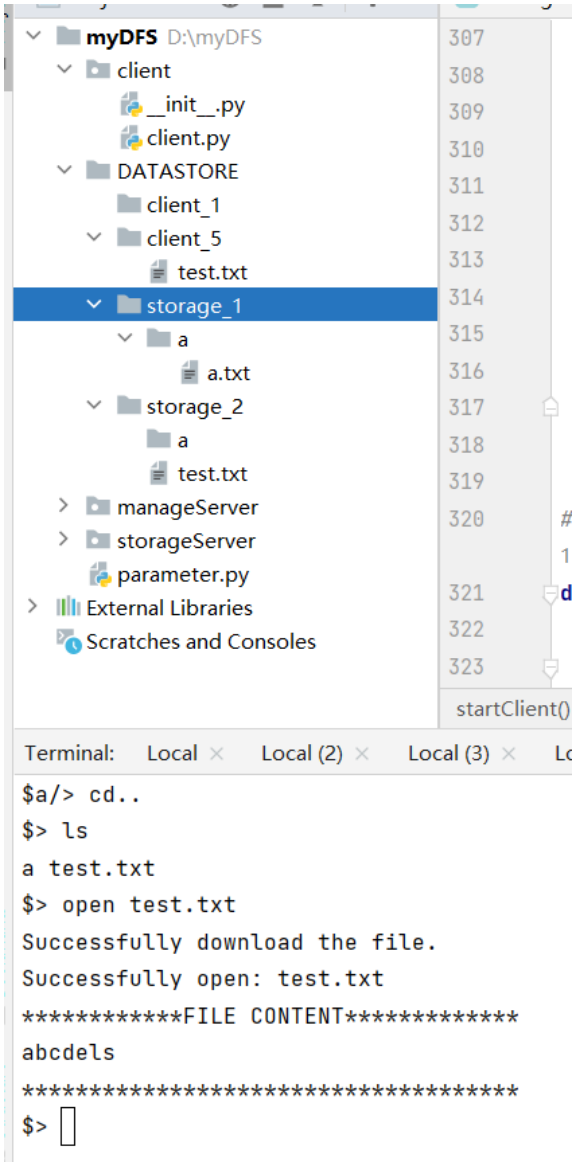


如图，可以看到该进程在存储器2下创建了文件texst.txt，而另一个进程连接的是存储器1

```
$a/> create a.txt
Client 5 successfully started and connected with server 1
Enter ctrl+c to exit: a
Enter ctrl+c to exit: Successfully create file:a.txt
$a/>
```

```
$a/> cd..
$> ls
a test.txt
$> open test.txt
Successfully download the file.
Successfully open: test.txt
************FILE CONTENT*************
abcdels
************************************
$>
```

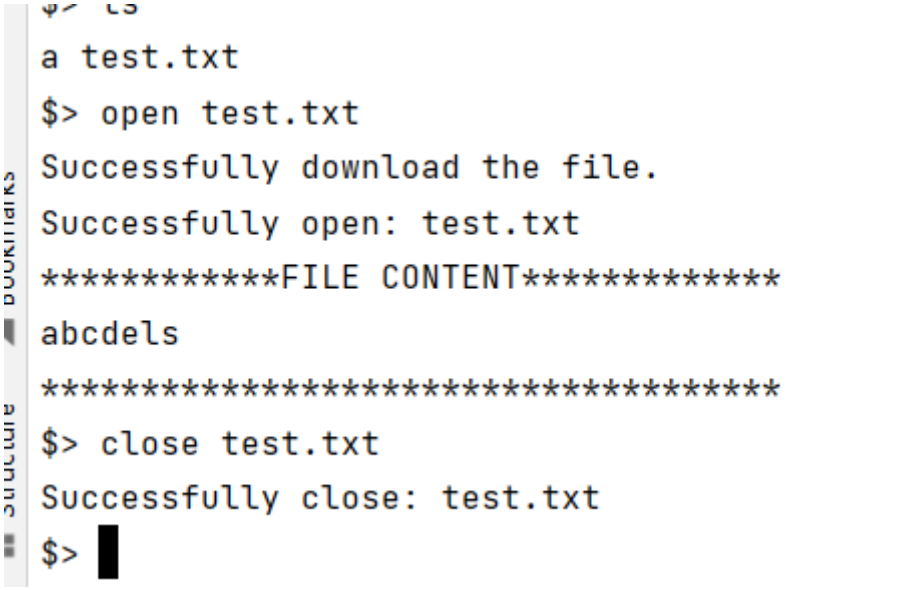
可以看到，在连接不同存储器的两个进程可以访问相同的文件。并且会缓存到用户本地：



5. 测试文件锁，此时进程test文件已经被一个进程打开，另一个进程访问该文件时会遇到锁，从而无法访问文件

```
$> create test.txt
Client 1 successfully started and connected with server 2
Enter ctrl+c to exit: abcde
Enter ctrl+c to exit: ls
Enter ctrl+c to exit: Successfully create file:test.txt
$> open test.txt
This file is locked by other client
$>
```

在读取进程使用close关闭文件后，客户1才能够访问该文件




```
$> open test.txt
This file is locked by other client
$> open test.txt
Successfully download the file.
Successfully open: test.txt
**********FILE CONTENT**********
abcdels
**********
$>
```

6. 更多操作见附件video

四、实验总结

最开始看到实验后毫无头绪，不知道该从何处做起。在网上查找资料后，看到了前人的实现。所以基于自己的理解，对他的实现做出了修改。我实现的是，存储服务器每一个存储的文件都可能是不一样的，只有在用户创建文件之后才会上传到服务器，并且由manageserver维护一个文件信息表，通过信息表来确定不同文件存储的服务器是哪个。并且实现了随机存储，可以一定程度上缓解各个服务器存储不平衡问题。

但还是有不足之处。

1. manageserver是集中管理，该节点断开连接之后，整个系统就会崩溃
2. 没能实现Paxos 共识方法
3. 在删除文件时，没有考虑该文件是否被打开，而是强行删除
4. 在manageserver下线后，重新上线时，若存储服务器中本身就有文件存在，manageserver是不会去读取该文件的

由于时间问题，以上几点不足没能优化。总体来说，通过本次实验，对grpc工作原理，分布式节点共识和一致性的理解加深。