实验课程: 分布式系统

实验名称: 分布式文件系统项目

专业名称: 计算机科学与技术

学生姓名: 吴臻

学生学号: 21307371

班级：计科3班

项目描述

> **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。**

基本要求

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；

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；

项目实现方案

运行环境

系统：Win11

编程语言：Python 3.10

IDE：PyCharm

通信方式：gRPC

```
pip install grpcio-tools
pip install --upgrade protobuf
```

编译proto文件：

```
python -m grpc_tools.protoc --python_out=. --grpc_python_out=. -I. XXX.proto
```

项目框架

> **本次项目需要构造三个类型的服务器，分别是 `DirectoryServer`（目录服务器）、`Lockserver`（锁服务器）和 `Fileserver`（文件服务器），目录服务器提供文件服务器的信息，锁服务器管理文件的并发访问，而文件服务器则负责存储和传输文件数据。**

1. **Directory Server（目录服务器）**：
   目录服务器负责存储和管理文件服务器的信息。它维护着一个文件服务器表，**记录了文件服务器的ip地址、端口、权限等数据信息。**当客户端需要访问文件服务器时，它可以通过向目录服务器发送请求获取文件服务器的相关信息，从而建立与文件服务器的联系。目录服务器起到了一个查询和定位的作用，**帮助客户端找到所需的文件服务器。**

```
E:\Anaconda3\python.exe E:\学习资料\大三上\分布式系统\大作业
DirServer is online
ID: 1   ip: localhost   port: 8083    password:111
ID: 2   ip: localhost   port: 8084    password:222
```

2. **Lock Server（锁服务器）**：
   锁服务器负责**管理文件的读写锁。**锁服务器一共提供两种类型的锁，分别是**共享锁和排它锁。**在多用户并发访问的情况下，为了保证数据的一致性（这里采用**瞬时一致性**），需要对文件进行加锁操作，以防止多个用户同时对同一文件进行写操作或读写冲突。锁服务器负责分配和管理文件的锁，并且在用户请求加锁或解锁时进行相应的处理。它起到了一个协调和控制的作用，确保文件的并发访问是安全和有序的。

3. **File Server（文件服务器）**：
   文件服务器承担了**存储和提供文件内容**的功能。它保存着实际的文件数据，并且能够根据客户端的请求来**读取、写入、删除文件**等操作。文件服务器实现了**备份功能（冗余）**，以保证文件的可靠性和可用性。客户端通过与文件服务器进行通信来实现文件的上传、下载和删除等操作。

4. **Client（客户端）**
   客户端在分布式文件系统中扮演着与服务器进行**通信和交互**的角色。它提供了用户接口，使用户能够通过命令与文件系统进行交互。

项目功能介绍和实现

> 代码实现请查看src文件夹

1. **用户登录与密码验证**

   i. 客户端首先**连接了文件锁服务器和目录服务器**，然后从目录服务器获取所有文件服务器的信息并展示给用户，让用户选择要连接的文件服务器。

   ii. 用户需要**输入要连接的文件服务器的ID和密码**。如果用户连续输入三次密码错误，则需要重新选择文件服务器。如果用户成功连接了文件服务器，则记录下该服务器的IP和port，并可以进行文件操作。

   iii. 客户端通过与目录服务器通信来获取文件服务器的信息，这样可以确保客户端能够连接到所有可用的文件服务器，并在其中选择一个来进行文件操作。同时，客户端还需要验证用户的身份和密码，以确保只有经过授权的用户才能够对文件进行操作。

   效果展示

```
Select one of the following servers:
ServerID        IP              port
1               localhost       8083
2               localhost       8084
Input ServerID to choose: 1
Please input password:11
Password Error
Please input password:11
Password Error
Please input password:111
Connecting to file server...
Successfully connect to FileServer localhost:8083
$
```

2. **查看当前位置的文件（ls）**

   i. 在文件服务器的 list() 函数中，它接收一个请求和上下文参数，并通过**os.listdir() 函数**获取指定路径下的所有文件名，并将其连接成一个字符串。最后，将文件名列表作为响应返回给客户端。

ii. 在客户端中，通过调用文件服务器的 list() 函数，向文件服务器发送一个列出当前路径下所有文件名的请求，并**接收到文件服务器的响应**。然后，将文件名列表返回给客户端进行进一步处理。

```
# 文件服务器
def list(self, request, context):
    # 返回一个包含该路径下的所有文件名的列表
    all_files = '    '.join(os.listdir(self.root_path + request.cur_path))
    return FS_pb2.List_Reply(list=all_files)

# 客户端
def list(self):
    # 列出当前路径下的所有文件名
    response = self.stub.list(FS_pb2.List_Request(cur_path = self.current_path)
    return response.list
```

效果展示

```
$ ls
c1a.txt    c2a.txt
```

3. **读写锁**

lock_type = 1 -> 共享锁

lock_type = 2 -> 排它锁

i. 当读写文件时，客户会调用**lockfile() 函数**，在这个函数中，它通过调用 `self.lockstub.lockfile()` 函数向文件服务器发送了一个锁定文件的请求。请求中包括了文件路径 `filepath` 、文件名 `filename` 和客户端 ID `self.id` 这些参数。该函数返回一个布尔值，表示锁定文件是否成功。

ii. 服务器会通过grpc在**lockfile函数**中接收到客户端发来的请求（包含信息），然后尝试调用 **addlock() 函数**，在 **addlock() 函数**中，它接收文件路径 `filepath` 、文件名 `filename` 、客户端ID `client_id` 和锁类型 `lock_type` 这些参数。首先，它通过调用 get_Lock_list() 函数获取当前的文件锁列表 `lock_list`。

iii. 然后，根据文件路径和文件名生成一个唯一的键 key，用于**标识该文件的锁状态**。如果要添加的锁类型是**排它锁**（lock_type == 2），并且已经存在同样的键且锁类型也为排它锁（lock_list[key]['lock_type'] == '2'）或是已经存在同样的键且锁类型为读共享锁（lock_dic[key]['lock_type'] == '1'），则抛出异常表示不允许再次上锁。

iv. 如果要添加的锁类型是**共享锁**（lock_type == 1），则只要没有存在的排它锁就可以添加成功。

```
message lockfileinfo{
    int32 client_id = 1; // 客户id
    string file_path =2;// 被锁的文件的服务器路径
    string filename = 3; // 被锁文件的名字
    int32 lock_type = 4; // 锁的类型
}
```

```
def addlock(self, filepath, filename, client_id, lock_type):
    # 添加文件锁
    lock_list = self.get_Lock_list()
    key = filepath + filename
    if key in lock_list and lock_list[key]['lock_type'] == '2':
        # 如果原来有锁，并且是排它锁，则不允许再次上锁
        raise Exception(f"Write_File({filepath}:{filename}) is locked ")

    if key in lock_list and lock_list[key]['lock_type'] == '1' and lock_type ==
        # 如果原来有读共享锁，并且现在要上排它锁，则不允许上锁
        raise Exception(f"Read_File({filepath}:{filename}) is locked ")

    # 新增锁信息到文件
    with open(self.lockfile_path, 'a', encoding='utf-8') as f:
        text = f"{filepath}\t{filename}\t{client_id}\t{lock_type}\n"
        f.write(text)
        if lock_type == 1:
            print(f"Add read_lock to ({filepath}:{filename})   client_id: {clie
        else:
            print(f"Add write_lock to ({filepath}:{filename})   client_id: {cli
```

效果展示

```
E:\Anaconda3\python.exe E:\学习资料\大三上\分布式系统\大作业\r
LockServer is online
Add write_lock to (server/:wz.txt)   client_id: 1
Write_File(server/:wz.txt) is locked
Unlock (server/:wz.txt)   client_id: 1
Add read_lock to (server/:wz.txt)   client_id: 2
Read_File(server/:wz.txt) is locked
Add read_lock to (server/:wz.txt)   client_id: 1
```

4. **同步与备份**

> **同步与备份主要是数据冗余，即当客户在一台服务器进行操作，比如：upload、download、delete、read、write时，会同时在其他服务器上同步此操作，来达到瞬时一致性。**

i. 获取要上传的文件的**原地址 source 和目标地址 target。**
检查文件是否存在，如果不存在则打印错误信息并返回。

ii. 通过调用 `self.dirstub.getfileserver(DS_pb2.Dir_Empty(empty=0))` 获取所有副本服务器的地址列表。

iii. 遍历服务器列表，对于每个服务器：

a. 如果是当前服务器自己，则跳过。

b. 建立与其他服务器的连接，**创建临时的 gRPC 通道和存根（stub）。**

c. 向其他服务器传递自己的操作（**注意只有主服务器才需要执行同步操作，副本操作无需进行同步操作**）

效果展示

```
Input ServerID to choose: 1
Please input password:111
Connecting to file server...
Successfully connect to FileServer localhost:8083
$ ls
c2a.txt   wz.txt
$ upload wz.txt
File uploaded successfully!
```

```
Input ServerID to choose: 2
Please input password:222
Connecting to file server...
Successfully connect to FileServer localhost:8084
$ ls
c2a.txt   s2a.txt   test.txt
$ ls
c2a.txt   s2a.txt   test.txt   wz.txt
```

5. **用户下载（download）、上传（upload）、删除（delete）**

> **用户下载是从服务器上下载文件，上传是将自己的本地文件上传到服务器，删除是删除服务器上的文件，注意：上传和删除涉及到前面提到的同步操作**

- **下载（download）**

  i. 先构建要下载的文件的远程路径 source 和本地路径 target。然后**获取远程服务器当前目录的所有文件列表。**

  ii. 调用 `self.lockfile(self.current_path, filename, 2)` 方法对文件进行**加锁操作(互斥锁）**。

  iii. 调用 `self.stub.download(FS_pb2.Download_Request(download_path=source))` 发起下载文件的请求，并获取响应。

  iv. 主服务器收到请求之后，进行相应操作，然后给客户返回成功或失败的响应。

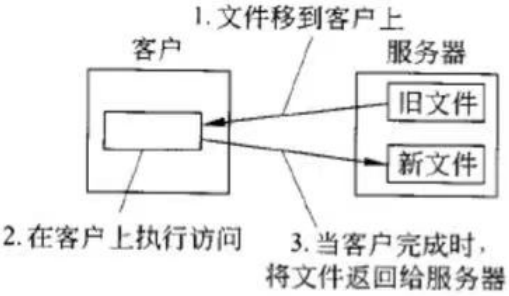
  v. 客户收到响应之后，调用 `self.unlockfile(self.current_path, filename)` 方法将文件解锁。

- **上传（upload）和删除（delete）**
  过程与下载（download）基本一致，主要区别在于主服务器收到请求之后，进行相应操作，并将该操作**同步**到其他副本服务器上，然后给客户返回成功或失败的响应

效果展示

```
$ ls
c2a.txt   wz.txt
$ download wz.txt
Download file(wz.txt) successfully
$ upload c1a.txt
Upload file(c1a.txt) successfully!
$ ls
c1a.txt   c2a.txt   wz.txt
$ delete c1a.txt
Delete file(c1a.txt) successfully
$ ls
c2a.txt   wz.txt
$
```

6. **用户读（read）、写（write）、关闭（close）**



> **读写功能是基于前面的下载（download）和上传（upload）实现的，当客户发出读写请求，客户会先将目标文件下载到本地，然后进行读或写，当使用close命令关闭时，客户会将文件上传到服务器，并同步到其他服务器。注意：当客户在操作目标文件时，会为该文件上锁，只有得到锁才具备操作的权限。**

效果展示

```
$ ls
c2a.txt   wz.txt
$ write wz.txt
Start writing
$ close wz.txt
File closed successfully.
$ write wz.txt
The file is being read or written by someone else
$ read wz.txt
Start reading
$ close wz.txt
File closed successfully.
```

实验结果

> **基本功能已经在上面的功能介绍中展示了，这里就不再重复，为了展示读写锁，录制了一个短视频（demo），下面对该视频进行讲解：**

1. 首先分别运行文件服务器（S）和两个客户（C1和C2），两个客户同时登录同一台文件服务器。
2. C1首先写文件（wz.txt),这时，可以看到，C2没法读或写该文件（体现排它锁）。
3. 然后C1关闭该文件，此时C2可以成功读取该文件。
4. 之后C1尝试写该文件，失败，因为C2正在阅读，但是C1同样可以读该文件（体现共享锁）。
5. 最后两个客户分别close该文件，释放锁。

实验总结

> **因为之前的实验有涉及到grpc的通信（消息订阅系统），所以本次实验项目采用了grpc来进行通信，虽然grpc只需要编写proto文件并编译，然后就可以调用编译得到的stub函数，但是真正操作起来还是有相当大的难度。特别是当多个服务器和客户端同时运行时，有时一个小小的变量错误就需要花费大量时间，我一开始是在linux虚拟机上进行实验的，后面因为调试过于复杂，于是转战到windows，用pycharm调试会方便很多，在查阅和参考了许多资料之后，终于完成该项目。**