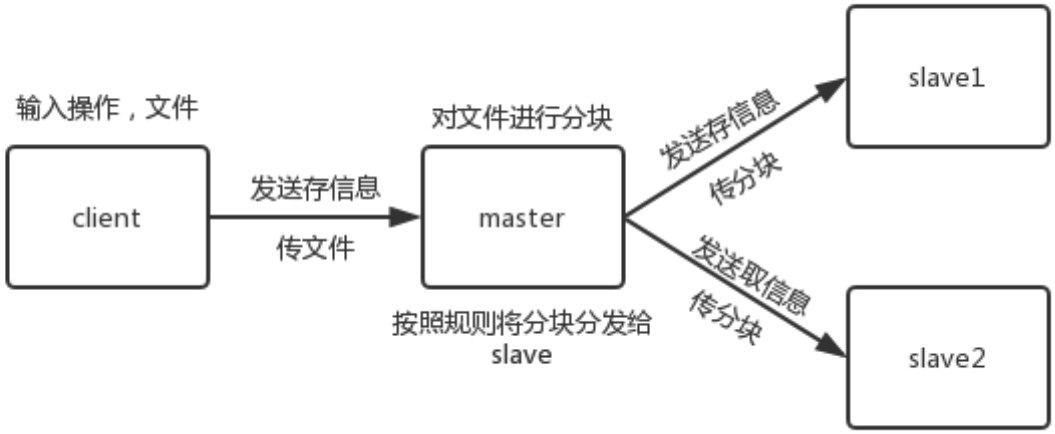
分布式文件系统实现

设计

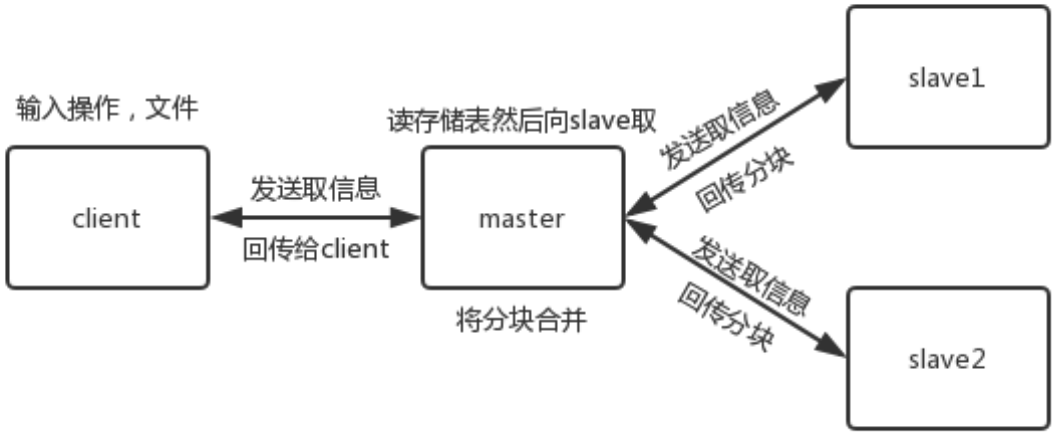
在三台服务器(*三台以上也是同理，所以只用实现三台就行*)上实现对客户端文件的存取，利用socket传输文件，保证文件传输的安全性。使用python分别编写四个程序
client.py，**master.py**，**slave1.py**，**slave2.py**分别运行在客户端（**即我的个人电脑**），主节点服务器（**thumm01**），从节点服务器（**thumm02**和**thumm03**）上，来互相配合完成对文件的存储和加载。

- 客户端用来跟用户交互，获得想进行的操作和操作的文件
- 主节点用来接收客户端发来的参数（操作类型，文件），然后对从节点进行调度
- 从节点接收主节点的消息接受或者发送文件

客户端往服务器分布式存文件流程



客户端往服务器分布式存文件流程



原理和实现

文件分块存放规则

把一个文件按50MB每块来分块并且按照1开始编号

如：一个500M文件按照50M（即52428800字节）分块，那么分块后的文件编号为1到10

由于我这里是在三台服务器上实现，所以每块的编号对3取余， 那么结果就是

1，2，0，1，2，0，1，2，0，1

三台服务器的存放为：

- **thumm01存放取余结果为0，1的块**
- **thumm02存放取余结果为1，2的块**
- **thumm03存放取余结果为0，2的块**

filelist0，filelist1和filelist2分别存放了保存在master，slave1，slave2上的文件路径，通过sendfile(filelist,DestIp)函数发送到指定地址

```
i = 1
count = 0
while filesize > count:
    part_filepath = '%s/part%04d' % (savedir, i)
    s = i % 3
    if (s == 0):
        filelist0.append(part_filepath)
        filelist2.append(part_filepath)
    elif (s == 1):
        filelist0.append(part_filepath)
        filelist1.append(part_filepath)
    elif (s == 2):
        filelist1.append(part_filepath)
        filelist2.append(part_filepath)
```

```
            filelistdel.append(part_filepath)
        i = i + 1
        count += 52428800
```

这样一个大文件均匀分块并且均匀存放在多台服务器上，而且每块有一个备份，设置了连接异常捕捉，发生异常时返回-1

首先从slave1取数据分块，如果连接失败则代表slave1可能宕机，则从slave2取

```
ret = recvfile(recvlist, slave[0])
#如果1号连接失败则从2号加载数据
if(ret==-1):
recvfile(recvlist, slave[1])
```

而且这样的存放规则不需要维护文件具体的记录表，因为只需要知道文件的大小就可以知道文件有多少分块，并且根据分块号除以3的余数就可以知道它在哪台服务器上，所以只需要纪录文件大小就行

各部分通信

定义一个信息头长度(4s代表**操作**有4字节，128s表示**文件路径**长度最多128字节，l代表**文件大小**)

```
fileinfo_size = struct.calcsize('4s128sl')
```

发送端先发送定长的打包后的信息（操作类型有save和load两种，代表存和取文件）

```
fhead = struct.pack('4s128sl', "load", filepath, filesize)
    socket.send(fhead)
```

接收端拆包来获得参数

```
from_recv = conn.recv(fileinfo_size)
    method, filename, filesize = struct.unpack('4s128sl', from_recv)
```

这样就得到了操作类型，文件路径，文件长度

主节点的文件分发和加载

向slave发送文件：

filelist是要发送的文件路径列表，DestIp则是要发送到的地址

由于send不可发送太多字节，采用python自带slice对文件进行分片发送

```
#发送文件
def sendfile(filelist,DestIp):
    for file in filelist:
        sk_slave = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sk_slave.settimeout(50)
        sk_slave.connect((DestIp, 7777))
        filesize = os.stat(file).st_size
        fhead = struct.pack('4s128sl', "save", file, filesize)
        sk_slave.send(fhead)

        fp = open(file, 'rb')
        for slice in fp:
            sk_slave.send(slice)
        fp.close()
```

向slave获取文件：

filelist是要获取的文件路径列表，DestIp则是要获取文件的地址

每次循环获取1024字节，然后一直到文件接收完毕

如果连接失败，代表从节点可能宕机，返回值-1，正常则返回0

```
#加载文件
def recvfile(filelist,DestIp):
    print "发送"+filelist+"到"+DestIp
    for file in filelist:

        sk_slave = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        sk_slave.settimeout(300)
        try:
            sk_slave.connect((DestIp, 7777))
        except Exception:
            return -1

        fhead = struct.pack('4s128sl', "load", file, 0)
        sk_slave.send(fhead)

        f=open(file,'wb')
        while 1:
            data = sk_slave.recv(1024)
            if not data:
                break
            f.write(data)
        f.close()
        sk_slave.close()
    return 0
```

文件的分块

将输入的文件按照分块大小分块并保存到输入的目录中

```
def split(fromfile,todir,chunksize):
    if not os.path.exists(todir):
        os.mkdir(todir)
    else:
        for fname in os.listdir(todir):
            os.remove(os.path.join(todir,fname))
    partnum = 0
    inputfile = open(fromfile,'rb')
    while True:
        chunk = inputfile.read(chunksize)
        if not chunk:
            break
        partnum += 1
        filename = os.path.join(todir,('part%04d'%partnum))
        fileobj = open(filename,'wb')
        fileobj.write(chunk)
        fileobj.close()
    return partnum
```

文件的合并

将输入的目录下所有文件按照序号顺序合并为一个文件

```
def joinfile(fromdir,filename,todir):
    if not os.path.exists(todir):
        os.mkdir(todir)
    if not os.path.exists(fromdir):
        print('Wrong directory')
    outfile = open(os.path.join(todir,filename),'wb')
    files = os.listdir(fromdir)
    files.sort()
    for file in files:
        filepath = os.path.join(fromdir,file)
        infile = open(filepath,'rb')
        data = infile.read()
        outfile.write(data)
        infile.close()
    outfile.close()
```

结果展示

分别在主节点（thumm01）和两台从节点（thumm02和thumm03）使用nohup 命令挂起python程序

其中python文件都是放在我账号根目录下运行的

`nohup python master.py &`

`nohup python slave.py &`

分布式存文件

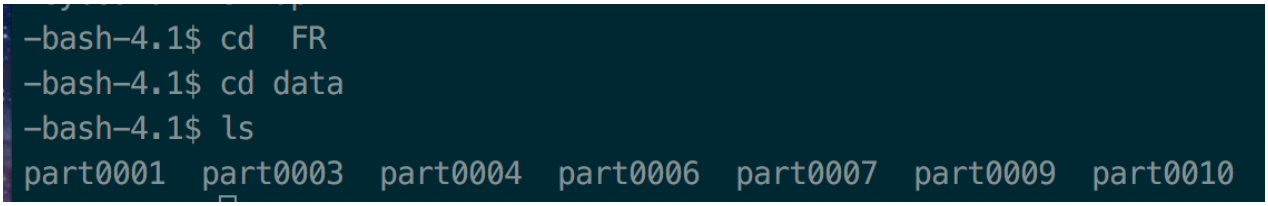
在客户机（我的个人电脑）运行client.py，作如下输入(data.csv是一个500MB左右文件)

```
  save data.csv
```

等待传输完毕后主节点显示

分别进入主节点和两台从节点的～ / FR/data文件夹后可以看到

(不记得什么操作把我第一台节点的名字给变成-bash-4.1了)



```
[2017210879@thumm02 ~]$ cd FR
[2017210879@thumm02 FR]$ cd data
[2017210879@thumm02 data]$ ls
part0001  part0002  part0004  part0005  part0007  part0008  part0010
```

```
[2017210879@thumm03 ~]$ cd FR
[2017210879@thumm03 FR]$ cd data
[2017210879@thumm03 data]$ ls
part0002  part0003  part0005  part0006  part0008  part0009
```

进入主节点的record文件下，查看data.txt存放了该文件大小记录（字节）

```
-bash-4.1$ cd record/
-bash-4.1$ more data.txt
524288000
```

分布式取文件

首先将开始存的文件转移到别的文件夹中，然后在我个人电脑上运行client.py后输入

```
load data.csv
```

由于我是从个人电脑与服务器交互，所以传输速度只有1MB左右每秒，比较慢

由于主节点保存了1，3，4，6，7，9，10号分块，所以只需要从slave1或者slave获取2，5，8号

分块主节点的输出为,可以看到从thumm02返回值为0，说明连接正常

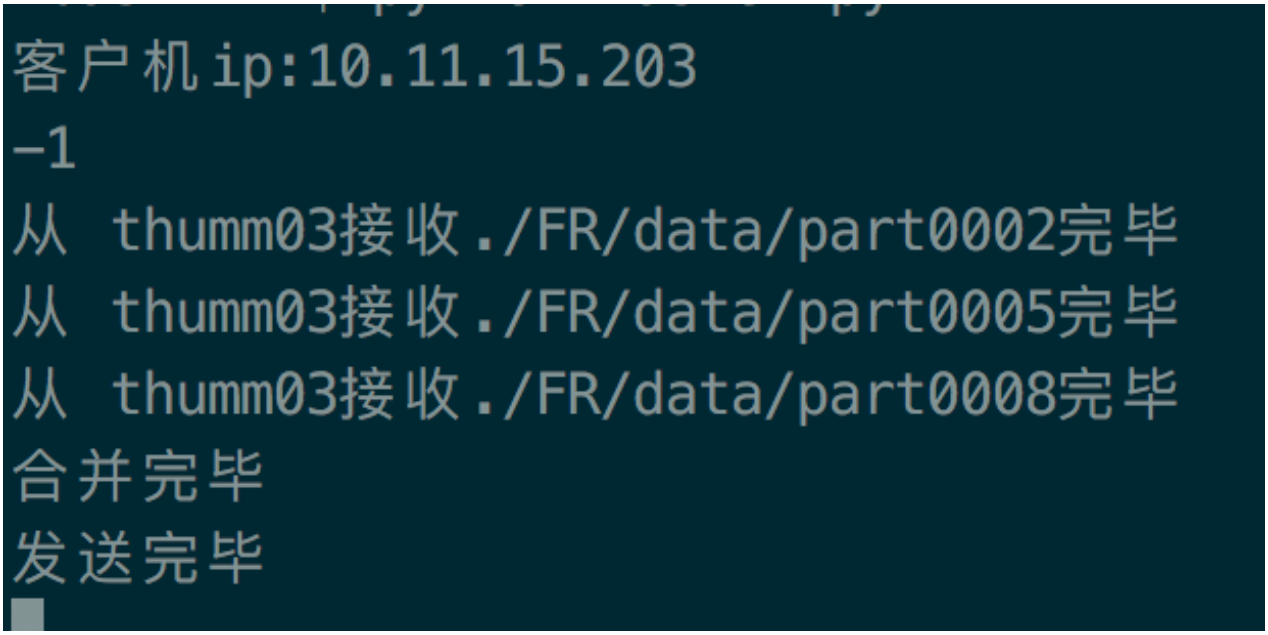
```
从 thumm02接收./FR/data/part0002完毕
从 thumm02接收./FR/data/part0005完毕
从 thumm02接收./FR/data/part0008完毕
0
合并完毕
发送完毕
```

在client端同目录下出现data.csv文件

在从节点slave1宕机的情况下的文件加载

为了模仿slave1节点宕机，我把主节点中的slave1地址由thumm02改为123，即一个连接不上的地址

然后再在client中输入load data.csv尝试加载文件，主节点的输出中为-1说明slave1连接不上,转而从slave2获取数据分块



文件分布式存储再加载后的完整性校验

原文件我放在/Users/joel/Desktop/data.csv

从分布式文件系统中加载的文件地址为/Users/joel/Desktop/DFS1/data.csv

使用diff命令对两个文件进行对比

没有任何输出，说明两个文件内容完全一致

文件存取用时分析

在主节点master.py中嵌入计时代码

```
import time;
time_start=time.time();

程序主体

time_end=time.time();
print time_end-time_start
```

文件分布式存储用时382.46秒，取文件用时49.5秒

三台服务器之间的数据传输很快（大概上百兆每秒），存储文件到服务器主要时间花在我的电脑和服务器之间的数据传输上，速度大概只有1M多每秒,而从服务器下载到我的电脑上则速度快很多