实验二 分布式文件系统

零、服务器集群更新说明

1.可使用的服务器集群

从lab2开始，我们上线了新的服务器集群，目前有两个集群(两个集群不互通，大家可以选择随意选择在集群一或二上实验^ ^):

集群一：ip地址：`10.103.9.11`，可用机器：01，03-06。登录集群一01的命令：`ssh xxx@10.103.9.11` （也就是和以下实验指导书的内容完全相同）

集群二：ip地址：`10.103.10.156`，可用机器：01-04。登录集群二01的命令：`ssh xxx@10.103.10.156 -p 8001`（**由于进入该集群的端口并非默认端口，所以在 `ssh` 指令后面一定要用 `-p` 要加上端口号！**）

其中 xxx为学号，默认密码也为学号。

2.注意事项

2.1 集群一需要重新设置免密登录

在第一次lab1的ddl之前，有不少同学配置的免密登录仅适用于集群 01,05,06这三台机器，所以当通过 `ssh` 访问其他节点时会出现以下情况：

```
2022214263@thumm01:~$ ssh thumm03
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:evavb+ZbgiSs4+3RkCd1m3dtb/EBQJ+1Ty5xPohUPx8.
Please contact your system administrator.
```

```
Add correct host key in /home/dsjxtjc/2022214263/.ssh/known_hosts to get rid
of this message.
Offending ECDSA key in /home/dsjxtjc/2022214263/.ssh/known_hosts:5
  remove with:
  ssh-keygen -f "/home/dsjxtjc/2022214263/.ssh/known_hosts" -R thumm03
ECDSA host key for thumm03 has changed and you have requested strict
checking.
Host key verification failed.
```

此时需要重新按照lab1中完成的任务重新配置节点01到其他节点的免密登录。

此外，如果启用集群二进行后续实验的话，为了避免后续实验出现未知情况，也请按照lab1重新配置节点01到其他节点的免密登录。

2.2 集群二的ssh连接端口和内部节点ip改变的问题

前面我们提到，集群二ssh所使用的端口为8001，而并非默认端口22，所以在使用 `ssh` 进入集群二时需要在最后添加 `-p 8001` 。此外，集群二节点之间的内部ip地址与集群一节点之间的内部ip地址也不同。

- 在集群一当中， thumm0X 的内部ip为 `192.168.0.10X` （X为1到6）
- 在集群二当中， thumm0X 的内部ip为 `192.168.1.10X` （X为1到4）

所以后续实验指导书中，涉及到集群的ssh连接端口以及集群节点内部ip的内容，在使用集群二的时候都需要进行对应修改（因为原本实验指导书默认使用的均为集群一）。

举例：在本次实验指导书中的 “三、掌握Hadoop DFS常用指令”-“2. 通过Web 查看Hadoop 运行情况”，提到需要在**本地**运行命令 `ssh xxx@10.103.9.11 -L 9870:192.168.0.101:9870` （其中xxx为学号），那么放在集群二当中，则需要运行 `ssh xxx@10.103.10.156 -p 8001 -L 9870:192.168.1.101:9870` 才能看到对应的结果。后续再有类似情况均进行类似处理。

2.3 集群节点更新通知

此后我们将随时在课程群中通知类似于某个集群/某个集群节点需要停用/启用的通知，请同学们即时在课程群查收消息，并在必要时更换集群继续进行实验。当实验中出现任何故障（例如无法登录、无法进行文件读写），请及时在课程群内告知我们。我们会尽快进行问题的排查与集群节点的修复。

一、实验目标

本次实验旨补全一个简单的Distributed File System (DFS) 并在其上实现MapReduce框架。具体任务如下：

- 了解 Hadoop 分布式文件系统常用指令（1 分）；
- 补全一个简单的分布式文件系统并添加容错功能（9 分）；
- 在自己设计的分布式文件系统上实现MapReduce 框架（5 分）。

二、实验任务与要求

- Hadoop
  - 掌握Hadoop简单指令，包括文件传输、查看
  - 通过Web 查看Hadoop 运行情况
- 分布式文件系统
  - 根据给定copyFromLocal代码，补充copyToLocal、ls、rm代码，实现分布式文件系统中下载、查看、删除
  - 实现数据块备份功能，对数据块进行多节点存储
  - Bonus——实现HeartBeat功能，模拟对节点故障的检测和故障后的文件表处理
- MapReduce

    - 实现矩阵乘法计算的MapReduce框架
    - Bonus——对框架的任一部分进行优化并说明理由
- 报告提交要求
    - 将命令、关键代码（文本）、结果截图放入报告，实验报告需为pdf 格式，连同代码文件一同打包成压缩文件（命名为`学号_姓名_实验一.*`，例如：`2021200000_张三_实验一.zip`），最后提交到网络学堂。压缩文件中文件目录应为：

```
.
└── 学号_姓名_实验二.pdf
└── MyDFS
    └── client.py
    └── common.py
    └── ...
```

- 迟交作业一周以内，以50% 比例计分；一周以上不再计分。另一经发现抄袭情况（包括往届），零分处理。

三、掌握Hadoop DFS常用指令

1. Hadoop使用方法

在服务器上，我们通过Linux 指令对本地文件系统进行操作，如使用`ls`查看文件/目录信息、使用`cp`进行文件复制、使用`cat`查看文件内容。在分布式文件系统中，也有一套相似的指令，接下来我们需要掌握一些基本的指令。（本题 1 分）

首先查看Hadoop DFS 支持的指令：

```
2020214210@thumm01:~$ hadoop fs
Usage: hadoop fs [generic options]
[-cat [-ignoreCrc] <src > ...]
[-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count >] <localsrc > ...
<dst >]
[-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src > ... <localdst >]
[-cp [-f] [-p | -p[topax]] [-d] <src > ... <dst >]
[-head <file >]
[-help [cmd ...]]
[-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [-e] [<path > ...]]
[-mkdir [-p] <path > ...]
[-moveFromLocal <localsrc > ... <dst >]
[-moveToLocal <src > <localdst >]
[-mv <src > ... <dst >]
[-rm [-f] [-r|-R] [-skipTrash] [-safely] <src > ...]
[-rmdir [--ignore -fail -on-non -empty] <dir > ...]
......
```

上面是DFS 中常用的指令，这些指令中有一些我们在本地文件系统中也用过，如`ls、cp、mv、rm、mkdir、cat、head`，还有一些指令是DFS 特有的，例如`copyFromLocal`、`copyToLocal`，主要用于DFS 与本地文件系统的数据交换。接下来使用`ls`指令查看DFS 中根目录下文件/文件夹的信息：

```
2020214210@thumm01:~$ 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
```

可以看到，现在DFS 根目录下一共有两项。其中`dsjxtjc`是一个文件夹，在这个文件夹下面有每位同学的文件夹，例如某位同学的学号是`202121xxxx`，那么TA对应的文件夹为`/dsjxtjc/202121xxxx/`。为了保证实验过程中不同用户之间不会产生干扰，每位同学只能在自己的文件夹下进行操作。下面查看自己的文件下的内容：

```
2020214210@thumm01:~$ hadoop fs -ls /dsjxtjc/2020214210
2020214210@thumm01:~$
```

接下来在本地创建一个`test.txt`文件：

```
2020214210@thumm01:~$ touch test.txt
2020214210@thumm01:~$ echo "Hello Hadoop" > test.txt
2020214210@thumm01:~$ cat test.txt
Hello Hadoop
2020214210@thumm01:~$
```

将本地文件传输至DFS 中：

```
2020214210@thumm01:~$ hadoop fs -copyFromLocal ./test.txt
/dsjxtjc/2020214210/
2021-10-11 20:15:04,136 INFO sasl.SaslDataTransferClient: SASL encryption
trust check: localHostTrusted = false, remoteHostTrusted = false
2020214210@thumm01:~$ hadoop fs -cat /dsjxtjc/2020214210/test.txt
2021-10-11 20:15:26,045 INFO sasl.SaslDataTransferClient: SASL encryption
trust check: localHostTrusted = false, remoteHostTrusted = false
Hello Hadoop
```

可以看到文件以及传输到DFS 上。`copyFromLocal/copyToLocal`用于本地文件系统与DFS之间文件的复制，`moveFromLocal/moveToLocal`用于本地文件系统与DFS 之间文件的移动，这些指令的详细用法可以使用`-help`指令查看，例如我们想了解`copyFromLocal`的用法：

```
2020214210@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_).
```

可以看到该指令有两个必填参数，第一个参数是本地路径，第二个参数是DFS 路径。

2. 通过Web 查看Hadoop 运行情况

在**本地**运行如下命令（将服务器的9870 端口映射到本地的9870 端口）：

```
(base) ➜  ~ ssh szxie@10.103.9.11 -L 9870:192.168.0.101:9870
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: Mon Oct 11 19:28:35 2021 from 10.61.237.158
szxie@thumm01:~$
```

在本地的浏览器中输入`localhost:9870`打开9870 端口，即可查看hadoop 运行情况，可通过此界面查看hadoop 的一些基本参数和job/task 的完成情况。

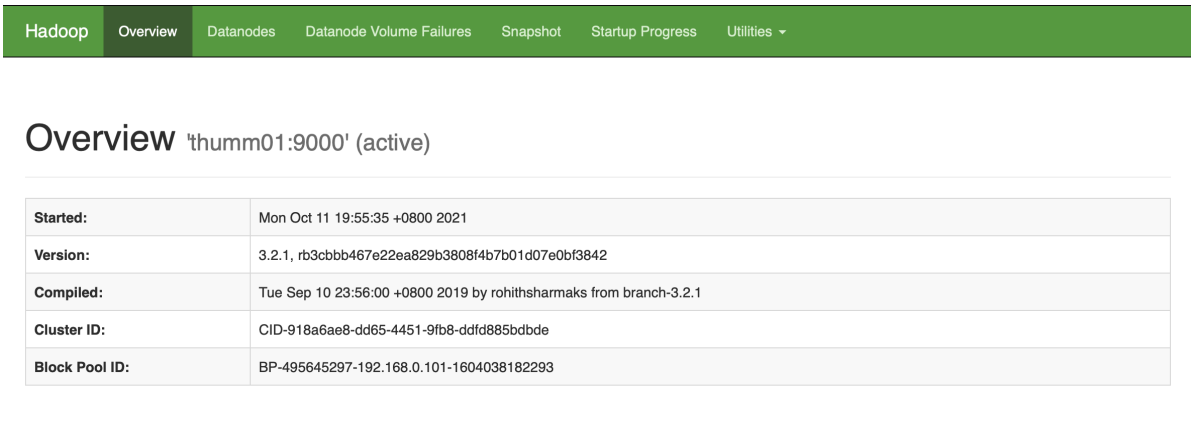


图1 Overview

图2 DataNode Information

四、分布式文件系统

GFS由一个`master`、多个`chunkserver`组成；用户通过`client`与GFS交互。以下用`NameNode`表示`master`，`DataNode`表示`chunkserver`。文件夹`MyDFS`中实现了一个简单的分布式文件系统，包含了NameNode、DataNode、Client三部分，其中NameNode负责记录文件块的位置（FAT表），DataNode负责数据的存储与读取，而Client则是用户与分布式文件系统交互的接口，详细原理请参考理论课内容和相关论文。（建议大家结合论文`The Google File System`或网上关于GFS的讲解来做这道题）

请根据提示和题设补全、修改代码，保证系统能完成以下指令：

- ls <dfs_path> : 显示当前目录/文件信息
- copyFromLocal <local_path> <dfs_path> : 从本地复制文件到DFS
- copyToLocal <dfs_path> <local_path> : 从DFS复制文件到本地
- rm <dfs_path> : 删除DFS上的文件
- format : 格式化DFS

0. MyDFS信息

0.0 目录结构

- MyDFS : 根目录
  - dfs : DFS文件夹，用于模拟DFS文件系统
    - name : 存放NameNode数据
    - data : 存放DataNode数据
  - test : 存放测试样例
    - test.txt
  - common.py ：全局变量
  - name_node.py : NameNode程序
  - data_node.py : DataNode程序
  - client.py : Client程序，用于用户与DFS交互

0.1 模块功能

- name_node.py
  - 保存文件的块存放位置信息
  - 获取文件/目录信息
  - get_fat_item： 获取文件的FAT表项
  - new_fat_item： 根据文件大小创建FAT表项
  - rm_fat_item： 删除一个FAT表项
  - format: 删除所有FAT表项
- data_node.py
  - load 加载数据块
  - store 保存数据块
  - rm 删除数据块
  - format 删除所有数据块
- client.py
  - ls : 查看当前目录文件/目录信息
  - copyFromLocal : 从本地复制数据到DFS
  - copyToLocal： 从DFS复制数据到本地
  - rm： 删除DFS上的文件

- format： 格式化DFS

0.2 操作示例

0. 进入MyDFS目录，**根据注意点1修改端口号**并执行如下命令

```
$ cd MyDFS/test
$ cp test.txt test_copyFromLocal.txt
```

1. 新建终端1，启动NameNode

```
$ python3 name_node.py
```

2. 新建终端2，启动DataNode

```
$ python3 data_node.py
```

3. 新建终端3，使用copyFromLocal指令（下面会解释该指令作用）

```
$ python3 client.py -copyFromLocal ./test/test_copyFromLocal.txt
test_copyFromLocal.txt
File size: 8411
Request: new_fat_item test_copyFromLocal.txt 8411
Fat:
blk_no,host_name,blk_size
0,localhost,4096
1,localhost,4096
2,localhost,219
```

其中blk_no为块号， host_name为该数据块存放的主机，blk_size为块的大小。这条指令作用是将./test/test_copyFromLocal.txt发送到dfs文件系统，文件系统将该文件切成三块，大小分别为4096, 4096, 219，并且都存储在localhost（thumm01）的dfs上。所以在./dfs/name和./dfs/data中会出现新文件。

1. copyFromLocal （例）

`copyFromLocal`的功能是将本地文件传到DFS之中。具体来说，`client`会把文件信息通过`new_fat_item`指令给NameNode，NameNode根据文件大小分配空间，并将相应空间信息以FAT表的形式返回给`client.py`（详见`name_node.py`中的`new_fat_item`函数）；接着，Client 根据FAT表和目标节点逐个建立连接发送数据块。

请注意，我们是用MyDFS/dfs文件夹来模拟文件系统，可视为该文件夹不可直接访问，必须通过接口函数（ls, copyFromLocal, copyToLocal, rm, format）进行操作。

```
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)
        for idx, row in fat.iterrows():
            data = fp.read(int(row['blk_size']))

            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段时间，避免粘包
            data_node_sock.send(bytes(data, encoding='utf-8'))
            data_node_sock.close()
        fp.close()
```

2. copyToLocal（2 分）

`copyToLocal` 是 `copyFromLocal` 的反向操作，请参考例题和 `name_node.py` 中的 `get_fat_item` 和 `data_node.py` 中的 `load` 函数，补全 `client.py` 中的 `copyToLocal` 函数。

```
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
```

执行如下命令，代表将dfs上的test_copyFromLocal.txt文件下载到本地 `./test/test_copyToLocal.txt`

```
$ python3 client.py -copyToLocal test_copyFromLocal.txt ./test/test_copyToLocal.txt
Request: get_fat_item test_copyFromLocal.txt
Fat:
blk_no,host_name,blk_size
0,localhost,4096
1,localhost,4096
2,localhost,219
```

3. ls（2 分）

Client 会向NameNode 发送请求，查看 `dfs_path` 下的文件或文件夹信息，请完善 `client.py` 中的 `ls` 函数（如下），使其实现上述功能，并能打印错误（使用 `try...error` 语句）。

```
def ls(self, dfs_path):
    cmd = "ls {}".format(dfs_path)
    print("Request: {}".format(request))
    # TODO: 将cmd发送给name node，接收name node返回的文件信息并打印
```

执行结果如下

```
$ python3 client.py -ls test_copyFromLocal.txt
b'blk_no,host_name,blk_size\n0,localhost,4096\n1,localhost,4096\n2,localhost,219\n'
```

4. rm（2 分）

`rm` 则是要删除相应路径的文件。请大家阅读 `name_node.py` 中的 `rm_fat_item` 和 `data_node.py` 中的 `rm` 函数补全 `client.py` 中的 `rm` 函数。

```
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；打印FAT表；根据FAT表逐个告诉目标DataNode删除对应数据块
```

执行结果如下，在./dfs/name和./dfs/data中的文件被删除

```
$ python3 client.py -rm test_copyFromLocal.txt
Request: rm_fat_item test_copyFromLocal.txt
Fat:
blk_no,host_name,blk_size
0,localhost,4096
1,localhost,4096
2,localhost,219

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

5. data replication（3 分）

目前common.py中DFS_REPLICATION为1，意为每个数据块只存储在一台主机上。实际上从系统稳定性考虑，每个数据块会被存放在多台主机。请修改DFS_REPLICATION和HOST_LIST，以及namenode.py、datanode.py、client.py中对应的部分，实现多副本块存储。推荐 `dfs_replicatio=3`，在5台机器上测试。

执行结果如下（注意要先将MyDFS拷贝到每个主机，都要启动datanode），报告中需要截图表示每个节点都能找到对应数据块

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

```
2,thumm01,219
2,thumm02,219
```

☆注意点

- `common.py`中的`DATA_NODE_PORT`和`NAME_NODE_PORT`可以修改为学号后四位和后五位，以免端口冲突；如果出现端口占用，可以使用命令`lsof -i:<port>`查看占用该端口的进程，如果是上次执行时进程没有关闭可以直接kill掉，如果是其他同学占用建议换个其他端口；
- 实验中可能会碰到粘包现象，可使用`time.sleep`处理；
- 所有任务提供的执行结果仅供参考，不要求完全一致，最后的报告中包含必要内容即可；
- 建议尽早开始实验，临近ddl机器应该会很卡。

五、MapReduce

请在分布式系统MyDFS的基础上搭建MapReduce框架，实现矩阵乘法计算。

0. 矩阵乘法原理

设矩阵A形状为(m,p)，矩阵B形状为(p,n)，$C=AB$为矩阵A与B的乘积，形状为(m,n)，其中矩阵C中的第i行第j列元素c_{ij}可以表示为

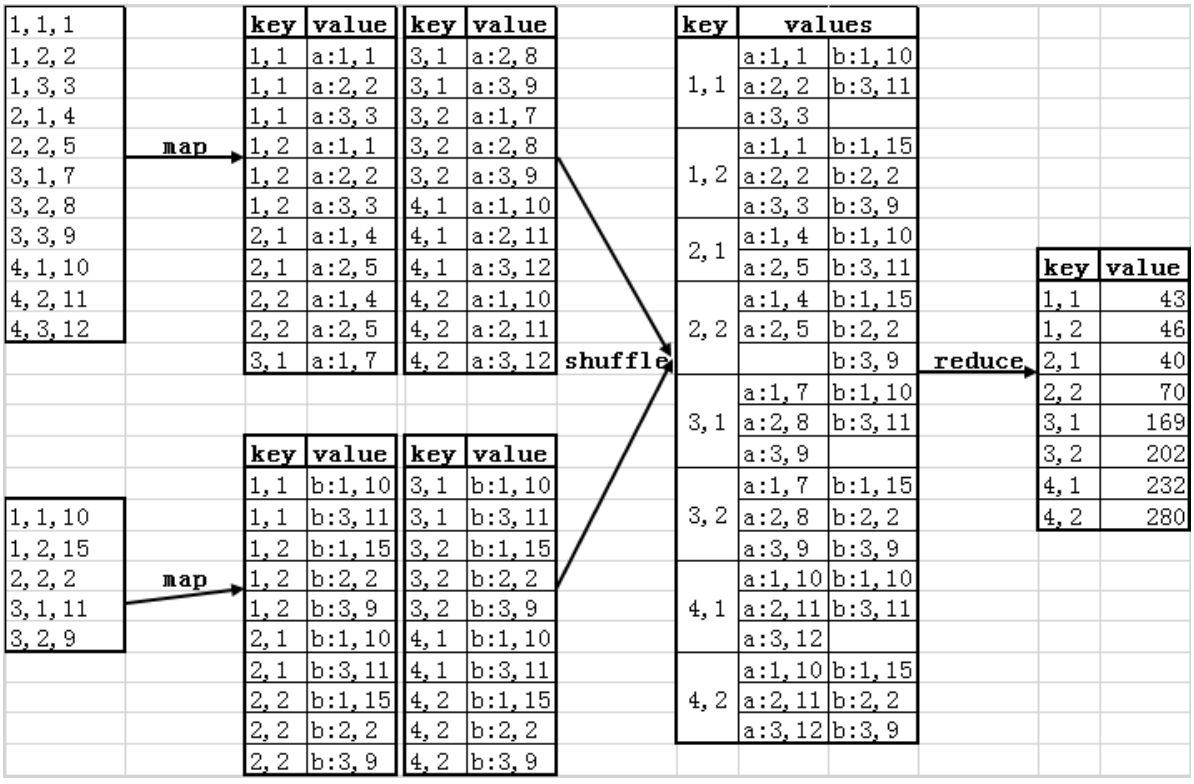
$$c_{ij}=\sum_{k=1}^{p}a_{ik}b_{kj}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{ip}b_{pj}$$

1. MapReduce实现矩阵乘法（5分）

以如下矩阵乘法为例

$$A=\begin{bmatrix}1&2&3\\4&5&0\\7&8&9\\10&11&12\end{bmatrix},B=\begin{bmatrix}10&15\\0&2\\11&9\end{bmatrix},C=AB=\begin{bmatrix}43&46\\40&70\\169&202\\232&280\end{bmatrix}$$

MapReduce处理流程如下



Map阶段
对于矩阵$A=(a_{ij})_{mp}$，每个元素a_{ij}需要参与n次计算。因此标识成n条$<key,value>$形式，
$key=(i,k),k=1,2,\dots,n,value=('a',j,a_{ij})$
对于矩阵$B=(b_{ij})_{pn}$，每个元素b_{ij}需要参与m次计算。因此标识成m条$<key,value>$形式，
$key=(k,j),k=1,2,\dots,m,value=('b',i,b\ ij\ )$

经过处理，用于计算c_{ij}需要的a,b就转变为有相同$key(i,j)$的数据对，通过$value$中$'a'$、$'b'$能区分元素是来自矩阵A还是矩阵B，以及具体的位置（在矩阵A的第几列，在矩阵B的第几行）。

Shuffle阶段
相同key的value会被加入到同一个列表中，形成$<key,list(value)>$对，传递给Reduce。

Reduce阶段
通过Map数据预处理和Shuffle数据分组两个阶段，Reduce阶段只需要知道两件事就行：

1. $<key,list(value)>$对经过计算得到的是矩阵C的哪个元素?
   因为Map阶段对数据的处理，将key构造成(i,j)形式，即在矩阵C中的位置，第i行j列。
2. $list(value)$中每个value来自于矩阵A和矩阵B的哪个位置?
   这个也在Map阶段进行了标记，对于 $\mathrm{value}(x,y,z)$，只需要找到y相同的来自不同矩阵（即x分别为'a'和'b'）的两个元素，取z相乘再加和即可。具体数据结构形式可参考向量（或数组），计算点积并求和。

☆注意点

- Map，Shuffle，Reduce三个操作可以相互独立也可以包含，只要给出的方法体现MapReduce的思想即可。比如可以只实现Map和Reduce函数(类)，Shuffle包含在Map或Reduce的实现中；
- Shuffle阶段实现较为自由，可以基于内存或中间文件等其他形式，选择你喜欢的就好，Map和Reduce阶段同理。

☆要求

请参考指导书给出的数据集格式，自行进行数据的生成以及运行结果正确性的验证，并在实验报告中给出生成数据的代码以及单机运行验证运行结果正确性的代码。

我们假定要计算的矩阵为 $C=AB$，其中 A 为 m 行 p 列，B 为 p 行 n 列。

考虑到mapreduce任务处理的数据量非常大，为了避免为服务器造成较大的负担，大家自行生成数据时按照 $m,p,n\leq 200$，A 和 B 中的非零元素不超过对应矩阵所有元素的千分之六（$6‰$）去考虑即可。

输入

输入文件给定若干行，每一行的格式为 `<Matrix>,<Row>,<Col>,<Number>`

其中 Matrix 为一个字母，只会是 `A` 或者 `B`，代表该元素来自矩阵 A 或者矩阵 B。

Row 代表该元素对应的行，Col 代表该元素对应的列，Number 为一个非零元素（为了简化实验要求和验证过程，只要是**正整数**即可）

对于矩阵 A 保证有 $1\leq \mathrm{Row}\leq m,1\leq \mathrm{Col}\leq p$

对于矩阵 B 保证有 $1\leq \mathrm{Row}\leq p,1\leq \mathrm{Col}\leq n$

为了考虑实验难度，我们 **不需要在输入文件中给出 m,p,n 的值**，将其放在你运行时输入的命令行指令即可。（但是如果有同学想进行尝试，那么将 m,p,n 额外放在输入文件的第一行也是可以的）

输出

按照 $\{\mathrm{Row}, \mathrm{Col}\}$ 的双关键字从小到大排序（即整体先对 Row 从小到大排序，在 Row 相同的情况下对 Col 进行从小到大排序），对矩阵 C 的 **全部非零元素** 进行输出。输出格式为 `<Row>,<Col>,<Number>`

需要注意的是，输入的时候可以不保证按照双关键字从小到大排序，也不需要保证 A 和 B 的非零元素不穿插出现。**但是一定要输入文件和输出文件均不出现重复位置的元素。**（例如：在输入文件同时出现两行 `A,3,4,5`，`A,3,4,15` 这样的情况）

命令行样例

假如我们需要运行发的 `py` 文件为 `client.py`，按照 m, p, n, 输入文件 , 输出文件形式的命令行格式进行输入，那么我们可以传入以下指令：

```
python3 client.py 4 3 2 input.txt output.txt
```

当然，这只是其中一种可供参考的方案，你也可以自行设置命令行格式，或者将 m, p, n 放在输入文件。

输入文件样例

这里只是根据上面的示意图，按照 A 和 B，以及行列双关键字排序之后的顺序给出的样例。如上所述，你不需要保证输入的有序性，只需要保证不出现重复位置的非零元素即可。事实上输入文件的有序性对于MapReduce的运行结果也不应当产生任何影响。

此外，该矩阵的12个元素中有11个非零的，在数据量增大之后，这样的稠密矩阵可能不适合进行实验。只需要按照上面的要求，自行生成稀疏矩阵即可。

```
A,1,1,1
A,1,2,2
A,1,3,3
A,2,1,4
A,2,2,5
A,3,1,7
A,3,2,8
A,3,3,9
A,4,1,10
A,4,2,11
A,4,3,12
B,1,1,10
B,1,2,15
B,2,2,2
B,3,1,11
B,3,2,9
```

输出文件样例

如上所述，最终的结果矩阵是对所有非零元素先按行再按列进行排序后再统一输出，这样可以简化正确性的比较。如果你的代码无法保证输出是按照该顺序进行排列、但是可以保证正确性的话，我们会在满分的基础上扣0.5分。

```
1,1,43
1,2,46
2,1,40
2,2,70
3,1,169
3,2,202
4,1,232
4,2,280
```

实验报告阐述&正确性比对

请在实验报告中详细叙述设计思想、数据分割方案、任务分配和整合方案等细节，并解释关键代码。最终的结果的**正确性**需要和单机的处理结果比对（建议数据集的生成以及单机处理也参照上述输入输出格式进行）。

六、Bonus

完成其中一项即可（2 分）

1. GFS中提到，NameNode会定期与每个DataNode进行交流，确认DataNode的状态，这个过程被称为`HeartBeat`，很多分布式系统都有这个功能。请大家修改`name_node.py`和`data_node.py`，添加`HeartBeat`功能，保证某个DataNode发生故障时NameNode能及时输出错误信息，并对丢失的文件进行备份，保证`data_replication`不变。请说明你的解决方案，解释相关代码并展示测试结果。实际测试中可以手动结束一个worker进程来模拟`DataNode`故障的现象。

   提示：可以考虑使用multiprocess module，NameNode应该维护所有DataNode的`回复时间`和`host_name`；在开启DataNode的同时开启轮询进程， 每隔一段时间需要收到DataNode的响应，否则认定该DataNode已经死去，当然还可以用其他方式比如NameNode每隔一段时间询问DataNode实现；

2. 请对比一下单机处理矩阵乘法以及你自己设计的`mapreduce`矩阵乘法的时间性能，并解释为什么会出现这样的情况。