电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2020091203003

姓 名 陈冠杰

（实验） 课程名称 大数据分析与智能计算

理论教师 罗瑜

实验教师 罗瑜

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：陈冠杰 学号：2020091203003指导教师：罗瑜**

**实验地点：xxxxxx 实验时间：XX.XX.XX**

**一、实验名称：实验1-环境安装及hadoop和spark比较**

**二、实验学时：4学时**

**三、实验目的：**

了解掌握hadoop与spark在Linux上的安装、配置以及实例运行

**四、实验原理：**

适用于 Linux 的 Windows 子系统 (WSL) 是 Windows 的一项功能，可用于在 Windows 计算机上运行 Linux 环境，而无需单独的虚拟机或双引导。 WSL 旨在为希望同时使用 Windows 和 Linux 的开发人员提供无缝高效的体验。WSL的优点包括 Windows 和 Linux 之间的无缝集成，启动时间短，资源占用量少，并且无需 VM 配置或管理，等等。学生曾用WSL作为在Windows系统下开发Linux项目的主要工具，如学生在企业实习环节参与的LarkSDK项目。

Hadoop 的设计思路来源于 Google 的 GFS 和 MapReduce。它是一个开源软件框架，通过在集群计算机中使用简单的编程模型，可编写和运行分布式应用程序处理大规模数据。

Spark 是一个基于内存计算的开源的集群计算系统，目的是让数据分析更加快速。Spark 提供了基于内存的计算集群，在分析数据时将数据导入内存以实现快速查询，“速度比”基于磁盘的系统，如比 Hadoop 快很多。Spark 最初是为了处理迭代算法，如机器学习、图挖掘算法等，以及交互式数据挖掘算法而开发的。在这两种场景下，Spark 的运行速度可以达到 Hadoop 的几百倍。

**五、实验内容：**

hadoop单机模式安装、Spark安装

测试安装

使用Hadoop MapReduce、Spark框架分别运行wordcount分析程序，来对MapReduce和Spark的性能进行对比。

**六、实验器材（设备、元器件）：**

适用于 Linux 的 Windows 子系统 (WSL)

jdk-8u391-linux-x64.tar.gz

hadoop-3.3.6.tar.gz

spark-3.5.0-bin-hadoop3.tgz

本实验使用的数据集，保存成word.txt文件

Python(3.7以上版本)

**七、实验步骤：**

一、Hadoop安装配置

需要先添加用来运行Hadoop进程的用户组hadoop及用户hadoop。

**1、添加用户及用户组**

创建用户和用户组hadoop

$ sudo mkdir -p /hadoop

$ sudo groupadd hadoop

$ sudo useradd -g hadoop -G hadoop -d /hadoop hadoop

$ sudo chown -R hadoop:hadoop /hadoop

$ sudo usermod -s /bin/bash hadoop

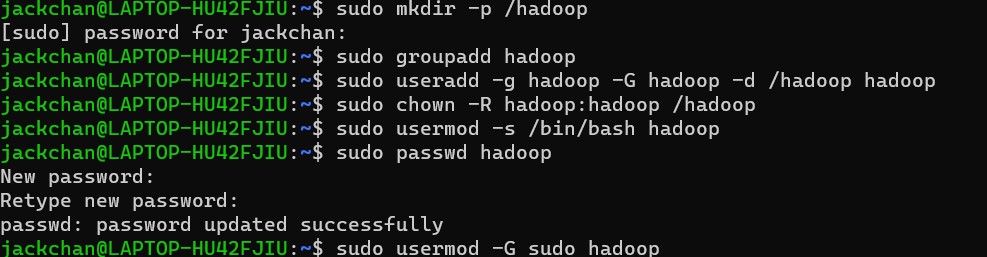
按照提示输入hadoop用户的密码，例如密码设定为hadoop。注意输入密码的时候是不显示的。

**2、添加sudo权限**

将hadoop用户添加进sudo用户组：

$ sudo usermod -G sudo hadoop

上述步骤执行结果如下

****

**3、安装及配置依赖的软件包**

（1）hadoop环境需要预安装openssh-server、java等，这些软件包在实验环境中如果没有，需要手工安装。

更新 Ubuntu:

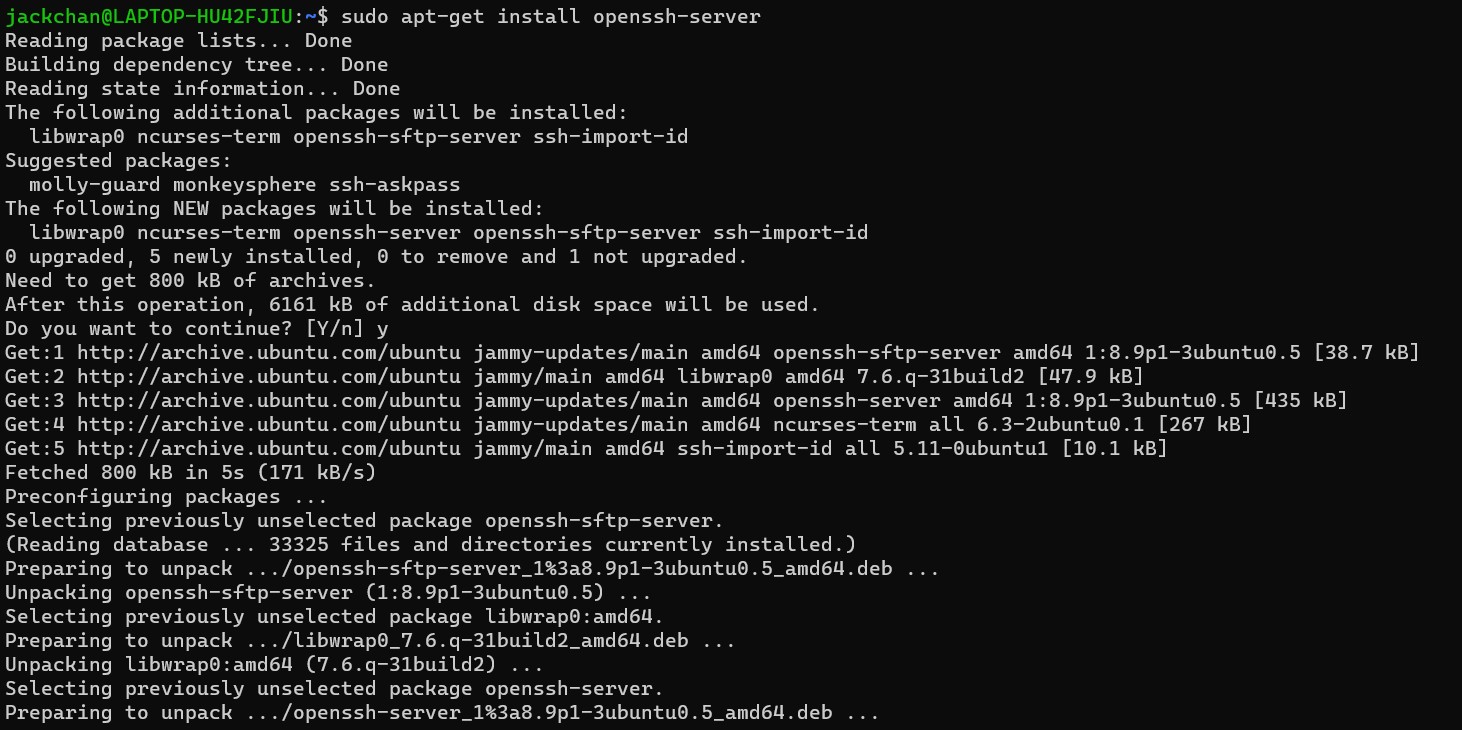
$ sudo apt-get update

$ sudo apt-get upgrade

安装并启动openssh-server：

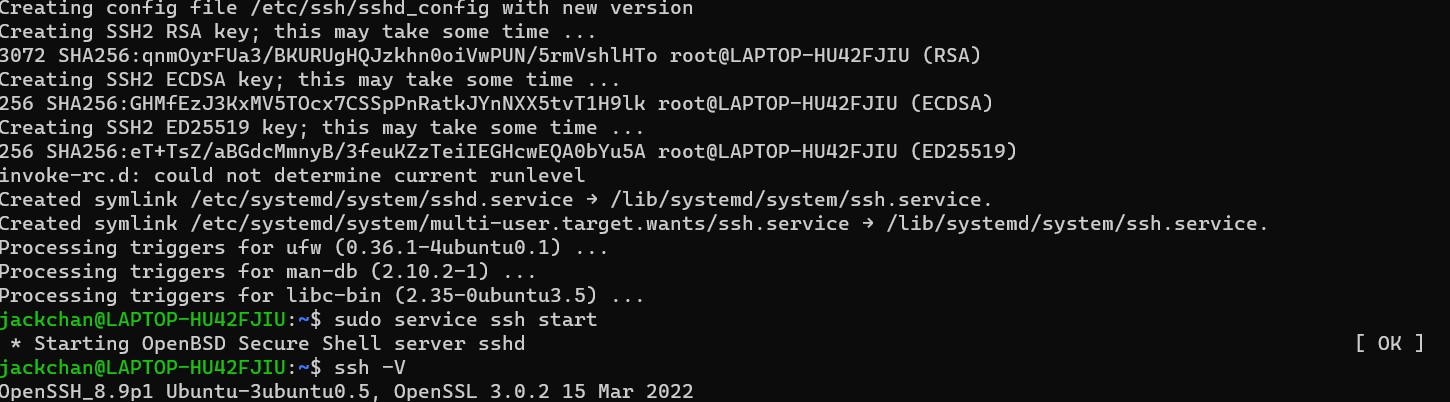
$ sudo apt-get install openssh-server -y

$ sudo service ssh start

****

验证环境执行下列指令：

$ ssh -V

****

（2）配置ssh免密码登录

在配置ssh免密登录之前，首先修改ssh的配置文件。

$ sudo nano /etc/ssh/sshd\_config

搜索“PermitRootLogin prohibit-password”，如果此句被注释则打开注释。

搜索“PubkeyAuthentication yes”，如果此句被注释则打开注释。

sudo nano /etc/ssh/sshd\_config

检查

PermitRootLogin yes # 远程root密码登录开启

StrictModes no

RSAAuthentication yes

PubkeyAuthentication yes

AuthorizedKeysFile .ssh/authorized\_keys

sudo nano /etc/ssh/ssh\_config

增加一句

PubkeyAcceptedKeyTypes +ssh-rsa

然后重启sshd 服务：

service sshd.service restart

随后配置ssh的免密登录。切换到hadoop用户，需要输入添加hadoop用户时配置的密码。后续步骤都将在hadoop用户的环境中执行。

$ su - hadoop

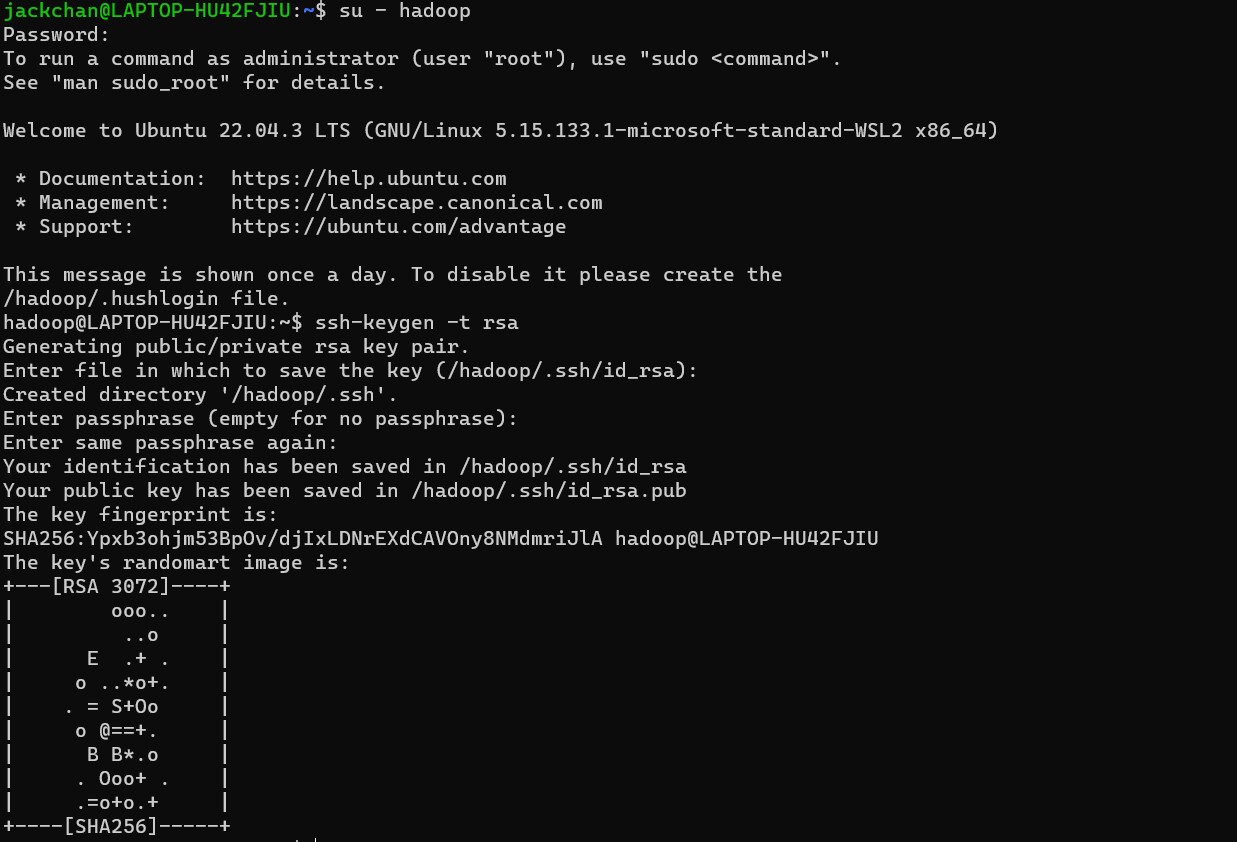
配置ssh环境免密码登录，在/hadoop目录下执行：

$ ssh-keygen -t rsa -P '' -f ~/.ssh/id\_rsa

$ cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

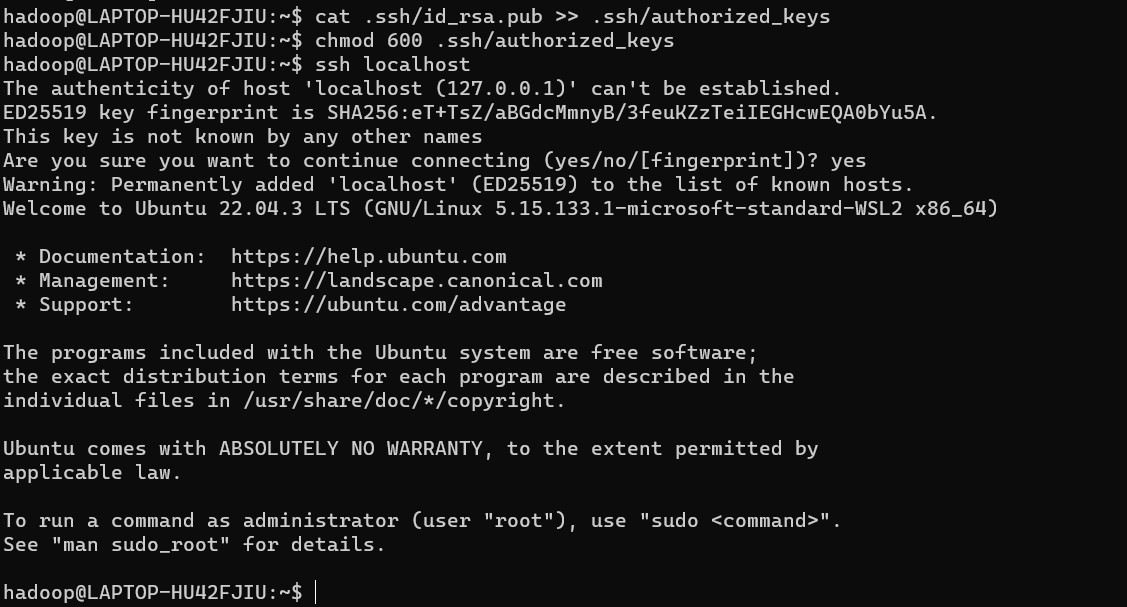
$ chmod 600 ~/.ssh/id\_rsa ~/.ssh/id\_rsa.pub

# 检查权限可用 ls -l -a命令



验证登录本机是否还需要密码，第一次需要输入yes，以后不需要密码就可以登录。

$ ssh localhost

****

验证完后退出远程登录

$ exit

**4、下载并安装JAVA和Hadoop**

开启WSL的镜像网络功能：在Windows的 %UserProfile%目录下(C:\Users\<UserName>)创建.wslconfig 文件并设置 networkingMode=mirrored，以启用镜像模式网络。启用此功能会将 WSL 更改为全新的网络体系结构，将 Windows 上的网络接口“镜像”到 Linux 中，以添加新的网络功能并提高兼容性。开启镜像网络以后，WSL内的网络连接可以享受与Windows下相同的速率。

JAVA 安装包下载地址（jdk1.8.0\_XX最新版本）：

https://www.oracle.com/java/technologies/downloads/

Hadoop安装包下载地址：

https://archive.apache.org/dist/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz

开启镜像网络后，使用wget命令下载文件，下载完成后在hadoop用户登录的环境中进行下列操作：

（1）复制压缩包到/hadoop目录下(theses are all in root dir)

$ sudo cp /home/xxx/Desktop/hadoop-3.3.6.tar.gz /hadoop/

$ sudo cp /home/xxx/Desktop/jdk-8u391-linux-x64.tar.gz /hadoop/

$ sudo cp /home/xxx/Desktop/spark-3.5.0-bin-hadoop3.tgz /hadoop/

（2）解压并安装

$ cd /hadoop

$ sudo chmod 777 /hadoop

$ tar -zxvf hadoop-3.3.6.tar.gz

$ tar -zxvf jdk-8u391-linux-x64.tar.gz

（3）打开一个新的终端并配置Hadoop环境变量

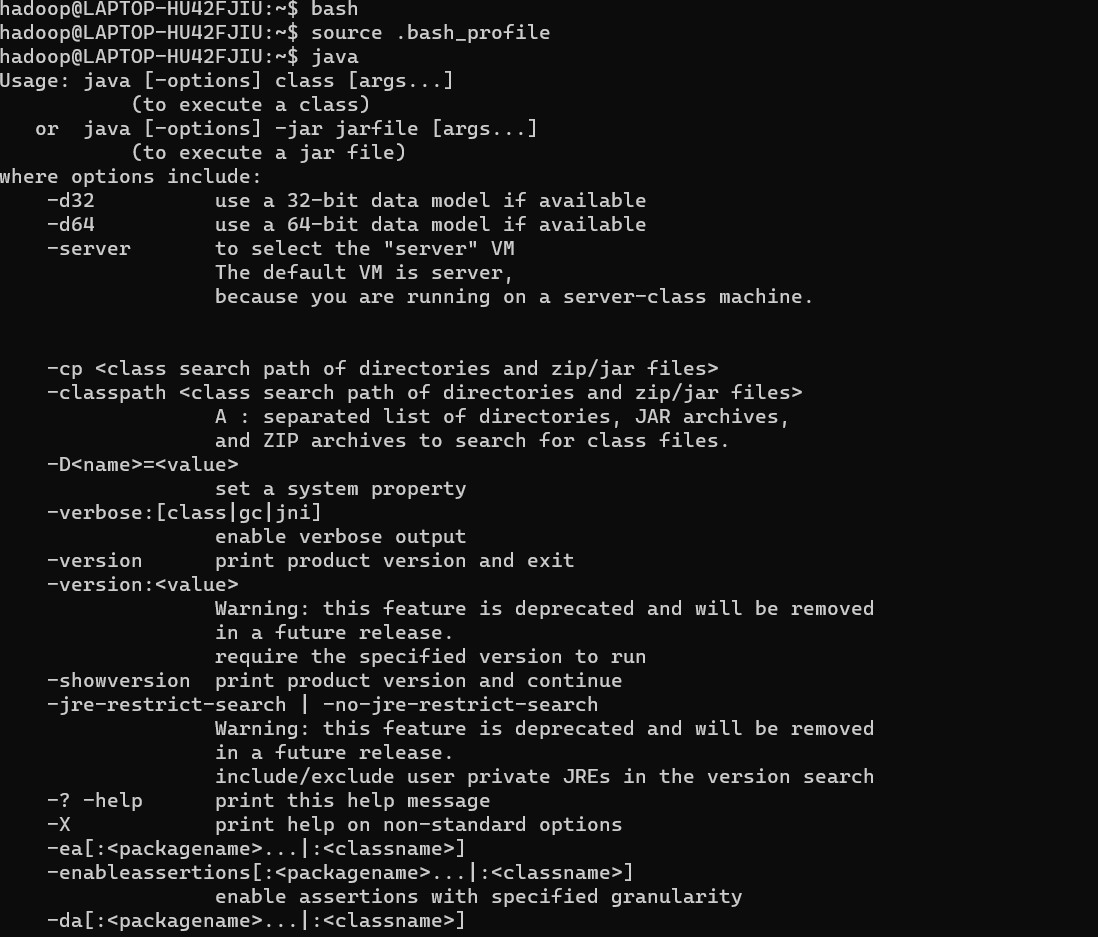
$ sudo nano /hadoop/.bash\_profile

增加以下内容：

|  |
| --- |
| #HADOOP START  export JAVA\_HOME=/hadoop/jdk1.8.0\_391  export HADOOP\_HOME=/hadoop/hadoop-3.3.6  export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin:$JAVA\_HOME/bin  export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME  export HADOOP\_COMMON\_HOME=$HADOOP\_HOME  export HADOOP\_HDFS\_HOME=$HADOOP\_HOME  export YARN\_HOME=$HADOOP\_HOME  export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native  export HADOOP\_OPTS="-Djava.library.path=$HADOOP\_HOME/lib"  #HADOOP END |

配置成功后，激活新加的环境变量

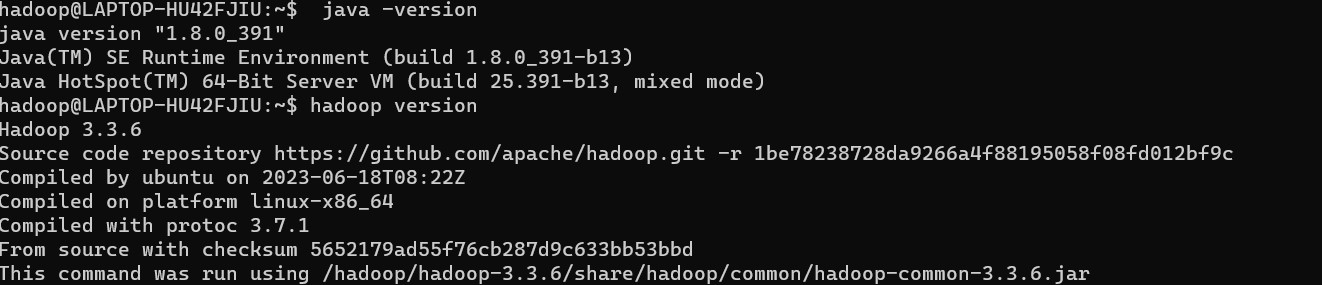
$ source ~/.bash\_profile



（4）查看JAVA和Hadoop版本

$ java -version

$ hadoop version



至此，Hadoop单机模式安装完成，可以通过下述步骤的测试来验证安装是否成功。

**5、实验结果**

创建输入的数据，暂时采用/etc/protocols文件作为测试。

$ mkdir /hadoop/hadoop\_project

$ cd /hadoop/hadoop\_project

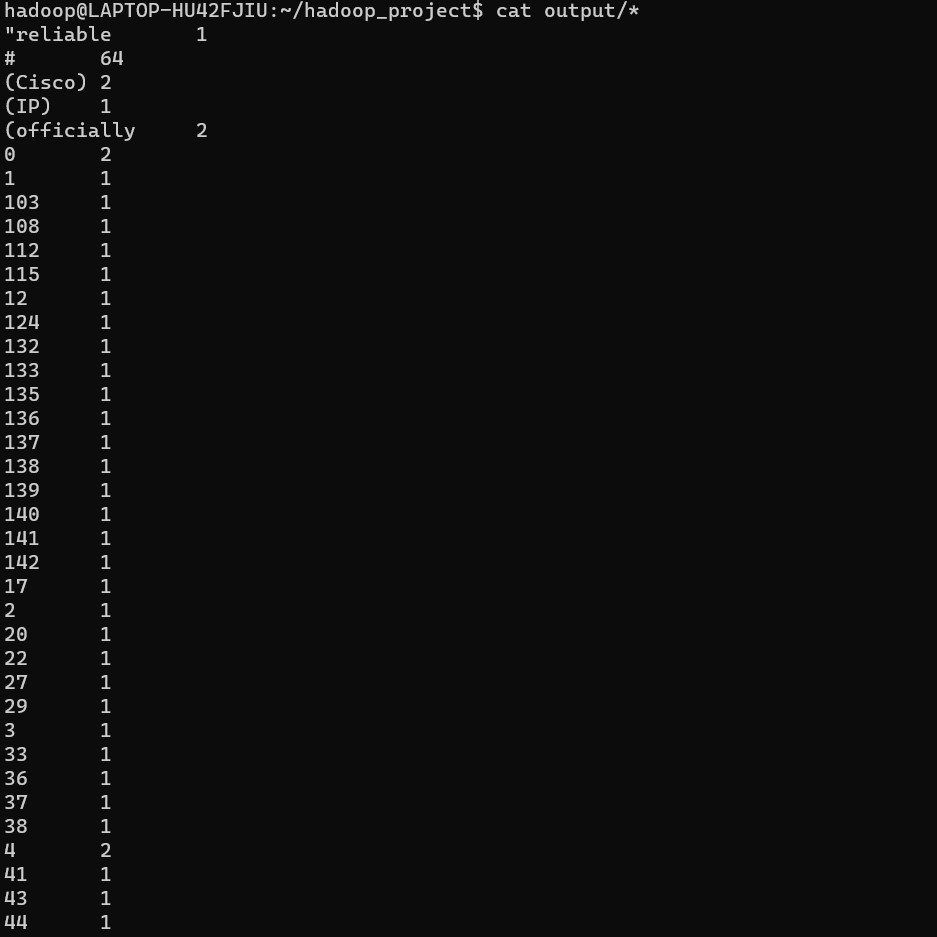
$ cp /etc/protocols input

执行Hadoop WordCount应用（词频统计）：

|  |
| --- |
| $ /hadoop/hadoop-3.3.6/bin/hadoop jar /hadoop/hadoop-3.3.6/share/hadoop/mapreduce/sources/hadoop-mapreduce-examples-3.3.6-sources.jar org.apache.hadoop.examples.WordCount input output |

查看生成的单词统计数据：

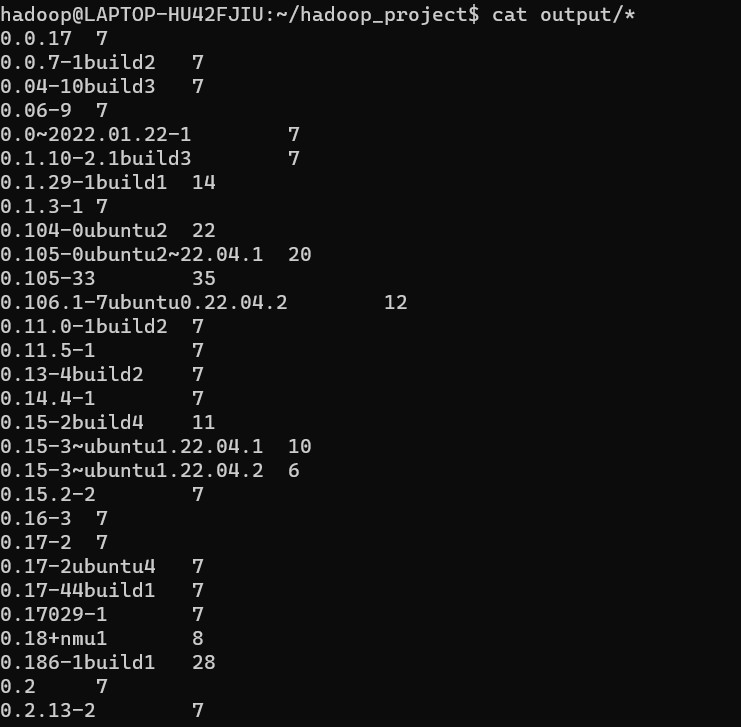
$ cat output/\*

****

**练习题**

请使用hadoop的wordcount对日志文件/var/log/dpkg.log进行词频统计。 将你执行的命令，和输出的结果粘贴到下面。

****

****

二、Hadoop伪分布式安装

**1、hadoop伪分布式配置**

hadoop的配置文件存放在/hadoop/hadoop-3.3.6/etc/hadoop下，要修改该目录下的文件core-site.xml和hdfs-site.xml来达到实现伪分布式配置。

在用户xxx（Ubuntu的初始登录账号）下，修改core-site.xml，将<configuration></configuration>修改为：

sudo nano /hadoop/hadoop-3.3.6/etc/hadoop/core-site.xml

|  |
| --- |
| <configuration>  <property>  <name>hadoop.tmp.dir</name>  <value>file:/hadoop/hadoop-3.3.6/tmp</value>  <description>Abase for other temporary directories.</description>  </property>  <property>  <name>fs.defaultFS</name>  <value>hdfs://localhost:9000</value>  </property>  </configuration> |

修改hdfs-site.xml，将<configuration></configuration>修改为：

sudo nano /hadoop/hadoop-3.3.6/etc/hadoop/hdfs-site.xml

|  |
| --- |
| <configuration>  <property>  <name>dfs.replication</name>  <value>1</value>  </property>  <property>  <name>dfs.namenode.name.dir</name>  <value>file:/hadoop/hadoop-3.3.6/tmp/dfs/name</value>  </property>  <property>  <name>dfs.datanode.data.dir</name>  <value>file:/hadoop/hadoop-3.3.6/tmp/dfs/data</value>  </property>  </configuration> |

配置完成后在/hadoop/hadoop-3.3.6下使用命令：

# 实现namenode的格式化

$ /hadoop/hadoop-3.3.6/bin/hdfs namenode -format

注意不要多次格式化，否则会在后面运行实例时报错“could only be written to 0 of the 1 minReplication nodes.”

解决方法如下：

1、 停止集群所有的服务。指令为：stop-all.sh

2、删除hdfs中配置的data目录下的所有文件（级core-site.xml中配置的hadoop.tmp.dir）。指令为：rm -rf /hadoop/hadoop-3.3.6/tmp/\*

3、重新格式化namenode。指令为：hadoop namenode -format

4、重新启动hadoop集群。指令为：start-all.sh

另外，需要修改/etc/hadoop/hadoop-env.sh中搜索并设置JAVA\_HOME。

sudo nano /hadoop/hadoop-3.3.6/etc/hadoop/hadoop-env.sh

|  |
| --- |
| export JAVA\_HOME=/hadoop/jdk1.8.0\_391 |

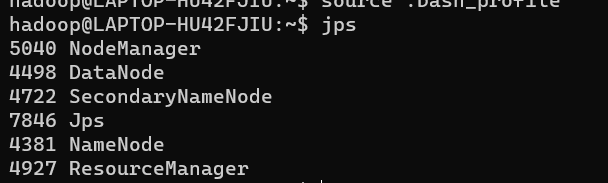
**2、启动hadoop（namenode节点）（start-all.sh在sbin里面）**

启动命令为：

$ /hadoop/hadoop-3.3.6/sbin/start-all.sh

检查是否运行成功，通过执行jps命令可以查看到hadoop的几个主要进程：

$ jps



三、安装与配置 Spark

**1、解压并安装Spark**

本次实验我们将spark安装在/hadoop下。下载安装包：

$ wget https://archive.apache.org/dist/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz

解压

$ cd /hadoop

$ tar -zxvf spark-3.5.0-bin-hadoop3.tgz

删除安装文件

$ rm -r spark-3.5.0-bin-hadoop3.tgz

修改文件名称

$ mv spark-3.5.0-bin-hadoop3 spark

**2、配置Hadoop环境变量**

在 Yarn 上运行 Spark 需要配置 HADOOP\_CONF\_DIR、 YARN\_CONF\_DIR 和 HDFS\_CONF\_DIR 环境变量

命令：

$ sudo nano /hadoop/.bash\_profile

在下面添加如下代码：

|  |
| --- |
| # SPARK START  export SPARK\_HOME=/hadoop/spark  export PATH=$PATH:$SPARK\_HOME/bin  # check the version of py4j  export PYTHONPATH=$SPARK\_HOME/python:$SPARK\_HOME/python/lib/py4j-0.10.9.7-src.zip:$PYTHONPATH  # if run pyspark on Ubuntu 20.04 LTS, it shows the error “python: command not found”  export PYSPARK\_PYTHON=python3  # SPARK END  export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop  export HDFS\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop  export YARN\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop |

保存关闭后，执行以下命令使得环境变量生效：

$ source /hadoop/.bash\_profile

**3、修改配置文件**

$ cd /hadoop/spark/conf/

$ cp spark-env.sh.template spark-env.sh

$ sudo nano /hadoop/spark/conf/spark-env.sh

在第一行“#!/usr/bin/env bash”下，写入以下内容

export SPARK\_MASTER\_HOST=127.0.0.1

export SPARK\_MASTER\_PORT=7077

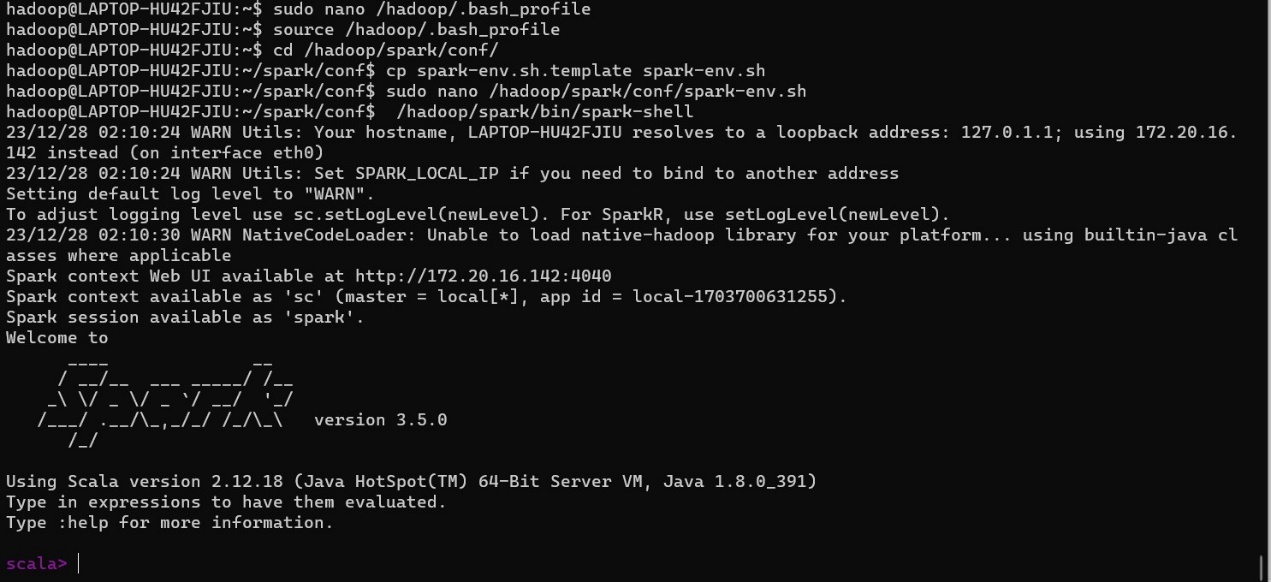
export SPARK\_WORKER\_CORES=1

export SPARK\_WORKER\_MEMORY=512M

**4、Spark的启动**

（1）进入spark-shell。进入Spark安装主目录，执行命令进入spark的shell界面：

$ /hadoop/spark/bin/spark-shell

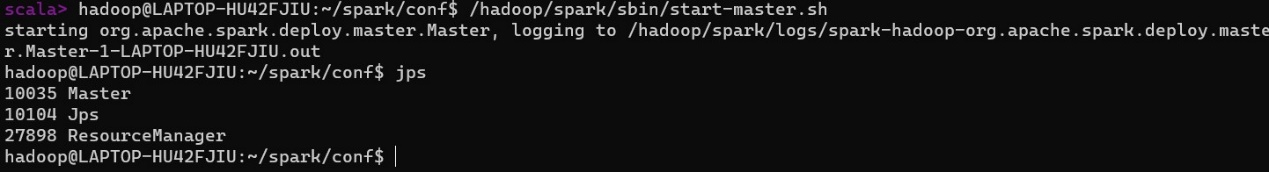


使用“CTRL+C”键退出spark-shell界面。

（2）启动spark

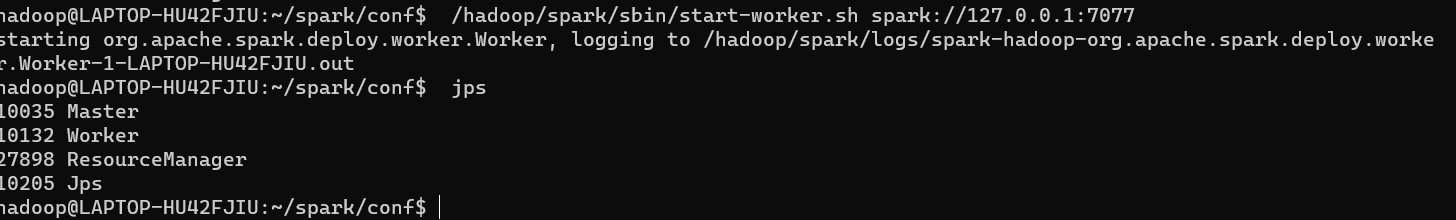
1）首先启动master

$ /hadoop/spark/sbin/start-master.sh

****

2）启动slave

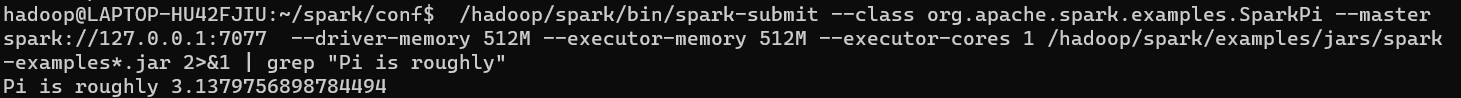
$ /hadoop/spark/sbin/start-worker.sh spark://127.0.0.1:7077

****

**5、验证Spark**

运行pi（）的实例

$ /hadoop/spark/bin/spark-submit --class org.apache.spark.examples.SparkPi --master spark://127.0.0.1:7077 --driver-memory 512M --executor-memory 512M --executor-cores 1 /hadoop/spark/examples/jars/spark-examples\*.jar 2>&1 | grep "Pi is roughly"

****

四、将本次实验的数据文件上传到HDFS文件系统

先建立一个/hadoop/data目录，把实验分析的word.txt文件放在此文件夹中。

$ mkdir /hadoop/data

$ cd /hadoop/data

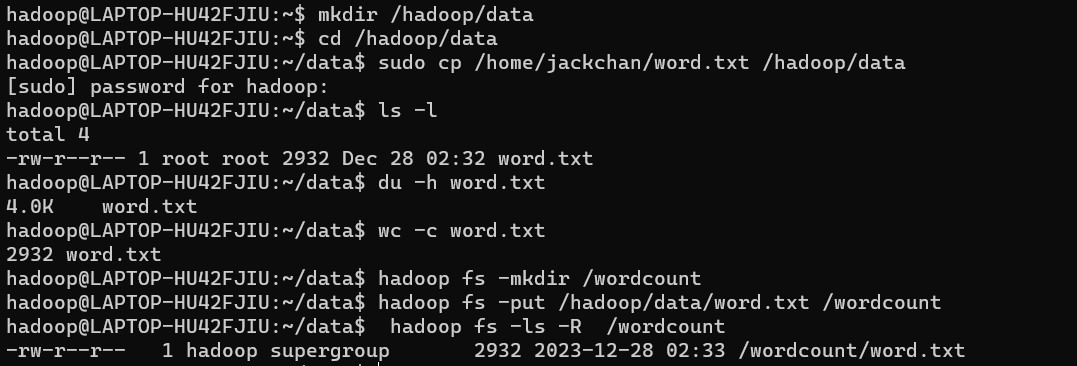
然后将word.txt上传至该目录下

**将文件上传到HDFS/wordcount：**

$ hadoop fs -mkdir /wordcount

$ hadoop fs -put /hadoop/data/word.txt /wordcount

$ hadoop fs -ls -R /wordcount

****

3、MapReduce实现WordCount实例（Python）

进入目录/hadoop/data。

**（1）首先编写MapReduce WordCount 代码。**

a. 编写map阶段的代码，创建一个Python程序，命名为“count\_mapper.py”，写入如下内容：

|  |
| --- |
| #!/usr/bin/env python3  import sys  # 从标准输入过来的数据  for line in sys.stdin:  # 将首位的空格去掉  line = line.strip()  # 将这一行文本切分成单词（按空格）  words = line.split()  # 读一个单词写出一个<单词,1>  for word in words:  print("%s\t%s" % (word, 1)) |

b. 编写Reduce阶段的代码，创建一个Python程序，命名为“count\_reducer.py”，写入如下内容：

|  |
| --- |
| #!/usr/bin/env python3  from operator import itemgetter  import sys  current\_word = None  current\_count = 0  word = None  # 从标准输入过来的数据  for line in sys.stdin:  # 去除左右空格  line = line.strip()  # 按照tab键进行切分，得到word和次数1  word, count = line.split('\t', 1)  # 得到的1是一个字符串，需要类型转化  try:  count = int(count)  except ValueError:  continue  # 如果本次读取的单词和上一次一样，对次数加1  if current\_word == word:  current\_count += count  else:  if current\_word:  # 输出统计结果  print ("%s\t%s" % (current\_word, current\_count))  current\_count = count  current\_word = word  # do not forget to output the last word if needed!  if current\_word == word:  print ("%s\t%s" % (current\_word, current\_count)) |

（2）程序编写完成后，首先在本地测试一下map和reduce，命令及图片如下：

$ head -20 /hadoop/data/word.txt | python3 count\_mapper.py | sort | python3 count\_reducer.py

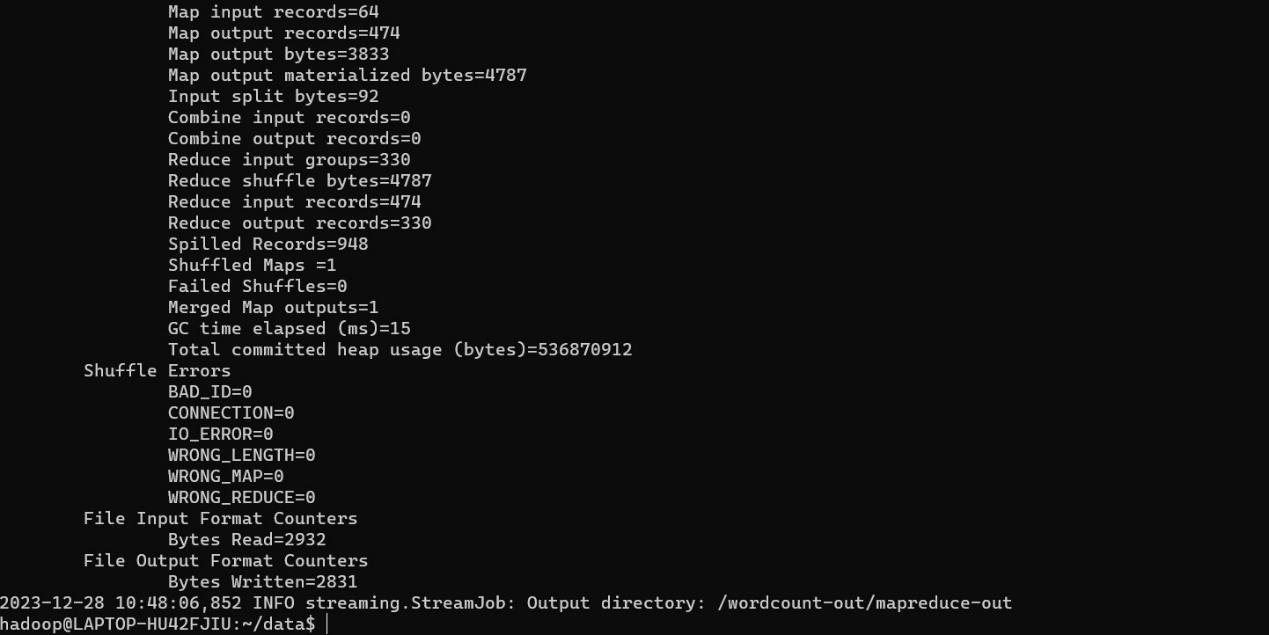
****

（3）运行该实例，命令如下：

$ hadoop jar /hadoop/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar -file count\_mapper.py -mapper count\_mapper.py -file count\_reducer.py -reducer count\_reducer.py -input /wordcount/word.txt -output /wordcount-out/mapreduce-out

（4）查看结果

$ hadoop fs -tail /wordcount-out/mapreduce-out/part-00000

****

4、Spark实现WordCount实例（python）

注意：使用spark-3.1.2-bin-hadoop3.2以上版本可能会提示“'NoneType' object has no attribute 'items'”错误，应该使用spark-3.5.0-bin-hadoop3版本。

（1）首先编写Spark WordCount 代码，创建一个Python程序，命名为“wordcount.py”，写入如下内容：

|  |
| --- |
| #!/usr/bin/env python3  #导入包  from pyspark import SparkContext  # 输入输出路径，输出路径不需要自己创建，系统会自动生成  inputFile = 'hdfs://localhost:9000/wordcount/word.txt'  outputFile = 'hdfs://localhost:9000/wordcount-out/spark-out'  sc = SparkContext('local', 'wordcount')  text\_file = sc.textFile(inputFile)  counts = text\_file.flatMap(lambda line: line.split(' ')).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)  counts.saveAsTextFile(outputFile) |

（2）运行该实例

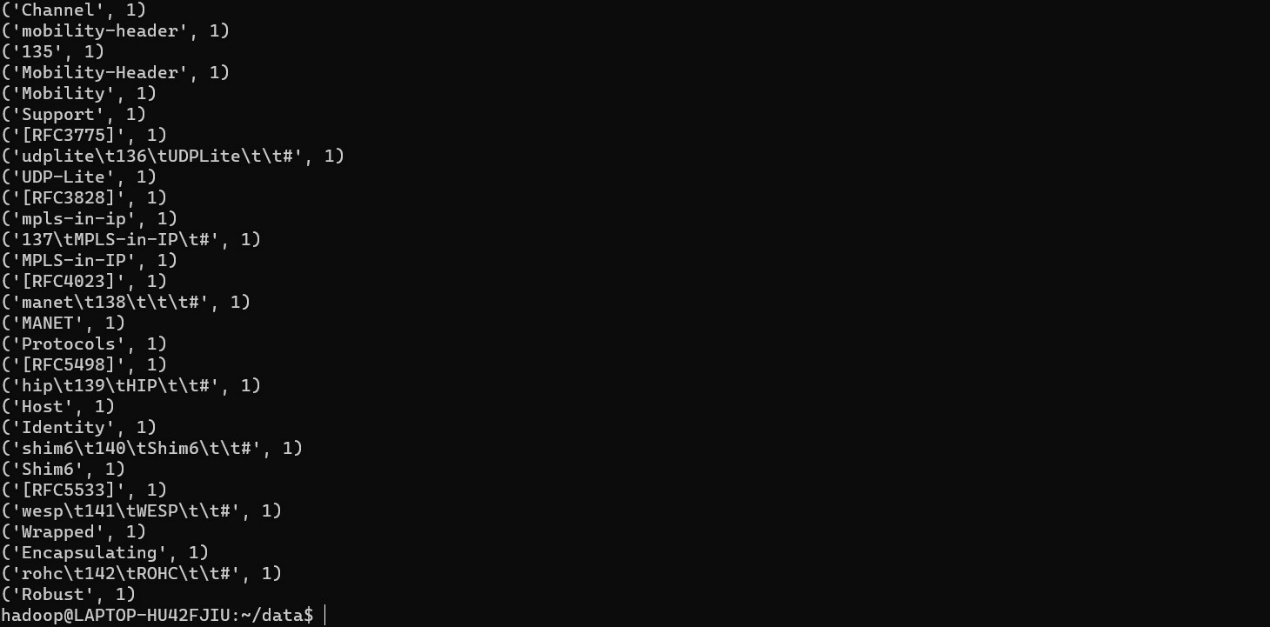
$ /hadoop/spark/bin/spark-submit --master spark://localhost:7077 /hadoop/data/wordcount.py

注：如果目录存在则先删除

$ hadoop fs -rm -r /wordcount-out/spark-out

（3）查看运行结果

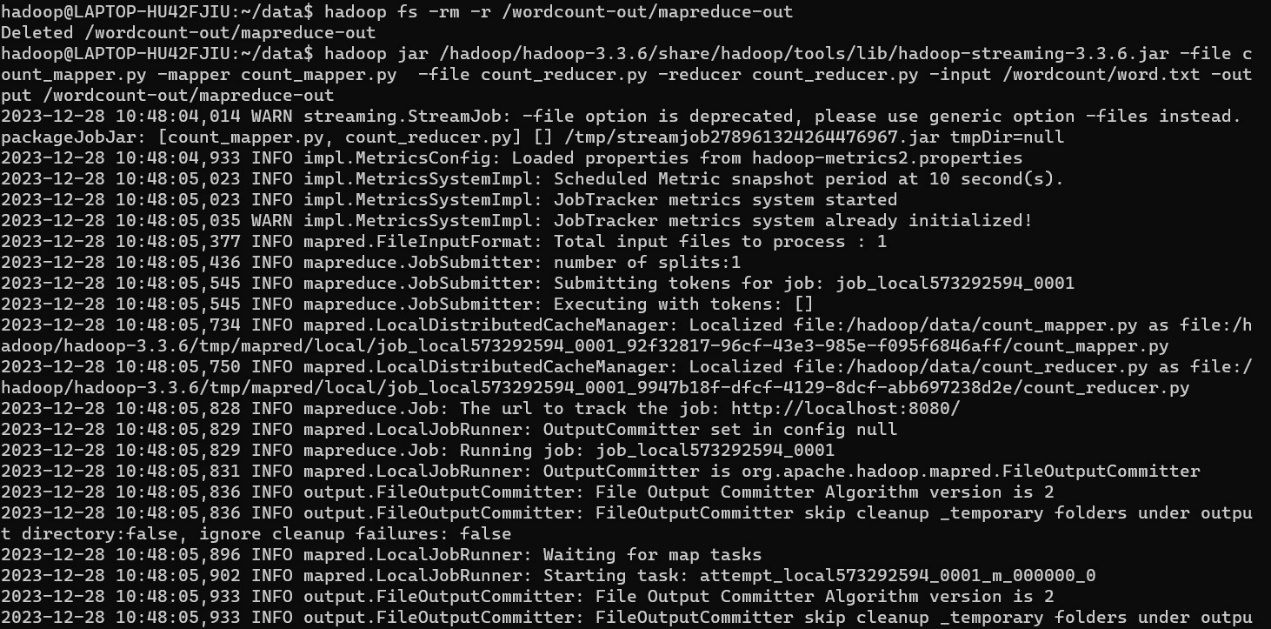
$ hadoop fs -tail /wordcount-out/spark-out/part-00000

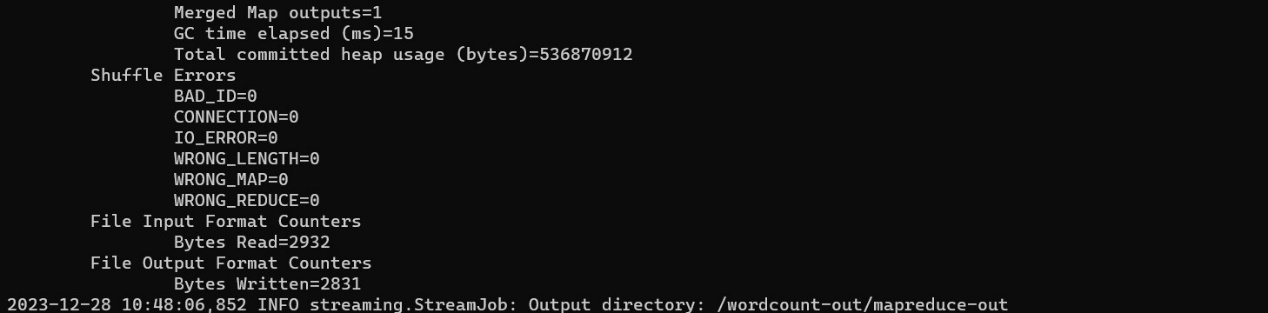
****

5、实验结果

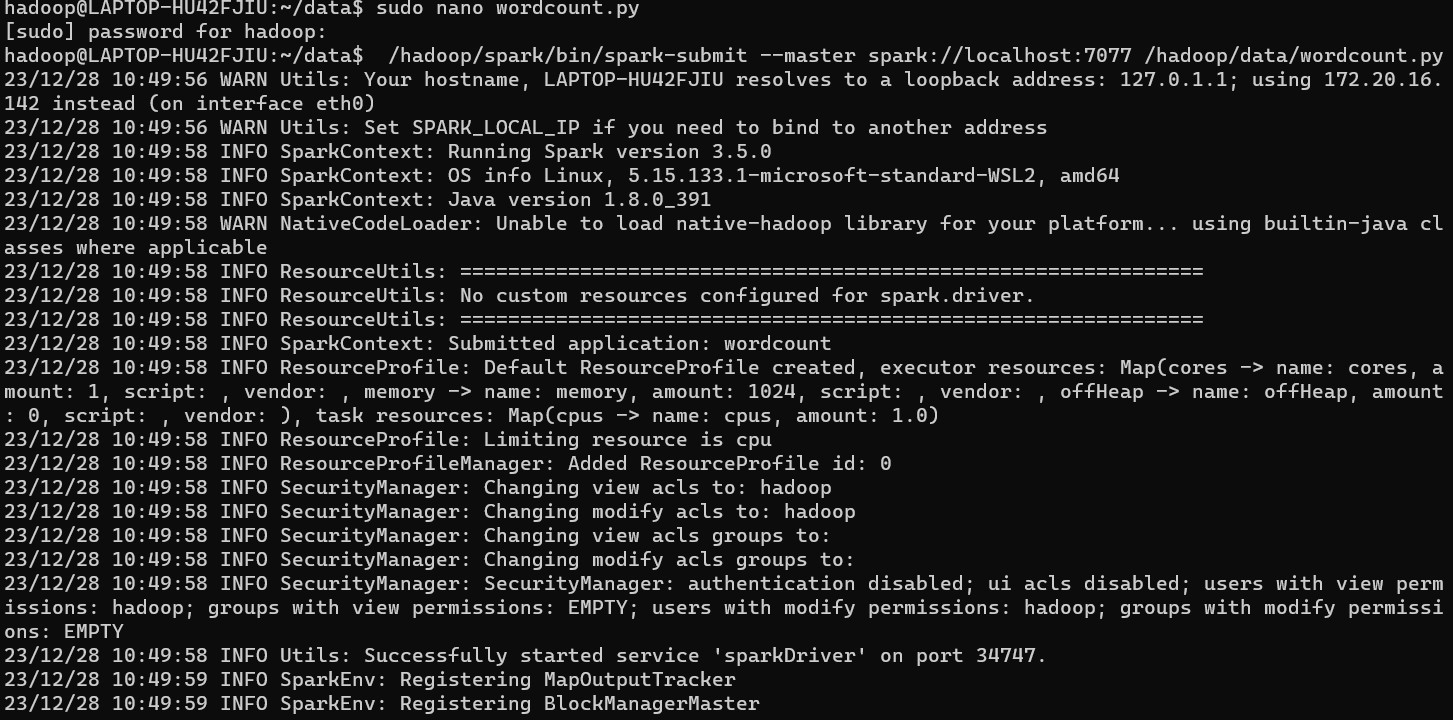
分析对比使用Hadoop的MapReduce和Spark两者的计算速度。

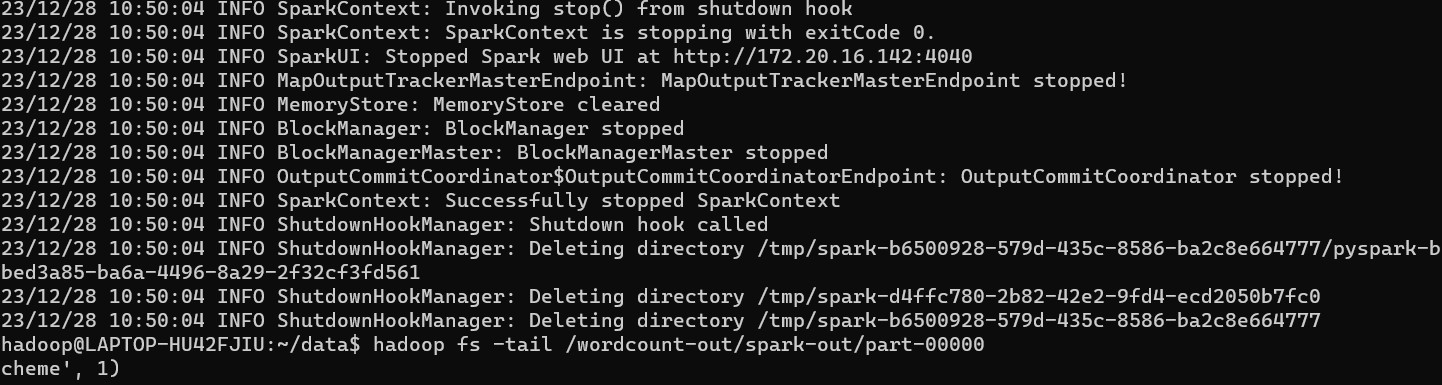
在命令行的回显中分别找到Hadoop与Spark命令运行的起止时间：

****

****

Hadoop命令运行的起止时间为10:48:04~10:48:06，约2s

****

****

Spark命令运行的起止时间为10:49:56~10:50:04，约6s

故在该实例下Hadoop的运行速度快于Spark

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

在该实例下，Hadoop的运行速度快于Spark

**九、总结及心得体会：**

本实验进行了hadoop单机模式安装、Spark安装、测试安装并使用Hadoop MapReduce、Spark框架分别运行wordcount分析程序，来对MapReduce和Spark的性能进行对比，学生收获颇丰。

**十、对本实验过程及方法、手段的改进建议：无**

**报告评分：**

**指导教师签字：**