# 实验1 Hadoop和Spark安装配置

1.1 实验介绍

1、实验内容

hadoop单机模式安装、Spark安装

测试安装

2、实验环境

jdk-8u351-linux-x64.tar

Hadoop3.3.4

spark-3.3.1-bin-hadoop3.tgz

3、适合人群

本课程难度为一般，属于初级级别课程，适合具有Linux基础的用户。

1.2 实验要求

Linux 环境已就绪（本示例采用的是 ubuntu-20.04.4环境）。

1.3 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

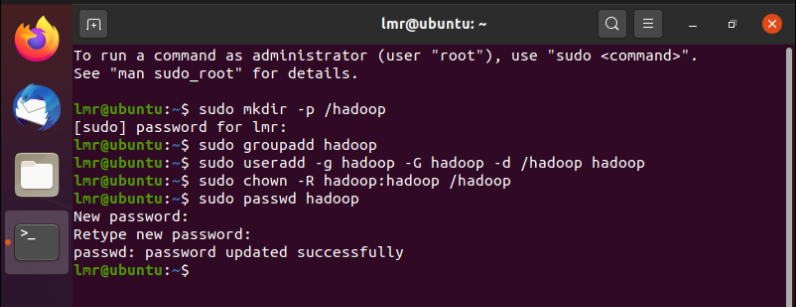
$ sudo chown -R hadoop:hadoop /hadoop

$ 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。

$ sudo passwd hadoop

|  |
| --- |
| Enter new UNIX password: hadoop  Retype new UNIX password: hadoop  passwd: password updated successfully |



**2、添加sudo权限**

将hadoop用户添加进sudo用户组：

$ sudo usermod -G sudo hadoop

**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:

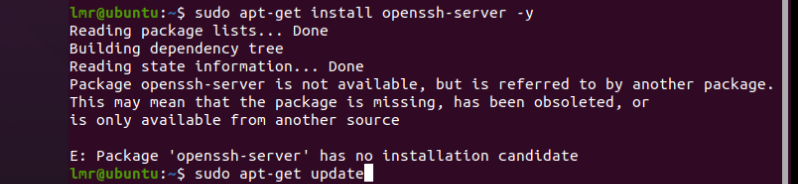
$ sudo apt-get update

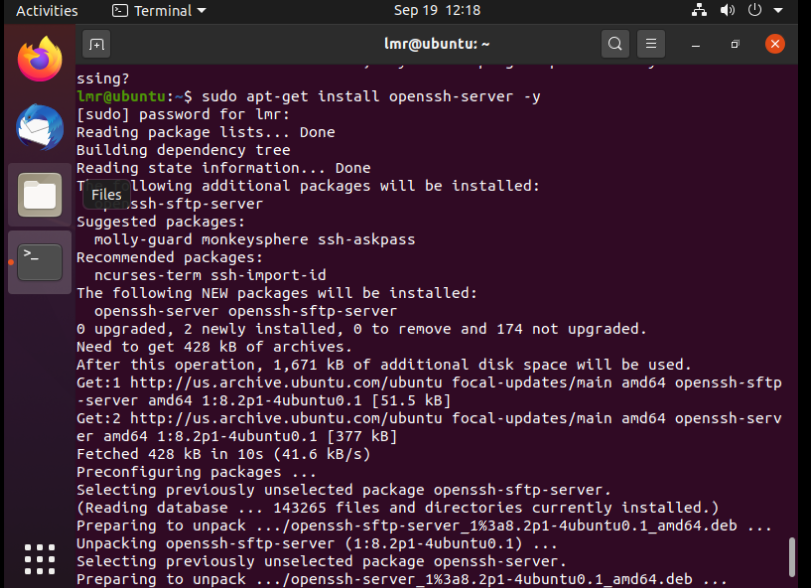
$ sudo apt-get upgrade

安装并启动openssh-verver：

$ sudo apt-get install openssh-server -y

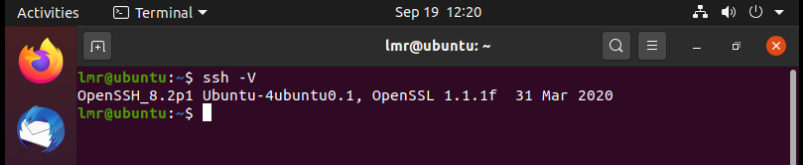
$ sudo service ssh start





验证环境执行下列指令：

$ ssh -V



（2）配置ssh免密码登录

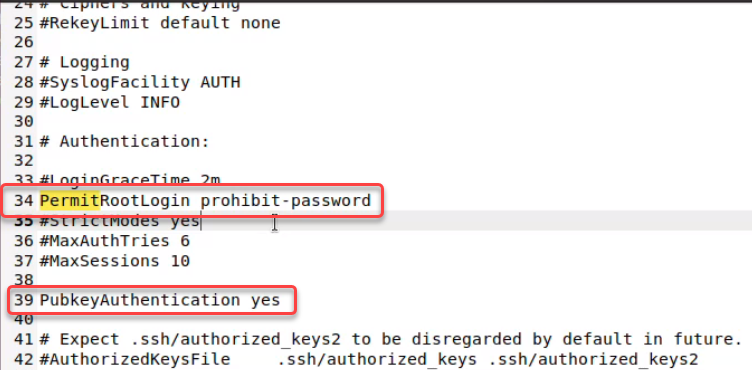
在配置ssh免密登录之前，首先修改ssh的配置文件。

$ sudo gedit /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。

完成后如下图所示：



然后重启sshd 服务：

systemctl restart sshd.service

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

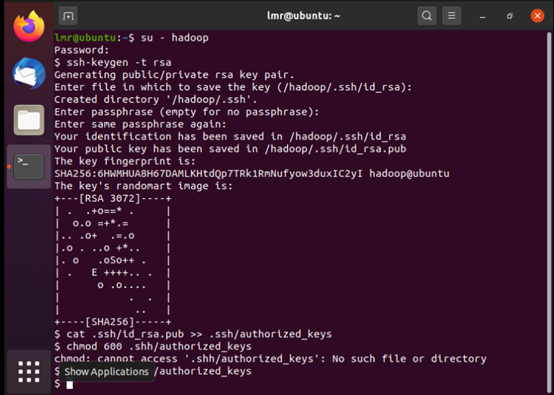
配置ssh环境免密码登录，在/hadoop目录下执行：

$ ssh-keygen -t rsa #一路回车

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

$ chmod 600 .ssh/authorized\_keys

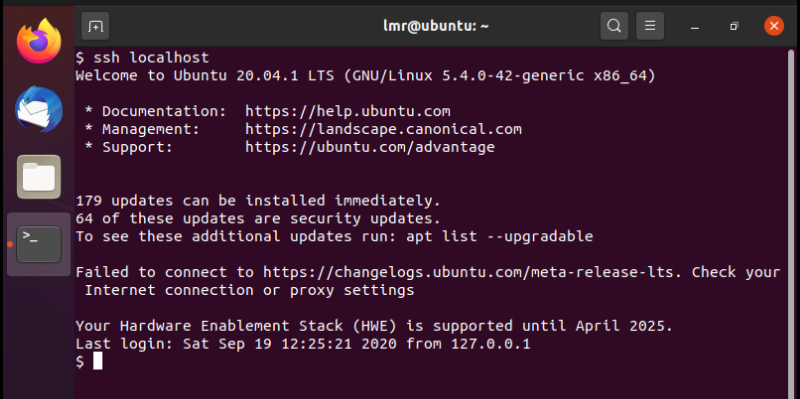
# 检查权限可用 ls -l -a命令



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

|  |
| --- |
| The authenticity of host 'localhost (::1)' can't be established.  ECDSA key fingerprint is 33:5d:12:e4:d5:59:8b:a3:a3:46:45:fd:16:f7:51:c8.  Are you sure you want to continue connecting (yes/no)? yes  Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.  The programs included with the Debian GNU/Linux system are free software;  the exact distribution terms for each program are described in the  individual files in /usr/share/doc/\*/copyright.  Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  permitted by applicable law. |



验证完后退出远程登录

$ exit

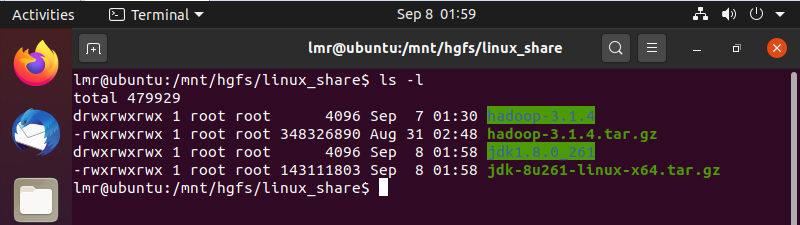
**4、下载并安装JAVA和Hadoop**

JAVA 安装包下载地址（jdk1.8.0\_XX最新版本）：https://www.oracle.com/java/technologies/downloads/#java8

Hadoop安装包下载地址：[https://archive.apache.org/dist/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz](https://archive.apache.org/dist/hadoop/common/hadoop-3.1.4/hadoop-3.1.4.tar.gz)

https://archive.apache.org/dist/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz

可以先下载好，存放在虚拟机的共享文件夹下。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-3.3.4.tar.gz /hadoop/

$ sudo cp /home/xxx/Desktop/jdk-8u351-linux-x64.tar.gz /hadoop/

$ sudo cp /home/xxx/Desktop/spark-3.3.1-bin-hadoop3.tgz /hadoop/

（2）解压并安装

$ cd /hadoop

$ tar -zxvf hadoop-3.3.4.tar.gz

$ chmod 755 /hadoop

$ tar -zxvf jdk-8u351-linux-x64.tar.gz

（3）open a new terminal and 配置Hadoop环境变量

$ sudo gedit /hadoop/.bash\_profile

增加以下内容：

|  |
| --- |
| #HADOOP START  export JAVA\_HOME=/hadoop/jdk1.8.0\_351  export HADOOP\_HOME=/hadoop/hadoop-3.3.4  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 |

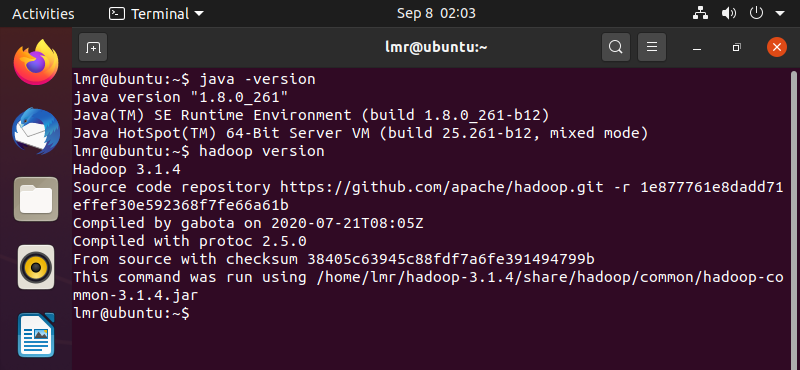
配置成功后，激活新加的环境变量

$ 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

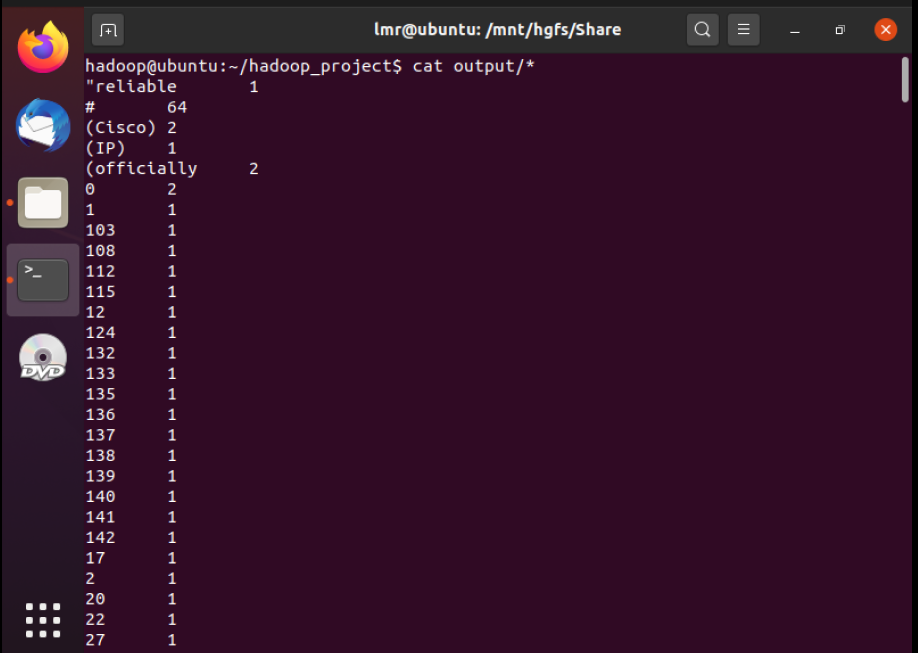
$ cp /etc/protocols input

执行Hadoop WordCount应用（词频统计）：

|  |
| --- |
| $ /hadoop/hadoop-3.3.4/bin/hadoop jar /hadoop/hadoop-3.3.4/share/hadoop/mapreduce/sources/hadoop-mapreduce-examples-3.3.4-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的文本框中。

**6、作业练习**

###输入你的作业代码###

###作业代码结束###

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-3.3.4/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 gedit /hadoop/hadoop-3.3.4/etc/hadoop/core-site.xml

|  |
| --- |
| <configuration>  <property>  <name>hadoop.tmp.dir</name>  <value>file:/hadoop/hadoop-3.3.4/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 gedit /hadoop/hadoop-3.3.4/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-3.3.4/tmp/dfs/name</value>  </property>  <property>  <name>dfs.datanode.data.dir</name>  <value>file:/hadoop/hadoop-3.3.4/tmp/dfs/data</value>  </property>  </configuration> |

配置完成后在/hadoop/hadoop-3.3.4下使用命令：

# 实现namenode的格式化

$ /hadoop/hadoop-3.3.4/bin/hdfs namenode -format

格式化失败注意权限问题“Sudo Unable to load native-hadoop library for your platform”。

$ chmod -R a+w /hadoop

$ /hadoop/hadoop-3.3.4/bin/hdfs namenode -formatls

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

sudo gedit /hadoop/hadoop-3.3.4/etc/hadoop/hadoop-env.sh

|  |
| --- |
| export JAVA\_HOME=/hadoop/jdk1.8.0\_351 |

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

启动命令为：

$ /hadoop/hadoop-3.3.4/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

|  |
| --- |
| ResourceManager  NodeManager  Jps  NameNode  SecondaryNameNode  DataNode |

注意：要配置了core-site.xml和hdfs-site.xml之后start-all.sh才会启动NodeManager和NameNode等。

如果，datanode启动失败，提示“Cannot set priority of datanode process”，可stop-all.sh后，修改以下文件再重启。

$ vi $HADOOP\_HOME/etc/hadoop/hadoop-env.sh

|  |
| --- |
| export HDFS\_DATANODE\_USER=hadoop  export HADOOP\_SECURE\_DN\_USER=hadoop  export HDFS\_NAMENODE\_USER=hadoop  export HDFS\_SECONDARYNAMENODE\_USER=hadoop  export YARN\_RESOURCEMANAGER\_USER=hadoop  export YARN\_NODEMANAGER\_USER=hadoop |

$ vi $HADOOP\_HOME/bin/hdfs

|  |
| --- |
| HADOOP\_SHELL\_EXECNAME="hadoop" |

如果jps命令后找不到namenode节点，还是配置文件的问题。

（1）提示Unable to load native-hadoop library for your platform 找不到库

$ hadoop checknative -a # 检查库（未解决）

（2）启动时报Error: JAVA\_HOME is not set and could not be found.

修改/etc/hadoop/hadoop-env.sh中设JAVA\_HOME。

通过web访问resourcemanager界面：http://localhost:8088。

通过web访问namenode HDFS Web界面：http://localhost:9870。

Ubuntu 关闭防火墙：ufw disable。

hadoop3.0以下版本web访问端口50070；3.0及以上web访问端口9870

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-3.0.1/spark-3.0.1-bin-hadoop3.2.tgz>

$ wget https://archive.apache.org/dist/spark/spark-3.3.1/spark-3.3.1-bin-hadoop3.tgz

解压

$ cd /hadoop

$ tar -zxvf spark-3.3.1-bin-hadoop3.tgz

删除安装文件

$ rm -r spark-3.3.1-bin-hadoop3.tgz

修改文件名称

$ mv spark-3.3.1-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 gedit /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-0.10.9.5-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 gedit /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

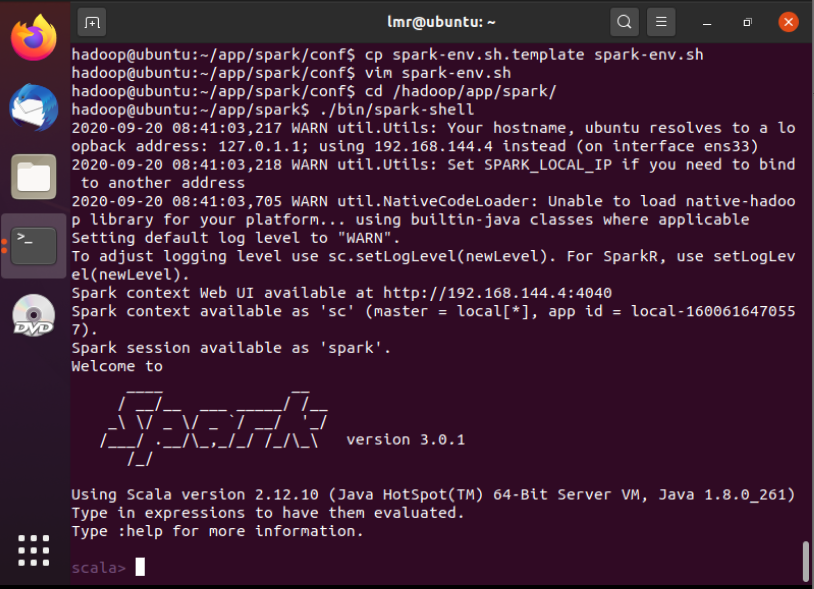
export SPARK\_WORKER\_CORES=1

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。

启动shell或者sql报错：

Error: A JNI error has occurred, please check your installation and try again

Exception in thread “main” java.lang.NoClassDefFoundError: org/slf4j/Logger

则在.bash\_profile加入以下环境变量：

#添加Hadoop的classpath到SPARK\_DIST\_CLASSPAHT中

$ export SPARK\_DIST\_CLASSPATH=$(hadoop classpath)

（2）启动spark

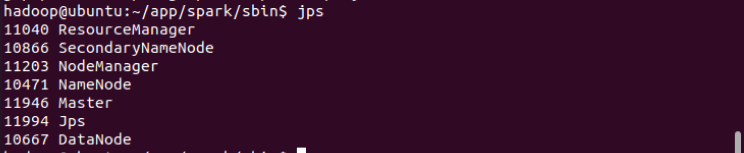
1）首先启动master

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



查看Master进程是否启动

$ jps



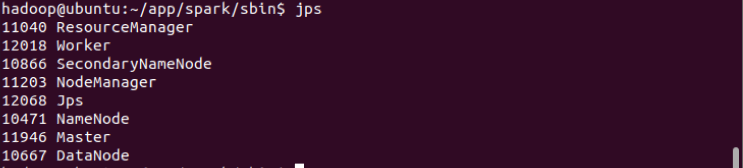
2）启动slave

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



查看Woker进程是否启动

$ jps



**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

结果在执行过程中的其中一行，需要大家仔细查看，如下图：

Pi is roughly 3.14\*\*

spark-submit可以提交任务到spark集群执行，也可以提交到hadoop 的yarn 集群执行。参数的含义是：

--class 应用程序的主类，仅针对 java 或 scala 应用。这里我们使用的是spark自带的计算pi的类。

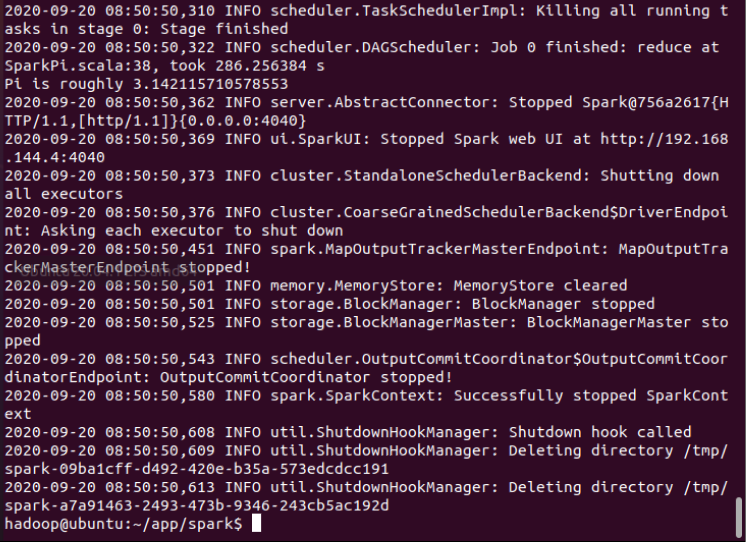
--master master 的地址，提交任务到哪里执行。

--driver-memory Driver内存，默认 1G。

--executor-memory 每个 executor 的内存，默认是1G。

--driver-cores Driver 的核数，默认是1。在 yarn 或者 standalone 下使用。

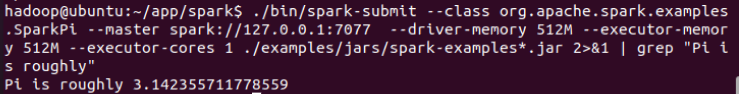
./examples/jars/spark-examples\*.jar 指的是/hadoop/spark/examples/jars下的spark-examples\*.jar包，运行pi的类就写在这些jar包里。



执行时会输出非常多的运行信息，输出结果不容易找到，可以通过 grep 命令进行过滤（命令中的 2>&1 可以将所有的信息都输出到 stdout 中，否则由于输出日志的性质，还是会输出到屏幕中）：

$ /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"

结果如下图（结果可能会有微小差别）：



**6、错误集合**

（1）Start-all.sh 找不到命令：

退到hadoop目录下，使用sbin/start-all.sh可以使用命令。

（2）ERROR: Attempting to operate on hdfs namenode as root

ERROR: but there is no HDFS\_NAMENODE\_USER defined. Aborting operation.

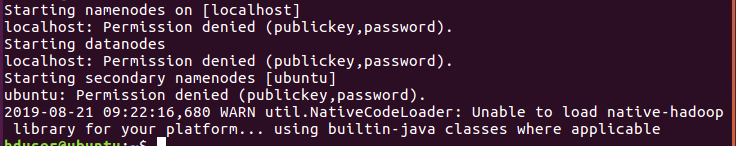
Starting datanodes

（3）Starting namenodes on [ubuntu]

ubuntu: root@ubuntu: Permission denied (publickey,password).

是当前用户无法无密码登陆ssh，要使用可以无密码登陆ssh的用户。

启动hadoop时会发生如下错误



首先运行

ssh localhost

正常情况下是免密登录的，如果还要输入密码的话，那就是ssh没有配置好。这里要说一下的是ssh7.0之后就关闭了dsa的密码验证方式，如果秘钥是通过dsa生成的话，需要改用rsa来生成秘钥。

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

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

再次运行

ssh localhost

（4）Starting namenodes on [ubuntu]

ubuntu: ERROR: Unable to write in /hadoop/logs. Aborting.

在进行Hadoop节点格式化时候出现Starting resourcemanager ERROR: Unable to write in /home/zhang/opt/hadoop-3.1.0/logs. Aborting.

原因：权限不够

加入-R 参数，将读写权限传递给子文件夹

$ sudo chmod -R 777 ./logs

或者出现Cannot create directory

同样是因为权限不够：将需要的权限给予相应的 文件夹

sudo chmod -R a+w /tmp

（5）Hadoop启动后找不到Namenode节点

A. 先关闭hadoop：sbin/stop-all.sh

B. 删除文件夹（hadoop2.7.3/下）的tmp/文件夹里边所有的东西

C. 删除日志： 删除logs文件夹下所有的东西：

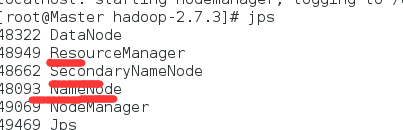
rm -rf logs/

mkdir logs

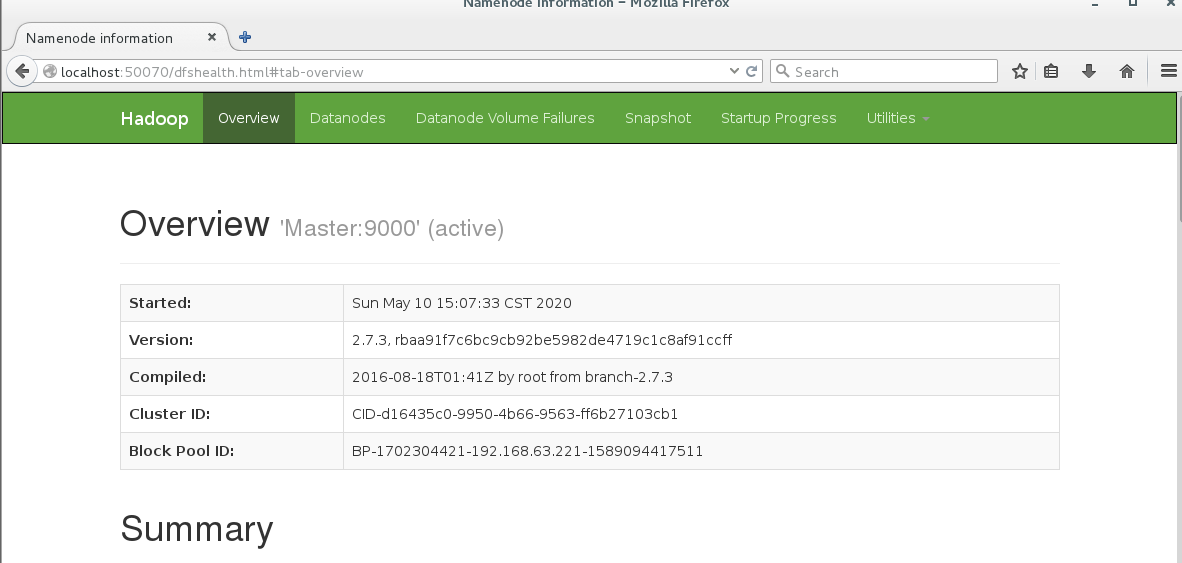
D. 重新格式化： bin/hadoop namenode -format

E. 格式化完成启动hadoop：

F. 启动成功：



G. 访问：http://localhost:50070

H. 

（6）刚开始可以运行hadoop ，运行了spark之后就不能运行Hadoop了：关闭spark的HDFS。

（7）ssh的问题

（8）Limeirong用户su hadoop之后和直接使用hadoop用户登录 环境变量效果不一样。

（9）Linux的vi有问题，配置文件编辑后没有效果：用其他编辑器。

（10）ssh无法免密登陆

终极解决方法：重装虚拟机。

另外的尝试：参阅网上很多方法后，发现步骤都差不多，但是却屡屡失败，设置完后仍然要输入密码，后面发现了是被登录机器的文件权限问题：

//用户权限

chmod 700 /home/username

//.ssh文件夹权限

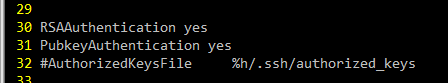
chmod 700 ~/.ssh/

// ~/.ssh/authorized\_keys 文件权限

chmod 600 ~/.ssh/authorized\_keys

还有可能这个文件内容被注释了（改成下图的状态）：

vi /etc/ssh/sshd\_config



如果还是无法登陆，可能是Linux禁用了root账户ssh登陆，解决方法：编辑配置文件：

vim /etc/ssh/sshd\_config

修改PermitRootLogin后面的no为yes，并且去掉前面的注释符，同时可以限制失败次数。注意：这个失败次数是对所有用户都起作用的。

#LoginGraceTime 2m

PermitRootLogin yes

#StrictModes yes

MaxAuthTries 3

#MaxSessions 10

最后重启sshd 服务：

systemctl restart sshd.service

参考：https://www.jianshu.com/p/8343c6b29735

xxx账号下：

sudo gedit /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 gedit /etc/ssh/ssh\_config

增加一句

PubkeyAcceptedKeyTypes +ssh-rsa

# 实验2 MapReduce与Spark内存计算的性能对比

2.1 实验介绍

1、实验内容

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。

2、实验环境

jdk-8u351-linux-x64

hadoop-3.3.4

spark-3.3.1-bin-hadoop3.tgz

自行准备本实验使用的数据集，保存成word.txt文件，尽量使用超过500M的文件。

1、查看数据集的大小：du -h word.txt

2、查看数据集的字符串数：wc -w word.txt

3、查看字符集的内容：more word.txt

3、适合人群

本课程难度为一般，属于初级级别课程，适合具有linux基础的用户。

2.2 实验要求

Linux 环境已就绪（本示例采用的是 ubuntu-20.04.4环境）。

1、Java （需要安装1.6.x及其以上版本)

2.、Hadoop和Spark

3、Python

本实验室使用了Hadoop 3.3.4版本以及Spark-3.3.1版本，Python版本必须在3.7以上。

2.3 概述

**一、Hadoop MapReduce**

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。

一个完成的MapReduce程序在Yarn中执行过程如下：

1、ResourcManager JobClient向ResourcManager提交一个job。

2、ResourcManager向Scheduler请求一个供MRAppMaster运行的container，然后启动它。

3、MRAppMaster启动起来后向ResourcManager注册。

4、ResourcManagerJobClient向ResourcManager获取到MRAppMaster相关的信息，然后直接与MRAppMaster进行通信。

5、MRAppMaster算splits并为所有的map构造资源请求。

6、MRAppMaster做一些必要的MR OutputCommitter的准备工作。

7、MRAppMaster向RM(Scheduler)发起资源请求，得到一组供map/reduce task运行的container，然后与NodeManager一起对每一个container执行一些必要的任务，包括资源本地化等。

8、MRAppMaster 监视运行着的task 直到完成，当task失败时，申请新的container运行失败的task。

9、当每个map/reduce task完成后，MRAppMaster运行MR OutputCommitter的cleanup 代码，也就是进行一些收尾工作。

10、当所有的map/reduce完成后，MRAppMaster运行OutputCommitter的必要的job commit或者abort APIs。

11、MRAppMaster退出。

**二、Spark**

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。

2.4 实验步骤

1、Hadoop、Spark安装配置

按照实验1的指导书按照配置hadoop和spark。

2、将本次实验的数据文件上传到HDFS文件系统

首先建立一个/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，查看该目录下是否有了word.txt文件使用以下命令：

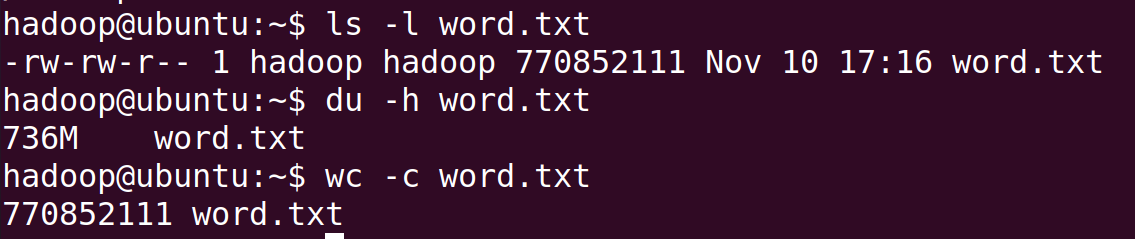
$ ls -l

查看数据集的大小：

$ du -h word.txt

查看数据集的字符串数：

$ wc -c word.txt



查看字符集的内容：

$ more word.txt

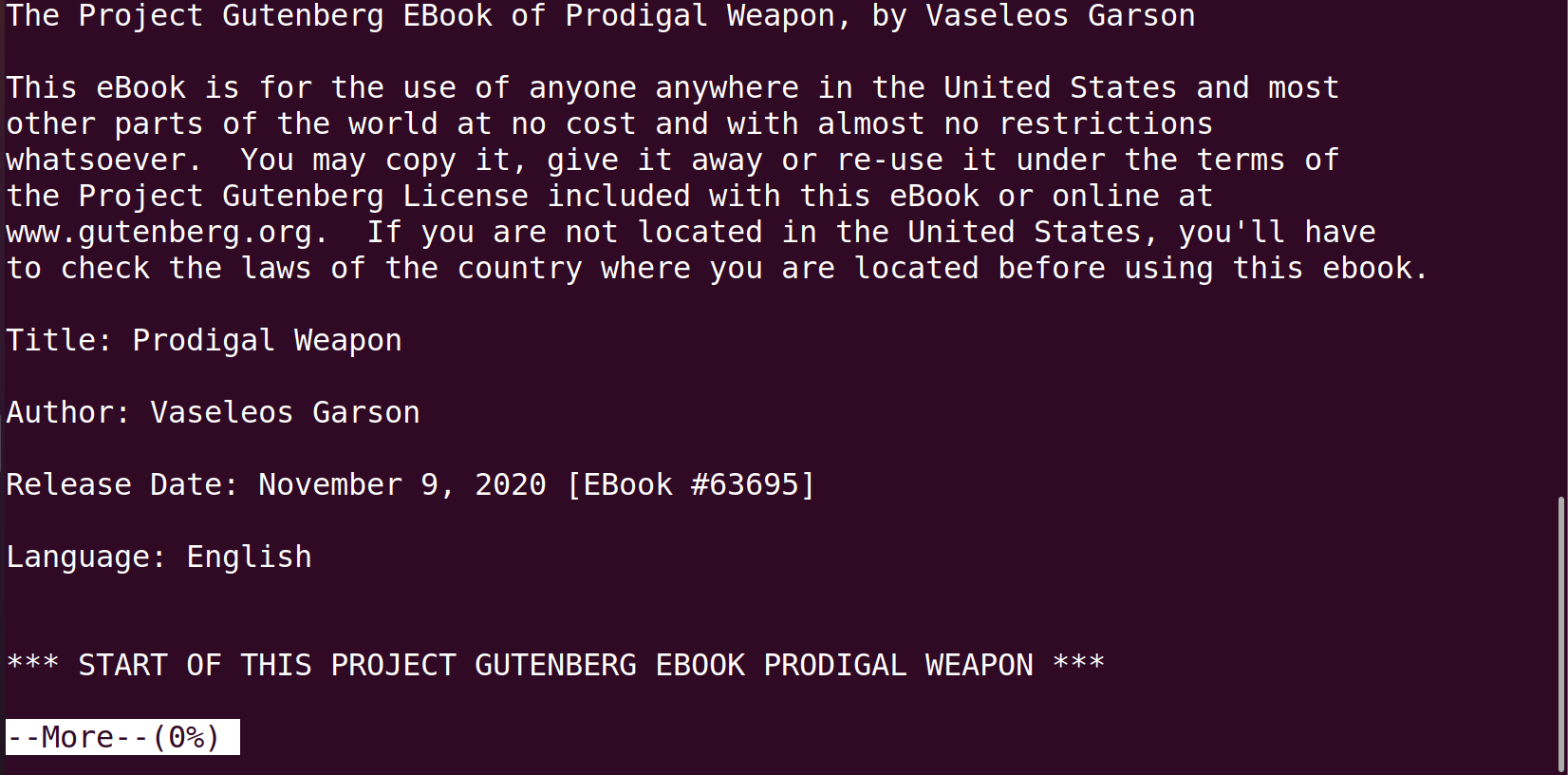
more命令说明：

（1）more命令会以一页一页的显示方便使用者逐页阅读，而最基本的指令就是按空白键（space）就往下一页显示，按 b 键就会往回（back）一页显示，而且还有搜寻字串的功能 。more命令从前向后读取文件，因此在启动时就加载整个文件。

（2）空格键 向下滚动一屏

（3）输入q退出more。

因此，可以按“空格键”看下一页，如需要退出按q。

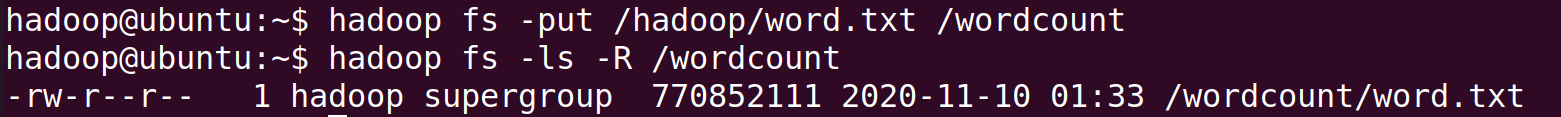


**将文件上传到HDFS/wordcount：**

$ 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：

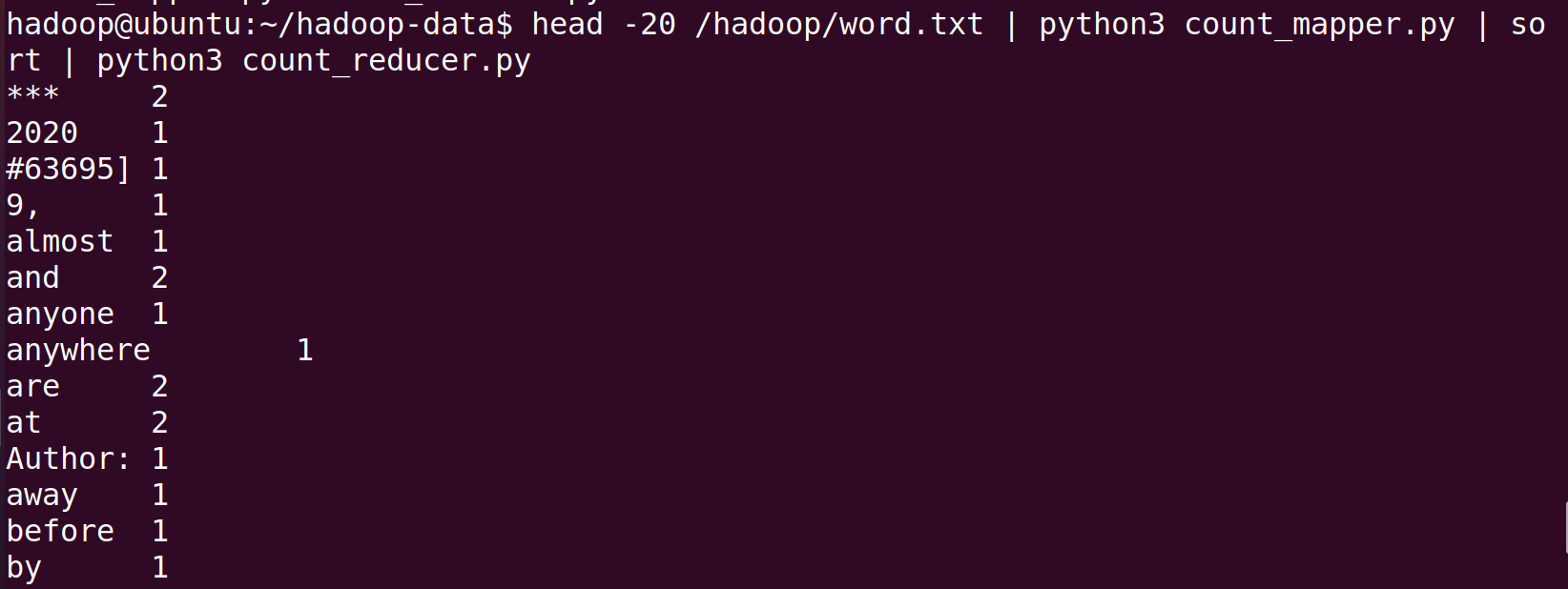
|  |
| --- |
| #!/usr/bin/env python3  import sys  # 从标准输入过来的数据  for line in sys.stdin:  # 将首位的空格去掉  line = line.strip()  # 将这一行文本切分成单词（按空格）  words = line.split()  # 读一个单词写出一个<单词,1>  for word in words:  print("%s\t%s" % (word, 1)) |

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



如上图，就证明map和reduce程序编写成功。

注意：如果py文件不具备权限，运行：

chmod a+w /hadoop/data

sudo chmod 777 ./count\_mapper.py

sudo chmod 777 ./count\_reducer.py

sudo chmod 777 ./wordcount.py

（3）运行该实例，命令如下：

$ hadoop jar /hadoop/hadoop-3.3.4/share/hadoop/tools/lib/hadoop-streaming-3.3.4.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

如果在windows上建立的py文件，提示了“java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess faile”，有可能需要执行下面两句才能正确运行

sed -i 's/\r$//' ./count\_mapper.py

sed -i 's/\r$//' ./count\_reducer.py

（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-3.3.1-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

如果程序运行有异常，请注意spark和python的版本是否匹配，例如spark 3.3.1版本需要python 3.7以上版本支持。例如Ubuntu 18.04.6 LTS的python版本号是3.6.9，则只支持到spark的spark-3.2.3版本。

5、实验结果

分析对比使用Hadoop的MapReduce和Spark两者的计算速度。

# 实验3 基于Hadoop和Spark的数据预测实践项目

1、Spark与Python结合

Python是数据分析最常用的语言之一，而Apache Spark是一个开源的强大的分布式查询和处理引擎。本实验讲解如何基于Python语言进行Spark Application编程，完成数据获取、处理、数据分析及可视化方面常用的数据分析方法与技巧，通过实验操作让学生掌握使用PySpark来分析数据。

通过初始安装Hadoop、Spark集群，并考虑到了当前应用场景需要支持Python程序运行在Spark集群之上，这时可以准备好对应Python软件包、依赖模块，在Hadoop集群中的每个节点上进行安装。这样，Hadoop集群的每个NodeManager上都具有Python环境，可以编写PySpark Application并在集群上运行。目前比较流行的是直接安装Python虚拟环境，使用Anaconda等软件，可以极大地简化Python环境的管理工作。

这种方式的缺点是，如果后续使用Python编写Spark Application，需要增加新的依赖模块，那么就需要在Hadoop集群的每个节点上都进行该新增模块的安装。而且，如果依赖Python的版本，可能还需要管理不同版本Python环境。目前实验Hadoop使用单机模式安装，如果数据量过大，学生电脑配置不高，会发生卡死现象。

2、实验设计

针对DataExpo2009数据集，通过Python、PySpark等工具，进行航空公司延误和取消分析。

该数据集包含1987年10月到2008年12月美国境内所有商业航班的航班到达和离开详细信息。这是一个大型数据集：总共有近1.2亿条记录，占用了1.6 GB的压缩空间和12 GB的未压缩时空间。

针对数据量太大的情况，可以选择kaggle上另一数据集DelayedFlights.csv进行分析，此数据集有1936758条数据，共有19家航空公司，是截取DataExpo2009一部分数据组成，有相同的数据字典。可选择前10个数据量最多的航空公司：

|  |  |
| --- | --- |
| **航空公司** | **数据量** |
| WN | 377602 |
| AA | 191865 |
| MQ | 141920 |
| UA | 141426 |
| OO | 132433 |
| DL | 114238 |
| XE | 103663 |
| CO | 100195 |
| US | 98425 |
| EV | 81877 |

可参考上表选择。航班有5205种类，最多的航班前10数据为：

|  |  |
| --- | --- |
| **航班** | **数据量** |
| LAX-SFO | 4739 |
| ORD-LGA | 4396 |
| ATL-LGA | 4058 |
| SFO-LAX | 4020 |
| LAS-LAX | 3516 |
| LGA-ATL | 3354 |
| ORD-EWR | 3347 |
| LGA-ORD | 3327 |
| ATL-EWR | 3283 |
| DFW-ORD | 3278 |

可参考上表选择航班。若是觉得193万条数据较少，也可选择DataExpo2009数据集中2008年的数据集，有7009728条数据。

实验需要读入并处理数据（可以存入HDFS、HBase等），然后制作一些图形和表格来进行数据分析，能更清楚地了解这个数据集中的内容。比如进行以下分析：

（1）查看飞机延误时间最长的前10名航班。

（2）计算延误的和没有延误的航空公司的比例。

（3）分析一天中、一周中延误最严重的飞行时间。

（4）短途航班和长途航班，哪种航班取消更严重？

（5）建立机器学习算法模型，预测未来航班取消情况。

3、模型选择

模型可选择：支持向量机（SVM）、随机森林（Random Forest）、梯度提升树(GBDT)、线性判别分析（Linear Discriminant Analysis）、伯努利贝叶斯分类（BernoulliNB）、Adaboost、XGBoost等。

模型评估方法可用：准确率和召回率、ROC 曲线、AUC（ROC 曲线下的面积）等，按照评分高低进行比较。

当然，如果要用深度学习来训练算法模型，由于数据集涉及到时间序列相关，建议选择长短期记忆模型（LSTM）。

4、DataExpo2009数据字典

|  |  |  |
| --- | --- | --- |
|  | Name | Description |
| 1 | Year | 1987-2008 |
| 2 | Month | 1-12 |
| 3 | DayofMonth | 1-31 |
| 4 | DayOfWeek | 1 (Monday) - 7 (Sunday) |
| 5 | DepTime | actual departure time (local, hhmm) |
| 6 | CRSDepTime | scheduled departure time (local, hhmm) |
| 7 | ArrTime | actual arrival time (local, hhmm) |
| 8 | CRSArrTime | scheduled arrival time (local, hhmm) |
| 9 | UniqueCarrier | unique carrier code |
| 10 | FlightNum | flight number |
| 11 | TailNum | plane tail number |
| 12 | ActualElapsedTime | in minutes |
| 13 | CRSElapsedTime | in minutes |
| 14 | AirTime | in minutes |
| 15 | ArrDelay | arrival delay, in minutes |
| 16 | DepDelay | departure delay, in minutes |
| 17 | Origin | origin IATA airport code |
| 18 | Dest | destination IATA airport code |
| 19 | Distance | in miles |
| 20 | TaxiIn | taxi in time, in minutes |
| 21 | TaxiOut | taxi out time in minutes |
| 22 | Cancelled | was the flight cancelled? |
| 23 | CancellationCode | reason for cancellation (A = carrier, B = weather, C = NAS, D = security) |
| 24 | Diverted | 1 = yes, 0 = no |
| 25 | CarrierDelay | in minutes |
| 26 | WeatherDelay | in minutes |
| 27 | NASDelay | in minutes |
| 28 | SecurityDelay | in minutes |
| 29 | LateAircraftDelay | in minutes |

5、环境要求

（见附录有完全基于jupyter notebook的环境配置）

1. Java （需要安装1.6.x及其以上版本)

2. Hadoop和Spark（前两次实验已有）

3. PySpark、Jupyter或者Pycharm

在这里，我们使用了Hadoop V3.3.4版本以及Spark-3.3.1版本。

5.1 PySpark的安装及测试

首先确认Spark的环境是否配置完成（实验一要求的spark环境）。

$ su - hadoop

口令输入：hadoop。

$ sudo gedit ~/.bash\_profile

看到配置文件里有如下spark环境内容

|  |
| --- |
| export SPARK\_HOME=/hadoop/spark  export PATH=$PATH:$SPARK\_HOME/bin |

$ echo $SPARK\_HOME

/hadoop/spark

上述输出确认hadoop的环境变量设置有效，如果无效则激活环境变量：

$ source ~/.bash\_profile

最后运行pyspark以测试PySpark是否可以成功启动。

PySpark有几种运行模式：

1、可以在本地直接运行，使用命令：

$ pyspark --master local[\*]

2、在Hadoop YARN上运行PySpark，在执行hadoop的“start-all.sh”之后，使用：

$ pyspark --master yarn --deploy-mode client

3、构建Spark Standalone集群，在Spark Standalone上运行PySpark，在执行了hadoop和spark的两个“start-all.sh”之后，使用命令：

$ pyspark --master spark://127.0.0.1:7077 --num-executors 1 --total-executor-cores 3 --executor-memory 512m

5.2 Jupyter Notebook安装及测试

Jupyter Notebook具备交互式界面，我们可以在Web界面输入Python命令后立刻看到结果。我们还可将数据分析的过程和运行后的命令与结果存储成笔记本，下次可以打开笔记本，重新执行这些命令，IPython Notebook笔记本可以包含文字、数学公式、程序代码、结果、图形、视频。

通过安装Anaconda软件包直接安装Jupyter Notebook。

**1、下载Anaconda**

https://repo.continuum.io/archive/index.html 进入网站后可以看到适用于Linux的安装包。可以选择Anaconda3-2022.10-Linux-x86\_64.sh，或者以上的其他版本。复制需要下载的anaconda的版本，在终端使用wget命令

|  |
| --- |
| $ wget https://repo.anaconda.com/archive/Anaconda3-2022.10-Linux-x86\_64.sh |

下载完成后，进行安装

|  |
| --- |
| $ bash Anaconda3-2022.10-Linux-x86\_64.sh -b |

-b是批次安装，会忽略Licese条款。

加入PySpark及Anaconda路径

|  |
| --- |
| $ sudo gedit /hadoop/.bash\_profile |

在打开的文档中，加入以下设置（这里配置的是ipython，next section配置的是jupyter，实际中使用jupyter即可）：

|  |
| --- |
| export PATH=$PATH:$SPARK\_HOME/bin:/hadoop/anaconda3/bin  export ANACONDA\_PATH=/hadoop/anaconda3  export PYSPARK\_DRIVER\_PYTHON=$ANACONDA\_PATH/bin/ipython  export PYSPARK\_PYTHON=$ANACONDA\_PATH/bin/python |

再次source生效。

|  |
| --- |
| $ source ~/.bash\_profile |

**2、Jupyter安装配置**

Jupyter Notebook是基于网页的用于交互计算的应用程序。其可被应用于全过程计算：开发、文档编写、运行代码和展示结果。简而言之，Jupyter Notebook是以网页的形式打开，可以在网页页面中直接编写代码和运行代码，代码的运行结果也会直接在代码块下显示。

配置PySpark driver

$ sudo gedit /hadoop/.bash\_profile

|  |
| --- |
| export PYSPARK\_DRIVER\_PYTHON=jupyter  export PYSPARK\_DRIVER\_PYTHON\_OPTS='notebook' |

保存关闭后，执行以下命令使得环境变量生效：

$ source /hadoop/.bash\_profile

重新输入pyspark后，在浏览器中就可以使用jupyter了。

**3、尝试在Notebook中使用Spark**

可以在“终端”程序中输入下列命令，创建并切换ipynotebook 工作目录。

$ mkdir -p ~/pythonwork/ipynotebook

$ cd ~/pythonwork/ipynotebook

进入工作目录后，输入以下命令

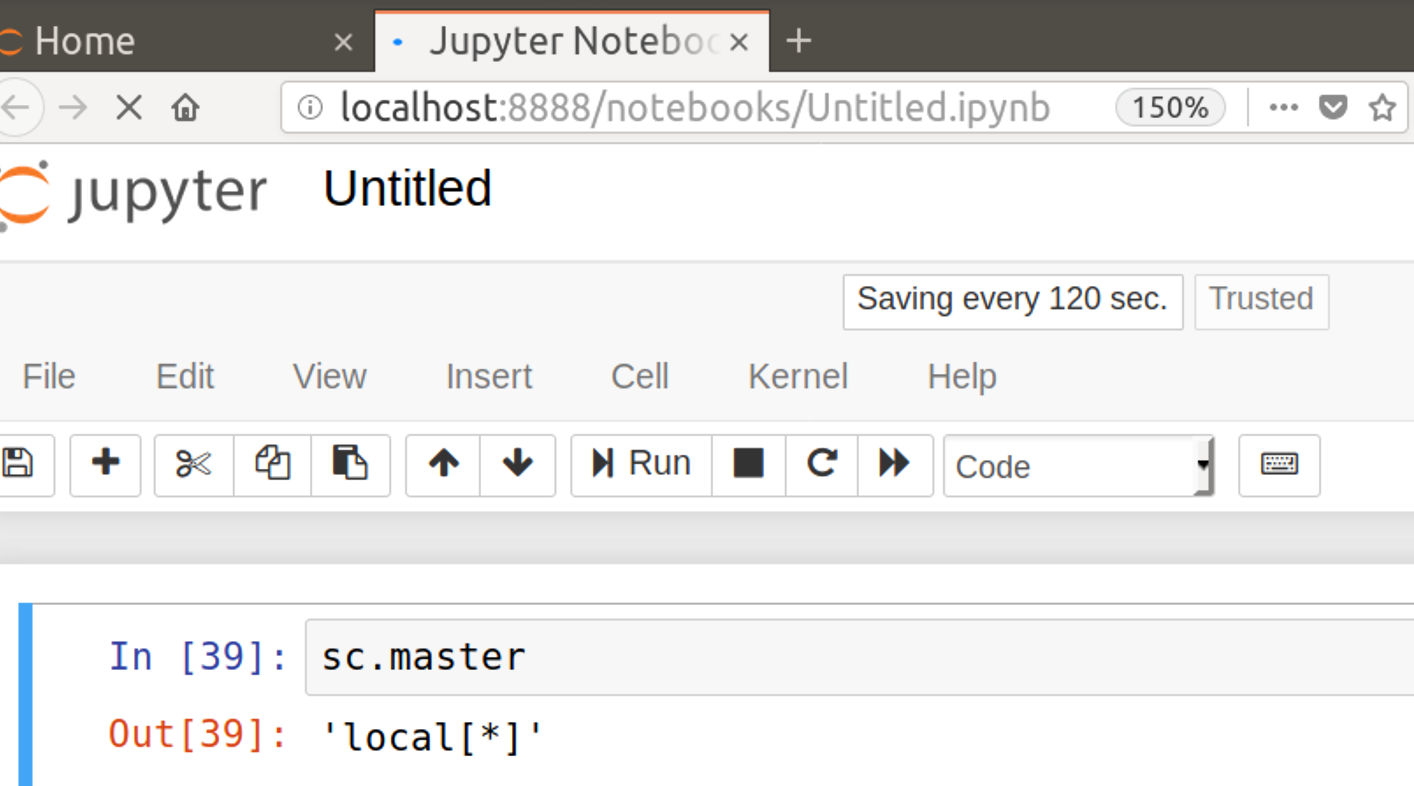
$ PYSPARK\_DRIVER\_PYTHON=ipython PYSPARK\_DRIVER\_PYTHON\_OPTS="notebook" pyspark

注意上面命令中的PYSPARK\_DRIVER\_PYTHON和PYSPARK\_DRIVER\_PYTHON\_OPTS如果在“.bash\_profile”中定义过，可忽略。

然后从浏览器启动Notebook界面，默认是以spark的本地模式运行，例如在新建的notebook中，输入代码：

sc.master

运行结果如下图所示：



6、实践内容要求

针对DataExpo2009数据集，通过Python、PySpark等工具，进行数据读取、数据处理及可视化展示，并进行以下分析：

（1）查看飞机延误时间最长的前10名航班。

（2）计算延误的和没有延误的航空公司的比例。

（3）分析一天中、一周中延误最严重的飞行时间。

（4）短途航班和长途航班，哪种航班取消更严重？

（5）建立机器学习算法模型，预测未来航班取消情况。

7、实践结果

（1）-（4）为统计分析展示结果，（5）为机器学习模型结果，请将结果代码和分析过程放在实验报告中。