東南大學
SOUTHEAST UNIVERSITY

云计算技术及应用
课程实践报告

标 题	大数据处理技术实践
学 号	232349
姓 名	李志成

东南大学计算机科学与工程学院

二零二四年五月

目录

大数据处理技术实践

1 实验目标

1）**掌握大数据环境配置与管理**：学习并实践如何在 Linux 系统下安装和配置 Hadoop 及 Spark 环境，包括必要的环境变量设置和基本的集群管理操作。

2）**实践 Hadoop 和 Spark 的基本应用**：通过完成 WordCount 和电影评分预测等实验，掌握在 Hadoop 和 Spark 环境下开发和运行大数据处理程序的基本方法。

2 实验环境准备

2.0 实验环境总览

操作系统：Ubuntu22.04
Java：openjdk-11
Python：Python 3.10.12
Hadoop：3.3.6
Spark：3.5.1

2.1 Hadoop 本体安装

2.1.1 安装 java

Hadoop 基础组件及其外围项目大多是基于 Java 的，要运行 Hadoop，我们需要在系统中安装 Java 运行时环境（JRE）。有多种 JRE 可供选择，我们使用 openjdk-11 作为 JRE。Hadoop 官方文档中说明了对 Java 版本的支持情况。Hadoop 框架使用 SSH 协议与本地（或远程计算机）进行通信，我们需要在本地计算机中安装 SSH 服务器守护程序。

```
sudo apt install openjdk-11-jdk openssh-server -y
```

2.1.2 创建 ssh 密钥并添加到自己的本地计算机

Hadoop 通常运行于分布式环境中。在这种环境下，它使用 SSH 协议与自己（或其他服务器）进行通信，为了能够配置 Hadoop 更安全地与服务器进行通信，我们为本地环境 Hadoop 用户生成公-私密钥对，以支持 SSH 中的密钥对认证：

```
ssh-keygen -b 4096 -C "for hadoop use"  # 为了方便这里可以直接一路回车
ssh-copy-id -p 22 localhost # 输入 yes
```

上述命令生成带备注信息的长度为 4096 位的 RSA 密钥，并将密钥复制至本地计算机中，用于支持 SSH 密钥对认证方式。

这时候可以尝试使用 ssh localhost，应该可以无密码登录。

```
lzc@lzc-virtual-machine:~$ ssh -p '22' 'localhost'
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

扩展安全维护（ESM）Applications 未启用。

0 更新可以立即应用。

启用 ESM Apps 来获取未来的额外安全更新
请参见 https://ubuntu.com/esm 或者运行: sudo pro status
```

2.1.3 下载并解压 Hadoop

Hadoop 的各种发行版本通常从 Hadoop 官方网站的下载页面(opens new window)下载。我们使用来自 Hadoop 官方网站的发行版本 3.3.6。我们准备将相关发行版本安装至文件系统的/opt 目录下。

首先从下载页面获得 Hadoop 的发行版文件：（利用清华源的镜像，速度更快）

```
wget
"https://mirrors.tuna.tsinghua.edu.cn/apache/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.
gz"
```

下载好压缩包后，解压到/opt 目录。

```
sudo tar -zxvf hadoop-3.3.6.tar.gz -C /opt
```

2.2 Hadoop 配置

Hadoop 并非是单个软件，而是一系列大数据存储及处理的工具集合。所以，有必要针对常用的基础组件进行配置，以完成 Hadoop 环境的搭建。

2.2.1 配置环境变量

安装 Hadoop 后，我们需要正确设置以下环境变量。可以将这些添加环境变量的命令添加至~/.bashrc 文件末尾，以避免每次登录时都需要执行，注意以下命令需要一次性复制和运行，不可以分多次：

```
cat << 'EOF' >> ~/.bashrc
export HADOOP_HOME=/opt/hadoop-3.3.6
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$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 PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
EOF
```

然后执行 source ~/.bashrc 使配置生效或者直接重启 shell。

2.2.2 配置 Hadoop 基础框架

顺利运行 Hadoop 组件，我们有必要告诉 Hadoop 有关 Java 运行时环境（JRE）的位置信息。Hadoop 框架的基础配置文件位于./etc/hadoop/hadoop-env.sh 中，我们只需要修改其中的 JAVA_HOME 行。如果你使用其他方式（比如非标准端口）通过 SSH 客户端连接到（本地）服务器，则需要额外修改 HADOOP_SSH_OPTS 行。

首先我们接下来的操作都需要以/opt/hadoop-3.3.6 作为当前路径，也就是说需要先 cd 到/opt/hadoop-3.3.6，另外注意以下命令同样需要一次性复制和运行。

（接下来会经常用到./etc/hadoop 这个目录，注意这个是相对于/opt/hadoop-3.3.6 的，而不是系统根目录下的/etc/hadoop，前者有一个.，表示相对路径。）

```
cat << 'EOF' >> ./etc/hadoop/hadoop-env.sh
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_SSH_OPTS="-p 22"
EOF
```

2.2.3 核心变量配置

Hadoop 生态的大多数组件采用.xml(opens new window)文件的方式进行配置。它们采用了统一的格式：即将配置项填写在每个配置文件的<configuration>与</configuration>之间，每个配置项通过以下的方式呈现：

```
<property>
<name>配置项名称</name>
<value>配置值</value>
</property>
```

例如，一个正常的 hadoop 配置文件应该类似如下结构，下文所说的放在<configuration></configuration> 之间都是如此：

```
<configuration>

<property>
<name>配置项名称 1</name>
<value>配置值 1</value>
</property>

<property>
<name>配置项名称 2</name>
<value>配置值 2</value>
</property>

  </configuration>
```

首先需要修改./etc/hadoop/core-site.xml，我们需要配置 fs.defaultFS 为 hdfs://localhost/，也就是说，需要把以下内容添加到./etc/hadoop/core-site.xml 的<configuration></configuration>之间。

```
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost/</value>
</property>
```

2.2.4 配置 HDFS（Hadoop 文件系统）

为了支持 Hadoop 的运行，我们需要使用能支持分布式的文件系统，HDFS 就是为了 Hadoop 使用的。

在使用 Hadoop 文件系统（HDFS）之前，我们需要显式配置 HDFS，以指定 NameNode 与 DataNode 的存储位置。我们计划将 NameNode 与 DataNode 存储于本地文件系统上，即存放于~/hdfs 中：

```
mkdir -p ~/hdfs/namenode ~/hdfs/datanode
```

然后，修改./etc/hadoop/hdfs-site.xml，将以下内容放到该文件的<configuration> </configuration>之间。

```
<property>
<name>dfs.replication</name>
<value>1</value>
</property>

<property>
<name>dfs.name.dir</name>
<value>/home/lzc/hadoop/namenode</value>
</property>

<property>
<name>dfs.data.dir</name>
<value>/home/lzc/hadoop/datanode</value>
</property>
```

首次启动 Hadoop 环境之前，我们需要初始化 Namenode 节点数据，使用以下命令：

```
hdfs namenode -format
```

应该可以看到很多输出信息。

2.2.5 配置 MapReduce（由 YARN 驱动）

MapReduce 是一种简单的用于数据处理的编程模型，YARN（Yet Another Resource Negotiator）是 Hadoop 的集群资源管理系统。

将以下内容放到./etc/hadoop/mapred-site.xml 的<configuration> </configuration>之间。

```
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
```

```
<property>
<name>yarn.app.mapreduce.am.env</name>
<value>HADOOP_MAPRED_HOME=/opt/hadoop-3.3.6/</value>
</property>

<property>
<name>mapreduce.map.env</name>
<value>HADOOP_MAPRED_HOME=/opt/hadoop-3.3.6/</value>
</property>

<property>
<name>mapreduce.reduce.env</name>
<value>HADOOP_MAPRED_HOME=/opt/hadoop-3.3.6/</value>
</property>
```

将以下内容放到./etc/hadoop/yarn-site.xml 的<configuration> </configuration>之间。

```
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>

<property>
<name>yarn.resourcemanager.hostname</name>
<value>localhost</value>
</property>
```

2.2.6 启动 Hadoop 集群

使用以下命令:

```
start-dfs.sh
start-yarn.sh
mr-jobhistory-daemon.sh start historyserver
```

可能会出现 ssh 的提示 Are you sure you want to continue connecting (yes/no/[fingerprint])?，直接 yes 就可以。

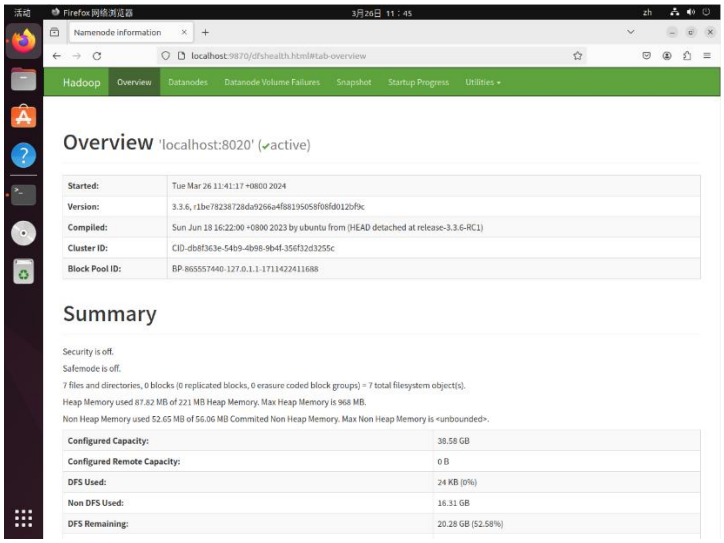
这将启动以下守护进程：一个 namenode、一个辅助 namenode、一个 datanode（HDFS）、一个资源管理器、一个节点管理器（YARN）以及一个历史服务器（MapReduce）。

为验证 Hadoop 相关服务启动成功，可以使用 jps 命令。

```
lzc@lzc-virtual-machine:/opt/hadoop-3.3.6/etc/hadoop$ jps
4176 DataNode
5202 Jps
4051 NameNode
4614 ResourceManager
4410 SecondaryNameNode
5132 JobHistoryServer
4734 NodeManager
```

jps 命令显示的内容与上述预期一致。

可以使用浏览器访问该主机的 9870 端口，以查看 HFS 的相关情况：



此时 hadoop 基础的组件已经安装完成。

2.2.7 关闭 Hadoop 集群

使用以下命令：

```
stop-yarn.sh
stop-dfs.sh
mr-jobhistory-daemon.sh stop historyserver
```

```
lzc@lzc-virtual-machine:/opt/hadoop-3.3.6/etc/hadoop$ stop-yarn.sh
Stopping nodemanagers
Stopping resourcemanager
lzc@lzc-virtual-machine:/opt/hadoop-3.3.6/etc/hadoop$ stop-dfs.sh
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [lzc-virtual-machine]
lzc@lzc-virtual-machine:/opt/hadoop-3.3.6/etc/hadoop$ mr-jobhistory-daemon.sh stop historyserver
WARNING: Use of this script to stop the MR JobHistory daemon is deprecated.
WARNING: Attempting to execute replacement "mapred --daemon stop" instead.
lzc@lzc-virtual-machine:/opt/hadoop-3.3.6/etc/hadoop$ jps
6777 Jps
```

可以看到，我们成功关闭了 Hadoop 集群服务。

2.3 添加所需的环境变量

使用 ip a 命令查看虚拟机的 IP 地址：

```
lzc@lzc-virtual-machine:/opt/hadoop-3.3.6/etc/hadoop$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:2c:76:01 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.61.133/24 brd 192.168.61.255 scope global dynamic noprefixroute ens33
       valid_lft 1363sec preferred_lft 1363sec
    inet6 fe80::9433:d545:d5b2:6ba6/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

使用 which hadoop 命令辅助找到 Hadoop 配置文件目录：

```
lzc@lzc-virtual-machine:/opt/hadoop-3.3.6/etc/hadoop$ which hadoop
/opt/hadoop-3.3.6/bin/hadoop
lzc@lzc-virtual-machine:/opt/hadoop-3.3.6/etc/hadoop$ cd /opt/hadoop-3.3.6/etc/hadoop
lzc@lzc-virtual-machine:/opt/hadoop-3.3.6/etc/hadoop$ ls
capacity-scheduler.xml      hadoop-policy.xml               kms-acls.xml          mapred-queues.xml.template     yarn-env.cmd
configuration.xsl           hadoop-user-functions.sh.example kms-env.sh           mapred-site.xml                yarn-env.sh
container-executor.cfg      hdfs-rbf-site.xml               kms-log4j.properties  shellprofile.d                 yarnservice-log4j.properties
core-site.xml               hdfs-site.xml                   kms-site.xml          ssl-client.xml.example         yarn-site.xml
hadoop-env.cmd              httpfs-env.sh                   log4j.properties      ssl-server.xml.example
hadoop-env.sh               httpfs-log4j.properties         mapred-env.cmd        user_ec_policies.xml.template
hadoop-metrics2.properties  httpfs-site.xml                 mapred-env.sh         workers
```

如图，Hadoop 的目录在 /opt/hadoop-3.3.6/bin/hadoop ，配置文件目录为 /opt/hadoop-3.3.6/etc/hadoop 。

使用 vim ~/.bashrc 命令编辑系统环境变量，手动添加 3 个环境变量：

```
export HADOOP_CONF_DIR=/opt/hadoop-3.3.6/etc/hadoop
export YARN_CONF_DIR=/opt/hadoop-3.3.6/etc/hadoop
export SPARK_LOCAL_IP=192.168.61.133
```

```
export HADOOP_HOME=/opt/hadoop-3.3.6
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$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 PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
export HADOOP_CONF_DIR=/opt/hadoop-3.3.6/etc/hadoop
export YARN_CONF_DIR=/opt/hadoop-3.3.6/etc/hadoop
export SPARK_LOCAL_IP=192.168.61.133
```

使用 source ~/.bashrc 命令让环境变量生效

2.4 Python 环境配置

默认情况下，Linux 会自带安装 Python，可以运行 python --version 命令或 python3 --version 命令查看：

```
lzc@lzc-virtual-machine:~$ python3 --version
Python 3.10.12
```

输入命令 which python 或 which python3 ，查看 Linux 默认安装的 Python 位置：

```
lzc@lzc-virtual-machine:~$ which python3
/usr/bin/python3
```

输入命令 python 或 python3 启动 python，并验证：

```
lzc@lzc-virtual-machine:~$ python3
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 3
>>> b = 4
>>> print(a + b)
7
```

2.5 Spark 安装

输入以下命令安装 Spark：

```
wget https://dlcdn.apache.org/spark/spark-3.5.1/spark-3.5.1-bin-hadoop3.tgz
```

输入命令 tar -zxvf spark-3.5.1-bin-hadoop3.tgz 解压 Spark。

Spark 开箱即用：进入解压后 Spark 目录下的 bin 目录，（通过 ./pyspark 命令）运行 bin/pyspark 程序, 可以提供一个交互式的 Python 解释器环境, 在这里面可以用 Python 语言调用 Spark API 进行计算。

```
lzc@lzc-virtual-machine:~/spark/bin$ ./pyspark
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
24/04/28 20:18:55 WARN Utils: Your hostname, lzc-virtual-machine resolves to a loopback addres
s: 127.0.1.1; using 192.168.61.133 instead (on interface ens33)
24/04/28 20:18:55 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/04/28 20:18:56 WARN NativeCodeLoader: Unable to load native-hadoop library for your platfor
m... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Python version 3.10.12 (main, Nov 20 2023 15:14:05)
Spark context Web UI available at http://192.168.61.133:4040
Spark context available as 'sc' (master = local[*], app id = local-1714306738111).
SparkSession available as 'spark'.
>>>
```

输入示例代码 sc.parallelize([1,2,3,4,5]).map(lambda x: x + 1).collect()，将数组内容都加 1：

```
lzc@lzc-virtual-machine:~/spark/bin$ ./pyspark
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
24/04/28 20:18:55 WARN Utils: Your hostname, lzc-virtual-machine resolves to a loopback addres
s: 127.0.1.1; using 192.168.61.133 instead (on interface ens33)
24/04/28 20:18:55 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/04/28 20:18:56 WARN NativeCodeLoader: Unable to load native-hadoop library for your platfor
m... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Python version 3.10.12 (main, Nov 20 2023 15:14:05)
Spark context Web UI available at http://192.168.61.133:4040
Spark context available as 'sc' (master = local[*], app id = local-1714306738111).
SparkSession available as 'spark'.
>>> sc.parallelize([1,2,3,4,5]).map(lambda x: x + 1).collect()
[2, 3, 4, 5, 6]
```

3 基于 Spark 的 WordCount 实验

3.1 启动 Hadoop 集群

使用以下命令启动 Hadoop 集群：

```
start-dfs.sh
start-yarn.sh
mapred --daemon start historyserver
```

这将启动以下守护进程：一个 namenode、一个辅助 namenode、一个 datanode（HDFS）、一个资源管理器、一个节点管理器（YARN）以及一个历史服务器（MapReduce）。为验证 Hadoop 相关服务启动成功，可以使用 jps 命令查看：

```
lzc@lzc-virtual-machine:~$ jps
30756 JobHistoryServer
30804 Jps
29749 DataNode
29958 SecondaryNameNode
30219 ResourceManager
30348 NodeManager
29613 NameNode
```

3.2 实验文件准备

准备一个 txt 文件 words.txt ，内容可自定义，以备用来做 WordCount 实验：

```
hello spark hdfs hadoop spark
hello hi python hdfs spark nifi
```

将 words.txt 文件上传至 HDFS 集群的特定路径。例如：

1） 使用命令 hdfs dfs -mkdir /input 在 HDFS 根目录下创建目录 input

2） 使用命令 hdfs dfs -ls / 检查是否创建成功：

```
lzc@lzc-virtual-machine:~$ hdfs dfs -ls /
Found 1 items
drwxrwx---   - lzc supergroup          0 2024-04-29 14:44 /tmp
lzc@lzc-virtual-machine:~$ hdfs dfs -mkdir /input
lzc@lzc-virtual-machine:~$ hdfs dfs -ls /
Found 2 items
drwxr-xr-x   - lzc supergroup          0 2024-04-29 14:56 /input
drwxrwx---   - lzc supergroup          0 2024-04-29 14:44 /tmp
```

3） 将 words.txt 文件上传至 HDFS 集群根目录下新创建的目录 input 下：

```
hdfs dfs -put words.txt /input/words.txt
```

```
lzc@lzc-virtual-machine:~$ hdfs dfs -put words.txt /input/words.txt
lzc@lzc-virtual-machine:~$ hdfs dfs -ls /input
Found 1 items
-rw-r--r--   1 lzc supergroup         62 2024-04-29 15:00 /input/words.txt
```

准备 PySpark 代码 WordCount.py ：

```
# coding:utf8
# 导入 PySpark 的配置和上下文模块
from pyspark import SparkConf, SparkContext
if __name__ == '__main__':
        # 创建一个 Spark 配置对象，并设置应用的名称为"WordCount"
        conf = SparkConf().setAppName("WordCount")
        # 使用之前设置的配置创建一个 SparkContext 对象，这是所有功能的入口点
        sc = SparkContext(conf=conf)
        # 从 HDFS 上读取一个文件，并创建一个 RDD（弹性分布式数据集）
        # hdfs://localhost/input/words.txt 是 HDFS 上文件的路径
        file_rdd = sc.textFile("hdfs://localhost/input/words.txt")
        # 对文件 RDD 中的每一行进行扁平化处理，即将每行的单词拆分成单独的单词，并创建新的 RDD
        words_rdd = file_rdd.flatMap(lambda line: line.split(" "))
        # 将每个单词映射为一个元组，其中单词是键，1 是值，表示单词出现的次数为 1
        words_with_one_rdd = words_rdd.map(lambda x: (x, 1))
        # 使用 reduceByKey 操作对具有相同键的元组进行归约，累加每个单词的出现次数
        # 这里 lambda 函数接受两个参数 a 和 b，并返回它们的和
        result = words_with_one_rdd.reduceByKey(lambda a, b: a + b)
        # collect()方法将分布式数据集中的数据收集到驱动程序节点
        print(result.collect())
```

3.3 提交计算任务

通过 spark 安装路径下的 bin/spark-submit 客户端以 local 模式提交任务给 spark 运行：

```
/home/lzc/spark-3.5.1-bin-hadoop3/bin/spark-submit --master local[*]
/home/lzc/py_script/WordCount.py
```

命令结构解读：（ spark 安装路径下的 bin/spark-submit 绝对路径）+（ --master local[*] ）+（代码 WordCount.py 的绝对路径）

任务执行成功：

```
24/04/29 15:16:33 INFO DAGScheduler: Job 0 finished: collect at /home/lzc/py_script/WordCount.py:21, took 2.119900 s
[('hdfs', 2), ('hadoop', 1), ('python', 1), ('hello', 2), ('spark', 3), ('hi', 1), ('nifi', 1)]
```

通过 spark 安装路径下的 bin/spark-submit 客户端以 yarn 模式提交任务给 spark 运行：

```
/home/lzc/spark-3.5.1-bin-hadoop3/bin/spark-submit --master yarn
/home/lzc/py_script/WordCount.py
```

命令结构解读：（ spark 安装路径下的 bin/spark-submit 绝对路径）+（ --master yarn ）+（代码 WordCount.py 的绝对路径）

任务执行成功：

```
24/04/29 15:38:53 INFO DAGScheduler: Job 0 finished: collect at /home/lzc/py_script/WordCount.py:21, took 5.441442 s
[('hdfs', 2), ('hadoop', 1), ('python', 1), ('hello', 2), ('spark', 3), ('hi', 1), ('nifi', 1)]
```

4 基于 Scala 的协同过滤电影评分预测实验

4.1 实验简介

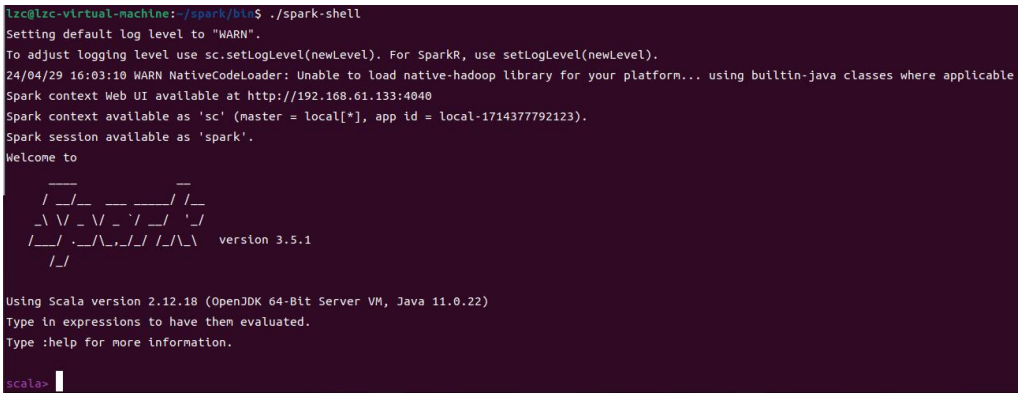
协同过滤是一种基于一组兴趣相同的用户或项目进行的推荐，它根据邻居用户(与目标用户兴趣相似的用户)的偏好信息产生对目标用户的推荐列表。

关于协同过滤的一个经典的例子就是看电影。如果你不知道哪一部电影是自己喜欢的或者评分比较高的，那么通常的做法就是问问周围的朋友，看看最近有什么好的电影推荐。而在问的时候，肯定都习惯于问跟自己口味差不多的朋友，这就是协同过滤的核心思想。因此，协同过滤是在海量数据中挖掘出小部分与你品味类似的用户，在协同过滤中，这些用户成为邻居，然后根据他们喜欢的东西组织成一个排序的目录推荐给你。

本实验使用 Scala 语言在 spark-shell 中完成。

4.2 进入 spark-shell 环境

进入到 spark 安装目录的 bin 目录下，输入 ./spark-shell 进入 spark-shell 环境:



4.3 导入所需的包

```
scala> import org.apache.spark.ml.evaluation.RegressionEvaluator
scala> import org.apache.spark.ml.recommendation.ALS
```

4.4 根据数据结构创建读取规范

创建一个 Rating 类型，即 [Int, Int, Float] ;然后建造一个把数据中每一行转化成 Rating 类的函数。

```
scala> case class Rating(userId: Int, movieId: Int, rating: Float)
scala> def parseRating(str: String): Rating = {
| val fields = str.split("::")
| assert(fields.size == 3)
| Rating(fields(0).toInt, fields(1).toInt, fields(2).toFloat)
| }
```

4.5 读取数据

导入 implicits ，读取 movielens 数据集，把数据转化成 Rating 类型；movielens 是 scala 自带的数据集。

```
scala> import spark.implicits._
scala> val ratings =
spark.sparkContext.textFile("file:///home/lzc/spark/data/mllib/sample_movielens_d
ata.txt").map(parseRating).toDF()
```

然后，我们把数据打印看一下，检查是否读取成功：

```
scala> ratings.show()
```

```
scala> ratings.show()
+------+-------+------+
|userId|movieId|rating|
+------+-------+------+
|     0|      2|   3.0|
|     0|      3|   1.0|
|     0|      5|   2.0|
|     0|      9|   4.0|
|     0|     11|   1.0|
|     0|     12|   2.0|
|     0|     15|   1.0|
|     0|     17|   1.0|
|     0|     19|   1.0|
|     0|     21|   1.0|
|     0|     23|   1.0|
|     0|     26|   3.0|
|     0|     27|   1.0|
|     0|     28|   1.0|
|     0|     29|   1.0|
|     0|     30|   1.0|
|     0|     31|   1.0|
|     0|     34|   1.0|
|     0|     37|   1.0|
|     0|     41|   2.0|
+------+-------+------+
only showing top 20 rows
```

4.6 构建模型并训练

将 movielens 数据集以 8:2 划分训练集和测试集：

```
scala> val Array(training, test) = ratings.randomSplit(Array(0.8,0.2))
```

使用 ALS 来建立推荐模型，这里我们构建了两个模型，一个是显性反馈，一个是隐性反馈：

```
scala> val alsExplicit = new
ALS().setMaxIter(5).setRegParam(0.01).setUserCol("userId").setItemCol("movieId").
setRatingCol("rating")

scala> val alsImplicit = new
ALS().setMaxIter(5).setImplicitPrefs(true).setUserCol("userId").setItemCol("movie
Id").setRatingCol("rating")
```

在 ML 中的实现有如下的参数：

numBlocks：是用于并行化计算的用户和商品的分块个数（默认为 10）

rank：是模型中隐语义因子的个数（默认为 10）

maxIter：是迭代的次数（默认为 10）

regParam：是 ALS 的正则化参数（默认为 1.0）

implicitPrefs：决定了是用显性反馈 ALS 的版本还是用适用隐性反馈数据集的版本（默认是 false，即用显性反馈）

alpha：是一个针对于隐性反馈 ALS 版本的参数，这个参数决定了偏好行为强度的基准（默认为 1.0）

nonnegative：决定是否对最小二乘法使用非负的限制（默认为 false）

可以调整这些参数，不断优化结果，使均方差变小。比如： imaxIter 越大， regParam 越小，均方差会越小，推荐结果较优。

接下来，把推荐模型放在训练数据上训练：

```
scala> val modelExplicit = alsExplicit.fit(training)


scala> val modelImplicit = alsImplicit.fit(training)
```

4.7 模型预测

使用训练好的推荐模型对测试集中的用户商品进行预测评分，得到预测评分的数据集:

```
scala> val predictionsExplicit = modelExplicit.transform(test)


scala> val predictionsImplicit = modelImplicit.transform(test)
```

将结果输出，对比真实结果与预测结果：

```
scala> predictionsExplicit.show()
```

```
scala> predictionsExplicit.show()
+------+-------+------+----------+
|userId|movieId|rating|prediction|
+------+-------+------+----------+
|     1|     16|   1.0| 1.5373151|
|     1|     21|   3.0| 1.6217059|
|     1|     36|   2.0| 2.3320606|
|     1|     57|   1.0| 1.6228204|
|     1|     78|   1.0|0.75977325|
|     1|     81|   1.0| 1.5203364|
|     1|     85|   3.0|-0.6404037|
|     2|     22|   1.0|-0.8504815|
|     2|     38|   1.0|  1.432808|
|     2|     54|   1.0|0.48037362|
|     2|     77|   1.0|-0.59617907|
|     2|     80|   1.0|   4.04151|
|     2|     88|   1.0|-0.3867868|
|     0|      3|   1.0|0.82268655|
|     0|     31|   1.0| 1.3085463|
|     0|     44|   1.0|0.88061965|
|     0|     45|   2.0| 1.4472697|
|     0|     61|   2.0| 1.6822193|
|     0|     68|   1.0| 3.4107647|
|     0|     94|   1.0| 1.3737634|
+------+-------+------+----------+
only showing top 20 rows
```

```
scala> predictionsImplicit.show()
```

```
scala> predictionsImplicit.show()
+------+-------+------+-----------+
|userId|movieId|rating|  prediction|
+------+-------+------+-----------+
|     1|     16|   1.0|   0.7996065|
|     1|     21|   3.0|  0.30570772|
|     1|     36|   2.0|  0.38876128|
|     1|     57|   1.0|-0.008767828|
|     1|     78|   1.0|  0.46393496|
|     1|     81|   1.0| 0.084658295|
|     1|     85|   3.0|  0.35942057|
|     2|     22|   1.0|   0.3127978|
|     2|     38|   1.0| -0.08139439|
|     2|     54|   1.0|  0.03473705|
|     2|     77|   1.0|  0.49821895|
|     2|     80|   1.0|   0.2916456|
|     2|     88|   1.0|  0.44200477|
|     0|      3|   1.0|  0.40153185|
|     0|     31|   1.0|-0.123872206|
|     0|     44|   1.0|  0.24522464|
|     0|     45|   2.0|  0.22329226|
|     0|     61|   2.0|  0.15047821|
|     0|     68|   1.0|  0.42929336|
|     0|     94|   1.0|  0.07206399|
+------+-------+------+-----------+
only showing top 20 rows
```

4.8 模型评估

通过计算模型的均方根误差来对模型进行评估，均方根误差越小，模型越准确：

```
scala> val evaluator = new
RegressionEvaluator().setMetricName("rmse").setLabelCol("rating").setPredictionCo
l("prediction")

scala> val rmseExplicit = evaluator.evaluate(predictionsExplicit)

scala> val rmseImplicit = evaluator.evaluate(predictionsImplicit)
```

```
scala> val evaluator = new RegressionEvaluator().setMetricName("rmse").setLabelCol("rating").setPredictionCol("prediction")
evaluator: org.apache.spark.ml.evaluation.RegressionEvaluator = RegressionEvaluator: uid=regEval_dc58e3687cce, metricName=rmse, throughOrigin=false

scala> val rmseExplicit = evaluator.evaluate(predictionsExplicit)
rmseExplicit: Double = 1.6619120673105368

scala> val rmseImplicit = evaluator.evaluate(predictionsImplicit)
rmseImplicit: Double = 1.8642955244407784
```

5 实验总结

在本次实验中，我的主要目标是掌握大数据环境的配置与管理，并实践 Hadoop 和 Spark 的基本应用。通过详细的步骤和严谨的实验操作，我成功配置了 Ubuntu 操作系统下的 Hadoop 和 Spark 环境，并通过实际的应用案例加深了对这些技术的理解。

1. 环境配置

配置环境是大数据实验的基础。我首先在 Linux 系统（Ubuntu 22.04）上安装了必要的软件包，包括 Java 和 Python 环境，以及 Hadoop 和 Spark 框架。在安装过程中，我遇到了一些挑战，比如环境变量的设置和 SSH 密钥的配置，但通过仔细阅读官方文档并实践，我成功解决了这些问题。

2. Hadoop 和 Spark 应用

在环境配置完成后，我进行了两个基本的大数据应用实验。第一个是使用 Hadoop 进行 WordCount 实验，这帮助我理解了如何在 Hadoop 环境下开发和运行大数据处理程序。

第二个应用是使用 Spark 进行电影评分预测的实验，这个实验不仅加深了我对 Spark 的使用理解，也让我体验到了大数据技术在实际应用中的强大能力。

3. 实践感悟

通过这次实验，我深刻体会到了大数据技术的复杂性和强大功能。虽然刚开始时配置环境的复杂性让我感到有些困难，但随着实验的深入，我逐渐掌握了核心技术和方法。实验不仅提升了我的技术技能，也增强了我解决问题的能力。

4. 未来展望

我计划继续深入学习更多关于大数据和人工智能的技术，希望能在未来的学习和研究中，将这些技术应用到更广泛的领域，解决实际问题。此外，我也希望能有机会参与更多的项目，以实际操作来不断提升自己的实战经验。

总之，这次实验是我学习道路上的一次重要实践，它不仅加深了我对大数据基本工具的理解，也为我未来的学术和职业生涯奠定了坚实的基础。