

**课程报告**

**课程名称：大数据处理**

**专业班级： 计算机本硕博2101班**

**学 号： U202115666**

**姓 名： 刘文博**

**指导教师： 石宣化**

**报告日期： 2024.6.16**

**计算机科学与技术学院**

**目 录**

[1. 课程实践总体说明 3](#_Toc169639497)

[i. 实践简介 3](#_Toc169639498)

[ii. 课程总结与课程建议 3](#_Toc169639499)

[2. 实践报告 1 5](#_Toc169639500)

[3. 实践报告 2 5](#_Toc169639501)

[4. 实践报告3 6](#_Toc169639502)

[5. 实践报告 4 8](#_Toc169639503)

[6. 实践报告5 12](#_Toc169639504)

* 1. 课程实践总体说明
     1. 实践简介

实践以vmware虚拟机为平台，让同学们动手在ubuntu环境下尝试用scala和java语言分别为hadoop和spark大数据处理框架编写包括hdfs文件系统中文件操作、wordcount、pagerank以及spark steaming应用。

具体而言，实验的内容为：

1. 环境搭建
   * 在VMware中安装和配置Ubuntu。
   * 安装和配置Java开发环境。
   * 安装和配置Hadoop。
   * 安装和配置Spark。
2. HDFS文件系统操作
   * 在HDFS中上传、下载、输出、查询、创建、删除、移动文件。
   * 在HDFS中查询目录下的所有文件，追加内容到文件中，删除、创建目录。
3. WordCount程序
   * 编写scala程序实现WordCount功能。
   * 分别使用命令行和Eclipse运行WordCount程序。
4. PageRank算法
   * 理解PageRank算法的基本原理和应用场景。
   * 编写Hadoop MapReduce程序实现PageRank算法。
   * 用web-Stanford数据集进行测试，查看程序执行结果。
5. Spark Streaming应用
   * 理解Spark Streaming的基本概念和使用场景。
   * 编写Spark Streaming程序，实现实时数据处理应用，进行实时WordCount。
   * 测试程序运行结果。
     1. 课程总结与课程建议

在完成这门课程实践之后，我有如下的收获：

* + 1. 对scala和java语言编写大数据处理程序有了初步的认识，学会使用Scala和Java编写简单的大数据处理程序，理解Scala和Java在大数据处理中的应用场景和各自的优势。
    2. 对spark、hadoop两个开源框架有了了解，能够使用其shell命令行进行基本的操作，掌握Spark和Hadoop的基本概念和架构，能够使用Spark和Hadoop的Shell命令行进行基本操作，如提交作业、查看作业状态、管理HDFS文件系统等。
    3. 掌握了scala的项目构建工具SBT（Scala Build Tool）的使用方法，能够自己写build.sbt文件管理项目依赖。
    4. 掌握了hadoop下的hdfs分布式文件系统和spark接口的使用，能够在HDFS上进行文件的上传、下载、删除等基本操作，掌握使用Spark的基本接口进行数据处理，如RDD操作、DataFrame和Dataset操作。
    5. 对包括mapreduce、函数式编程等大数据处理思想有了更深入的理解，理解MapReduce编程模型的基本原理和应用场景，掌握函数式编程的基本概念，并能够在Scala和Spark中应用这些思想进行数据处理。
    6. 能够实现一些大数据处理的经典应用，例如pagerank、wordcount，理解这些算法的工作原理，并能够根据需求进行优化和扩展。
    7. 能够应对一些scala和java语言的debug，能够解决在编写大数据处理程序时遇到的常见问题和错误。
    8. 知道了如何管理项目所需要的jars包，能够根据项目需求添加、更新和删除依赖，确保项目的正常运行

总的来说，在做完这门实践之后，我对大数据处理的核心概念和技术有了深入的理解，并能够独立完成一些经典的大数据处理任务。这些技能和经验将为同学们未来在大数据领域的学习和工作打下坚实的基础。

课程建议：

课程所用的教程比较早，其中所有的jars包版本过于陈旧，一些方法和类的定义可能和最新的jars包不同带来项目依赖的问题，且任务书所给的代码大多数都有一些bug不能直接运行。希望能更新教程的内容或者给出提示，同时纠正任务书中代码的bug。

* 1. 实践报告 1

大数据处理实验的第一部分是linux、hadoop、spark等工具的配置。

基本上按照任务书所给教程来就可以了。

但在配置环境变量时可以进行一些简化，我最终修改的.bashrc 如下所示：

export JAVA\_HOME=/usr/lib/jvm/jdk1.8.0\_162

export JRE\_HOME=${JAVA\_HOME}/jre

export CLASSPATH=.:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

export HADOOP\_HOME=/usr/local/hadoop

export HADOOP\_OPTS="-Djava.library.path=$HADOOP\_HOME/lib/native"

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native

export SPARK\_HOME=/usr/local/spark

export PATH=${JAVA\_HOME}/bin:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin:$SPARK\_HOME/bin:/usr/local/sbt:$PATH

export TERM=xterm-color

其中加入了sbt和spark所在的目录到路径中去，方便后续执行sbt和spark相关的命令。

最后的export TERM=xterm-color命令能够解决在执行sbt package命令时报错java.lang.NumberFormatException: For input string: "0x100的问题。

* 1. 实践报告 2

大数据处理实验的第二部分是对hdfs文件系统的操作。

该实验所有操作的scala代码都需要在spark中执行且完成前面实验一对spark环境的配置。

故需要在我的sparkapp文件夹下创建11个子项目分别进行打包运行来测试scala代码的运行结果。Sparkapp目录的结构如下图 1所示。

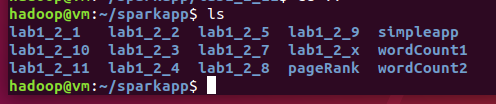


图 1 sparkapp目录结构

每个子文件夹都按照SBT要求的目录格式进行构建。用sbt package命令进行打包然后提交到spark中进行运行查看运行结果。

当然，在此之前需要先按照教程在eclipse中编写HDFSexample程序。

在编写HDFSexample程序时，遇到了报错

java.lang.RuntimeException: java.lang.ClassNotFoundException: Class org.apache.hadoop.hdfs.DistributedFileSystem not found

这是因为教程中给的 jar 包并不完整，需要在项目的 build path 中 configure buildpath 中再添加一个 hadoop-hdfs-client-xxx 包。

其次任务书中给出的scala代码存在许多bug，在编译时需要进行修改：

1. 所有代码的import语句后都没有定义类，需要用一个类把 import 后面的代码包起来
2. 向HDFS中上传任意文本文件的代码中，需要将 val buffer = new Array 修改为 val buffer = new Array[[Byte](https://file+.vscode-resource.vscode-cdn.net/d:/_Study/bp/4096)]
3. 从HDFS中下载指定文件的代码中，将 val localFile 修改为 var,并且在每次循环中更新它
4. 将HDFS中指定文件的内容输出到终端中的代码中，报错 Class org.apache.hadoop.hdfs.DistributedFileSystem not found,在该项目的 build.sbt的依赖项中加上所需的jars包(既可以用本地hadoop文件夹下的jars包，也可以远程托管)
5. 向HDFS中指定的文件追加内容，由用户指定内容追加到原有文件的开头或结尾的代码中，需要将两种情况的代码合并到一块，在运行时由用户指定追加到开头还是结尾，sbt 运行指令形如sbt "run end /user/hadoop/text.txt /mnt/hgfs/shared/a.txt"
6. 删除HDFS中指定的目录，由用户指定目录中如果存在文件时是否删除目录的代码与上面的同理，需要合并代码并由用户指定是否强制删除。

最终将把sbt打包得到jar包用spark-submit命令提交到spark中测试运行结果。

* 1. 实践报告3

实验的第三部分是分别在spark-shell中和scala项目中实现wordCount功能。

在spark-shell中：

首先需要在/usr/local/spark目录下执行bin/spark-shell命令打开spark-shell，

之后按照任务书中的命令进行wordcount操作。

值得注意的是，如果直接按任务书中那样new一个SparkContext对象会报错没有SparkContext对象，在此之前需要在spark-shell中执行import org.apache.spark.SparkContext命令来导入相关对象的定义。

之后停止原有的sc，即执行sc.stop()命令，然后才可以执行val sc = new SparkContext。

因为本地路径下没有文件<file://home/stu/softwore/hadoop/README.txt>，故用命令val textFile = sc.textFile("file:///mnt/hgfs/shared/a.txt")进行替换，指定本机下的 a.txt 文件。

统计文件中各个单词的计数，得到结果：

res3: Array[(String, Int)] = Array((T,1), (d,61), (z,2), (",36), (4,17), (8,6), (p,102), (L,6), (x,23), (R,10), (B,1), (6,6), (P,5), (t,152), (.,142), (0,7), (b,5), (h,68), (2,24), (" ",198), ($,16), ("",6), (>,16), (n,142), (f,25), (~,1), (j,10), (J,1), ((,32), (v,25), (F,4), (V,8), (:,42), (,,6), (l,116), (<,16), (N,10), (r,141), (D,3), (w,2), (),32), (=,9), (e,208), (s,131), (/,30), (M,18), (7,4), (5,4), (a,224), (\_,3), (O,12), (;,1), (y,10), (A,13), (u,26), (#,2), (i,92), (I,13), (o,191), (k,33), (9,5), (3,15), (%,13), (K,1), (q,5), (-,16), (S,31), (C,28), (E,2), (1,24), (g,45), (W,3), (+,4), (c,99), (m,21))

在scala项目中：

与前面的实践内容类似，同样根据任务书中的参考代码2用sbt工具构建scala原生项目。

但直接运行任务书中的代码会报错：

[error] /home/hadoop/sparkapp/wordCount/src/main/scala/wordCount.scala:1: object spark is not a member of package org.apache

[error] import org.apache.spark.rdd.RDD

[error] ^

[error] /home/hadoop/sparkapp/wordCount/src/main/scala/wordCount.scala:2: object spark is not a member of package org.apache

[error] import org.apache.spark.{SparkConf, SparkContext}

[error] ^

[error] two errors found

[error] (compile:compileIncremental) Compilation failed

[error] Total time: 4 s, completed Jun 13, 2024 9:22:49 AM

故修改sbt文件加入/usr/local/spark/jars中的本地包依赖即可解决

修改后的build.sbt内容如下：

name := "Simple Project"

version := "1.0"

scalaVersion := "2.12.18"

val hadoopVersion = "3.1.3"

val hadoopOrg = "org.apache.hadoop"

// 定义一个函数来获取多个目录中的所有 JAR 文件

def jarDependencies(directories: Seq[File]): Seq[File] = {

directories.flatMap { directory =>

if (directory.exists && directory.isDirectory) {

directory.listFiles.filter(\_.getName.endsWith(".jar"))

} else {

Seq.empty[File]

}

}

}

/\*

libraryDependencies ++= Seq(

"org.apache.spark" %% "spark-core" % "3.5.1",

hadoopOrg % "hadoop-common" % hadoopVersion,

hadoopOrg % "hadoop-nfs" % hadoopVersion,

hadoopOrg % "hadoop-hdfs" % hadoopVersion,

hadoopOrg % "hadoop-hdfs-nfs" % hadoopVersion,

hadoopOrg % "hadoop-hdfs-client" % hadoopVersion

)

\*/

// 指定三个目录

val hadoopLibDirectory1 = file("/usr/local/hadoop/share/hadoop/hdfs/lib")

val hadoopLibDirectory2 = file("/usr/local/hadoop/share/hadoop/common/lib")

val sparkLibDirectory = file("/usr/local/spark/jars")

unmanagedJars in Compile ++= {

jarDependencies(Seq(hadoopLibDirectory1, hadoopLibDirectory2, sparkLibDirectory))

}

unmanagedJars in Compile ++= Seq(

file("/usr/local/hadoop/share/hadoop/common/hadoop-common-3.1.3.jar"),

file("/usr/local/hadoop/share/hadoop/common/hadoop-nfs-3.1.3.jar"),

file("/usr/local/hadoop/share/hadoop/hdfs/hadoop-hdfs-3.1.3.jar"),

file("/usr/local/hadoop/share/hadoop/hdfs/hadoop-hdfs-nfs-3.1.3.jar"),

file("/usr/local/hadoop/share/hadoop/hdfs/hadoop-hdfs-client-3.1.3.jar")

)

resolvers ++= Seq(

"Apache Releases" at "https://repository.apache.org/content/repositories/releases/",

"Maven Central" at "https://repo1.maven.org/maven2/"

)

根据任务书中的参考代码 1 直接创建 scala 项目，运行也会报错，将路径修改为val filePath = "/mnt/hgfs/shared/a.txt"即可解决。

* 1. 实践报告 4

实验的第四部分是

基于eclipse软件，编写pageRank应用。pageRank应用的代码如下：

import java.util.ArrayList;

import java.util.List;

import org.apache.spark.api.java.\*;

import org.apache.spark.api.java.function.Function2;

import org.apache.spark.SparkConf;

import scala.Tuple2;

public class PageRank {

public static void main(String[] args) {

//set spark conf

SparkConf conf = new SparkConf().setAppName("PageRank").setMaster("local[\*]");

JavaSparkContext sc = new JavaSparkContext(conf);

//load graph data

JavaRDD<String> lines = sc.textFile("hdfs://localhost:9000/user/hadoop/web-Stanford.txt")

.zipWithIndex()

.filter(tuple -> tuple.\_2 >= 4)

.map(Tuple2::\_1);

//fromnode as key tonode as value

JavaPairRDD<Long, Long> links = lines.mapToPair(line -> {

String[] parts = line.split("\\s+");

return new Tuple2<>(Long.parseLong(parts[0]), Long.parseLong(parts[1]));

});

JavaPairRDD<Long, Iterable<Long>> groupedLinks = links.groupByKey().cache();

// 获取节点总数 N

long N = groupedLinks.keys().count();

// 初始化每个节点的 PageRank 值

JavaPairRDD<Long, Double> initialRanks = groupedLinks.mapValues(v -> 1.0 / N);

// 定义阻尼系数和误差阈值

double dampingFactor = 0.85;

double tolerance = 0.0001;

double delta = Double.MAX\_VALUE;

int maxIterations = 500;

JavaPairRDD<Long, Double> ranks = initialRanks;

int iterations = 0;

// PageRank 迭代计算，直到收敛

while (delta > tolerance && iterations < maxIterations) {

JavaPairRDD<Long, Double> contributions = groupedLinks.join(ranks).flatMapToPair(tuple -> {

long nodeId = tuple.\_1;

Iterable<Long> neighbors = tuple.\_2.\_1;

double rank = tuple.\_2.\_2;

List<Tuple2<Long, Double>> results = new ArrayList<>(); // store the result

int size = 0;

for (Long neighbor : neighbors)

{

size++;

}

for(Long neighbor : neighbors)

{

results.add(new Tuple2<>(neighbor, rank / size));

}

return results.iterator();

});

JavaPairRDD<Long, Double> newRanks = contributions

.reduceByKey((Function2<Double, Double, Double>) (a, b) -> a + b)

.mapValues((org.apache.spark.api.java.function.Function<Double, Double>) v -> (1 - dampingFactor) / N + dampingFactor \* v);

// 计算 PageRank 值的变化

delta = ranks.join(newRanks)

.mapToDouble(tuple -> Math.abs(tuple.\_2.\_1 - tuple.\_2.\_2))

.sum();

// 更新PageRank值

ranks = newRanks;

iterations += 1;

System.out.println("Iteration " + iterations + " completed with delta = " + delta);

}

// 显示结果

// List<Tuple2<Long, Double>> output = ranks.collect();

// for (Tuple2<Long, Double> tuple : output) {

// System.out.println("Node " + tuple.\_1 + " has rank " + tuple.\_2 + ".");

// }

JavaRDD<String> result = ranks.map(tuple-> "Node " + tuple.\_1 + " had rank " + tuple.\_2);

result.saveAsTextFile("/home/hadoop/eclipse-workspace/pageRank/pagerank-result");

sc.stop();

}

}

代码思路：

首先指定在本地集群中运行。然后导入web-Stanford.txt的内容，将其中开头的4行丢掉(因为开头的四行是注释不是实际数据)。

之后将每一行用空格分割成两个部分（即fromnode和tonode），用一个pair进行保存。然后将fromnode作为关键字进行合并，得到同一个fromnode的所有tonode的列表（同样是一个pair，前者为fromnode，后者是tonode构成的列表）。

对fromnode的不同值进行统计，即为图中所有有边节点的个数。

之后构造一个所有节点的初始的pagerank的数据结构，同样是一个pair，前者为节点的编号，后者为1/n，n为上一步统计的节点的总数。

再定义阻尼系数为0.85，误差阈值为0.0001，最大迭代次数为500，初始时的误差为MAX。准备开始pageRank的迭代过程。

迭代过程如下：

因为每个出节点向其邻居均匀地贡献自身的pagerank值，故我们先将将节点的rank值与出节点建立映射，之后在第一个 for循环中统计其邻居的数量，在第二个计算出节点对每个邻居的贡献值，用flatMaptoPair函数将所有节点对其每个邻居的贡献值合并到一个大的resultRDD中，即为contributions。

之后将contributions和原来的rank值用mapreduce方法进行运算得到更新后的rank值，再计算两次迭代中所有节点的rank值变化的绝对值之和，如果小于误差阈值就终止迭代过程。

最后将rank值的结果输出到项目目录下的pagerank-result.txt中。

为了方便地调用spark的接口，需要在项目的 build path 中 configure buildpath 中导入 hadoop 目录下的相关 jars 包和 spark/jars 目录下的所有 jars 包。值得注意的是，当出现多个 jars 包的不同版本时，运行时会报错类或方法的定义找不到，应该仔细检查导入的 jars 包看是否有重名的，将其中旧版本的 jars 包移除。（spark和hadoop路径下的jars包有很多是相同的，但hadoop中的jars包一般版本会比较旧，一定要移除）

* 1. 实践报告5

实验的第五部分是基于Spark Streaming变成模型实现实时wordCount。

同样直接基于任务书中的代码构建scala项目即可/

但如果直接按照任务书中的代码运行会报错Error initializing SparkContext :A master URL must be set in your configuration。

故在任务指导书的参考代码的基础上指定在本地集群上运行

val spark = SparkSession.builder

.appName("StructuredNetworkWordCount")

.master("local[\*]")

.getOrCreate()

同时，教程中的 spark 和 scala 版本太老，要想运行应该修改 simple.sbt 为

name := "Simple Project"

version := "1.0"

scalaVersion := "2.12.18"

libraryDependencies += "org.apache.spark" %% "spark-sql" % "3.5.1"

其中scalaVersion和spark-sql后的版本号对应这我的虚拟机中安装的scala和spark的版本。

在运行应用前要在另一终端先执行nc -lk 9999命令,使用 netcat 工具监听 9999 端口，否则执行应用可能会报connection refused错误。

这之后在另一终端输入hello world, hello Beijing, hello world即可，得到如下输出

+--------+-----+

| value|count|

+--------+-----+

| world,| 1|

| hello| 3|

|Beijing,| 1|

| world| 1|

+--------+-----+