

**大数据处理实验报告**

实验三：MapReduce的基本操作

专业班级： 计算机2011班

学 号： U202010755

姓 名： 路昊东

指导教师： 郑渤龙

报告日期： 2022年 3 月 24 日

**计算机科学与技术学院**

**《大数据处理》课程实验报告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 实验地点 | | 南一楼804 | 课程名称 | | 大数据处理 | | |
| 实验题目 | | MapReduce的基本操作 | 成绩 |  | | 指导教师 | 郑渤龙 |
| 教师评价 | □ 实验过程正确； □ 源程序/实验内容提交； □ 程序结构/实验步骤合理；  □ 实验结果正确； □ 语法、语义/命令正确； □ 报告规范；  其他： | | | | | | |
| **一、实验目的**   1. 了解MapReduce的用途 2. 掌握MapReduce的基本命令   **二、实验内容**   1. 实验环境配置 2. MapReduce (55’) 3. Spark (35’) 4. 附加题 (学有余力可以课下自行尝试，不算分) 5. 实验总结 (10’)   **三、实验环境**  **一. 华为云环境：**  1. 区域：“华北-北京四”   1. MRS集群： 2. 名称：mrs\_dong 3. 集群版本：1.9.2 4. 类型：分析集群 5. 组件：HBase、Hive、Tez、Spark 6. 计费模式：按需计费 7. 可用区：可用区2 8. 弹性公网IP绑定：是 9. CPU架构：鲲鹏计算（集群高可用关闭；core节点数：1） 10. 委托、数据盘加密默认，告警“关闭” 11. kerberos认证：关闭 12. 登录方式：密码 13. 虚拟私有云：默认创建 14. 弹性公网 IP： 15. 计费模式：按需计费 16. 线路：全动态 BGP 17. 公网带宽：按流量计费 18. 带宽大小：5M 19. 购买数量：1 20. 安全组规则：master1节点：入方向全部放通 21. **实验的本地软硬件环境如下：** 22. CPU型号：AMD Ryzen 7 4800H 23. CPU主频：2.9GHz 24. 核心/线程数：八核心/十六线程 25. 内存容量：16GB（8GB×2） 26. 操作系统：ubuntu20.04   **四、实验过程或步骤（源程序）**  下面是本次实验的具体内容及步骤、实验的详细记录、实验结果分析：  **1. 实验环境配置**   1. 服务购买： 2. 登录控制台 3. 购买MRS服务 4. 购买弹性公网IP 5. 绑定EIP 6. 创建集群 7. 配置安全组 8. 登录服务器远程master节点 9. **MapReduce** 10. **进入hadoop**   **①**打开**cloudshell**登录云服务器，命令行输入：  **cd /opt/client/HDFS/hadoop**  **②** 终端截图如下：    这里使用**cd**命令，进入目录**/opt/client/HDFS/hadoop**中。   1. **添加环境变量**   **①**命令行输入：  **export HADOOP="/opt/client/HDFS/hadoop/share/hadoop"**  **export CLASSPATH ="$HADOOP/common/ hadoop-common-2.8.3-mrs-1.9.0. jar:$HADOOP/mapreduce/hadoop-mapreduce-client-core-2.8.3-mrs-1.9.0.**  **jar:$HADOOP/common/lib/comm ons-cli-1.2.jar:$CLASSPATH"**  **②** 终端截图如下：    这里使用**export**命令，用于设置或者显示环境变量。我们知道，在 shell 中执行程序时，shell 会提供一组环境变量。**export** 可新增，修改或删除环境变量，供后续执行的程序使用。**export** 的效力仅限于该次登陆操作。  **export [-fnp][变量名称]=[变量设置值]** 为其使用语法，参数**[-fnp]**其中**-f** 代表[变量名称]中内容为函数名称。**-n** 删除指定的变量（变量实际上并未删除，只是不会输出到后续指令的执行环境中）。**-p** 列出所有的shell赋予程序的环境变量。  此处就定义了**HADOOP**和**CLASSPATH**两个环境变量并赋值。   1. **创建java程序WordCount.java,在里面输入代码** 2. 命令行输入：   **vim WordCount.java**  自动创建**WordCount.java**文件   1. 在编辑界面中使用a进行插入：   **import java.io.IOException;**  **import java.util.StringTokenizer;**  **import org.apache.hadoop.conf.Configuration;**  **import org.apache.hadoop.fs.Path;**  **import org.apache.hadoop.io.IntWritable;**  **import org.apache.hadoop.io.Text;**  **import org.apache.hadoop.mapreduce.Job;**  **import org.apache.hadoop.mapreduce.Mapper;**  **import org.apache.hadoop.mapreduce.Reducer;**  **import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;**  **import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;**  **public class WordCount {**  **public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {**  **private final static IntWritable one = new IntWritable(1);**  **private Text word = new Text();**  **public void map(Object key, Text value, Context context) throws IOException, InterruptedException {**  **StringTokenizer itr = new StringTokenizer(value.toString());**  **while (itr.hasMoreTokens()) {**  **word.set(itr.nextToken());**  **context.write(word, one);**  **}**  **}**  **}**  **public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {**  **private IntWritable result = new IntWritable();**  **public void reduce(Text key, Iterable<IntWritable> values,Context context) throws IOException, InterruptedException {**  **int sum = 0;**  **for (IntWritable val : values) {**  **sum += val.get();**  **}**  **result.set(sum);context.write(key, result);**  **}**  **}**  **public static void main(String[] args) throws Exception {**  **Configuration conf = new Configuration();**  **Job job = Job.getInstance(conf, "word count");**  **job.setJarByClass(WordCount.class);**  **job.setMapperClass(TokenizerMapper.class);**  **job.setCombinerClass(IntSumReducer.class);**  **job.setReducerClass(IntSumReducer.class);**  **job.setOutputKeyClass(Text.class);**  **job.setOutputValueClass(IntWritable.class);**  **FileInputFormat.addInputPath(job, new Path(args[0]));**  **FileOutputFormat.setOutputPath(job, new Path(args[1]));**  **System.exit(job.waitForCompletion(true) ? 0 : 1);**  **}**  **}**   1. 输入完成按键**Esc**退出编辑 2. 使用**:wq**指令保存退出。 3. 终端截图如下：       此时使用**ls**指令查看**hadoop**目录下的文件如下：    可见多出**WordCount.java**文件，写入成功。   1. **编译WordCount.java**   **①**命令行输入：  **javac WordCount.java**  **或者**加入参数，编译如下：  **javac WordCount.java -cp $(hadoop classpath)**  **②** 终端截图如下：    我们知道，**javac**为编译指令，**-cp**参数设置类路径。  此时使用**ls**指令查看**hadoop**目录下的文件，可见多出**WordCount.class**等与编译有关的文件，表明编译成功。   1. **创建文件test1，内容为hello hust，文件test2，内容为hello学号，将他们放入hdfs的/input文件夹内。**   **①**命令行输入：  **vim test1**  **vim test2**  **hdfs dfs -mkdir /input**  **hdfs dfs -put test1 /input**  **hdfs dfs -put test2 /input**  **hdfs dfs -ls /input**  **②** 终端截图如下：      这里使用**vim**指令创建并写入两个新文件**test1、test2**，之后使用**hdfs**的相关指令在根目录下创建**input**文件夹，使用**-put**指令将本地文件移入**input**文件夹下，最后使用**-ls**命令查看相关指令，得到两个文件的权限、写入时间等信息，可见文件创建、移动成功。   1. **打包为jar包**   **①**命令行输入：  **jar cf WordCount.jar WordCount\*.class**  **②** 终端截图如下：    我们知道，**jar**为打包指令，关于打包，这个压缩包和Winzip的压缩格式是一样的，区别在于**jar**压缩的文件默认多一个**META-INF**的文件夹,该文件夹下包含一个**Manifest.mf(清单文件)**的文件。通常来说**jar**命令得到的压缩包有三种(压缩格式完全一样,只是后缀名不同而已):  　　A、**\*.jar** - 它里面包含N个class文件。  　　B、**\*.war (web)** - 它是一个web应用打包生成的。  　　C、**\*.ear(Enterprise)** -它是一个企业应用打包生成的包  这条指令中，**-c** 表示创建新的归档文件，**-f** 用来指定归档文件名（为压缩包指定名字）。此外还有**-t、-x、-u、-v、-m、-e**等参数。  此时使用**ls**指令查看**hadoop**目录下的文件，可见多出**WordCount.jar**等与编译有关的文件，表明编译成功。   1. **运行WordCount.jar将hdfs的/input作为输入，/output作为输出，并打印/output目录下的文件，显示出词频统计的结果**   **①** 命令行输入：  **export HADOOP\_CLASSPATH = $HADOOP\_CLASSPATH:/opt/client/HDFS/hadoop/ WordCount.jar**  **hadoop jar WordCount.jar WordCount hdfs:///input hdfs:///output**  **hdfs dfs -cat /output/part-r-00000p**  **②** 前两指令（运行jar包）终端截图如下：      这里使用**export** 命令，新建环境变量**HADOOP\_CLASSPATH**到**/opt/client/HDFS/ hadoop/**目录下的**jar**包**WordCount.jar**。通过了解操作系统**PATH**环境变量的用途，上述步骤代表部署**hadoop**的时候已经将**$HADOOP\_HOME/bin**目录配置到了操作系统的**PATH**环境变量中。所以**$HADOOP\_HOME/bin**目录下的**hadoop**文件是可以直接运行的。  随后使用**hadoop jar**指令：这里**hadoop**指**$HADOOP\_HOME/bin**下的**shell**脚本名，**jar**是**hadoop**脚本需要的**command**参数，**WordCount.jar**是要执行的**jar**包在本地文件系统中的完整路径，参递给**RunJar**类。**WordCount**为main方法所在的类，参递给**RunJar**类。**/input**为传递给**WordCount**类，作为DFS文件系统的路径，指示输入数据来源。**/output**为传递给WordCount类，作为DFS文件系统的路径，指示输出数据路径。  这里指运行当前配置的**hadoop**的环境变量的，刚打包好的 **jar**包，的将**hdfs**的**/input**目录下的**test1，test2**文件作为输入，**/output**作为文件输出的目的目录。  最后的**hdfs dfs -cat**指令使用**cat**参数，意为显示文件内容到标准输出上，后面的**part-r-00000p**文件为上述需要输出的词频统计结果：  **U202010755 1**  **hello 2**  **hust 1**  终端截图如图所示：    可见结果正确，词频统计成功！   1. **Spark** 2. **打开spark**   **①** 命令行输入：  **pyspark**  **②** 终端截图如下：    当看到**Welcome** 以及**SparkSession available as 'spark'.>>>** 的字样，表示成功打开了**spark**。   1. **读取hdfs文件内容**   **①** 命令行输入：  **lines = spark.read.text("hdfs:///input").rdd.map(lambda r: r[0])**  **②** 终端截图如下：    这里使用了**spark-textFile**，查看**textFile**函数，  **/\*\***  **\* Read a text file from HDFS, a local file system (available on all nodes), or any**  **\* Hadoop-supported file system URI, and return it as an RDD of Strings.**  **\*/**  **def textFile(**  **path: String,**  **minPartitions: Int = defaultMinPartitions): RDD[String] = withScope {**  **assertNotStopped()**  **hadoopFile(path, classOf[TextInputFormat], classOf[LongWritable], classOf[Text],**  **minPartitions).map(pair => pair.\_2.toString).setName(path)**  **}**  函数中，**path: String** 是一个**URI**，可以是HDFS、本地文件（全部的节点都可以），或者其他**Hadoop**支持的文件系统（URI返回的是一个字符串类型的RDD，也就是是**RDD**的内部形式是**Iterator[(String)]）**；**minPartitions= math.min (defaultParallelism, 2)** 是指定数据的分区，如果不指定分区，当你的核数大于2的时候，不指定分区数那么就是 2。当你的数据大于128M时候，Spark是为每一个快（block）创建一个分片。  这里命令意为从hdfs中读取整个**input**文件夹的文件内容，由上图得知，读取成功。   1. **词频统计**   **①** 命令行输入：  **counts = lines.flatMap(lambda x: x.split(' ')).map(lambda x: (x,**  **1）).reduceByKey(lambda x, y: x + y)**  **output = counts.collect()**  **②** 终端截图如下：    这里使用**spark**的词频统计，读文本文件生成**RDD lines**。命令行中**flatmap**()意为将一行一行的文本分割成单词，**map**()意为转换成键值对，**reduceByKey**()意为统计词频，之后使用**collect**()收集结果。  此外spark相关语法还有去掉停用词、使用filter过滤单词长度、转换大小写、按字母或者词频排序等功能，等待后续探索。   1. **输出词频统计结果**   **①** 命令行输入：  **output**  **②** 终端截图如下（显示统计结果如下图所示）：    **>>> output**  **[(u'hust', 1), (u'hello', 2), (u'U202010755', 1)]**  通过与上步骤使用hadoop统计词频结果功能相比，结果相同，均为正确，故spark统计词频完全正确！   1. **附加题**   **1.将以上两个文件存入 hdfs 2.编写 mapreduce 的程序， 输出每门课的平均成绩。 3.编写 mapreduce 的程序， 输出每位同学有多少门课成绩低于 75 分。**   1. **将以上两个文件存入 hdfs**   **①** 仍然使用上文编辑test1、test2文件的格式将文件编辑保存存入hdfs的input文件夹下。  **②** 终端截图如下：    观察到，文件写入成功。   1. **编写 mapreduce 的程序， 输出每门课的平均成绩** 2. 这里使用分别统计数学和英语成绩平均成绩的方法，建立两个类**MathAverageCount**和**EnglishAverageCount**，仿照之前**WordCount**的格式，修改一下**map()**方法读取**token**的形式与方式以及**intSumReducer()**方法**get**的形式与求平均方法。 3. 编写完源代码之后，在本地编译、配置环境变量、打包程序为jar包，然后在hadoop下运行，注意运行时要删除之前的/output文件夹。   **②** 终端截图如下：  middle_img_v2_80df5e23-abc1-4fd7-a4d2-1b0e3e8ca8cg  origin_img_v2_258f46bb-8f8b-4ddb-bb92-bd46ada1317g  这里可以看到，结果正确，故程序编写正确。   1. **编写 mapreduce 的程序， 输出每位同学有多少门课成绩低于 75 分**   **①** 这里依旧沿用上步骤的方法，但是要本地自己创建一个类，更改一下映射关系和上步骤更改的两个程序，同样能够输出正确的结果。  **②** 终端截图如下：  origin_img_v2_2f6711b9-59e2-4089-8cf8-ea95d86e27bg  这里可以看到，结果正确，故程序编写正确。  **五、出现的问题与解决方案**  **问题与解决方案**   1. 本次实验问题主要出现在**mapreduce**环节。   开始由于没有打包**jar**包，在运行**hadoop**的时候自然无法看到**.jar**文件导致运行失误，如图所示：      之后使用**jar**指令，对于**cf**参数使用不大了解，最后通过学习掌握了基本知识，程序运行成功。  同时我了解到，由于本次实验使用的**mrs**版本为**1.9.2**，若是选择版本号为**3**的**mrs**，由于内部的**jdk**版本发生改变，示例程序将不能够正确使用，若要正确运行，**main**方法中的**args[0]、args[1]**必须改成**args[1]和args[2]**，如图所示：     1. 本次实验仍旧遇到了一些**系统问题**，比如MRS创建集群20%自动失败、绑定IP出现问题、删除资源时删除虚拟私有云失败等等，最终通过一次次地重启和等待得到解决。   **六、实验总结**  本次实验同样使我收获颇丰。  首先，也是结合课上的学习与实验的实践，我对于MapReduce有了更为深入的了解。我们都知道，hadoop要分布式包括两部分，一是分布式文件系统hdfs,一部是分布式计算框，就是mapreduce,缺一不可，也就是说，可以通过mapreduce很容易在hadoop平台上进行分布式的计算编程。  **关于mapreduce到底是什么，这里举一个例子，**如果想统计下过去10年计算机论文出现最多的几个单词，看看大家都在研究些什么，那收集好论文后，该怎么办呢？当然可以写一个小程序，把所有论文按顺序遍历一遍，统计每一个遇到的单词的出现次数，最后就可以知道哪几个单词最热门了。 这种方法在数据集比较小时，是非常有效的，而且实现最简单，用来解决这个问题很合适。第二种方法是写一个多线程程序，并发遍历论文。这个问题理论上是可以高度并发的，因为统计一个文件时不会影响统计另一个文件。当我们的机器是多核或者多处理器，方法二肯定比方法一高效。但是写一个多线程程序要比方法一困难多了，我们必须自己同步共享数据，比如要防止两个线程重复统计文件。  第三种方法是把作业交给多个计算机去完成。我们可以使用方法一的程序，部署到N台机器上去，然后把论文集分成N份，一台机器跑一个作业。这个方法跑得足够快，但是部署起来很麻烦，我们要人工把程序copy到别的机器，要人工把论文集分开，最痛苦的是还要把N个运行结果进行整合（当然我们也可以再写一个程序）。**MapReduce本质上就是上述的方法三。**但是如何拆分文件集，如何copy程序，如何整合结果这些都是框架定义好的。我们只要定义好这个任务（用户程序），其它都交给MapReduce。  我们还知道MapReduce擅长处理大数据，本人感觉这其实来自于MapReduce**“分而治之”的思想**。通过查阅资料我了解到，在mapreduce里面，**Mapper**负责“分”，即把复杂的任务分解为若干个“简单的任务”来处理。“简单的任务”包含三层含义：一是数据或计算的规模相对原任务要大大缩小；二是就近计算原则，即任务会分配到存放着所需数据的节点上进行计算；三是这些小任务可以并行计算，彼此间几乎没有依赖关系。同时**Reducer**负责对map阶段的结果进行汇总。至于需要多少个Reducer，用户可以根据具体问题，通过在mapred-site.xml配置文件里设置参数mapred.reduce.tasks的值，缺省值为1。  **在本次实验中，**我首先对于MapReduce的基本命令有了一定的了解比如hadoop执行jar包等，同时巩固了一些linux的具体操作指令比如对文件的增删查改操作。**让我收获最大的是**了解了MapReduce中自定义map和reduce函数的原理和方法，并进行了实践。我了解到**Mapper 类**，有四个泛型：KEYIN、VALUEIN、KEYOUT、VALUEOUT，前面两个KEYIN、VALUEIN 指的是map 函数输入的参数key、value 的类型；后面两个KEYOUT、VALUEOUT 指的是map 函数输出的key、value 的类型。  Mapper之后的**Reducer 类**，也有四个泛型，同理，分别指的是reduce 函数输入的key、value类型（这里输入的key、value类型通常和map的输出key、value类型保持一致）和输出的key、value 类型。  写完了mapper和reduce相关的处理函数，在**Main方法**中，我主要做了三件事：一是指定输入、输出目录；二是指定自定义的Mapper类和Reducer类；三是提交作业。  通过详细阅读示例代码，我发现在**Mapper**类和**Reducer**类中都使用了**Hadoop**自带的基本数据类型，例如**String**对应**Text**，**long**对应**LongWritable**，**int**对应**IntWritable**。搜集资料发现这是因为**HDFS**涉及到序列化的问题，**Hadoop**的基本数据类型都实现了一个**Writable**接口，而实现了这个接口的类型都支持序列化。  同时在附加题中我尝试了使用**ToolRunner**类改写成绩统计类，将代码与配置隔离、使得修改**MapReduce**参数不需要修改**java**代码、打包、部署，提高工作效率！！  当然除了**MapReduce**，本节课我还体验了**spark**的用法，查阅资料得知，**Spark**计算快于**MapReduce**：这其中根本原因在于 **DAG** 计算模型。一般而言，**DAG** 相比**MapReduce** 在大多数情况下可以减少**shuffle**次数。**Spark** 的 **DAGScheduler** 相当于一个改进版的 **MapReduce**，如果计算不涉及与其他节点进行数据交换，**Spark** 可以在内存中一次性完成这些操作，也就是中间结果无须落盘，减少了磁盘 IO 的操作。但是，如果计算过程中涉及数据交换，**Spark** 也是会把 **shuffle** 的数据写磁盘的！同时**MapReduce**是将中间结果保存到磁盘中，减少了内存占用，牺牲了计算性能。**Spark**是将计算的中间结果保存到内存中，可以反复利用，提高了处理数据的性能。同时**Spark**是粗粒度资源申请，而**MapReduce**是细粒度资源申请。不过课上短暂的体验让我感觉两者都是数据处理的好方法！  总之，本次实验属于是对于我理论学习的一个比较好的实践，依然感谢老师给了我们免费体验实践的机会，让我们学到了有趣的新知识，我将继续学习有关大数据处理的知识，拓展自己，提升自己的学习深度与广度！ | | | | | | | |