

**大数据处理实验报告**

实验三：MapReduce的基本操作

专业班级： CS2002

学 号： U202015324

姓 名： 屈绍博

指导教师： 石宣化

报告日期： 2022.4.8

**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 |  |
| 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**  **1）集群** 版本：MRS 1.9.2 类型：分析集群 组件选择：Hadoop 2.8.3, HBase 1.3.1, Hive 2.3.3 and Tez 0.9.1 **（2）CPU架构** Master节点 鲲鹏通用计算增强型： 4 vCPUs | 16 GB | kc1.xlarge.4 系统盘：通用型SSD 100 GB x 1 数据盘：通用型SSD 200 GB x 1 分析Core节点 鲲鹏通用计算增强型： 4 vCPUs | 16 GB | kc1.xlarge.4 系统盘： 通用型SSD 100 GB x 1 数据盘： 通用型SSD 100 GB x 1  在Windows11操作系统上，使用chrome浏览器登录网页端华为云CloudShell，使用控制台提供的VNC登录服务器，在linux环境下进行实验。  **四、实验过程或步骤（源程序）**   * + - 1. **Map Reduce**  1. **进入Hadoop**   输入指令cd /opt/client/HDFS/hadoop进入Hadoop    **图1.1 进入hadoop**   1. **添加环境变量(15’)**输入指令export HADOOP="/opt/client/HDFS/hadoop/share/hadoop"   以及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"  来添加环境变量    **图1.2 添加环境变量**   1. **创建Java程序并输入代码**   首先利用vim指令创建Java程序    **图1.3 创建Java程序文件**  并在程序中输入所给代码    **图1.4 输入所给代码**   1. **编译刚刚的Java程序**   利用javac指令编译Java程序    **图1.5 编译Java程序**   1. **创建文件并放入input文件夹中**   利用vim指令c创建文件并输入数据    **图1.6 创建文件**    **图1.7 文件内容**    **图1.8 文件内容**    **图1.9 文件转存**   1. **运行 WordCount.jar，显示词频统计结果**   调用指令  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-00000来显示统计结果    **图1.10 调用指令统计词频**    **图1.11 显示词频统计结果**   * + - 1. **Spark**  1. **打开spark**   调用pyspark指令打开spark    **图2.1 进入spark**   1. **读取hdfs文件内容**   调用lines = spark.read.text("hdfs:///input").rdd.map(lambda r: r[0])指令读取文件内容，结果如下。    **图2.2 读取hdfs文件内容**   1. **词频统计**   调用指令counts = lines.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y) output = counts.collect()对词频进行统计    **图2.3 操作结果**   1. **结果输出**   调用output指令对词频统计结果进行输出    **图2.4 输出统计结果**   * + - 1. **附加题**  1. **文件的保存**   利用vim指令，新建文件并将其存入hdfs   1. **编写mapreduce程序输出平均成绩** 2. **编写mapreduce程序输出每位同学有几门课低于75分**   在这个人物中，我们仅需要对TokenizerMapper类进行重写，代码如下：  package readuse;  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 sub\_1 = new Text();          private Text sub\_2 = new Text();          private Text sub\_3 = new Text();          public void map(Object key, Text value, Context context) throws IOException, InterruptedException {                  int avg = 0;                  int sum = 0;                  StringTokenizer itr = new StringTokenizer(value.toString());                  while (itr.hasMoreTokens()) {                      sub\_1.set(itr.nextToken());                      sub\_2.set(itr.nextToken());                      sub\_3.set(itr.nextToken());                      int tmp = Integer.parseInt(sub\_3);                  //     int tmp = (sub\_3.charAt(0) - '0') \* 10 + sub\_3.charAt(1) - '0';                      avg += tmp;                      sum++;                      one.set(1);                      if(tmp < 75) context.write(sub\_1, one);                  }                  if(sum==3) {                      one.set(avg / 3);                      context.write(sub\_2, one);                      sum=0;                      avg=0;                  }          }  }          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);      }  }  代码解析：  由于我们在每行中要读取三个成绩数据，分别是学号，科目和成绩，所以我们可以直接构造三个字符串对象对每行中的数据进行读取:sub\_1存储学号，sub\_2存储科目，sub\_3存储成绩，我们仅需对成绩进行判断后对相应数据进行操作。  由于我们利用nextToken读入的是字符串格式，我们要将最后一个数据sub\_3转化为int形存储在tmp中，对tmp进行判定才能进行下一步操作。  因为我们要计算平均分以及统计每位同学成绩低于75分的科目数。  平均分比较简单，我们仅需要每次读取数据后将转化得到的tmp加到avg中即可，最后再进行一步除法运算得出平均分，将所有的数据进行加和之后，我们将avg除以3再将其利用write指令向context中写入sub\_2对应的科目以及平均分，即可记录每个科目的平均分。  对75分以下科目的统计，我们可以每次读取数据后对tmp判断，若tmp低于75，我们则利用write指令向context中写入sub\_1对应的学号以及1，代表这个sub\_1学生有一科低于75分，这样再读取全部数据后我们便可以知道每个学生有几个科目低于75分。  由于我们每个文件中存储了三个成绩数据，我们还需要设定一个sum标记来存储当前读取到第几条数据，若sum等于3，相当于全部的数据已经被读取，这时我们再进行求平均数计算。  若程序运行正常，我们会在context中得到各个学号和低于75分的标记，每有一科低于75，就会在context中出现一次学号+1的内容，以及每一个科目的平均分，经过reduce的操作，我们就可以得到所有学生低于75分的科目的数量以及每个科目的平均分。  测试结果：    **图3.1 测试结果**  **五、出现的问题与解决方案**  在编写附加题代码时，我遇到了一些问题，由于我仅修改了TokenizerMapper类，在map方法中实现了计算低于75分和计算平均数两个功能，如果不加判断的话每次都出成绩都会进行计算平均数的操作，导致context表中有多组课程-分数的数据，导致输出错误，所以我在map方法中添加了读入次数判定，只有在读入数据达到一定次数后才会将总成绩进行除法运算得到平均值并写入context中。  但这也并不是最终解决方案，在读取每个file时，我们并不知道其中有多少成绩数据，甚至每个file中的成绩数量都不尽相同，所以这样的修改也是仅仅可以通过测试集的样例，并不能适应所有情况，不能简单地对程序做读入次数判定后便输出了事，程序还需要做进一步修改。  **六、实验总结**  本次实验中，我们首次接触到了mapreduce中的程序，也可以将我们之前学过的Java知识运用到实验中来，修改mapreduce处理程序的过程也十分有趣，让我了解到了许多新指令，更深层次地理解了mapreduce底层思想并在编程中加以运用，让我感觉收获颇丰且蛮有乐趣。 | | | | | | | |