

華中科技大學

大数据处理实验报告

实验三：MapReduce 的基本操作

专业班级： CS2005 班

学 号： U202090063

姓 名： 董玲晶

指导教师： 石宣化

报告日期： 2022.04.06

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	石宣化
教师评价	<div><input type="checkbox"/> 实验过程正确；<input type="checkbox"/> 源程序/实验内容提交；<input type="checkbox"/> 程序结构/实验步骤合理； <input type="checkbox"/> 实验结果正确；<input type="checkbox"/> 语法、语义/命令正确；<input type="checkbox"/> 报告规范； 其他：</div>				
<div><h3>一、实验目的</h3><div><div>1. 了解 MapReduce 的用途</div><div>2. 掌握 MapReduce 的基本命令</div></div><h3>二、实验内容</h3><div><div>1. 实验环境配置</div><div>2. MapReduce (55')</div><div>3. Spark (35')</div><div>4. 附加题 (学有余力可以课下自行尝试, 不算分)</div><div>5. 实验总结 (10')</div></div><h3>三、实验环境</h3><div><div>1. 软件：系统搭载 Spark2.2.2, Hadoop2.8.3, HBase1.3.1, Hive2.3.3, Tez0.9.1, 使用弹性公网 IP 访问 MRS。</div><div>2. 硬件：使用 MRS1.9.2 分析集群。Master 节点和分析 Core 节点均搭载 4 个 Cortex 虚拟 CPU, 16G 内存, 以及高 IO/100G 数据盘和系统盘。</div></div></div>					

四、实验过程或步骤（源程序）

3.1 MapReduce

3.1.1 进入 Hadoop，如图 3.1.a 所示。

```
# cd /opt/client/HDFS/Hadoop
```

```
Last login: Thu Jan  1 08:00:10 1970  
[root@node-master1j0qw ~]# cd /opt/client/HDFS/hadoop
```

图 3.1.a

3.1.2 添加环境变量，如图 3.1.b 所示。

```
# 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:$HADOOP/common/lib/commons-cli-1.2.jar:$CLASSPATH"
```

```
[root@node-master1j0qw hadoop]# export HADOOP="/opt/client/HDFS/hadoop/share  
/hadoop"  
[root@node-master1j0qw hadoop]# export CLASSPATH="$HADOOP/common/hadoop-comm  
on-2.8.3-mrs-1.9.0.jar:$HADOOP/mapreduce/hadoop-mapreduce-client-core-2.8.3-  
mrs-1.9.0.jar:$HADOOP/common/lib/commons-cli-1.2.jar:$CLASSPATH"  
[root@node-master1j0qw hadoop]#
```

图 3.1.b

3.1.3 用 vim 命令创建一个 java 程序 WordCount.java, 在里面输入如图 3.1.c 所示代码，命令如图 3.1.d 所示。

```
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();
    }
}
```

图 3.1. c-(1)

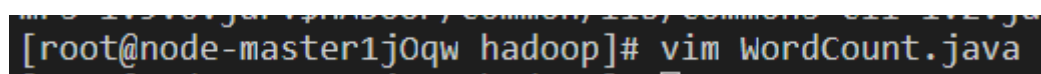
```

        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);
    }
}

```

图 3.1. c-(2)



```

[root@node-master1j0qw hadoop]# vim WordCount.java

```

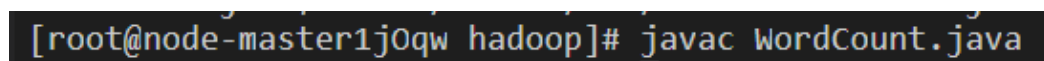
图 3.1. d

3.1.4 编译 WordCount.java 文件，如图 3.1. e 和 3.1. f 和 3.1. g 所示.

```

# javac WordCount.java
# jcr cvf WordCount.jar WordCount.class added manifest
# jar cvf WordCount.jar WordCount*.class added manifest

```



```

[root@node-master1j0qw hadoop]# javac WordCount.java

```

图 3.1. e

```
[root@node-master1j0qw hadoop]# javac WordCount.java
[root@node-master1j0qw hadoop]# jar cvf WordCount.jar WordCount.class
added manifest
adding: WordCount.class(in = 1487) (out= 810)(deflated 45%)
[root@node-master1j0qw hadoop]#
```

图 3.1. f

```
[root@node-master1j0qw hadoop]# jar cvf WordCount.jar WordCount*.class
added manifest
adding: WordCount$IntSumReducer.class(in = 1731) (out= 734)(deflated 57%)
adding: WordCount$TokenizerMapper.class(in = 1732) (out= 752)(deflated 56%)
adding: WordCount.class(in = 1487) (out= 810)(deflated 45%)
[root@node-master1j0qw hadoop]#
```

图 3.1. g

3.1.5 用 vim 命令创建文件 test1，内容为 hello hust 文件 test2，内容为 hello U202090063（本人学号），将他们放入 hdfs 的 /input 文件夹下。最后用 ls 命令显示 /input 文件夹下的文件详情，可发现已经放入成功。以上操作如图 3.1. h 所示。

```
# vim test1.txt
# vim test2.txt
# hdfs dfs -mkdir /input
# hdfs dfs -put test1.txt /input
# hdfs dfs -put test2.txt /input
# hdfs dfs -ls /input
```

```
[root@node-master1j0qw hadoop]# vim test1.txt
[root@node-master1j0qw hadoop]# vim test2.txt
[root@node-master1j0qw hadoop]# hdfs dfs -mkdir /input
[root@node-master1j0qw hadoop]# hdfs dfs -put test1.txt /input
[root@node-master1j0qw hadoop]# hdfs dfs -put test2.txt /input
[root@node-master1j0qw hadoop]# hdfs dfs -ls /input
Found 2 items
-rw-r--r--  1 root ficommon      11 2022-03-31 19:59 /input/test1.txt
-rw-r--r--  1 root ficommon     17 2022-03-31 20:00 /input/test2.txt
```

图 3.1. h

3.1.6 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。如图 3.1. i 和 3.1. j 所示。

由图可发现 hello 出现了两次，hust 出现了一次，U202090063 出现了一次，与写入内容相符，统计正确。

```
# hadoop jar WordCount.jar WordCount hdfs:///input hdfs:///output
```

```
# hdfs dfs -cat /output/part-r-00000
```

```
[root@node-master1j0qw hadoop]# export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:/opt/client/HDFS/hadoop/WordCount.jar
[root@node-master1j0qw hadoop]# hadoop jar WordCount.jar WordCount hdfs:///input hdfs:///output
```

图 3.1.i

```
[root@node-master1j0qw hadoop]# hdfs dfs -cat /output/part-r-00000
U202090063      1
hello      2
hust      1
[root@node-master1j0qw hadoop]#
```

图 3.1.j

3.2 Spark

3.2.1 打开 Pyspark, 如图 3.2.a 所示。

```
# Pyspark
```

```
Welcome to                               .---.  
                                     /---\   \_.-.  
      /\_/\_/_\_./_\_/_\_/_\_/_\_./_\_/_\_-/_\_\_/. version 2.2.2-mrs-1.9.0  
     _/_/_/_/_.\_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/  
  
Using Python version 2.7.15 (default, Apr 1 2019 00:00:00)  
SparkSession available as 'spark'.  
>>> █
```

图 3.2. a

3.2.2 读取刚刚放入 hdfs /input 文件夹下的文件内容，如图 3.2.b 所示。

```
# lines = spark.read.text("hdfs:///input").rdd.map(lambda r: r[0])
```

```
>>> lines = spark.read.text("hdfs:///input").rdd.map(lambda r: r[0])
```

图 3.2. b

3.2.3 对文件里的内容做词频统计，如图 3.2.c 所示。

从 print 输出的内容可发现 hello 出现了两次, hust 出现了一次, U202090063 出现了一次, 与写入内容相符, 统计正确。

```
# counts = lines.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)
```

```
# output = counts.collect()
```

3.2.4 打印输出统计结果, 如图 3.2.d 所示。

```
# print(output)
```

```
>>> counts = lines.flatMap(lambda x: x.split(' ')).map(lambda x: (x, ...  
... 1)).reduceByKey(lambda x, y: x + y)  
>>> output = counts.collect()
```

图 3.2.c

```
>>> print(output)  
[(u'hust', 1), (u'hello', 2), (u'U202090063', 1)]
```

图 3.2.d

3.3 附加题

3.3.1 用 vim 指令创建两个文件 file1 和 file2, 文件内容如 3.3.a 所示, 指令操作如图 3.3.b 所示。

```
# vim file1.txt
```

```
# vim file2.txt
```

file1:

20210001 Math 90

20210002 Math 80

20210003 Math 70

file2:

20210001 English 80

20210002 English 70

20210003 English 60

图 3.3.a


```
[root@node-master1j0qw ~]# vim file1.txt
[root@node-master1j0qw ~]# vim file2.txt
```

图 3.3. b

3.3.2 在 hdfs 中创建一个文件夹并命名为 addition，创建完毕后将上述两个文件放入/addition 文件夹内，用 ls 命令查看/addition 文件夹下的文件详情，如图 3.3. c 所示。

```
# hdfs dfs -mkdir /addition
# hdfs dfs -put file1.txt /addition
# hdfs dfs -put file2.txt /addition
# hdfs dfs -ls /addition
```

```
[root@node-master1j0qw ~]# hdfs dfs -mkdir /addition
[root@node-master1j0qw ~]# hdfs dfs -put file1.txt /addition
[root@node-master1j0qw ~]# hdfs dfs -put file2.txt /addition
[root@node-master1j0qw ~]# hdfs dfs -ls /addition
Found 2 items
-rw-r--r--  1 root ficommon      51 2022-03-31 20:20 /addition/file1.txt
-rw-r--r--  1 root ficommon      60 2022-03-31 20:20 /addition/file2.txt
```

图 3.3. c

3.3.3 用 map 的思想读取刚刚放入 hdfs /addition 文件夹下的文件内容，用 split 函数按空格切分读取的每一行获得学生成绩的科目信息和分数信息，同时每条数据计 1；再用 reduceByKey 函数将学生的成绩相加，计数标志 1 相加就得到学生的总人数。最后打印输出结果。Mapreduce 语句如图 3.2. d 所示，结果如图 3.3. e 所示。

由结果可知，数学科目的平均分是 80，英语科目的平均分是 70。

```
# score = line.map(lambda x:(x.split(' ')[1],
(int(x.split(' ')[2], 1))).reduceByKey(lambda (x, y), (m, n):(x+m,
y+n))

# output = [(x, y/z) for (x, (y, z)) in score.collect()]

# print(output)

>>> score = lines.map(lambda x: (x.split(' ')[1], (int(x.split(' ')[2]), 1))).reduceByKey(lambda (x,y),(m,n):(x+m, y+n))
>>> output = [(x, y/z) for (x,(y,z)) in score.collect()]
```

图 3.3. d

```
>>> print(output)
[(u'Math', 80), (u'English', 70)]
>>> □
```

图 3.3. e

3.3.4 用 map 的思想读取刚刚放入 hdfs /addition 文件夹下的文件内容，用 split 函数按空格切分读取的每一行获得学生成绩的科目名称和分数，如果该成

绩大等于 75 计数标志设置为 0 否则设置为 1；再用 reduceByKey 函数将计数标志相加；最后打印输出结果。Mapreduce 语句如图 3.2.f 所示，结果如 3.3.g 所示。

由结果可知学号为 20210001 的学生有 0 门低于 75 分，学号为 20210002 的学生有 1 门低于 75 分，学号为 20210003 的学生有两门低于 75 分。

```
# counts = lines.map(lambda x: (x.split(' ')[0], x if int(x.split(' ')[2])>=75 else 1)).reduceByKey(lambda x, y: x+y)

# output = counts.collect()

# print(output)

((u'20210001', 0), (u'20210003', 2), (u'20210002', 1))
>>> counts = lines.map(lambda x: (x.split(' ')[0], 0 if int(x.split(' ')[2])>=75 else 1)).reduceByKey(lambda x, y: x + y)
>>> output = counts.collect()
[]
```

图 3.3.f

```
>>> print(output)
[(u'20210001', 0), (u'20210003', 2), (u'20210002', 1)]
>>> []
```

图 3.3.g

五、出现的问题与解决方案

1. WordCount.java 文件一直编译不成功。一开始是显示有不正确的缩进，从 pdf 文件复制的时候会打乱缩进和代码格式。花了很长的时间调整了缩进之后还是编译不成功，看了很久之后怀疑是环境出了问题。最后发现是 export 出问题了，想到了之前虚拟机的操作，于是手动改了 bash_profile 之后再重新 source 了一遍，编译成功。

2. 附加题的内容刚拿到的时候感觉还是比较有挑战性的。首先因为受到前面实验内容的影响，我的第一反应是用 java 语言来解决问题，问题是我没有学过 java，前面编译 WordCount.java 文件用的指令也是现场上网搜索的，虽然说各种语言大差不差，但还是有一些不太一样，对我来说有一些困难（平常用 python 比较多）。后来我想到 spark 有 python 的 api，而且前面的有一部分内容也是用 python 语言完成的，所以我想到了其实计算的实现可以不拘泥于 java 语言，或许用 python 语言也可以很好地实现，甚至有可能还更加的精简。

3. 当下定决心要用 python 开始写之后我在如何实现的方面碰到了难题，有点不知所措，于是我回头看前面的例子，发现所给代码的核心其实就在于 map 和 reduce 思想的运用（抱歉前面光顾着一股脑做实验也没认真看实验的内容和标题，这里自我反省一下）。由于以前看过一些有关数据处理的资料，我想到了 python 的 map 和 ReduceByKey 函数。

六、实验总结

本次实验做完之后总体自我感觉难度不大，大部分内容都给了保姆级别的教程（谢谢助教）。尽管难度不大，但我认为收获颇多。

首先，我深入理解了 MapReduce 的核心思想，初步掌握了 MapReduce 的基本命令与操作。在这里对助教和老师要求我们做附加题的行为表示肯定，其实附加题前面的内容确实是保姆级教程，所以自己在做的时候就只是照猫画虎、走马观花，没有真正地去理解和感悟实验本身的内容，做附加题之后因为要自己自己去写代码才真正地去学习和思考。

其次，印象比较深的第一个内容是编译 WordCount.java 文件，确实也是真的 debug 了好久，没有想到之前觉得没什么用处以后再也不会用到的虚拟机的操作知识竟然派上了用场，深刻体会到了永远不要忽视小细节，还有就是应该多尝试各种事情，不要总觉得用不到就不去做和尝试。而第二个内容就是最后的附加题，最后写出来还是蛮有成就感的！