

# 分布式计算实验报告

## 一、实验目的

学习基于 MapReduce 框架的分布式计算程序设计方法。

## 二、实验题目

### 题目 1

输入文件为学生成绩信息，包含了必修课与选修课成绩，格式如下：

班级 1, 姓名 1, 科目 1, 必修, 成绩 1 <br> （注：<br> 为换行符）

班级 2, 姓名 2, 科目 1, 必修, 成绩 2 <br>

班级 1, 姓名 1, 科目 2, 选修, 成绩 3 <br>

....., ....., ....., ..... <br>

编写两个 Hadoop 平台上的 MapReduce 程序，分别实现如下功能：

1. 计算每个学生必修课的平均成绩。
2. 按科目统计每个班的平均成绩。

### 题目 2

输入文件的每一行为具有父子/父女/母子/母女/关系的一对人名，例如：

Tim, Andy <br>

Harry, Alice <br>

Mark, Louis <br>

Andy, Joseph <br>

....., ..... <br>

假定不会出现重名现象。

编写 Hadoop 平台上的 MapReduce 程序，找出所有具有 grandchild-grandparent 关系的人名组。

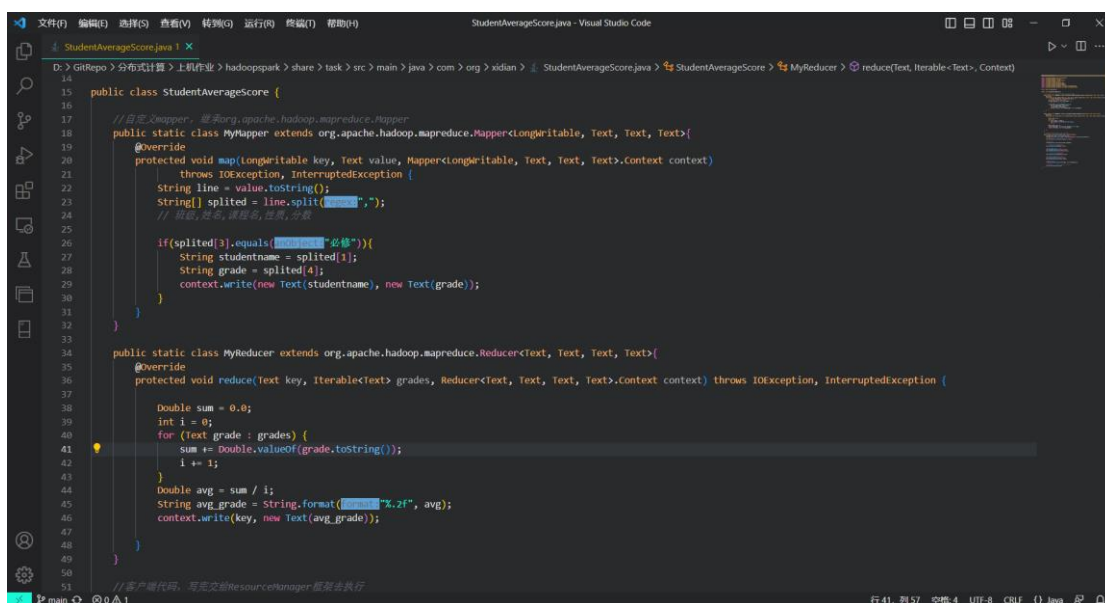
## 三、实验思路

题目 1:

要求 1:

在 Map 阶段，我们读入文件后，将每一行的内容按”，”分隔后写入字符串数组中，以姓名为键，以成绩为值。在筛选过程中我们加入筛选条件，只录入”必修”的科目。

在 Reduce 阶段，我们对每个键对应的值循环求和，同时计算值的个数，不难得出其成绩的平均值。

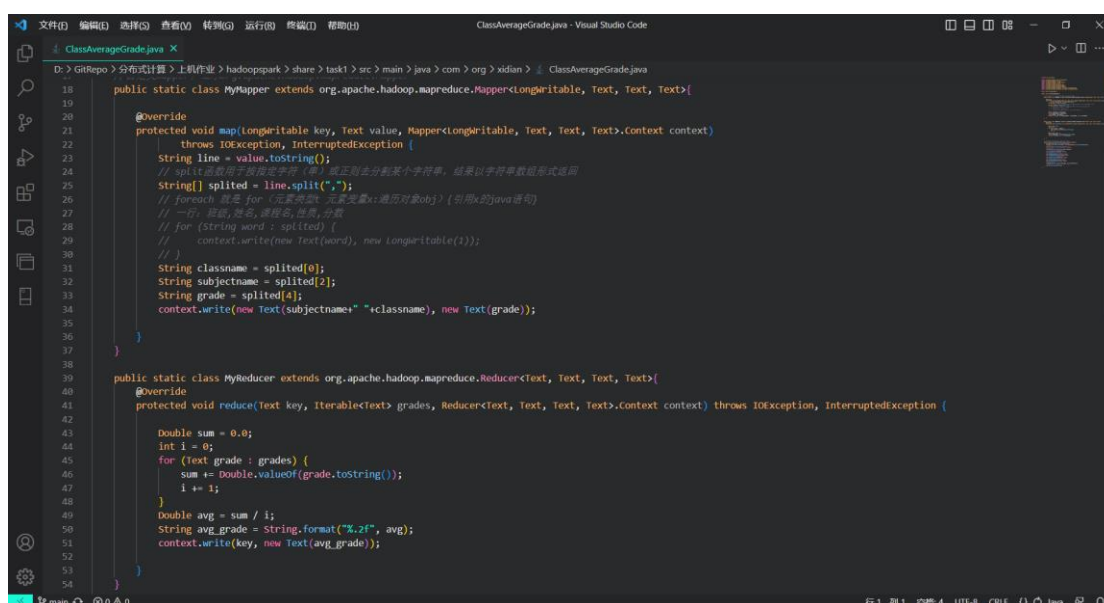


```
14
15 public class StudentAverageScore {
16
17     //自定义Mapper，继承org.apache.hadoop.mapreduce.Mapper
18     public static class MyMapper extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text, Text, Text>{
19         @Override
20         protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, Text>.Context context)
21             throws IOException, InterruptedException {
22             String line = value.toString();
23             String[] splitted = line.split(",");
24             // 姓名, 姓名, 课程名, 成绩, 分数
25
26             if(splitted[3].equals("必修")){
27                 String studentname = splitted[1];
28                 String grade = splitted[4];
29                 context.write(new Text(studentname), new Text(grade));
30             }
31         }
32     }
33
34     public static class MyReducer extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text>{
35         @Override
36         protected void reduce(Text key, Iterable<Text> grades, Reducer<Text, Text, Text, Text>.Context context) throws IOException, InterruptedException {
37
38             Double sum = 0.0;
39             int i = 0;
40             for (Text grade : grades) {
41                 sum += Double.valueOf(grade.toString());
42                 i++;
43             }
44             Double avg = sum / i;
45             String avg_grade = String.format("%.2f", avg);
46             context.write(key, new Text(avg_grade));
47         }
48     }
49
50     // 基于源代码，通过资源管理器部署并执行
51 }
```

要求 2:

在 Map 阶段，我们读入文件后，将每一行的内容按”，”分隔后写入字符串数组中，与要求 1 的步骤大致相同，但是这一次我们以科目+班级这个整体作为键，以成绩为值。

在 Reduce 阶段，我们对每个键对应的值循环求和，同时计算值的个数，最终得到以班级和科目为统计标准的平均值。

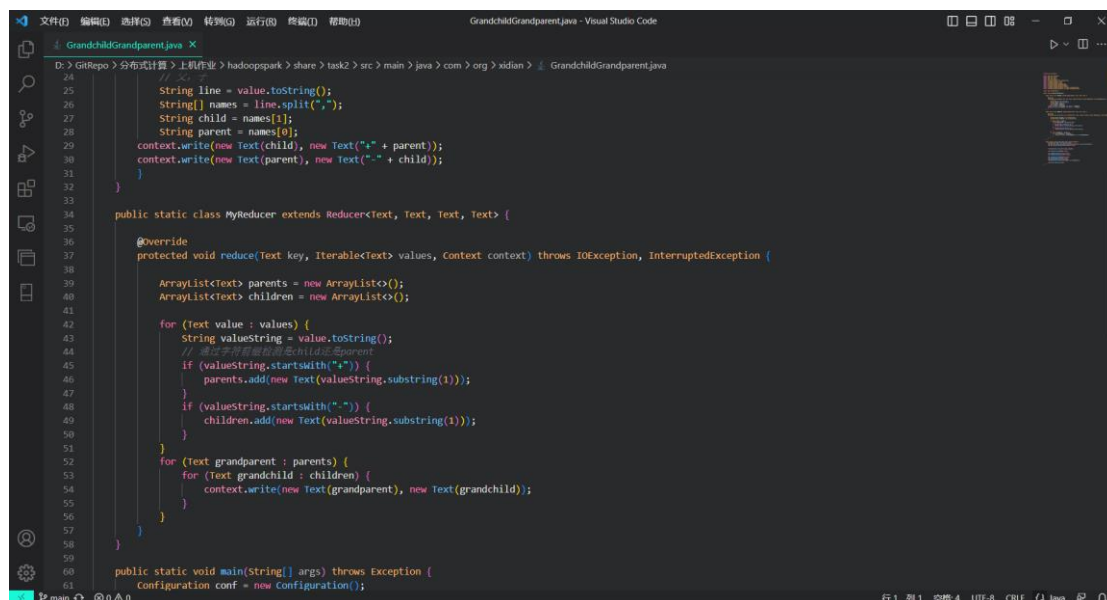


```
18
19 public class ClassAverageGrade {
20
21     //自定义Mapper，继承org.apache.hadoop.mapreduce.Mapper
22     public static class MyMapper extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text, Text, Text>{
23         @Override
24         protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, Text>.Context context)
25             throws IOException, InterruptedException {
26             String line = value.toString();
27             // split 函数用于按指定字符（单）或正则表达式分割某个字符串，结果以字符串数组形式返回
28             String[] splitted = line.split(",");
29             // foreach 就是 for（元素类型：元素变量xx遍历对象obj）（引用x的java语句）
30             // 一行，姓名, 姓名, 课程名, 成绩, 分数
31             // for (String word : splitted) {
32                 context.write(new Text(word), new LongWritable(1));
33             }
34
35             String classname = splitted[0];
36             String subjectname = splitted[2];
37             String grade = splitted[4];
38             context.write(new Text(subjectname+" "+classname), new Text(grade));
39         }
40     }
41
42     public static class MyReducer extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text>{
43         @Override
44         protected void reduce(Text key, Iterable<Text> grades, Reducer<Text, Text, Text, Text>.Context context) throws IOException, InterruptedException {
45
46             Double sum = 0.0;
47             int i = 0;
48             for (Text grade : grades) {
49                 sum += Double.valueOf(grade.toString());
50                 i++;
51             }
52             Double avg = sum / i;
53             String avg_grade = String.format("%.2f", avg);
54             context.write(key, new Text(avg_grade));
55         }
56     }
57 }
```

## 题目 2:

在 Map 阶段，我们对其进行预处理。对 txt 文档的每一行，先用 split 函数将其用“,”分隔成若干字符串，并存于数组 splited 中。splited[0]为父母姓名，splited[1]为子女姓名。为了区分父母与子女，我为父母的姓名前加了“+”，为子女的名字前加了“-”作为区分的方式。将父母与子女的信息全部录入。

在 Reduce 阶段，循环遍历每一个键对应的所有值，如果是以“+”开头的那么我们将其加入到创建的 parents 列表中，如果是以“-”开头的，那么我们将其加入到所创建的 children 列表中。注意写入列表前需要将自行添加的标记符号去掉，因此使用了 subString() 方法。最终循环输出祖孙对即可。



```
24 // +
25 String line = value.toString();
26 String[] names = line.split(",");
27 String child = names[1];
28 String parent = names[0];
29 context.write(new Text(child), new Text("+ " + parent));
30 context.write(new Text(parent), new Text("- " + child));
31 }
32 }
33
34 public static class MyReducer extends Reducer<Text, Text, Text, Text> {
35
36     @Override
37     protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
38
39         ArrayList<Text> parents = new ArrayList<>();
40         ArrayList<Text> children = new ArrayList<>();
41
42         for (Text value : values) {
43             String valueString = value.toString();
44             // 去除 + 和 - 标记
45             if (valueString.startsWith("+")) {
46                 parents.add(new Text(valueString.substring(1)));
47             }
48             if (valueString.startsWith("-")) {
49                 children.add(new Text(valueString.substring(1)));
50             }
51         }
52         for (Text grandparent : parents) {
53             for (Text grandchild : children) {
54                 context.write(new Text(grandparent), new Text(grandchild));
55             }
56         }
57     }
58 }
59
60 public static void main(String[] args) throws Exception {
61     Configuration conf = new Configuration();
```

## 四、实验环境的搭建

1. 由于我使用的是 Ubuntu 系统，因此选择使用 Linux 的环境配置法。在 Linux 下安装 Docker 引擎，Docker 安装完毕后在设置界面配置国内镜像仓库地址。
2. 导入包含实验环境的 Docker 镜像：(1) 利用 docker load --input hadoopsparkv2.tar 将压缩包 hadoopsparkv2.tar 中的 Dockers 镜像 ubuntu-jdk8-hadoop-spark:v2 镜像导入的本机的 Docker 引擎中。(2) 将 hadoopspark.zip 中的内容解压到 xxxx/hadoopspark 目录下，然后在命令

终端模式下将当前目录切换到 xxxx/hadoopspark。

3. 启动实验环境：首先开启 Docker Desktop 。将当前目录切换到 xxxx/hadoopspark ，然后运行命令 `docker-compose up -d` 和 `docker ps`，可以看到基于 `ubuntu-jdk8-hadoop-spark:v2` 镜像的 Docker 容器（虚拟机）在运行。

4. 用 `ssh -p 2222 root@localhost` 命令通过 ssh 协议从本机（宿主机）远程登录到 `hadoopspark_singlenode` 虚拟机内部，密码为 123456。

5. 用 `start-dfs.sh` 命令启动 HDFS 分布式文件系统。

6. 若要关闭实验环境，先确保当前虚拟机中所有 job 均已完成，然后在命令行中输入 `logout` 登出虚拟机。最后在 `xxxx/hadoopspark` 目录下使用 `docker-compose down` 关闭实验环境。

## 五、实验程序的执行方法

1. 在本地使用 `mvn clean + mvn package` 命令将程序打成 jar 包。

2. 在平台使用指令运行程序。

## 六、心得体会

通过实验，我深刻理解了 HDFS 和 MapReduce 的工作原理，学会了如何搭建和配置分布式环境，并编写和调试 MapReduce 作业。这种实际操作的经验对于理解分布式系统的设计和优化非常有帮助。在实际的操作中，我也遇到了一些大大小小的挫折，比如在 Linux 上配置环境的时候，我多次被镜像源困扰，我更换了多次镜像后仍然无法安装 Docker 环境，我怀疑我可能是缺少了某种依赖，但是这个依赖又无法正常的安装，为此我纠结了好久，不过最后，我通过强制卸载依赖再重新安装的方式解决了这个问题，最终耗费了接近一天的时间来解决这个报错，因此这次分布式实验还是很令人难忘的。