**实验报告（第6次）**

实验名称 MapReduce的编程实践2 实验时间 第 周第 次课

同组同学 小组分工

**一、实验目的**

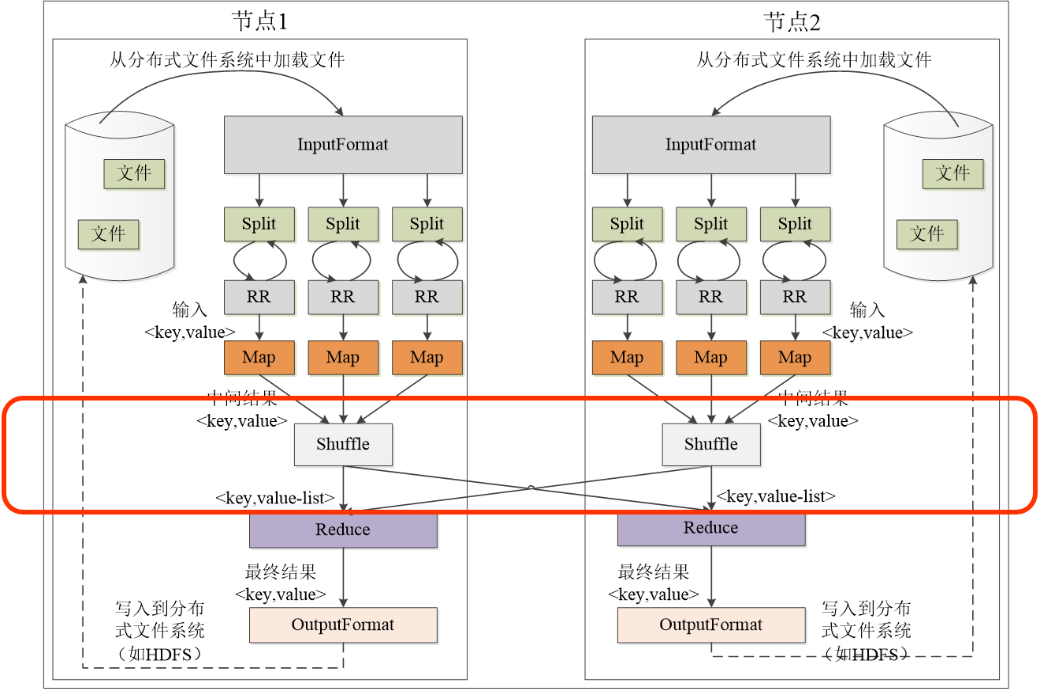
1. 通过实验掌握基本的MapReduce编程方法；
2. 掌握用MapReduce解决一些常见的数据处理问题，包括数据去重、数据排序和数据挖掘等。

**二、实验仪器设备或材料**

1. Ubuntu 22.04.3
2. Hadoop 3.1.3
3. HBase 2.2.2
4. JDK 1.8.0
5. Eclipse

**三、实验原理**

Map函数+Reduce函数+Shuffle过程



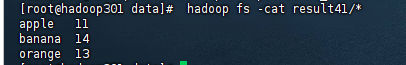
**四、实验内容与步骤**

1.Java API编程实现商品销售总量计算

**题目**：统计每种商品的总销售数量。

数据文件：sales\_data.txt

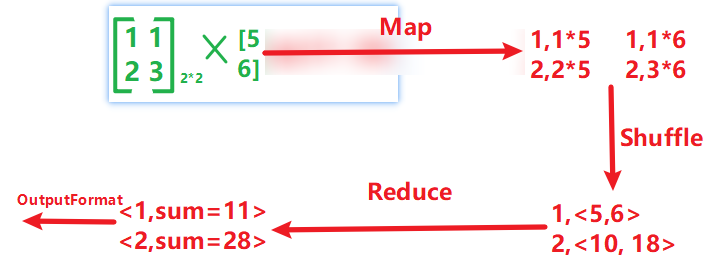
结果：



代码：

import java.io.IOException;  
import java.util.Iterator;  
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;  
import org.apache.hadoop.util.GenericOptionsParser;  
public class fuirt\_sale {  
  
 public static void main(String[] args) throws Exception {  
 Configuration conf = new Configuration();  
 Job job = Job.*getInstance*(conf, "fuirt\_sale");  
 job.setJarByClass(fuirt\_sale.class);  
 job.setMapperClass(fuirt\_sale.TokenizerMapper.class);  
 job.setReducerClass(fuirt\_sale.IntSumReducer.class);  
  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(IntWritable.class);  
 FileInputFormat.*addInputPath*(job, new Path("hdfs://192.168.133.111:9000/user/root/sales\_data.txt"));  
 FileOutputFormat.*setOutputPath*(job, new Path("hdfs://192.168.133.111:9000/user/root/result7"));  
 System.*exit*(job.waitForCompletion(true)?0:1);  
 }  
 public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {  
 private final static IntWritable *quantity* = new IntWritable();  
 private Text fruit = new Text();  
 public TokenizerMapper() {  
 }  
 public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context) throws IOException, InterruptedException {  
 String[] tokens = value.toString().split(" ");  
 需补充  
 context.write(需补充);  
 }  
 }  
  
 public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {  
 private IntWritable result = new IntWritable();  
 public IntSumReducer() {  
 }  
 public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {  
 int sum = 0;  
 IntWritable val;  
 for(Iterator i$ = values.iterator(); i$.hasNext(); sum += val.get()) {  
 val = (IntWritable)i$.next();  
 }  
 this.result.set(sum);  
 context.write(key, this.result);  
 }  
 }  
}

2. Java API编程实现矩阵-向量乘法



数据文件：Matrix.txt

结果：



代码：

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;  
  
  
import java.io.IOException;  
  
public class Matrix\_vector {  
  
 public static class MatrixVectorMapper extends Mapper<Object, Text, IntWritable, IntWritable> {  
 private IntWritable rowKey = new IntWritable();  
 private IntWritable productValue = new IntWritable();  
 private int[] vector = new int[10]; // 假设最大向量长度为10  
  
 @Override  
 protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {  
 String[] tokens = value.toString().split(",");  
  
 // 处理向量元素  
 if (tokens[0].equals("B")) {  
 int index = Integer.*parseInt*(tokens[1]);  
 int vectorValue = Integer.*parseInt*(tokens[2]);  
 vector[index] = vectorValue; // 存储向量值  
 System.*out*.println("Vector index: " + index + ", Value: " + vectorValue);  
 }  
 // 处理矩阵元素  
 else if (tokens[0].equals("A")) {  
 int row = Integer.*parseInt*(tokens[1]);  
 int col = Integer.*parseInt*(tokens[2]);  
 int matrixValue = Integer.*parseInt*(tokens[3]);  
  
 // 计算乘积  
 int vectorValue = vector[需补充]; // 获取对应的向量元素  
 int product = 需补充; // 计算乘积  
 rowKey.set(需补充);  
 productValue.set(需补充);  
  
 // 输出 row 和 product  
 context.write(rowKey, productValue);  
  
 // 调试输出  
 System.*out*.println("Row: " + row + ", Matrix Value: " + matrixValue + ", Vector Value: " + vectorValue + ", Product: " + product);  
 }  
 }  
 }  
  
 public static class MatrixVectorReducer extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {  
 @Override  
 protected void reduce(IntWritable key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {  
 int sum = 0;  
 for (IntWritable val : values) {  
 sum += val.get();  
 }  
 context.write(key, new IntWritable(sum));  
 }  
 }  
  
 public static void main(String[] args) throws Exception {  
 Configuration conf = new Configuration();  
 Job job = Job.*getInstance*(conf, "Matrix\_vector");  
  
 job.setJarByClass(Matrix\_vector.class);  
 job.setMapperClass(MatrixVectorMapper.class);  
 job.setReducerClass(MatrixVectorReducer.class);  
  
 job.setOutputKeyClass(IntWritable.class);  
 job.setOutputValueClass(IntWritable.class);  
  
 FileInputFormat.*addInputPath*(job, new Path("hdfs://192.168.133.111:9000/user/root/Matrix.txt"));  
 FileOutputFormat.*setOutputPath*(job, new Path("hdfs://192.168.133.111:9000/user/root/result8"));  
 System.*exit*(job.waitForCompletion(true) ? 0 : 1);  
 }  
}

3. Java API编程实现共同好友计算

数据文件：friends.txt

数据讲解：

A:B,C,D,E,F,M

表示用户A有好友B,C,D,E,F,M

B:C,D,L,E

表示用户B有好友C,D,L,E

需要计算这些用户中，他们两两用户的共同好友是哪些。

比如A和B有哪些共同好友。

A和C有哪些共同好友。

需要两两用户都要组合遍历计算。

注意对称性，A和B组合是和B和A组合是一样的，只需要计算一次。

计算思路：

1.Map阶段

A:B,C,D,E,F,M

1.1通过字符:分割成两部分，A和B,C,D,E,F,M

1.2再次通过,字符分割B,C,D,E,F,M，获取到一个字符串数组[B,C,D,E,F,M]。

1.3遍历字符串数组，两两组合，B-C、B-D、B-E、B-F、B-M、C-D、C-E、C-F、C-M、

E-F、E-M、F-M

1.4构建K-V结构，(B-C,A)、(B-D,A)、(B-E,A)、(B-F,A)、(B-M,A)、(C-D,A)、(C-E,A)、(B-D,A)、(C-F,A)、(C-M,A)、(E-F,A)、(E-M,A)、(F-M,A)

B:C,D,L,E同样处理

1.1通过字符:分割成两部分，B和C,D,L,E

1.2再次通过,字符分割C,D,L,E，获取到一个字符串数组[C,D,L,E]。

1.3遍历字符串数组，两两组合，C-D、C-L、C-E、D-L、D-E、L-E

1.4构建K-V结构，(C-D,B)、(C-L,B)、(C-E,B)、(D-L,B)、(D-E,B)、(L-E,B)

...................................

其他数据都是这样处理

2.Reduce阶段

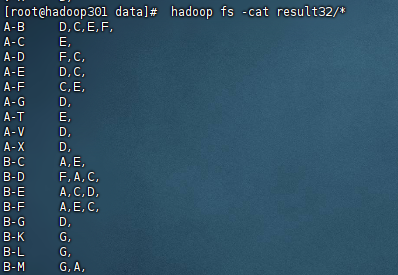
2.1相同的key进行归并

(C-D,A)和(C-D,B)的key值相同，可以归并成(C-D,<A,B>)

这样就可以计算出，用户C和D,他们共同好友是A，B。

其它数据处理逻辑一样。

结果部分截图：



代码：

import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.mapreduce.Reducer;  
import java.io.IOException;  
import java.util.Arrays;  
  
  
public class Mutual\_friend {  
 public static class Map extends Mapper<LongWritable, Text,Text, Text> {  
  
 protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  
 //拆分数据  
 String[] split = value.toString().split(":");  
 System.*out*.print(split[0]);  
 System.*out*.print(split[1]);  
 String[] split1 = split[1].split(",");  
 Arrays.*sort*(split1);//排序去重  
 for (int i = 0; i < split1.length-1; i++) {  
 //利用两个for循环对结果一一匹配  
 for (int j = i+1; j <需补充; j++) {  
 context.write(new Text(split1[i]+"-"+split1[j]),new Text(需补充));  
 }  
 }  
 }  
  
 }  
 public static class Reduce extends Reducer<Text, Text,Text,Text> {  
  
 protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {  
 StringBuffer sb=new StringBuffer();  
 for (Text te:values) {  
 sb.append(te).append(",");  
 }  
 context.write(key, new Text(sb.toString()));  
 }  
 }  
 public static void main(String[] args) throws Exception {  
 Configuration conf = new Configuration();  
 Job job = Job.*getInstance*(conf,"friends");  
 job.setJarByClass(Mutual\_friend.class);  
 //设置Mapper类  
 job.setMapperClass(Map.class);  
 //设置Reducer类  
 job.setReducerClass(Reduce.class);  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(Text.class);  
 FileInputFormat.*addInputPath*(job, new Path("hdfs://192.168.133.111:9000/user/root/friends.txt"));  
 FileOutputFormat.*setOutputPath*(job, new Path("hdfs://192.168.133.111:9000/user/root/result32"));  
 System.*exit*(job.waitForCompletion(true)?0:1);  
 }  
}

**五、实验结果与分析**

（截取主要运行结果界面即可）

**六、结论与体会**

**七、教师评语**