**作业三**

**任务：**

**1.用Mapper和Reducer连接迪拜机场免税店的销售数据和区域数据**

**2.** **将连接后的销售数据根据区域分区**

**MAP阶段代码**

public static class MultiTaskMapper extends Mapper<LongWritable, Text, Text, Text> {

private Text outKey = new Text();

private Text outValue = new Text();

private Map<String, String> airportData = new HashMap<>();

@Override

protected void setup(Context context) throws IOException, InterruptedException {

BufferedReader br = null;

try {

// 获取缓存文件

URI[] cacheFiles = context.getCacheFiles();

if (cacheFiles != null && cacheFiles.length > 0) {

String line;

Path cachePath = new Path(cacheFiles[0].toString());

// 读取缓存文件

br = new BufferedReader(new InputStreamReader(

FileSystem.get(context.getConfiguration()).open(cachePath)));

while ((line = br.readLine()) != null) {

String[] tokens = line.split(",", -1);

// 确保有足够的字段

if (tokens.length >= 4) {

String region = tokens[0].trim();

String city\_of\_airport = tokens[1].trim().toUpperCase();

String country = tokens[2].trim();

String gdp = tokens[3].trim();

// 将机场信息存入Map

String value = region + "," + country + "," + gdp;

airportData.put(city\_of\_airport, value);

}

}

}

} catch (IOException e) {

System.err.println("Exception reading cache file: " + e);

} finally {

if (br != null) {

br.close(); // 关闭BufferedReader

}

}

}

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

// 跳过表头

if (key.get() == 0 && value.toString().contains("city\_of\_airport")) {

return;

}

String line = value.toString();

String[] fields = line.split(",", -1);

// 确保有足够的字段

if (fields.length >= 42) {

String city\_of\_airport = fields[1].trim().toUpperCase();

String airportInfo = airportData.get(city\_of\_airport);

if (airportInfo != null) {

// 连接数据以供任务1使用

String joinedData = line + "," + airportInfo;

// 任务1结果

outKey.set("Task1");

outValue.set(joinedData);

context.write(outKey, outValue);

String[] airportInfoFields = airportInfo.split(",", -1);

String region = airportInfoFields[0].trim();

outKey.set(region);

outValue.set(joinedData);

context.write(outKey, outValue);

}

}

}

}

**PARTITIONER阶段代码**

public static class RegionPartitioner extends Partitioner<Text, Text> {

private static Map<String, Integer> regionMap = new HashMap<>();

static {

// 定义区域与分区的映射关系

regionMap.put("INDIA SUB-CONTINENT", 0);

regionMap.put("EUROPE", 1);

regionMap.put("AFRICA", 2);

}

@Override

public int getPartition(Text key, Text value, int numPartitions) {

String keyStr = key.toString();

// 根据区域决定分区

if (regionMap.containsKey(keyStr)) {

return regionMap.get(keyStr) % numPartitions;

} else {

return 0; // 默认分区

}

}

}

**REDUCER阶段代码**

public static class MultiTaskReducer extends Reducer<Text, Text, NullWritable, Text> {

private Text result = new Text();

private MultipleOutputs<NullWritable, Text> multipleOutputs;

@Override

protected void setup(Context context) {

// 初始化MultipleOutputs以支持多输出

multipleOutputs = new MultipleOutputs<>(context);

}

@Override

protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

String keyStr = key.toString();

// 处理任务1的结果

if ("Task1".equals(keyStr)) {

for (Text val : values) {

result.set(val.toString());

multipleOutputs.write(NullWritable.get(), result, "Task1/output");

}

} else {

// 处理任务2的结果

for (Text val : values) {

result.set(val.toString());

multipleOutputs.write(NullWritable.get(), result, "Task2/" + keyStr + "/part");

}

}

}

@Override

protected void cleanup(Context context) throws IOException, InterruptedException {

multipleOutputs.close(); // 关闭MultipleOutputs

}

}



图1 任务3-1输出结果

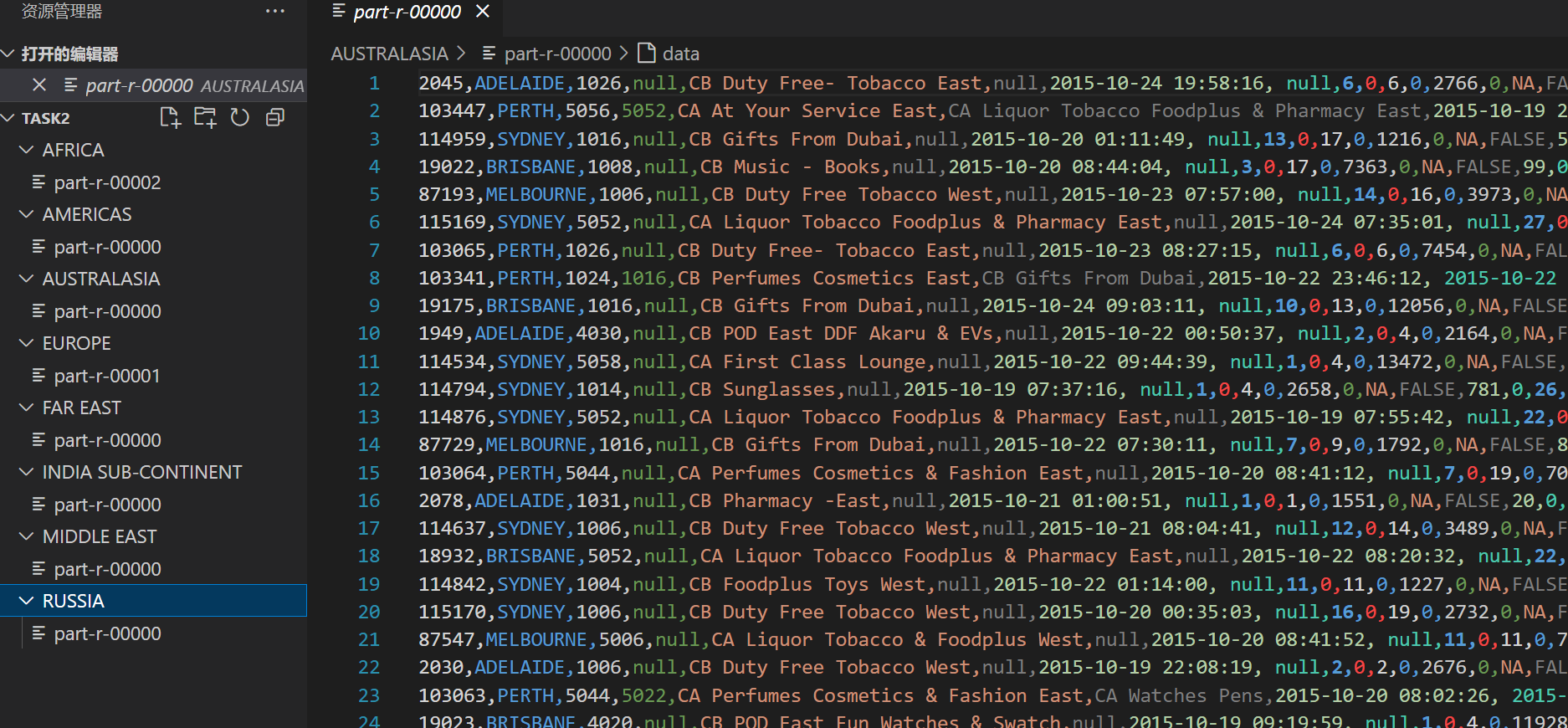


图2 任务3-2输出结果

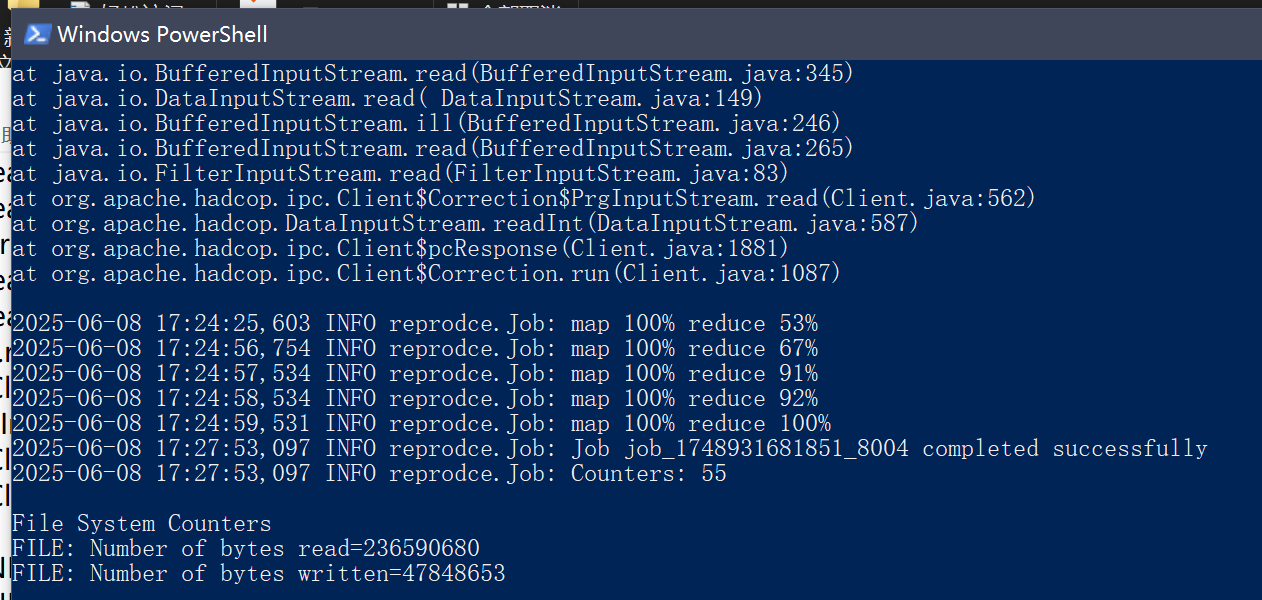


图3 MapReduce任务运行界面

总结：通过设计和实现MapReduce任务，我显著提高了数据处理能力，深入理解了组织模式和连接模式。这一过程不仅增强了我的编程技能和实践能力，还让我学会了如何有效地将复杂问题分解为可管理的子任务，优化数据流，并合理连接不同数据源的信息，从而提升了整体的工作效率和代码的可维护性。