大数据综合处理实验
实验三

组长：韩畅，组员：李展烁、王一之、闫旭芃

2020 年 5 月 6 日

1 实验规划与设计

1.1 任务分配

171860551, 韩畅：组长，项目规划，程序框架及大部分功能，主持程序运行调试、优化与提交
171860550, 王一之：代码管理，报告撰写，代码注释添加、整理
171860549, 闫旭芃：完善部分功能，添加代码注释，报告主体撰写
171840565, 李展烁：程序集群运行、调试、优化

1.2 任务要求

使用 MapReduce 完成两张表的 join 操作，存入到 Hive 中。

1.3 设计思路

实现两张表的join操作，可以利用mapreduce的特性，读入order和product将两张表到map中，将key打包成自定义的数据类型输出，并且按照两表中关联的条件排序依据，将两表满足join条件的数据并携带数据所来源的文件信息，发往同一个reduce task，在reduce中进行数据的串联

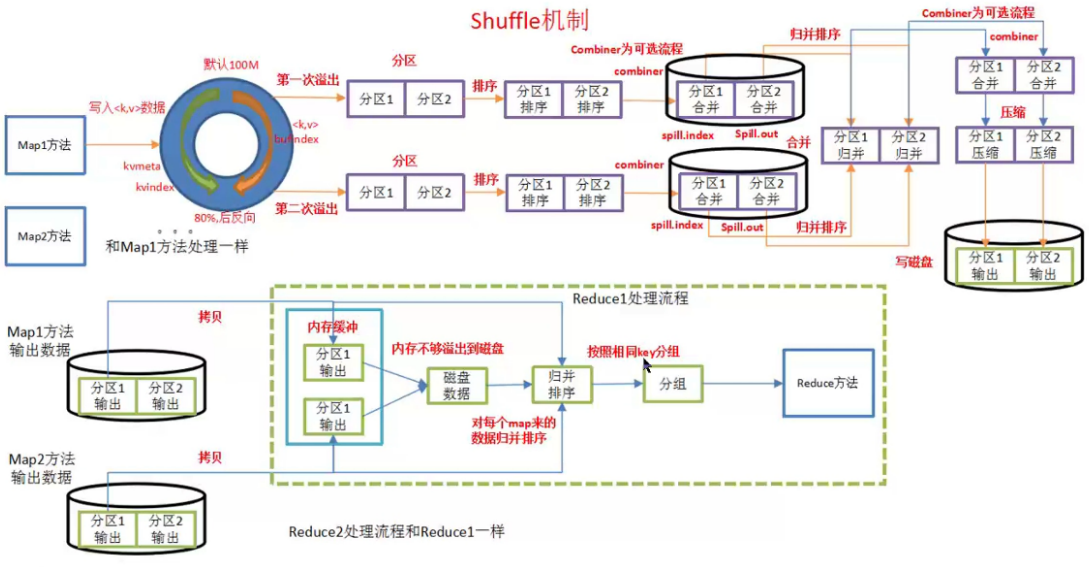

图 1. mapreduce shuffle工作流程图

1.3.1 自定义数据类型

自定义一个bean存储表中每一种数据：oid,odata,oamount,pid,pname,price

```
public class OrderBean implements WritableComparable<OrderBean> {

    private String oId;//订单id
    private String oData;//订单日期
    private String pId;//商品id
    private String pName;//商品名称
    private String price;//商品价格
    private String oAmount;//数量
```

图 2. 自定义bean

实现一些必要的函数，并且实现compareTo()，目的是为了按照pid进行排序分组，每一组然后按照pname排序,因为o表中不存在pname所以，p表中的内容就会被排在o表内容的前面

```java
@Override
public int compareTo(OrderBean o) {


    int comRes = this.pId.compareTo(o.pId);//先按照pid排序
    if (comRes == 0) {
        return o.pName.compareTo(this.pName);//在按照pname排序并且将product内容放在order前面
    } else {
        return comRes;
    }
}
```

(a) 实现compareTo

```java
// OID\ODATA\PID\PNAME\PRICE\OAMOUNT
public void setAll(String s1, String s2, String s3, String s4, String s5, String s6) {//设置值
    setOId(s1);
    setOData(s2);
    setPId(s3);
    setPName(s4);
    setPrice(s5);
    setOAmount(s6);
}
public String toString(){//不加toStirng函数，最后输出内存的地址
    return oId + "\t" + oData + "\t" + pId + "\t" + pName + "\t" + price + "\t" + oAmount;
}
```

(b) 部分其他函数(1)

```java
@Override
public void readFields(DataInput d) throws IOException {

    this.oId = d.readUTF();
    this.oData = d.readUTF();
    this.pId = d.readUTF();
    this.pName = d.readUTF();
    this.price = d.readUTF();
    this.oAmount = d.readUTF();
}

@Override
public void write(DataOutput d) throws IOException {

    d.writeUTF(oId);
    d.writeUTF(oData);
    d.writeUTF(pId);
    d.writeUTF(pName);
    d.writeUTF(price);
    d.writeUTF(oAmount);
}
```

(c) 部分其他函数(2)

图 3. orderbean部分代码实现

1.3.2 主功能Map设计思路

分辨读入的数据来源于哪个文件，分别给相应的值赋值，不存在的值则赋值为空。输出的key打包成之前自定义的ordebean类型

```
public class JoinMapper extends Mapper<LongWritable, Text, OrderBean, NullWritable> {

    private OrderBean ob = new OrderBean();
    private String fileName;

    @Override
    protected void setup(Mapper<LongWritable, Text, OrderBean, NullWritable>.Context context)
            throws IOException, InterruptedException {

        FileSplit fs = (FileSplit) context.getInputSplit();//读入文件获取文件名
        fileName = fs.getPath().getName();

        super.setup(context);//解析缓存中的数据
    }

    @Override
    protected void map(LongWritable key, Text value,
            Mapper<LongWritable, Text, OrderBean, NullWritable>.Context context)
            throws IOException, InterruptedException {

        String[] fields = value.toString().split(" ");
        // OID\ODATA\PID\PNAME\PRICE\OAMOUNT
        if (fileName.equals("product.txt")) {//product表数据赋值
            ob.setAll("", "", fields[0], fields[1], fields[2], "");
        } else {//order表数据赋值
            ob.setAll(fields[0], fields[1], fields[2], "", "", fields[3]);
        }
        context.write(ob, NullWritable.get());

    }
}
```

图 4. Mapper部分实现

1.3.3 重写Partitioner

对map发出的数据进行分区，根据pid进行分区

```
public class JoinPartitioner extends Partitioner<OrderBean, NullWritable> {
    @Override
    public int getPartition(OrderBean ob, NullWritable nw, int i) {

        return Integer.parseInt(ob.getPId()) - 1;

    }
}
```

图 5. Partitioner部分实现

1.3.4 排序

排序会根据key排序，由于key是自定义数据类型orderbean，所以在orderbean中需要自定义排序方法（见上）
compareTo函数首先按照pid排序，然后按照pname排序。由于product中pid 是唯一的，所以相同的pid中只会有一个pname，并且会被排序到最前面，形成特定的结构

oid	odata	pid	oamount	pid	pname	price
1001	20190731	4	2	1	chuizi	3999
1002	20190731	3	100	2	huawei	3999
1003	20190731	2	40	3	xiaomi	2999
1004	20190731	2	23	4	apple	5999
1005	20190801	4	55			
1006	20190801	3	20			
1007	20190801	2	3			
1008	20190801	4	23			
1009	20190802	2	10			
1010	20190802	2	2			
1011	20190802	3	14			
1012	20190802	3	18			

(a) 按照pid进行排序

oid	odata	pid	pname	price	oamount
		2	huawei	3999	
1003	20190731	2			40
1004	20190731	2			23
1007	20190801	2			3
1009	20190802	2			10
1010	20190802	2			2
		3	xiaomi	2999	
...	...	...	...	...	...

(b) 排序结果示例

图 6. 排序示例

1.3.5 自定义分组

一个reduce任务，默认只会接收到一个key的数据，所以我们要把相同pid的数据分到一个组里面处理

oid	odata	pid	pname	price	oamount
		2	huawei	3999	
1003	20190731	2			40
1004	20190731	2			23
1007	20190801	2			3
1009	20190802	2			10
1010	20190802	2			2
		3	xiaomi	2999	

图 7. 自定义分组示例

```java
public class JoinComparator extends WritableComparator{

    public JoinComparator() {

        super(OrderBean.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {//自定义分组

        OrderBean oa = (OrderBean)a;
        OrderBean ob = (OrderBean)b;
        return oa.getPId().compareTo(ob.getPId());//相同pid分到一组
    }
}
```

图 8. 重写Comparator

1.3.6 主功能Reduce设计思路

输入的第一对中的key为要join的对应的Pname及Price，将此值分别赋给之后的所有键值对中的key，并写入结果。

```java
public class JoinReducer extends Reducer<OrderBean, NullWritable, OrderBean, NullWritable>{

    @Override
    protected void reduce(final OrderBean key, final Iterable<NullWritable> val,
            Reducer<OrderBean, NullWritable, OrderBean, NullWritable>.Context context)
            throws IOException, InterruptedException {
        final Iterator<NullWritable> iterator = val.iterator();
        iterator.next();
        final String pName = key.getPName();
        final String price = key.getPrice();//得到Pname及Price
        while (iterator.hasNext()) {
            iterator.next();
            key.setPName(pName);
            key.setPrice(price);//为所有项写入
            context.write(key, NullWritable.get());
        }

    }
}
```

图 9. 重写Reducer

1.3.7 Key-Value类型协调

value设为空NullWritable，自定义key数据类型OrderBean，将数据全部封装到这里面，方便后续排序，分区，分组

1.4 代码演示

1.4.1 Map阶段代码演示

已经包含在设计思路图片中，见图4

1.4.2 Reduce阶段代码演示

同上,见图9

2 实验结果展示

2.1 结果文件截图

2.2 Hive输出结果截图

2.3 Web UI 报告内容展示

3 实验经验总结与改进方向