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：
171860550, 王一之：
171860549, 闫旭芃：
171840565, 李展烁：

1.2 任务要求

使用 MapReduce 完成两张表的 join 操作

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

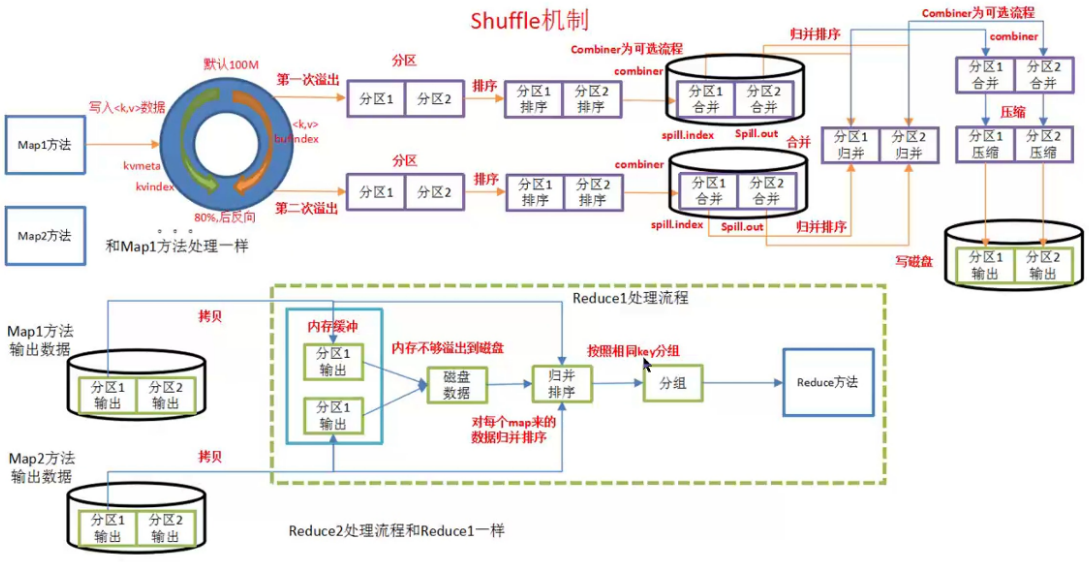


图 1. mapreduce shuffle 工作流程图

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) {
    // TODO Auto-generated method stub

    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
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 {
    // TODO Auto-generated method stub
    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 {
    // TODO Auto-generated method stub
    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 {
        // TODO Auto-generated method stub
        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 {
        // TODO Auto-generated method stub
        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());
        // super.map(key, value, context);
    }

}
```

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) {
        // TODO Auto-generated method stub
        return Integer.parseInt(ob.getPId()) - 1;
        // return 0;
    }
}
```

图 5. mapper 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

```
public class JoinComparator extends WritableComparator{

    public JoinComparator() {
        // TODO Auto-generated constructor stub
        super(OrderBean.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {//自定义分组
        // TODO Auto-generated method stub
        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

TODO: 主功能 Reduce 设计思路

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

1.4.2 Reduce 阶段代码演示

2 实验结果展示

2.1 结果文件截图

2.2 结果文件在 HDFS 上的路径

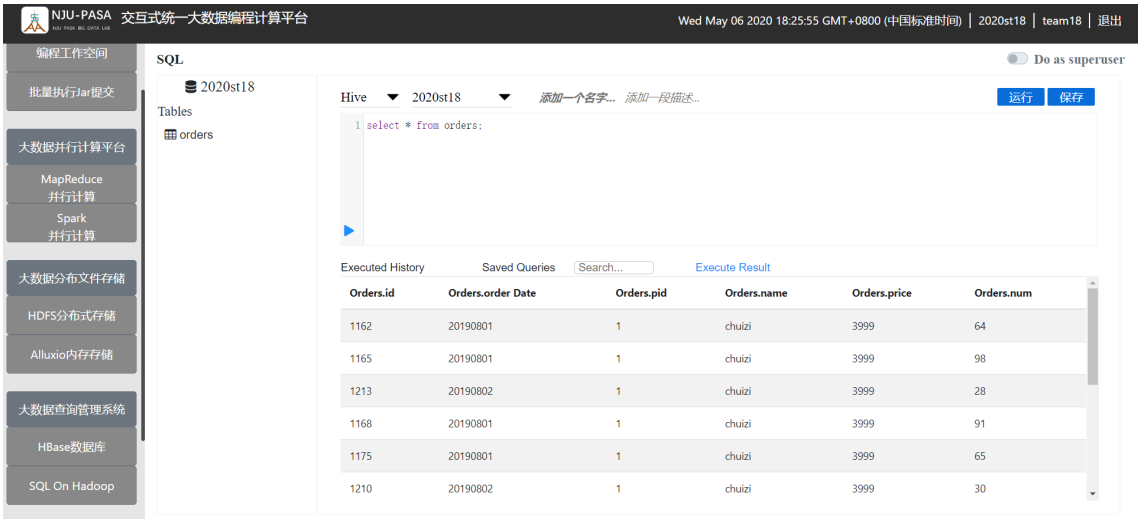
/user/2020st18/output2/

2.3 所有命令

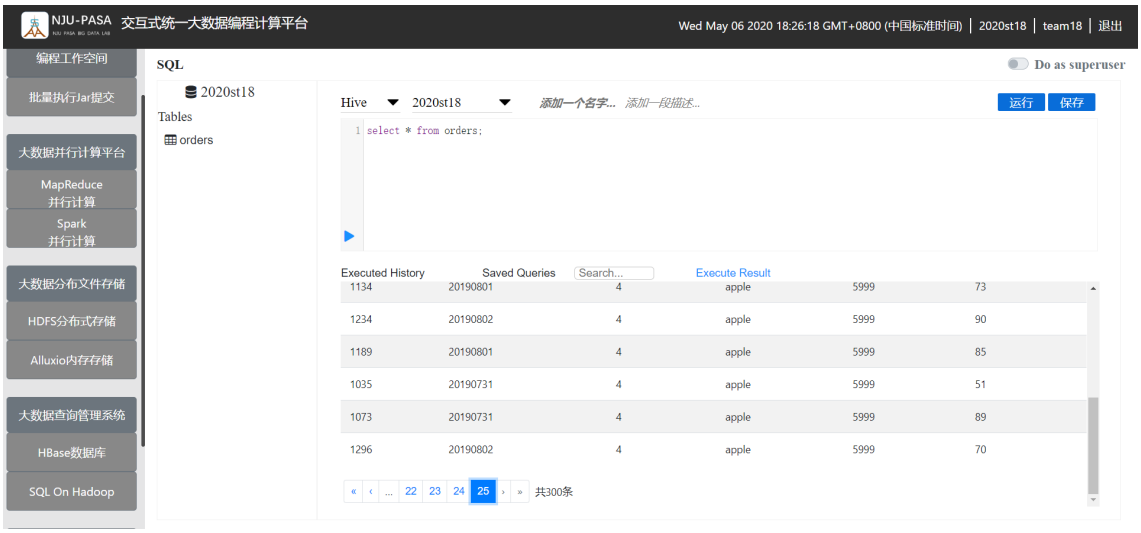
jar 包执行命令：hadoop jar /home/2020st18/HiveMyJoin.jar MyJoin.JoinDriver /data/exercise_3 output2

hive 建表命令：create table orders(id int,order_date string,pid string,name string,price int,num int) row format delimited fields terminated by ''location '/user/2020st18/output2/';

2.4 hive 输出结果文件的部分截图



(a) orders 表开头



(b) orders 表结尾

图 9. Hive 执行结果

2.5 Web UI 报告内容展示

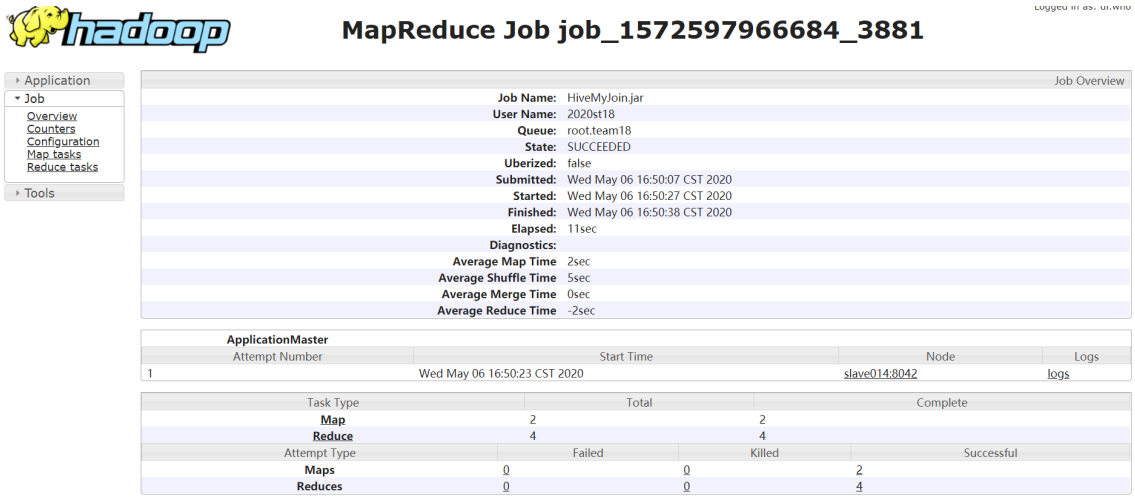


图 10. WebUI 执行报告

3 实验经验总结与改进方向

reference