

实验三: MyJoin_Hive

组号:2020st32

小组成员:

171860607 白晋斌

171860033 黄诗涵

171180541 祖东珏

联系方式:810594956@qq.com

分工:因本次实验较为简单,小组成员决定分别独立完成代码编写,实验报告中分别讲述了三位同学的代码思路. 此外,三位同学的源代码、JAR 包、JAR 包执行方式说明详见附件.

目录

实验任务.....	3
实验过程.....	3
1. Map 和 Reduce 的设计思路.....	3
1.1 黄诗涵	3
1.2 祖东珏	4
1.3 白晋斌	6
2. 集群测试	7
2.1 运行 Jar 包，输出到/user/2020st32/output_3_1.....	7
2.2 查看结果	7
3. Hive 管理输出结果	8
3.1 建表	8
3.2 SELECT 语句查询内容.....	8
WebUI 执行报告.....	9
实验感悟.....	10
参考资料.....	10

实验任务

1. 使用 MapReduce 完成/data/exercise_3 目录下两张表 order.txt 和 product.txt 的 join 操作。
2. 使用 Hive 管理输出结果
3. 能查询到建的表并用 select 语句查询内容

实验过程

1. Map 和 Reduce 的设计思路

此部分分别展示三位同学的代码设计思路与伪代码。

1.1 黄诗涵

order.txt 是订单信息，包括“订单 ID” (oid)、“订单日期”(odate)、“商品 ID” (pid)、“购买数量”(oamount)

product.txt 是商品信息，包括“商品 ID” (pid)、“商品名称”(pname)，“商品单价”(price)

要对两张表进行 join 操作也就是将商品信息里的名称和单价填充到有对应 pid 的订单信息中，相当于做笛卡尔积。

定义一个 public class OrderBean implements WritableComparable<OrderBean>

类中包含 oid,odate,pid,pname,price,oamount

将它作为 mapper 的输出，reducer 的输入。

1.1.1 自定义 compareTo 函数

```
1. public int compareTo(OrderBean other)
2. {
3.     if(pid 不相等)
4.         return pid.compareTo(other.pid);
5.     else
6.     {
7.         if(pname 为空)
8.         {
9.             if(other.pname 为空)
10.                return compareOid(other); //两个都是订单信息，用 oid 排序
11.            else return -pname.compareTo(other.pname); //将商品信息排在订单信息之前
12.        }
13.        else
14.        {
15.            if(other.pname 为空)
16.                return -pname.compareTo(other.pname);
17.            else return price.compareTo(other.price); //pid 相同的商品信息根据价格排序
18.        }
19.    }
20. }
```

1.1.2 自定义排序

需要定义 public class JoinComparator extends WritableComparator

然后设置 job.setGroupingComparatorClass(JoinComparator.class);

JoinComparator 重写 compare 方法，直接调用上面的 compareTo 即可

有了以上的数据结构的排序方法，mapper 函数要做的就是将读入的信息填入 OrderBean 并作为 key 传给 reducer，程序就会自动按照自定义的排序算法给 key 排序。

reducer 会按如下顺序接收到 mapper 的输出：

```
hadoop@ubuntu:~/hadoop_installs/hadoop-2.7.1$ hadoop fs -cat /output22/part-r-00000
1      chuizi  3999
2      huawei   3999
1003   20190731 2      40
1004   20190731 2      23
1007   20190801 2      3
1009   20190802 2      10
1010   20190802 2      2
1002   20190731 3      xiaomi 2999 100
1006   20190801 3      20
1011   20190802 3      14
1012   20190802 3      18
1001   20190731 4      apple  5999 2
1005   20190801 4      55
1008   20190801 4      23
```

显然 pid 相同的被划分为了一组，并且第一组就是商品信息。

1.1.3 reducer

```
1. public static class JoinReducer extends Reducer<OrderBean,Text,Text,Text> {
2.     OrderBean productInfo=new OrderBean();    //保存商品信息
3.     @Override
4.     protected void reduce(OrderBean key, Iterable<Text> values,Context context)
       throws IOException,InterruptedException {
5.         if(key.type().compareTo("product")==0) { //收到商品信息，更新
6.             productInfo.setPrice(key.getPrice());
7.             productInfo.setPname(key.getPname());
8.         }
9.         else { //收到订单信息，将保存的商品信息填入
10.             key.setPname(productInfo.getPname());
11.             key.setPrice(productInfo.getPrice());
12.             for(Text value:values)
13.                 context.write(new Text(key.toString()),new Text(""));
14.         }
15.     }
16. }
17. }
```

填入商品信息后按如下格式打印：

```
hadoop@ubuntu:~/hadoop_installs/hadoop-2.7.1$ hadoop fs -cat /output24/part-r-00000
1003   20190731 2      huawei  3999  40
1004   20190731 2      huawei  3999  23
1007   20190801 2      huawei  3999  3
1009   20190802 2      huawei  3999  10
1010   20190802 2      huawei  3999  2
1002   20190731 3      xiaomi 2999  100
1006   20190801 3      xiaomi 2999  20
1011   20190802 3      xiaomi 2999  14
1012   20190802 3      xiaomi 2999  18
1001   20190731 4      apple  5999  2
1005   20190801 4      apple  5999  55
1008   20190801 4      apple  5999  23
```

1.2 祖东珏

本次实验中需要实现 product 表和 order 表的 join 操作，product 表和 order 表的主键均为商品 ID (pid)，因此基本思路是将 product 表和 order 表中 pid 相同的记录分配到同一个节点进行 reduce 操作，将 product 的商品名称 (pname) 和商品单价 (price) 填入订单中。

问题在于，如果直接使用 pid 作为 key，虽然能够正确将对应的记录分配到同一个 Reducer 中，但是无法知道哪一条记录是 product 表中的，需要遍历当前 Reducer 中的所有记录，时间复杂度高。因此，可以利用 MapReduce 过程中系统的排序来对相同 pid 的记录进行排序，将 product 表中的记录放在最前面，后面依次跟上 order 中的记录。

1.2.1 数据类型

本实验中定义一个数据类型 Record

```
enum RecordType {
    Unknown, Product, Order
}

public class Record implements WritableComparable<Record> {
    public Long pid;
    public String pname;
    public Float price;
    public Long oid;
    public String odata;
    public Long oamount;
    RecordType type;

    public Record() {...}

    public int compareTo(Record o) {...}

    @Override
    public void write(DataOutput dataOutput) throws IOException {...}

    @Override
    public void readFields(DataInput dataInput) throws IOException {...}

    @Override
    public String toString() { return oid + " " + odata + " " + pid + " "
}
```

i.Record 类型中有一条记录的所有内容：商品 ID、商品名称、商品单价、订单 ID、订单日期、购买数量。添加了一条数据用以标示记录属于 Product 或 Order。

ii.为了实现排序后 product 表记录在最前，order 表记录按照订单 ID 排序的功能，实现了 compareTo()函数：

```
public int compareTo(Record o) {
    int idcompare = this.pid.compareTo(o.pid);
    if(idcompare != 0)
        return idcompare;
    else {
        if(this.type == RecordType.Product && o.type == RecordType.Product)
            return this.pname.compareTo(o.pname);
        else if(this.type == RecordType.Order && o.type == RecordType.Order)
            return this.oid.compareTo(o.oid);
        else if(this.type == RecordType.Product && o.type == RecordType.Order)
            return 1;
        else
            return -1;
    }
}
```

iii.剩余函数用以满足 WritableComparable 接口要求和输出要求

1.2.2 Map

本实验中的 Map 函数较为简单，仅需要根据数据来自 product 或 order 表进行填充并发送即可。

```
@Override
public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    String[] attributes = value.toString().split(" ");
    Record record = new Record();
    if(filename.equals("product.txt")) {...}
    else if(filename.equals("order.txt")) {...}
    else
        throw new IOException("Unknown Filename!" + filename.toString());
    context.write(record, NullWritable.get());
}
```

1.2.3 Partition

由于使用自定义数据类型，可能导致 pid 相同的记录被分配到不同的 Reducer 中，因此重新实现了以 id 作为分类依据的 Partitioner

```
// Partitioner
public static class MyJoinPartitioner extends HashPartitioner<Record, NullWritable> {
    @Override
    public int getPartition(Record key, NullWritable value, int numReduceTasks) {
        return key.pid.intValue() % numReduceTasks;
    }
}
```

1.2.4 Reduce

在实现了上述功能后，Reducer 会依次接收到按 pid 排序的一系列记录，其中每个 pid 中第一个接收到的记录为 product，此时记录下 pname 和 price 用以后续填充。在接收到 order 记录后用之前记录的 pname 和 price 进行填充后即可输出。

```
@Override
public void reduce(Record key, Iterable<NullWritable> values, Context context) throws IOException, InterruptedException {
    if(key.type == RecordType.Product) {
        curPname = key.pname;
        curPrice = key.price;
    }
    else if(key.type == RecordType.Order) {
        key.pname = curPname;
        key.price = curPrice;
        context.write(key, NullWritable.get());
    }
    else
        throw new IOException("Record type Unknown!");
}
```

1.3 白晋斌

1.3.1 Map

要根据数据来自 product 或 order 表,分别切分并提取对应部分,将 pid 作为 key,其他部分作为 value,并增加一个标示符记忆来自哪个表,我们将次标识符和 value 一并作为 map 的输出 value。

```
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
    String fileName = ((FileSplit)context.getInputSplit()).getPath().getName().split(regex: "(.txt)|(.TXT)")[0];
    String line = value.toString();
    if (line == null || line.equals("")) return;
    String[] lines = line.split(DELIMITER);
    if(fileName.equals("order")){
        String id1=lines[0];
        String order_date2=lines[1];
        String pid3=lines[2];
        String number6=lines[3];
        context.write(new Text(pid3),new Text( string: "order"+DELIMITER+id1+DELIMITER+order_date2+DELIMITER+pid3+DELIMITER+number6));
    }
    else if(fileName.equals("product")){
        String pid3=lines[0];
        String name4=lines[1];
        String price5=lines[2];
        context.write(new Text(pid3),new Text( string: "product"+DELIMITER+pid3+DELIMITER+name4+DELIMITER+price5));
    }
}
}
```

1.3.2 Reduce

因为同一 pid 被分到同一 reduce 函数,我们可以直接遍历 value,根据标识符存为两个 vector,按照左连接的定义,我们遍历 order 的 vector,对每个 order 遍历 product 的 vector 生成输出,如果找不到对应的 product 则设置对应部分为 NULL.

之后将连接出的新数据作为 key 输出即可.

```
protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException{
    Vector<String> order = new Vector<String>();
    Vector<String> product = new Vector<String>();

    for(Text each_val:values) {
        String each = each_val.toString();
        if(each.startsWith("order")) {
            order.add(each);
        } else if(each.startsWith("product")) {
            product.add(each);
        }
    }

    for(String ord:order){
        String[] ords=ord.split(DELIMITER);
        String id1=ords[1];
        String order_date2=ords[2];
        String pid3=key.toString();
        String number6=ords[4];
        if (product.size() == 0) {
            context.write(new Text( string: id1+DELIMITER+order_date2 + DELIMITER + pid3+DELIMITER+"NULL"+DELIMITER+"NULL"+DELIMITER+number6), new Text());
        } else {
            for(String prod:product){
                String[] prods=prod.split(DELIMITER);
                String name4=prods[2];
                String price5=prods[3];
                context.write(new Text( string: id1+DELIMITER+order_date2 + DELIMITER + pid3+DELIMITER+name4+DELIMITER+price5+DELIMITER+number6), new Text());
            }
        }
    }
}
}
```

1.3.3 高级算法

可直接用 DataJoin 类实现 Map 端 Join 和 Reduce 端 Join.在此不做赘述,该方式可大大提高程序鲁棒性.

2. 集群测试

2.1 运行 Jar 包, 输出到/user/2020st32/output_3_1

请输入MR提交命令, 例: `hadoop jar /home/data/WordCount.jar WordCount /zj/input /zj/output`

```
hadoop jar /home/2020st32/MyJoin.jar /data/exercise_3 /user/2020st32/output_3_1
```

2.2 查看结果

在大数据分布文件存储->HDFS 分布式存储中找到输出的文件

```
1005 20190731 1 chuizi 3999 60
1016 20190731 1 chuizi 3999 84
1017 20190731 1 chuizi 3999 73
1019 20190731 1 chuizi 3999 56
1024 20190731 1 chuizi 3999 35
1026 20190731 1 chuizi 3999 100
1027 20190731 1 chuizi 3999 84
1030 20190731 1 chuizi 3999 89
1031 20190731 1 chuizi 3999 74
1033 20190731 1 chuizi 3999 19
1036 20190731 1 chuizi 3999 69
1046 20190731 1 chuizi 3999 28
1047 20190731 1 chuizi 3999 52
1051 20190731 1 chuizi 3999 87
1055 20190731 1 chuizi 3999 81
```

Cancel

Download

3. Hive 管理输出结果

3.1 建表

输入语句: create table order2020st32 (id string,order_date string,pid string,name string,price string,num string) row format delimited fields terminated by ' ' location '/user/2020st32/output_3_1/';

创建表后能看到出现一个名字为 order2020st32 的 table.

Tab Name

order2020st32

3.2 SELECT 语句查询内容

Hive 2020st32 添加一个名字... 添加一段描述... 更新 运行 保存

```
1 create table order2020st32 (id string,order_date string,pid string,name string,price string,num string) row format delimited fields
2 show tables;
3 select * from order2020st32;
```

Executed History	Saved Queries	Execute Result
1030	20190731	1 chuizi 3999 89
1031	20190731	1 chuizi 3999 74
1033	20190731	1 chuizi 3999 19
1036	20190731	1 chuizi 3999 69
1046	20190731	1 chuizi 3999 28

< 1 2 3 4 ... >

共300条

Hive
2020st32
添加一个名字... 添加一段描述...
运行 保存

```
1 select * from order2020st32;
```


Executed History Saved Queries [Execute Result](#)

Order2020st32.id	Order2020st32.order Date	Order2020st32.pid	Order2020st32.name	Order2020st32.price	Order2020st32...
1005	20190731	1	chuizi	3999	60
1016	20190731	1	chuizi	3999	84
1017	20190731	1	chuizi	3999	73
1019	20190731	1	chuizi	3999	56
1024	20190731	1	chuizi	3999	35
1026	20190731	1	chuizi	3999	100

WebUI 执行报告

Show 20 entries
Search

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress
application_1572597966684_3361	2020st32	My Join	MAPREDUCE	root.team32	Fri Apr 24 16:15:04 +0800 2020	Fri Apr 24 16:15:21 +0800 2020	FINISHED	SUCCEEDED	



Application application_1572597966684_3361

Cluster

- About
- Nodes
- Node Labels
- Applications
 - NEW
 - NEW SAVING
 - SUBMITTED
 - ACCEPTED
 - RUNNING
 - FINISHED
 - FAILED
 - KILLED
- Scheduler

Tools

Kill Application

Application Overview

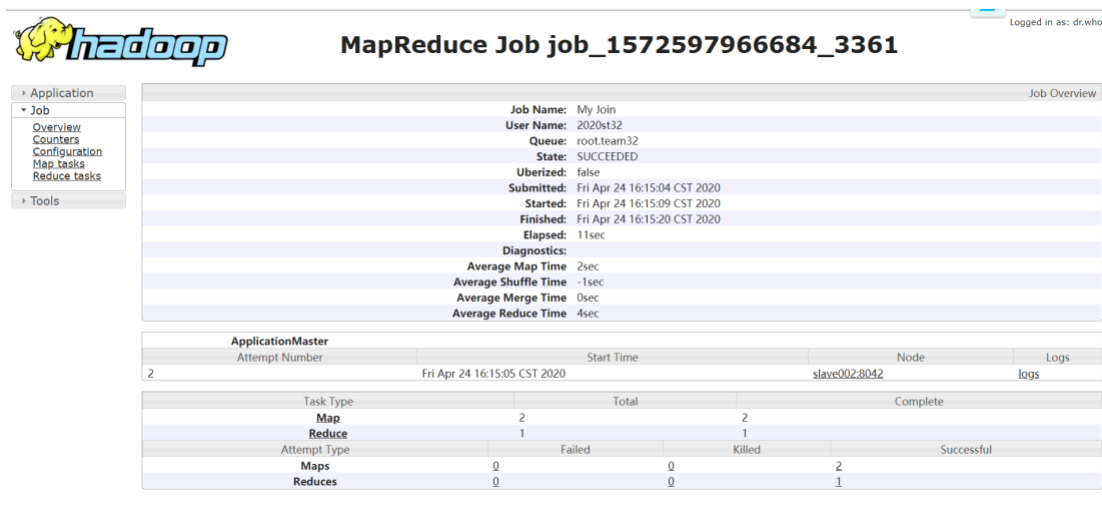
User: 2020st32
Name: My Join
Application Type: MAPREDUCE
Application Tags:
YarnApplicationState: FINISHED
Queue: root.team32
FinalStatus Reported by AM: SUCCEEDED
Started: Fri Apr 24 16:15:04 +0800 2020
Elapsed: 17sec
Tracking URL: History
Diagnostics:

Application Metrics

Total Resource Preempted: <memory:0, vCores:0>
Total Number of Non-AM Containers Preempted: 0
Total Number of AM Containers Preempted: 0
Resource Preempted from Current Attempt: <memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt: 0
Aggregate Resource Allocation: 60072 MB-seconds, 34 vcore-seconds

Show 20 entries
Search:

Attempt ID	Started	Node	Logs	Blacklisted Nodes
appattempt_1572597966684_3361_000002	Fri Apr 24 16:15:04 +0800 2020	http://slave002:8042	Logs	N/A



实验感悟

1. MapReduce wordcount 测试卡死在 running job,原因竟然是

```

20/04/26 23:56:07 INFO input.FileInputFormat: Total input paths to process : 2
20/04/26 23:56:07 INFO mapreduce.JobSubmitter: number of splits:2
20/04/26 23:56:07 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1587226935256_0013
20/04/26 23:56:08 INFO impl.YarnClientImpl: Submitted application application_1587226935256_0013
20/04/26 23:56:08 INFO mapreduce.Job: The url to track the job: http://192.168.1.104:8088/proxy/application_1587226935256_0013/
20/04/26 23:56:08 INFO mapreduce.Job: Running job: job_1587226935256_0013
  
```

之前执行 MapReduce 的时候强制按 CTRL+C 退出了，tmp 里面保存了 job 的一些信息，而 safemode 是无法删除 tmp 里面的这些信息的

解决方案:重新格式化.

参考资料[2][3]

2.生成的 jar 包一度很大怎么办?参考[4][5]

参考资料

- [1] <https://stackoverflow.com/questions/40315820/error63-40-java-incompatible-types-org-apache-hadoop-mapreduce-job-cannot>
- [2] <https://www.cnblogs.com/TTYb/p/8274246.html>
- [3] <https://blog.csdn.net/hukun910903/article/details/79736257>
- [4] <https://jingyan.baidu.com/article/48b558e3f8f6637f39c09a44.html>
- [5] <https://blog.csdn.net/ming19951224/article/details/81416387>