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 14 日

1 实验规划与设计

1.1 任务分配

171860551, 韩畅：组长
171860550, 王一之：
171860549, 闫旭芃：
171840565, 李展烁：

1.2 任务要求

使用 MapReduce 完成对数据的二次排序。

1.3 设计思路

map 过程将每行的两个值处理为自定义数据结构 Pair_Int 作为 key，为减少传输量，value 为空。重写 partitioner，保证分区时第一个值相同的值分配到同一个 ruducer 上。在 Pair_Int 中设置比较函数，每个分区会调用此比较函数，此时便完成了二次排序。重载 Comparator，保证分组时第一个值相同的结果在同一批次任务中处理,reduce 阶段将传入的值转化为 text 直接输出即可。

1.3.1 自定义数据类型

自定义一个 Pair_Int 存储数值对

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

    private int lf = 0;
    private int rt = 0;
```

图 1. 自定义 Pair_Int

重载读写方法，，重载 toString 方法方便最终输出，以二次排序的规则重载 compareTo 方法，同时重载 WritableComparator 提供字节流比较

```java
@Override
public int compareTo(Pair_Int pI) {
    // 第一列的值升序排列，而第二列的值降序排列

    if (lf != pI.lf) {
        return lf < pI.lf ? -1 : 1;
    } else if (rt != pI.rt) {
        return rt > pI.rt ? -1 : 1;
    } else {
        return 0;
    }
}
```

(a) 重载比较

```java
public static class SSComparator extends WritableComparator {
    public SSComparator() {
        super(Pair_Int.class);
    }

    // 使用二进制字典序排列
    @Override
    public int compare(byte[] arg0, int arg1, int arg2, byte[] arg3, int arg4, int arg5) {

        return compareBytes(arg0, arg1, arg2, arg3, arg4, arg5);
    }
}

// 在WritableCoparator里激活新定义的comparator
static {
    WritableComparator.define(Pair_Int.class, new SSComparator());
}
```

(b) 重载比较 2

图 2. Pair_Int 部分代码实现 1

```java
// 必须重载的读入和输出方法，为了利用编码范围，需要加上偏移量
@Override
public void readFields(DataInput dI) throws IOException {

    lf = dI.readInt() + Integer.MIN_VALUE;
    rt = Integer.MAX_VALUE - dI.readInt();
}

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

    dO.writeInt(lf - Integer.MIN_VALUE);
    dO.writeInt(Integer.MAX_VALUE - rt);
}
```

(a) 重载读写

```java
@Override
public String toString() {
    return this.lf + "\t" + this.rt;
}

@Override
public int hashCode() {
    // 由于第二列的值的范围不超过100，因此使用此函数即可确保任意不同的一二列值对的哈希码不同
    return lf * 100 + rt;
}

@Override
public boolean equals(Object obj) {

    // 当且仅当两列完全相同时相等，不同类型，存在不相等均返回false
    if (obj instanceof Pair_Int) {
        Pair_Int p = (Pair_Int) obj;
        return p.lf == lf && p.rt == rt;
    } else {
        return false;
    }
}
```

(b) 部分其他函数

图 3. Pair_Int 部分代码实现 2

1.3.2 主功能 Map 设计思路

读入数据，每行的两个值以自定义数据结构 Pair_Int 的方式作为 key，value 都为空

```java
public class SecondSortMapper
        extends Mapper<Text, Text, Pair_Int, NullWritable> {

    private final Pair_Int key = new Pair_Int();

    @Override
    public void map(Text key, Text value,Context context) throws IOException, InterruptedException {
      this.key.set(Integer.parseInt(key.toString()), Integer.parseInt(value.toString()));
      context.write(this.key, NullWritable.get());//key中为intpair，value为空
    }
}
```

图 4. Mapper 实现

1.3.3 重写 Partitioner

重写 Partitioner 自定义分区，根据第一个值进行分区，保证第一个值相同在同一个 reducer 上

```java
public class L1Partitioner extends Partitioner<Pair_Int, IntWritable> {
    @Override
    // 保证相同的第一列值必然分配到同一个reducer上
    public int getPartition(Pair_Int key, IntWritable val, int num) {

        return Math.abs(key.getLf() * 100) % num;
    }
}
```

图 5. Partitioner 部分实现

1.3.4 排序

mapreduce 此阶段对每个分区内进行排序，key 为自定义类型，所以在 Pair_Int 中自定义排序方法，见2
compareTo 函数首先第一个值，若相同则比较第二个值，完成第一列升序，第二列降序的比较。

1.3.5 自定义分组

为保证结果正确，第一个值相同的结果要在同一个 reduce 任务中处理因此重载 RawComparator 进行自定义分组

```java
public class SecondSortGroupingComparator implements RawComparator<SecondarySort.Pair_Int> {
    @Override
    public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
        return WritableComparator.compareBytes(b1, s1, Integer.BYTES, b2, s2, Integer.BYTES);
    }

    //根据第一个字段分组
    @Override
    public int compare(SecondarySort.Pair_Int p1, SecondarySort.Pair_Int p2) {
        return Integer.compare(p1.getLf(), p2.getLf());
    }
}
```

图 6. 重写 Comparator

1.3.6 主功能 Reduce 设计思路

输入的 key 中已经是排好序的结果，自定义类型已经实现了 tostring，直接输出即可

```java
public class SecondSortReducer extends Reducer<Pair_Int, NullWritable, Pair_Int, NullWritable> {

    @Override
    public void reduce(Pair_Int key, Iterable<NullWritable> values, Context context)
            throws IOException, InterruptedException {
        for (NullWritable ignored : values) {//int对在key中，value为空，直接忽略
            context.write(key, NullWritable.get());
        }
    }
}
```

图 7. 重写 Reducer

1.3.7 Key-Value 类型协调

map 输出类型为 intPair, IntWritable，value 实际为空 (NullWritable)
reduce 输出类型为 intPair, IntWritable，value 同样为空 (NullWritable)，但 key 中会调用 Pair_Int 的 tostring，输出实际为 text

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3

1.4.2 Reduce 阶段代码演示

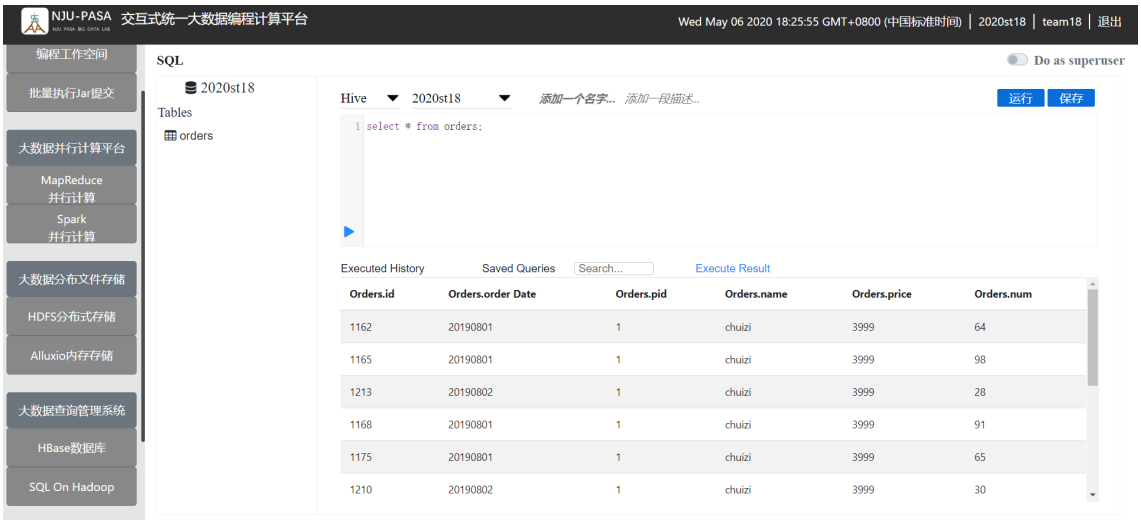
同上, 见图6

2 实验结果展示

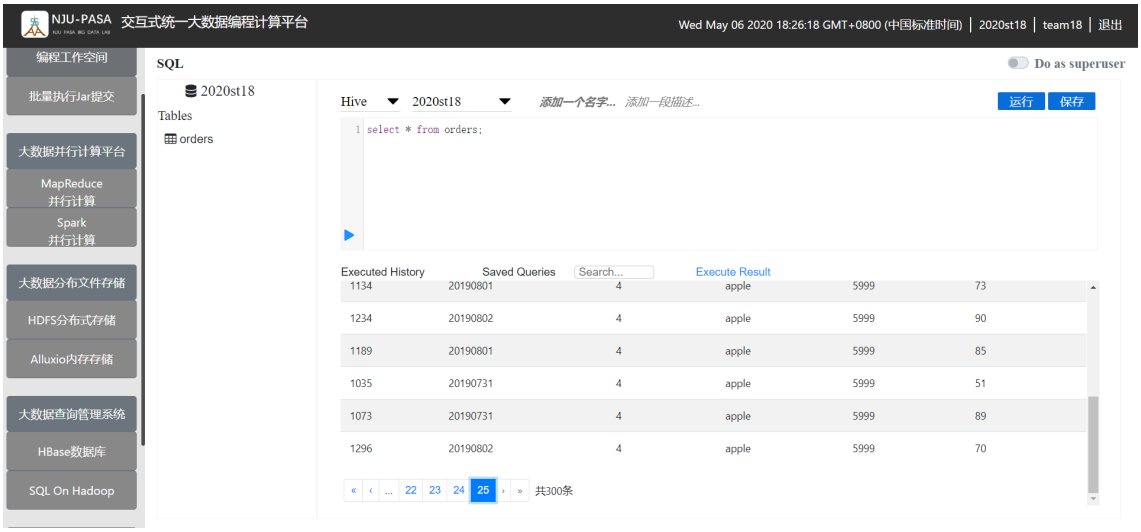
2.1 结果文件在 HDFS 上的路径

todo

2.2 输出结果文件的部分截图



(a) 结果开头



(b) 结果结尾

图 8. 执行结果

2.3 Web UI 报告内容展示

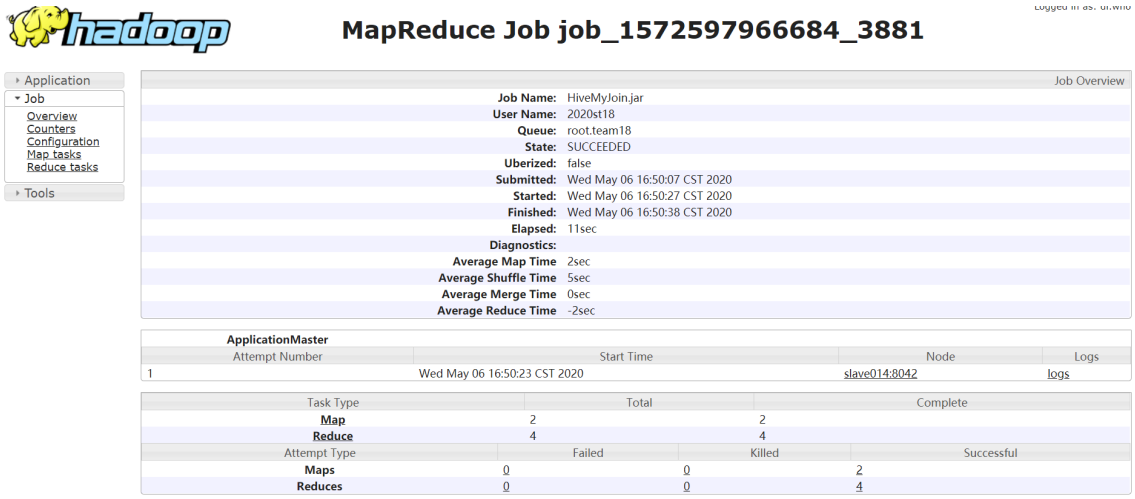


图 9. WebUI 执行报告

3 实验经验总结与改进方向

- todo
- todo