Day05_MapReduce的排序

大数据-张军锋 Day05 MapReduce排序 全排序 二次排序

数据的排序

> reducer的排序为单节点排序
> 使用新的分区策略解决多个reduce的排序

全排序

过程:

- 先进行排序,排序完取个中间值,作为分区的条件,就会保证是有序的

- 定义抽样:
  - 对大数据进行数据的统计分布分区时,如果进行全部直接排序的话,需要运行所有的文件就会浪费时间,则出现了抽样
  - 抽样从数据中取出样本,然后对样本进行排序,找到中值,作为数据的中值,进行分区
- 分区
  - 在进行计算的时候要将map输出的数据分配到不同的reduce中,而mapper分配划分数据过程就是Partition,负责分区数据的类叫做Partitioner
  - 设置全排序的分区和分区文件的保存路径,则Partitioner程序就会读取分区文件来完成按顺序进行分区
- 加载分区文件到分布式缓存中,则缓存文件会自动把分区文件加载分配到每一个map节点的内存中
- 设置reducer节点的个数,默认为1个
- 要是想倒序排序,方法之一是指定job的自己定义的继承了compare的类
- 设置读取文件输出文件:
  - map端的输入会把文本文件读取成kv对,按照分割符把一行分为两部分,前面为key后面为value
  - 如果分隔符不存在则整行都是key,value为空
  - 默认分隔符为/t,手动指定分隔符参数:mapreduce.input.keyvaluelinerecordreader.key.value.separator
  - 如果用KeyValueTextInputFormat,则参数类型为Text,Text,以文本字符串来存储,则不能满足用数字排序存储
  - SequenceFile在保存数据的同时会保留数据的类型,本身就是以kv方式保存数据的,则解决了KeyValueTextInputFormat的问题
- 将随机抽样写入分区文件,在job启动之前启动抽样程序,并将抽样程序得到的中值写入分区文件中

全局排序代码

- 自定义的排序

```
public static class WritableDescComparetor extends IntWritable.Comp
arator {

    @Override
    public int compare(byte[] arg0, int arg1, int arg2, byte[] arg
3, int arg4, int arg5) {
        return -super.compare(arg0, arg1, arg2, arg3, arg4, arg5);
    }

}
```

- 全排序的job

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    //定义抽样
    InputSampler.Sampler<IntWritable, Text> sampler = new InputSamp
ler.RandomSampler<>(0.6, 5);
    FileSystem fileSystem = FileSystem.get(conf);
    //设置分区文件路径,抽样的结果写入
    Path partitionFile = new Path("/bd14/totalsor/_partition");
    //设置后,全排序的partition程序就会读取这个分区文件来完成按顺序进行分区
    partitionFile.getFileSystem(conf).delete(partitionFile, true);
    TotalOrderPartitioner.setPartitionFile(conf, partitionFile);
    Job job = Job.getInstance(conf);
    job.setJarByClass(TotalSort.class);
    job.setJobName("进行排序");
    job.setMapperClass(Mapper.class);
    job.setReducerClass(TotalSortReduce.class);
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(Text.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    //把分区文件加入分布式缓存中
    job.addCacheFile(partitionFile.toUri());
    //设置分区器
    job.setPartitionerClass(TotalOrderPartitioner.class);
    //设置reducer节点个数
    job.setNumReduceTasks(2);
    //如果要倒序排序,方法之一是指定job的sortcompare方法
    job.setSortComparatorClass(WritableDescComparetor.class);
    Path inPath = new Path("/bd14/two/part-r-00000");
    Path outPath = new Path("/bd14/totalsor1");
    fileSystem.delete(outPath, true);
    //设置文件输出文件为SequenceFile
    job.setInputFormatClass(SequenceFileInputFormat.class);
    FileInputFormat.addInputPath(job, inPath);
    FileOutputFormat.setOutputPath(job, outPath);
    //将随机抽样写入分区文件
    InputSampler.writePartitionFile(job, sampler);
    //启动job
    System.exit(job.waitForCompletion(true) ? 0 : 1);

}
```

二次排序

原理过程

- 概念:
  - 先排序第一个字段,第一个字段相同的时候排序第二个字段
- 过程:
  - 将两个字段合并成一个对象,作为key发送到reducer节点上
  - 在这个对象类型中自定义排序规则,重写compare方法(这个对象类型实现WritableComparable类)
    - 要重写readFields,write,compareTo方法
    - readFields序列化
    - write反序列化
    - 序列化和反序列化的顺序必须保持一致
    - compareTo比较,在比较方法中实现二次排序
  - 自定义分区,用来把第一个字段的key值分区到同一个节点上,返回值为数字,根据数据的求余,可以进行比较均衡的分,返回的数字为reduce的标号
  - 在map中把数据封装到定义的对象类型中
  - 在reducer中再进行数据的提取
  - 定义分组比较器,让不同的key值的第一个字段相同的kv调用同一个reduce方法
    - 继承WritableComparator
    - 重写compare方法自定义排序规则
    - 在构造方法中向父类传递要比较的数据类型
  - 设置job中的分组比较器为自己定义的

二次排序代码:

1 . 定义封装类型和排序规则:

```
public static class SecondarySortCompareble implements WritableComp
arable<SecondarySortCompareble>{
    private String  firstFiled;
    private int secondFiled;
    public String getFirstFiled() {
        return firstFiled;
    }
    public void setFirstFiled(String firstFiled) {
        this.firstFiled = firstFiled;
    }
    public int getSecondFiled() {
        return secondFiled;
    }
    public void setSecondFiled(int secondFiled) {
        this.secondFiled = secondFiled;
    }
    //序列化
    @Override
    public void readFields(DataInput in) throws IOException {
        this.firstFiled = in.readUTF();
        this.secondFiled = in.readInt();
    }
    //反序列化
    @Override
    public void write(DataOutput out) throws IOException {
        out.writeUTF(firstFiled);
        out.writeInt(secondFiled);
    }
    //比较方法
    @Override
    public int compareTo(SecondarySortCompareble o) {
        if(this.firstFiled.equals(o.getFirstFiled())){
            /*if(this.secondFiled>o.getSecondFiled()){
                return 1;
            }else if(this.secondFiled<o.getSecondFiled()){
                return -1;
            }else{
                return 0;
            }*/
            return this.secondFiled - o.getSecondFiled();
        }else{
            return this.firstFiled.compareTo(o.getFirstFiled());
        }
    }
}
```

2 . 自定义分区:

```
public static class SecondarySortPartition extends Partitioner<Seco
ndarySortCompareble, NullWritable> {
    @Override
    public int getPartition(SecondarySortCompareble key, NullWritab
le value, int numpartition) {
        int reduceNo = (key.getFirstFiled().hashCode()&Integer.MA
X_VALUE) % numpartition;
        return reduceNo;
    }
}
```

3 . 自定义分组比较器:

```
public static class SecondarySortGroupComparator extends WritableCo
mparator {
    //构造方法里面向父类传递比较器要比较的数据类型
    public SecondarySortGroupComparator(){
        super(SecondarySortCompareble.class,true);
    }
       //重写compare方法自定义排序规则
    @Override
    public int compare(WritableComparable a, WritableComparable b)
{
        SecondarySortCompareble ssca = (SecondarySortCompareble)a;
        SecondarySortCompareble sscb = (SecondarySortCompareble)b;
        return ssca.getFirstFiled().compareTo(sscb.getFirstFile
d());
    }
}
```

4 .job进行工作:

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf);
    job.setJarByClass(SecondarySort.class);
    job.setJobName("二次排序");
    job.setMapperClass(SecondarySortMap.class);
    job.setReducerClass(SecondarySortReduce.class);
    job.setMapOutputKeyClass(SecondarySortCompareble.class);
    job.setMapOutputValueClass(NullWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    Path inPath = new Path("/secondaryorder");
    Path outPath = new Path("/bd14/second");
    outPath.getFileSystem(conf).delete(outPath, true);
    FileInputFormat.addInputPath(job, inPath);
    FileOutputFormat.setOutputPath(job, outPath);
    job.setInputFormatClass(KeyValueTextInputFormat.class);
    job.setPartitionerClass(SecondarySortPartition.class);
    //设置分组比较器
    job.setGroupingComparatorClass(SecondarySortGroupComparator.cla
ss);
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

抽样,求中值,根据中值定义partition