好程序员
从平凡到卓越，为梦想而拼搏
GoodProgrammer.org

2.4.15、自定义的partitioner

partitioner定义：

partitioner的作用是将mapper 输出的key/value拆分为分片（shard），每个reducer对应一个分片。

默认情况下，partitioner先计算key的散列值（hash值）。然后通过reducer个数执行取模运算：key.hashCode%(reducer个数)。这样能够随机地将整个key空间平均分发给每个reducer,同时也能确保不同mapper产生的相同key能被分发到同一个reducer。

目的：

可以使用自定义Partitioner来达到reducer的负载均衡， 提高效率。

适用范围：

需要非常注意的是：必须提前知道有多少个分区。比如自定义Partitioner会返回4个不同int值，而reducer number设置了小于4，那就会报错。所以我们可以通过运行分析任务来确定分区数。例如，有一堆包含时间戳的数据，但是不知道它能追朔到的时间范围，此时可以运行一个作业来计算出时间范围。

注意：

在自定义partitioner时一定要注意防止数据倾斜。

2.4.16、案例：按照手机归属地进行分区保存数据分析结果

手机号码属于各个省，如何将各个省的上网数据写到一个文件中呢？

手机号码属于哪儿个省市可查的，会有数据字典。

那么有了这种数据字典如何实现呢?

我们要解决的问题就是如何把相同省份的手机号码发送到同一个reduce中去处理

这种就是分区组件的实现，默认的分区规则是key的hash值%reduce的数量，那这种规则肯定实现不了我们的需求

那这种组件提供了一个抽象类，我们可以通过继承这个类重写分区方法来改变数据的分发规则，即省份相同的数据发送到同一个reduce中

这个类就是Partitioner

```
//<Text,flowBean>数据类型和map输出的数据类型一致，看图，是将map的输出数据分发
public class provincePartitioner extends Partitioner<Text,flowBean>{

    //如何调用？每一个键值对就调用一次，每次都查询去数据库中查询数据字典太耗费资源
    //所以我们模拟数据字典加载到内存
    public int getPartition(Text key, flowBean bean, int numPartitions) {
    }
    //我们可以把数据字典的数据加载到内存，便于数据的分发
    //模拟数据字典（查询手机号属于哪儿个省份），定义一个map用于数据字典数据的存储
    private static HashMap<String,Integer> pmap = new HashMap<>();
    //静态初始化
    static{
        //假设136是0号省份，被分发到0号reduce处理
        pmap.put("136",0);
        pmap.put("137",1);
```

千锋好程序大数据学院

```
        pmap.put("138",2);
        pmap.put("139",3);
    }
     @Override
    public int getPartition(Text key, flowBean bean, int numPartitions) {
        String prefix = key.toString().substring(0,3);
        Integer partNum = pmap.get(prefix);
        //有可能有不属于这些分区的的数据，那就给一个默认分区
        return (partNum==null ? 4:partNum);
    }
}
```

然后将自定义分区设置进job中job.setPartitionerClass(ProvincePatitioner.class);

2.4.17、TOPN案例

求每个人评分最高的10部电影

定义FlownBean

```
//先实现序列化，后期更改
public class RateBean implements WritableComparable<RateBean> {
    private String movie;
    private String rate;
    private String timeStamp;
    private String uid;
    public String getMovie() {
        return movie;
    }

    public void setMovie(String movie) {
        this.movie = movie;
    }

    public String getRate() {
        return rate;
    }

    public void setRate(String rate) {
        this.rate = rate;
    }

    public String getTimeStamp() {
        return timeStamp;
    }

    public void setTimeStamp(String timeStamp) {
        this.timeStamp = timeStamp;
    }

    public String getUid() {
        return uid;
    }

    public void setUid(String uid) {
        this.uid = uid;
    }
```

千锋好程序大数据学院

```
    @Override
    public void write(DataOutput out) throws IOException {
        out.writeUTF(movie);
        out.writeUTF(rate);
        out.writeUTF(timeStamp);
        out.writeUTF(uid);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
        this.movie = in.readUTF();
        this.rate = in.readUTF();
        this.timeStamp = in.readUTF();
        this.uid = in.readUTF();
    }
    @Override
    public String toString() {
        return movie + "\t" + rate;
    }
    @Override
    public int compareTo(RateBean o) {
        return -this.rate.compareTo(o.rate);
    }
}
```

定义mapper

```
public class RateMapper extends Mapper<LongWritable, Text,Text,RateBean> {
    Text k = new Text();//先不创建
    ObjectMapper objectMapper;//先不创建，创建
    @Override
    protected void setup(Context context) throws IOException,
InterruptedException {
        objectMapper = new ObjectMapper();//赋值
    }

    @Override
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        //ObjectMapper objectMapper = new ObjectMapper();//拿到外面，仅创建一次就可
以
        RateBean rateBean =
objectMapper.readValue(value.toString(),RateBean.class);
        System.out.println(k.toString());
        k.set(rateBean.getUid());
        context.write(k,rateBean);
    }
}
```

定义reduce

```
public class RateReduce extends Reducer<Text,RateBean,Text,RateBean> {
    @Override
    protected void reduce(Text key, Iterable<RateBean> values, Context context)
throws IOException, InterruptedException {
        //定义一个list,对list排序
```

千锋好程序大数据学院

```
        List<RateBean> rateBeanList = new ArrayList<RateBean>();
        Configuration conf = context.getConfiguration();
        int topN = conf.getInt("topN",5);
        for (RateBean rateBean:values){
            RateBean newBean = new RateBean();
            newBean.setMovie(rateBean.getMovie());
            newBean.setRate(rateBean.getRate());
            newBean.setTimeStamp(rateBean.getTimeStamp());
            newBean.setUid(rateBean.getUid());
            rateBeanList.add(newBean);
        }
        Collections.sort(rateBeanList);
        for (int i=0;i<topN;i++){
            context.write(key,rateBeanList.get(i));
        }
    }
}
```

定义runner

```
public class RateRunner  {
    public static void main(String[] args) {
        try {
            Configuration conf = new Configuration();
            conf.set("topN",args[0]);
            Job job = Job.getInstance(conf,"rate");
            job.setMapperClass(RateMapper.class);
            job.setReducerClass(RateReduce.class);

            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(RateBean.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(RateBean.class);
            FileInputFormat.addInputPath(job,new Path(args[1]));
            Path out = new Path(args[2]);
            FileSystem fs = FileSystem.get(conf);
            if (fs.exists(out)){
                fs.delete(out,true);
            }
            FileOutputFormat.setOutputPath(job,out);
            int res = job.waitForCompletion(true)?0:1;
            System.exit(res);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

千锋好程序大数据学院

2.4.18、二次排序

实现思路

将相同UID的数据发送到同一个reduce中，自定义分区

分区内按照UID和rate排序，UID相同的按照rate进行升序排序，自定义数据类型的排序

reduce中接收的数据已经是排好序的了，分组是比较key是否相同，比较两个条件uid和rate

这样的话不能分到同一个组中，如何把相同UID的数据分到一组，就需要我们自定义分组

自定义RateBean,并实现排序和序列化

```
public class RateBean implements WritableComparable<RateBean> {
    private String movie;
    private String rate;
    private String timeStamp;
    private String uid;
    public String getMovie() {
        return movie;
    }

    public void setMovie(String movie) {
        this.movie = movie;
    }

    public String getRate() {
        return rate;
    }

    public void setRate(String rate) {
        this.rate = rate;
    }

    public String getTimeStamp() {
        return timeStamp;
    }

    public void setTimeStamp(String timeStamp) {
        this.timeStamp = timeStamp;
    }

    public String getUid() {
        return uid;
    }

    public void setUid(String uid) {
        this.uid = uid;
    }

    @Override
    public void write(DataOutput out) throws IOException {
        out.writeUTF(movie);
        out.writeUTF(rate);
        out.writeUTF(timeStamp);
        out.writeUTF(uid);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        this.movie = in.readUTF();
        this.rate = in.readUTF();
        this.timeStamp = in.readUTF();
        this.uid = in.readUTF();
    }
```

千锋好程序大数据学院

```
    @Override
    public String toString() {
        return uid + "\t" + movie + "\t" + rate;
    }

    @Override
    public int compareTo(RateBean o) {
        if (this.uid.equals(o.uid)){
            return -this.rate.compareTo(o.rate);
        }else {
            return this.uid.compareTo(o.uid);
        }
    }
}
```

自定义分区

```
public class RatePartitioner extends Partitioner<RateBean, NullWritable> {
    @Override
    public int getPartition(RateBean rateBean, NullWritable nullWritable, int
numPartitions) {
        //怕出现负数
        return (rateBean.getUid().hashCode()&Integer.MAX_VALUE) % numPartitions;
    }
}
```

自定义分组

```
public class RateGroupingComparator extends WritableComparator {
    public RateGroupingComparator(){
        super(RateBean.class,true);
    }
    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        RateBean bean1 = (RateBean)a;
        RateBean bean2 = (RateBean)b;
        return bean1.getUid().compareTo(bean2.getUid());
    }
}
```

map

千锋好程序大数据学院

```
public class RateMapper extends Mapper<LongWritable, Text,RateBean,
NullWritable> {

    ObjectMapper objectMapper;
    @Override
    protected void setup(Context context) throws IOException,
InterruptedException {
        objectMapper = new ObjectMapper();
    }
    @Override
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        RateBean rateBean =
objectMapper.readValue(value.toString(),RateBean.class);
        context.write(rateBean,NullWritable.get());
    }
}
```

reduce

```
public class RateReduce extends Reducer<RateBean, NullWritable,RateBean,
NullWritable> {
    @Override
    protected void reduce(RateBean key, Iterable<NullWritable> values, Context
context) throws IOException, InterruptedException {
        int topN = context.getConfiguration().getInt("topN",3);
        int count = 0;
        for (NullWritable value:values) {
            context.write(key,NullWritable.get());
            count++;
            if (count==3){
                return;
            }
        }
    }
}
```

runner

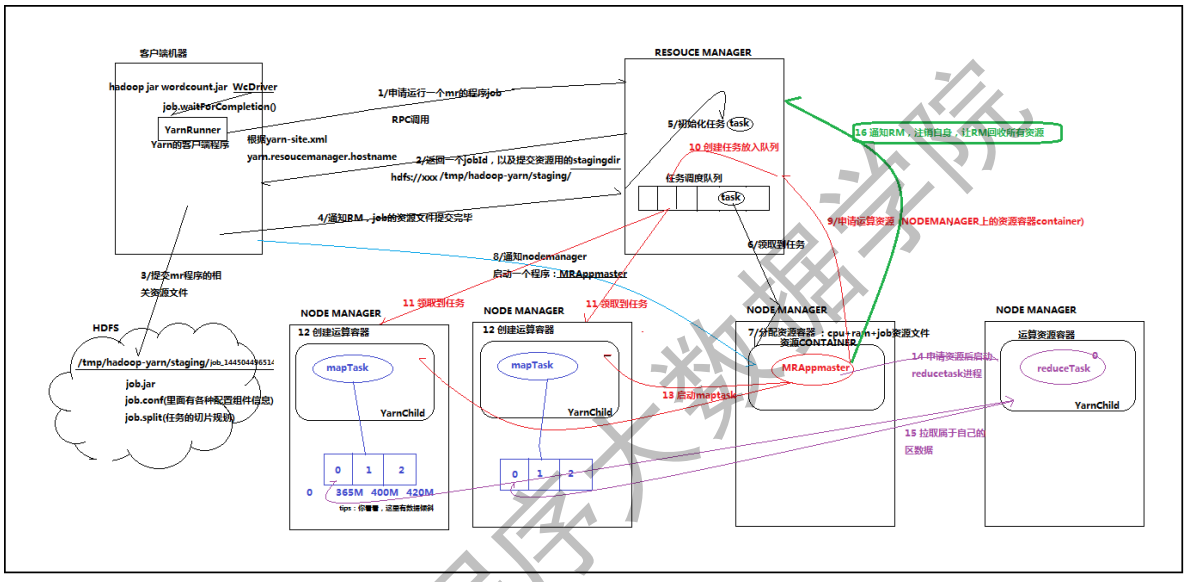
```
public class RateRunner  {
    public static void main(String[] args) {
        try {
            Configuration conf = new Configuration();
            conf.set("topN",args[0]);
            Job job = Job.getInstance(conf,"rate");
            job.setMapperClass(RateMapper.class);
            job.setReducerClass(RateReduce.class);

            job.setMapOutputKeyClass(RateBean.class);
            job.setMapOutputValueClass(NullWritable.class);
            job.setOutputKeyClass(RateBean.class);
            job.setOutputValueClass(NullWritable.class);
            job.setGroupingComparatorClass(RateGroupingComparator.class);
            job.setPartitionerClass(RatePartitioner.class);
            FileInputFormat.addInputPath(job,new Path(args[1]));
            Path out = new Path(args[2]);
```

千锋好程序大数据学院

```
            FileSystem fs = FileSystem.get(conf);
            if (fs.exists(out)){
                fs.delete(out,true);
            }
            FileOutputFormat.setOutputPath(job,out);
            int res = job.waitForCompletion(true)?0:1;
            System.exit(res);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2.4.19、job提交流程



2.4.20、MR的调优

1 资源相关参数

//以下参数是在用户自己的mr应用程序中配置在mapred-site.xml就可以生效

(1) mapreduce.map.memory.mb: 一个Map Task可使用的资源上限（单位:MB），默认为1024。如果Map Task实际使用的资源量超过该值，则会被强制杀死。

(2) mapreduce.reduce.memory.mb: 一个Reduce Task可使用的资源上限（单位:MB），默认为1024。如果Reduce Task实际使用的资源量超过该值，则会被强制杀死。

(3) mapreduce.map.cpu.vcores: 每个Map task可使用的最多cpu core数目, 默认值: 1

(4) mapreduce.reduce.cpu.vcores: 每个Reduce task可使用的最多cpu core数目, 默认值: 1

(5) mapreduce.map.java.opts: Map Task的JVM参数，你可以在此配置默认的java heap size等参数, e.g.

“-Xmx1024m -verbose:gc -Xloggc:/tmp/@taskid@.gc”（@taskid@会被Hadoop框架自动换为相应的taskid），默认值: “”

(6) mapreduce.reduce.java.opts: Reduce Task的JVM参数，你可以在此配置默认的java heap size等参数, e.g.

“-Xmx1024m -verbose:gc -Xloggc:/tmp/@taskid@.gc”, 默认值: “”

//shuffle性能优化的关键参数，应在yarn启动之前就配置好

千锋好程序大数据学院

(13) mapreduce.task.io.sort.mb 100 //shuffle的环形缓冲区大小，默认100m

(14) mapreduce.map.sort.spill.percent 0.8 //环形缓冲区溢出的阈值，默认80%

//应该在yarn启动之前就配置在服务器的yarn-site.xml配置文件中才能生效

(7) yarn.scheduler.minimum-allocation-mb 1024 给应用程序container分配的最小内存

(8) yarn.scheduler.maximum-allocation-mb 8192 给应用程序container分配的最大内存

(9) yarn.scheduler.minimum-allocation-vcores 1

(10)yarn.scheduler.maximum-allocation-vcores 32

(11)yarn.nodemanager.resource.memory-mb 8192 每台NodeManager最大可用内存

(12)yarn.nodemanager.resource.cpu-vcores 8 每台NodeManager最大可用cpu核数

2 容错相关参数

(1) mapreduce.map.maxattempts: 每个Map Task最大重试次数，一旦重试参数超过该值，则认为Map Task运行失败，默认值：4。

(2) mapreduce.reduce.maxattempts: 每个Reduce Task最大重试次数，一旦重试参数超过该值，则认为Map Task运行失败，默认值：4。

(3) mapreduce.map.failures.maxpercent: 当失败的Map Task失败比例超过该值为，整个作业则失败，默认值为0. 如果你的应用程序允许丢弃部分输入数据，则该该值设为一个大于0的值，比如5，表示如果有低于5%的Map Task失败（如果一个Map Task重试次数超过mapreduce.map.maxattempts，则认为这个Map Task失败，其对应的输入数据将不会产生任何结果），整个作业扔认为成功。

(4) mapreduce.reduce.failures.maxpercent: 当失败的Reduce Task失败比例超过该值为，整个作业则失败，默认值为0.

(5) mapreduce.task.timeout: Task超时时间，经常需要设置的一个参数，该参数表达的意思为：如果一个task在一定时间内没有任何进入，即不会读取新的数据，也没有输出数据，则认为该task处于block状态，可能是卡住了，也许永远会卡主，为了防止因为用户程序永远block住不退出，则强制设置了一个该超时时间（单位毫秒），默认是300000。如果你的程序对每条输入数据的处理时间过长（比如会访问数据库，通过网络拉取数据等），建议将该参数调大，该参数过小常出现的错误提示是“AttemptID:attempt_14267829456721_123456_m_000224_0 Timed out after 300 secsContainer killed by the ApplicationMaster.”。

十倍大数据学院