



Shuffle

过程：默认一个分区对应一个map任务，假如现在有10台机器，每台机器上面有3个分区，那么Mapper的时候就会同时又30个map任务在运行，他是怎么运行的呢？

根据代码的设置，job.SetInputFormatClass(TextInputFormat.class)这个是默认的可以不写，

使用InputFormat进行读取数据，他会把数据转换成《key，value》的格式，这个样子各个map任务就有数据输入了，然后调用用户自己编写的Mapper的map方法来处理这里面的数据，当map端不断处理数据的时候，他会把处理的结果写入到内存缓冲区中，这个内存缓冲区默认是100mb，mapreduce.task.io.sort.mb进行设置，当放入内存缓冲区的数据达到内存缓冲区的一定比例,mapreduce.map.sort.spill.percent默认是0.8的时候就会把现在的数据写入到磁盘中，他怎么写入磁盘的呢？他首先会把数据进行分区partition，这个partition的个数和reduce的个数是一样的，然后在每个分区中按照key进行排序，最后才写入到磁盘，随着map任务的执行，这个样子磁盘中会有许多的小文件，当到达一定的个数的时候mapreduce.task.io.sort.factor的时候，默认是10个的时候就会把这些文件进行合并成为一个文件，合并文件的时候也会进行按照分区进行合并，在分区内部进行排序。当map任务执行完成的时候，磁盘中就会有许多这个的文件，最后进行合并成一个大的文件（这个文件在本地的机器上面，这个时候可以设置压缩方式，或者combiner），合并也是会按照分区在分区内部进行排序，此时，这个map任务真正结束了。这个是map端的shuffle，接着会启动reduce端的shuffle。Reduce会通过网络copy自己要处理的数据，例如这个是时候一个reduce就会从前面的30个机器上copy数据，copy的时候也要进行合并，合并的时候也要进行排序，然后分区group，将相同key的value放在一起，最后交给自己编写的reduce去处理。

所以总的来说。自己设置的partitioner会作用在许多地方，并不是一处地方。



例如我们设置分区如下：

job.setOutputValueClass(KpiWritable.class);

job.setNumReduceTasks(2);

static class KpiPartitioner extends HashPartitioner<Text, KpiWritable>{

@Override

public int getPartition(Text key, KpiWritable value, int numReduceTasks) {

return (key.toString().length()==11)?0:1;

}

}

他就把数据分成两类，这个样子就实现了数据的分组。然后进行reduce，这个样子就相当于在组内进行reduce，实现了分组reduce。常见的场景例如分组排序。

例如我们可以自定义排序，其实就是它key implements WritableComparable<key的类型>实现它里面的排序方法，这个样子我们可以自定义key，例如key包含两个字段，我们可以自定义排序算法，来实现二次排序，如下：

static class NewK2 implements WritableComparable<NewK2>{

Long first;

Long second;

public NewK2(){}

public NewK2(long first, long second){

this.first = first;

this.second = second;

}

@Override

public void readFields(DataInput in) throws IOException {

this.first = in.readLong();

this.second = in.readLong();

}

@Override

public void write(DataOutput out) throws IOException {

out.writeLong(first);

out.writeLong(second);

}

/\*\*

\* 当k2进行排序时，会调用该方法.

\* 当第一列不同时，升序；当第一列相同时，第二列升序

\*/

@Override

public int compareTo(NewK2 o) {

final long minus = this.first - o.first;

if(minus !=0){

return (int)minus;

}

return (int)(this.second - o.second);

}

@Override

public int hashCode() {

return this.first.hashCode()+this.second.hashCode();

}

@Override

public boolean equals(Object obj) {

if(!(obj instanceof NewK2)){

return false;

}

NewK2 oK2 = (NewK2)obj;

return (this.first==oK2.first)&&(this.second==oK2.second);

}

}

刚才说到我们可以设置压缩方式,就是在map的最后阶段.可以通过两种方式来进行设置.一种是通过配置文件来设置,一种就是通过程序里面的conf.set来设置.











