**浙 江 理 工 大 学**

**实**

**践**

**工**

**作**

**日**

**志**

计算机科学与技术（人工智能）学院

学 院：\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

年 级：

计算机科学与技术

2021

专 业：

姓 名：

陈昊天

学 号：

2021329600006

实践时间： 2024 年 6 月 16 日至 2024 年 6 月 18 日

**实 践 工 作 日 志**

|  |  |
| --- | --- |
| |  | | --- | | 2024年06月16日 |   **需求：对每一个maptask的输出局部汇总**  统计过程中对每一个maptask的输出进行局部汇总，以减小网络传输量即采用Combiner功能。  增加一个WordcountCombiner类继承Reducer      **需求4：大量小文件的切片优化**  将输入的大量小文件合并成一个切片统一处理。  在WordcountDriver中增加如下代码    运行程序，并观察运行的切片个数为1    **流量汇总案例**  需求2：将统计结果按照手机归属地不同省份输出到不同文件中  （1）Mapreduce中会将map输出的kv对，按照相同key分组，然后分发给不同的reducetask。默认的分发规则为：根据key的hashcode%reducetask数来分发  （2）如果要按照我们自己的需求进行分组，则需要改写数据分发（分组）组件Partitioner  自定义一个CustomPartitioner继承抽象类：Partitioner  （3）在job驱动中，设置自定义partitioner： job.setPartitionerClass(CustomPartitioner.class)  在需求1的基础上，增加一个分区类    在驱动函数中增加自定义数据分区设置和reduce task设置 |
| 指导老师评语： |

**实 践 工 作 日 志**

|  |  |
| --- | --- |
| |  | | --- | | 2024年06月17日 |   需求3：将统计结果按照总流量倒序排序（全排序）  （1）把程序分两步走，第一步正常统计总流量，第二步再把结果进行排序  （2）context.write(总流量，手机号)  （3）FlowBean实现WritableComparable接口重写compareTo方法  （1）FlowBean对象在在需求1基础上增加了比较功能  package com.bigdata.mapreduce.sort; import java.io.DataInput; import java.io.DataOutput; import java.io.IOException; import org.apache.hadoop.io.WritableComparable;  public class FlowBean implements WritableComparable<FlowBean> {   private long upFlow;  private long downFlow;  private long sumFlow;   *// 反序列化时，需要反射调用空参构造函数，所以必须有* public FlowBean() {  super();  }   public FlowBean(long upFlow, long downFlow) {  super();  this.upFlow = upFlow;  this.downFlow = downFlow;  this.sumFlow = upFlow + downFlow;  }   public void set(long upFlow, long downFlow) {  this.upFlow = upFlow;  this.downFlow = downFlow;  this.sumFlow = upFlow + downFlow;  }   public long getSumFlow() {  return sumFlow;  }   public void setSumFlow(long sumFlow) {  this.sumFlow = sumFlow;  }   public long getUpFlow() {  return upFlow;  }   public void setUpFlow(long upFlow) {  this.upFlow = upFlow;  }   public long getDownFlow() {  return downFlow;  }   public void setDownFlow(long downFlow) {  this.downFlow = downFlow;  }   */\*\*  \* 序列化方法  \* @param out  \* @throws IOException  \*/* @Override  public void write(DataOutput out) throws IOException {  out.writeLong(upFlow);  out.writeLong(downFlow);  out.writeLong(sumFlow);  }   */\*\*  \* 反序列化方法 注意反序列化的顺序和序列化的顺序完全一致  \* @param in  \* @throws IOException  \*/* @Override  public void readFields(DataInput in) throws IOException {  upFlow = in.readLong();  downFlow = in.readLong();  sumFlow = in.readLong();  }   @Override  public String toString() {  return upFlow + "\t" + downFlow + "\t" + sumFlow;  }   @Override  public int compareTo(FlowBean o) {  *// 倒序排列，从大到小* return this.sumFlow > o.getSumFlow() ? -1 : 1;  } }  （2）编写mapper  package com.bigdata.mapreduce.sort; import java.io.IOException; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Mapper;  public class FlowCountSortMapper extends Mapper<LongWritable, Text, FlowBean, Text>{  FlowBean bean = new FlowBean();  Text v = new Text();   @Override  protected void map(LongWritable key, Text value, Context context)  throws IOException, InterruptedException {   *// 1 获取一行* String line = value.toString();   *// 2 截取* String[] fields = line.split("\\s+");   *// 3 封装对象* String phoneNbr = fields[0];  long upFlow = Long.*parseLong*(fields[1]);  long downFlow = Long.*parseLong*(fields[2]);   bean.set(upFlow, downFlow);  v.set(phoneNbr);   *// 4 输出* context.write(bean, v);  } }  （3）编写reducer  package com.bigdata.mapreduce.sort; import java.io.IOException; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Reducer;  public class FlowCountSortReducer extends Reducer<FlowBean, Text, Text, FlowBean>{   @Override  protected void reduce(FlowBean key, Iterable<Text> values, Context context)  throws IOException, InterruptedException {   *// 循环输出，避免总流量相同情况* for (Text text : values) {  context.write(text, key);  }  } }  （4）编写driver  package com.bigdata.mapreduce.sort; import java.io.IOException; import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Job; import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class FlowCountSortDriver {   public static void main(String[] args) throws ClassNotFoundException, IOException, InterruptedException {   *// 1 获取配置信息，或者job对象实例* Configuration configuration = new Configuration();  Job job = Job.*getInstance*(configuration);   *// 6 指定本程序的jar包所在的本地路径* job.setJarByClass(FlowCountSortDriver.class);   *// 2 指定本业务job要使用的mapper/Reducer业务类* job.setMapperClass(FlowCountSortMapper.class);  job.setReducerClass(FlowCountSortReducer.class);   *// 3 指定mapper输出数据的kv类型* job.setMapOutputKeyClass(FlowBean.class);  job.setMapOutputValueClass(Text.class);   *// 4 指定最终输出的数据的kv类型* job.setOutputKeyClass(Text.class);  job.setOutputValueClass(FlowBean.class);   *// 5 指定job的输入原始文件所在目录* FileInputFormat.*setInputPaths*(job, new Path(args[0]));  FileOutputFormat.*setOutputPath*(job, new Path(args[1]));   *// 7 将job中配置的相关参数，以及job所用的java类所在的jar包， 提交给yarn去运行* boolean result = job.waitForCompletion(true);  System.*exit*(result ? 0 : 1);  } } |
| 指导老师评语： |

**实 践 工 作 日 志**

|  |  |
| --- | --- |
| |  | | --- | | 2024年06月18日 |   **辅助排序和⼆次排序**  有如下订单数据  订单id 商品id 成交金额  0000001 Pdt\_01 222.8  0000001 Pdt\_06 25.8  0000002 Pdt\_03 522.8  0000002 Pdt\_04 122.4  0000002 Pdt\_05 722.4  0000003 Pdt\_01 222.8  0000003 Pdt\_02 33.8  现在需要求出每一个订单中最贵的商品。  （1）利用“订单id和成交金额”作为key，可以将map阶段读取到的所有订单数据按照id分区，按照金额排序，发送到reduce。  （2）在reduce端利用groupingcomparator将订单id相同的kv聚合成组，然后取第一个即是最大值。  代码实现  package com.bigdata.mapreduce.order; import java.io.DataInput; import java.io.DataOutput; import java.io.IOException; import org.apache.hadoop.io.WritableComparable;  public class OrderBean implements WritableComparable<OrderBean> {   private int order\_id; *// 订单id号* private double price; *// 价格* public OrderBean() {  super();  }   public OrderBean(int order\_id, double price) {  super();  this.order\_id = order\_id;  this.price = price;  }   @Override  public void write(DataOutput out) throws IOException {  out.writeInt(order\_id);  out.writeDouble(price);  }   @Override  public void readFields(DataInput in) throws IOException {  order\_id = in.readInt();  price = in.readDouble();  }   @Override  public String toString() {  return order\_id + "\t" + price;  }   public int getOrder\_id() {  return order\_id;  }   public void setOrder\_id(int order\_id) {  this.order\_id = order\_id;  }   public double getPrice() {  return price;  }   public void setPrice(double price) {  this.price = price;  }   *// 二次排序* @Override  public int compareTo(OrderBean o) {   int result;   if (order\_id > o.getOrder\_id()) {  result = 1;  } else if (order\_id < o.getOrder\_id()) {  result = -1;  } else {  *// 价格倒序排序* result = price > o.getPrice() ? -1 : 1;  }   return result;  } }  package com.bigdata.mapreduce.order; import java.io.IOException; import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.NullWritable; import org.apache.hadoop.mapreduce.Job; import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class OrderDriver {   public static void main(String[] args) throws Exception, IOException {   *// 1 获取配置信息* Configuration conf = new Configuration();  Job job = Job.*getInstance*(conf);   *// 2 设置jar包加载路径* job.setJarByClass(OrderDriver.class);   *// 3 加载map/reduce类* job.setMapperClass(OrderMapper.class);  job.setReducerClass(OrderReducer.class);   *// 4 设置map输出数据key和value类型* job.setMapOutputKeyClass(OrderBean.class);  job.setMapOutputValueClass(NullWritable.class);   *// 5 设置最终输出数据的key和value类型* job.setOutputKeyClass(OrderBean.class);  job.setOutputValueClass(NullWritable.class);   *// 6 设置输入数据和输出数据路径* FileInputFormat.*setInputPaths*(job, new Path(args[0]));  FileOutputFormat.*setOutputPath*(job, new Path(args[1]));   *// 10 设置reduce端的分组* job.setGroupingComparatorClass(OrderGroupingComparator.class);   *// 7 设置分区* job.setPartitionerClass(OrderPartitioner.class);   *// 8 设置reduce个数* job.setNumReduceTasks(3);   *// 9 提交* boolean result = job.waitForCompletion(true);  System.*exit*(result ? 0 : 1);  } }  package com.bigdata.mapreduce.order; import org.apache.hadoop.io.WritableComparable; import org.apache.hadoop.io.WritableComparator;  public class OrderGroupingComparator extends WritableComparator {   protected OrderGroupingComparator() {  super(OrderBean.class, true);  }   @SuppressWarnings("rawtypes")  @Override  public int compare(WritableComparable a, WritableComparable b) {   OrderBean aBean = (OrderBean) a;  OrderBean bBean = (OrderBean) b;   int result;  if (aBean.getOrder\_id() > bBean.getOrder\_id()) {  result = 1;  } else if (aBean.getOrder\_id() < bBean.getOrder\_id()) {  result = -1;  } else {  result = 0;  }   return result;  } }  package com.bigdata.mapreduce.order; import java.io.IOException; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.NullWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Mapper;  public class OrderMapper extends Mapper<LongWritable, Text, OrderBean, NullWritable> {  OrderBean k = new OrderBean();   @Override  protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {   *// 1 获取一行* String line = value.toString();   *// 2 截取* String[] fields = line.split("\t");   *// 3 封装对象* k.setOrder\_id(Integer.*parseInt*(fields[0]));  k.setPrice(Double.*parseDouble*(fields[2]));   *// 4 写出* context.write(k, NullWritable.*get*());  } }  package com.bigdata.mapreduce.order; import org.apache.hadoop.io.NullWritable; import org.apache.hadoop.mapreduce.Partitioner;  public class OrderPartitioner extends Partitioner<OrderBean, NullWritable> {   @Override  public int getPartition(OrderBean key, NullWritable value, int numReduceTasks) {   return (key.getOrder\_id() & Integer.*MAX\_VALUE*) % numReduceTasks;  } }  package com.bigdata.mapreduce.order; import java.io.IOException; import org.apache.hadoop.io.NullWritable; import org.apache.hadoop.mapreduce.Reducer;  public class OrderReducer extends Reducer<OrderBean, NullWritable, OrderBean, NullWritable> {   @Override  protected void reduce(OrderBean key, Iterable<NullWritable> values, Context context)  throws IOException, InterruptedException {   context.write(key, NullWritable.*get*());  } }  运行：  hadoop jar order-1.0-SNAPSHOT.jar com.bigdata.mapreduce.order.OrderDriver /order/input /order/output |
| 指导老师评语： |