

实验报告

**大数据处理技术**

报 告 人 姓 名 刘飞鸿

学 号 S2010W0748

学 科 专 业 电子信息

报告提交日期 2020年12月1日

# 实验 6 MapReduce 初级编程实践

## 一、实验目的

* 通过实验掌握基本的 MapReduce 编程方法；
* 掌握用 MapReduce 解决一些常见的数据处理问题，包括数据去重、数据排序和数据挖掘等。

## 二、实验平台

* + 操作系统：Linux（建议 Ubuntu16.04）
  + Hadoop 版本：2.7.1

## 三、实验步骤

#### （一）编程实现文件合并和去重操作

对于两个输入文件，即文件 A 和文件 B，请编写 MapReduce 程序，对两个文件进行合并，并剔除其中重复的内容，得到一个新的输出文件 C。下面是输入文件和输出文件的一个样例供参考。

输入文件 A 的样例如下：

|  |  |
| --- | --- |
| 20180101 | x |
| 20180102 | y |
| 20180103 | x |
| 20180104 | y |
| 20180105 | z |
| 20180106 | x |

输入文件 B 的样例如下：

|  |  |
| --- | --- |
| 20180101 | y |
| 20180102 | y |
| 20180103 | x |
| 20180104 | z |
| 20180105 | y |

根据输入文件 A 和 B 合并得到的输出文件 C 的样例如下：

|  |  |
| --- | --- |
| 20180101 | x |
| 20180101 | y |
| 20180102 | y |
| 20180103 | x |
| 20180104 | y |
| 20180104 | z |
| 20180105 | y |
| 20180105 | z |
| 20180106 | x |

#### （二）编写程序实现对输入文件的排序

现在有多个输入文件，每个文件中的每行内容均为一个整数。要求读取所有文件中的整数，进行升序排序后，输出到一个新的文件中，输出的数据格式为每行两个整数，第一个数字为第二个整数的排序位次，第二个整数为原待排列的整数。下面是输入文件和输出文件的一个样例供参考。

输入文件 1 的样例如下：

33

37

12

40

输入文件 2 的样例如下：

4

16

39

5

输入文件 3 的样例如下：

1

45

25

根据输入文件 1、2 和 3 得到的输出文件如下：

1 1

2 4

3 5

4 12

5 16

6 25

7 33

8 37

9 39

10 40

11 45

#### （三）对给定的表格进行信息挖掘

下面给出一个 child-parent 的表格，要求挖掘其中的父子辈关系，给出祖孙辈关系的表格。

输入文件内容如下：

child

parent

Steven

Lucy

Steven

Jack

Jone

Lucy

Jone

Jack

Lucy

Mary

Lucy

Frank

Jack

Alice

Jack

Jesse

David

Alice

David

Jesse

Philip

David

Philip

Alma

Mark

David

Mark

Alma

输出文件内容如下：

grandchild

grandparent

Steven

Alice

Steven

Jesse

Jone

Alice

Jone

Jesse

Steven

Mary

Steven

Frank

Jone

Mary

Jone

Frank

Philip

Alice

Philip

Jesse

Mark

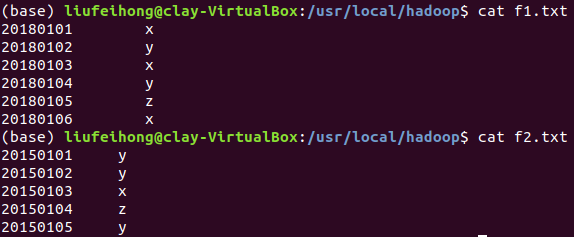
Alice

Mark

Jesse

## 四、实验报告

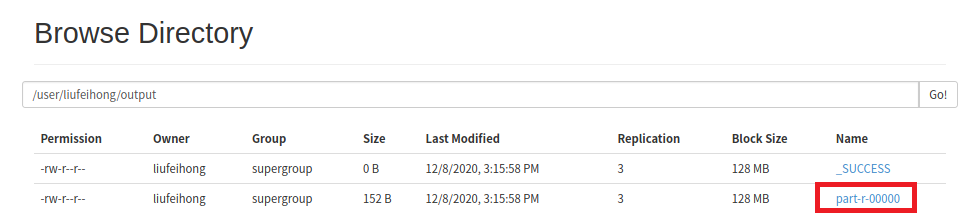
1. **编程实现文件合并和去重操作:**
2. 创建f1.txt和f2.txt两个文件，文件内容如下



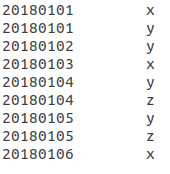
1. 在” /usr/local/hadoop”文件夹下创建input文件夹
2. 将文件上传到hdfs的/input文件夹下
3. 我们编写的源代码如下:

|  |
| --- |
| package map\_reduce;  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.Mapper;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class FileMerge {  public static class Map extends Mapper<Object, Text, Text, Text> {  private static Text text = new Text();  @Override  public void map(Object key, Text value, Context context) throws IOException, InterruptedException {  text = value;  context.write(text, new Text(""));  }  }  public static class Reduce extends Reducer<Text, Text, Text, Text> {  @Override  public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {  context.write(key, new Text(""));  }  }  public static void main(String[] args) throws Exception {  Configuration conf = new Configuration();  conf.set("fs.defaultFS", "hdfs://localhost:9000");  String[] otherArgs = new String[]{"input", "output"};  if (otherArgs.length != 2) {  System.err.println("Usage: Merge and duplicate removal <in> <out>");  System.exit(2);  }  Job job = Job.getInstance(conf, "Merge");  job.setJarByClass(FileMerge.class);  job.setMapperClass(Map.class);  job.setReducerClass(Reduce.class);  job.setOutputKeyClass(Text.class);  job.setOutputValueClass(Text.class);  FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  System.exit(job.waitForCompletion(true) ? 0 : 1);  }  } |

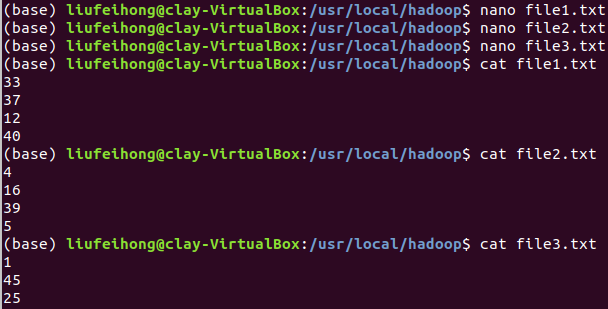
1. 得到的结果如下:



输出的part-r-00000文件的内容如下,与预想的结果一致！



1. **编写程序实现对输入文件的排序**
   1. 创建文件file1.txt、file2.txt和file3.txt



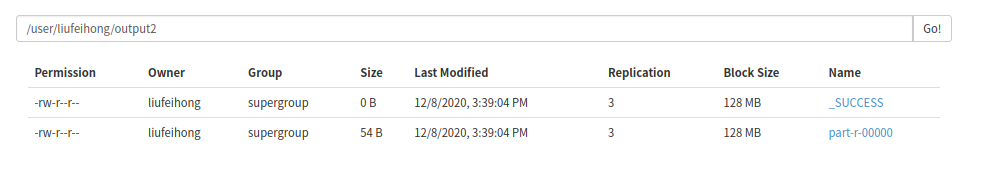
* 1. 在HDFS建立input2文件夹并上传



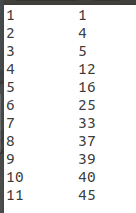
* 1. 编写的代码如下

|  |
| --- |
| package map\_reduce;  import java.io.IOException;  import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class ContentSort {  public static class Map extends Mapper<Object, Text, IntWritable, IntWritable> {  private static IntWritable data = new IntWritable();  @Override  public void map(Object key, Text value, Context context) throws IOException, InterruptedException {  String line = value.toString();  data.set(Integer.parseInt(line));  context.write(data, new IntWritable(1));  }  }  public static class Reduce extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {  private static IntWritable linenum = new IntWritable(1);  @Override  public void reduce(IntWritable key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {  for (IntWritable val : values) {  context.write(linenum, key);  linenum = new IntWritable(linenum.get() + 1);  }  }  }  public static void main(String[] args) throws Exception {  Configuration conf = new Configuration();  conf.set("fs.defaultFS", "hdfs://localhost:9000");  String[] otherArgs = new String[]{"input2", "output2"};  if (otherArgs.length != 2) {  System.err.println("Usage: ContentSort <in> <out>");  System.exit(2);  }  Job job = Job.getInstance(conf, "ContentSort");  job.setJarByClass(ContentSort.class);  job.setMapperClass(Map.class);  job.setReducerClass(Reduce.class);  job.setOutputKeyClass(IntWritable.class);  job.setOutputValueClass(IntWritable.class);  FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  System.exit(job.waitForCompletion(true) ? 0 : 1);  }  } |

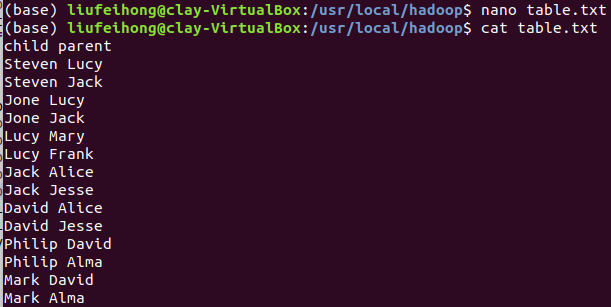
* 1. 得到的输出结果如下:



打开part-r-00000文件的内容如下,与预想的结果一致！



1. **对给定的表格进行信息挖掘**
   1. 创建文件table.txt



* 1. 建立文件夹并将文件上传

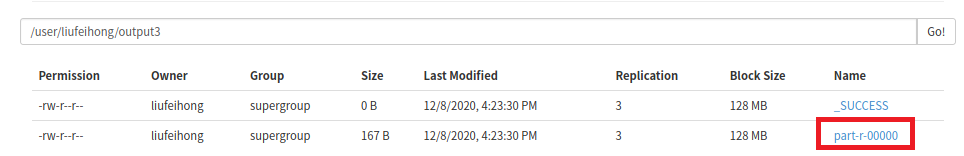




* 1. 书写的代码如下

|  |
| --- |
| package map\_reduce;  import java.io.IOException;  import java.util.ArrayList;  import java.util.List;  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.Mapper;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class STJoin {  public static int time = 0;  public static class Map extends Mapper<Object, Text, Text, Text> {  @Override  public void map(Object key, Text value, Context context) throws IOException, InterruptedException {  String line = value.toString();  String[] childAndParent = line.split(" ");  List<String> list = new ArrayList<>(2);  for (String childOrParent : childAndParent) {  if (!"".equals(childOrParent)) {  list.add(childOrParent);  }  }  if (!"child".equals(list.get(0))) {  String childName = list.get(0);  String parentName = list.get(1);  String relationType = "1";  context.write(new Text(parentName), new Text(relationType + "+"  + childName + "+" + parentName));  relationType = "2";  context.write(new Text(childName), new Text(relationType + "+"  + childName + "+" + parentName));  }  }  }  public static class Reduce extends Reducer<Text, Text, Text, Text> {  @Override  public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {  if (time == 0) {  context.write(new Text("grand\_child"), new Text("grand\_parent"));  time++;  }  List<String> grandChild = new ArrayList<>();  List<String> grandParent = new ArrayList<>();  for (Text text : values) {  String s = text.toString();  String[] relation = s.split("\\+");  String relationType = relation[0];  String childName = relation[1];  String parentName = relation[2];  if ("1".equals(relationType)) {  grandChild.add(childName);  } else {  grandParent.add(parentName);  }  }  int grandParentNum = grandParent.size();  int grandChildNum = grandChild.size();  if (grandParentNum != 0 && grandChildNum != 0) {  for (int m = 0; m < grandChildNum; m++) {  for (int n = 0; n < grandParentNum; n++) {  context.write(new Text(grandChild.get(m)), new Text(  grandParent.get(n)));  }  }  }  }  }  public static void main(String[] args) throws Exception {  Configuration conf = new Configuration();  conf.set("fs.defaultFS", "hdfs://localhost:9000");  String[] otherArgs = new String[]{"input3", "output3"};  if (otherArgs.length != 2) {  System.err.println("Usage: Single Table Join <in> <out>");  System.exit(2);  }  Job job = Job.getInstance(conf, "Single table Join ");  job.setJarByClass(STJoin.class);  job.setMapperClass(Map.class);  job.setReducerClass(Reduce.class);  job.setOutputKeyClass(Text.class);  job.setOutputValueClass(Text.class);  FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  System.exit(job.waitForCompletion(true) ? 0 : 1);  }  } |

* 1. 输出的结果如下



打开part-r-00000文件的内容如下,与预想的结果一致！

