**武汉工商学院**

**实验报告**

系：物联网系 专业：物联网工程 班级：20物联本一 课程名称：云计算技术

|  |  |  |  |
| --- | --- | --- | --- |
| 实验名称 | 实验三 MapReduce | | |
| 学号 | 20423030 | 姓名 | 谢康 |
| 指导老师 | 王勇 |  |  |
| 实验目的 | 1. 通过实验掌握基本的MapReduce编程方法。 2. 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。 3. 通过操作MapReduce的实验，模仿实验内容，深入理解MapReduce的过程，和shuffle的具体意义。 | | |
| **一、实验环境**  操作系统:Ubuntu-16.04、Hadoop版本:2.6.0、JDK版本:1.8、IDE:Eclipse  **二、实验内容及步骤**  **任务1：编程实现文件合并和去重操作：**   1. **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。** 2. **输入文件f1.txt的样例如下：**   20150101 x  20150102 y  20150103 x  20150104 y  20150105 z  20150106 x   1. **输入文件f2.txt的样例如下：**   20150101 y  20150102 y  20150103 x  20150104 z  20150105 y   1. **根据输入文件f1和f2合并得到的输出文件的样例如下**：   20150101 x  20150101 y  20150102 y  20150103 x  20150104 y  20150104 z  20150105 y  20150105 z  20150106 x   1. **实验过程:** 2. 创建文件f1.txt和f2.txt（将上面样例内容复制进去）   http://upload-images.jianshu.io/upload_images/3245878-7706918c8f4bb046.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240   1. 在HDFS建立input文件夹（执行这步之前要开启hadoop相关进程）   http://upload-images.jianshu.io/upload_images/3245878-d8b89509aa826bd1.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240  加参数 -p   1. 上传样例到HDFS中的input文件夹   http://upload-images.jianshu.io/upload_images/3245878-364524429a6dfdc2.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240   1. 接着打开eclipse创建MapReduce项目（具体实现步骤参考HDFS实验）   **实验代码:**  **package** com.Merge;  **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;  /\*\*  \* Created by Xie kang on 2023/4/14.  \*/  **public** **class** Merge {  /\*\*  \*对A、B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  \*/  //重载map函数，直接将输入中的value复制到输出数据的key上  **public** **static** **class** Map **extends** Mapper<Object, Text, Text, Text> {  **private** **static** Text *text* = **new** Text();  **public** **void** map(Object key, Text value, Context content) **throws** IOException, InterruptedException {  *text* = value;  content.write(*text*, **new** Text(""));  }  }  //重载reduce函数，直接将输入中的key复制到输出数据的key上  **public** **static** **class** Reduce **extends** Reducer<Text, Text, Text, Text> {  **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"};//设置输入参数  **if** (otherArgs.length != 2) {  System.***err***.println("Usage:Merge and duplicate removal <in> <out>");  System.*exit*(2);  }  Job job = Job.*getInstance*(conf,"Merge and duplicate removal");  job.setJarByClass(Merge.**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);  }  }  **测试结果：**    **拓展实验：**  WordCount.java  **package** com.WordCount;  **import** java.io.IOException;  **import** java.io.IOException;  **import** java.util.StringTokenizer;  **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;  **import** org.apache.hadoop.util.GenericOptionsParser;  /\*\*  \* Created by Xie kang on 2023/4/14.  \*/  **public** **class** WordCount {  **public** **static** **class** TokenizerMapper  **extends** Mapper<Object, Text, Text, IntWritable>{  **private** **final** **static** IntWritable ***one*** = **new** IntWritable(1);  **private** Text word = **new** Text();  /\*\*  \* Mapper中的map方法：  \* void map(K1 key, V1 value, Context context)  \* 映射一个单个的输入k/v对到一个中间的k/v对  \* 输出对不需要和输入对是相同的类型，输入对可以映射到0个或多个输出对。  \* Context：收集Mapper输出的<k,v>对。  \* Context的write(k, v)方法:增加一个(k,v)对到context  \* 程序员主要编写Map和Reduce函数.这个Map函数使用StringTokenizer函数对字符串进行分隔,通过write方法把单词存入word中  \* write方法存入(单词,1)这样的二元组到context中  \*/  **public** **void** map(Object key, Text value, Context context  ) **throws** IOException, InterruptedException {  StringTokenizer itr = **new** StringTokenizer(value.toString());  **while** (itr.hasMoreTokens()) {  word.set(itr.nextToken());  context.write(word, ***one***);  }  }  }  **public** **static** **class** IntSumReducer  **extends** Reducer<Text,IntWritable,Text,IntWritable> {  **private** IntWritable result = **new** IntWritable();  /\*\*  \* Reducer类中的reduce方法：  \* void reduce(Text key, Iterable<IntWritable> values, Context context)  \* 中k/v来自于map函数中的context,可能经过了进一步处理(combiner),同样通过context输出  \*/  **public** **void** reduce(Text key, Iterable<IntWritable> values,  Context context  ) **throws** IOException, InterruptedException {  **int** sum = 0;  **for** (IntWritable val : values) {  sum += val.get();  }  result.set(sum);  context.write(key, result);  }  }    **public** **static** **void** main(String[] args) **throws** Exception {  /\*\*  \* Configuration：map/reduce的j配置类，向hadoop框架描述map-reduce执行的工作  \*/  Configuration conf = **new** Configuration();  conf.set("fs.defaultFS", "hdfs://localhost:9000");  String[] otherArgs = **new** String[]{"wordcount\_input","wordcount\_output"};//设置输入参数  **if** (otherArgs.length != 2) {  System.***err***.println("Usage:Merge and duplicate removal <in> <out>");  System.*exit*(2);  }  Job job = **new** ~~Job~~(conf, "word count");//设置一个用于定义的job名称  job.setJarByClass(WordCount.**class**);  job.setMapperClass(TokenizerMapper.**class**);//为job设置Mapper类  job.setCombinerClass(IntSumReducer.**class**);//为job设置Combiner类  job.setReducerClass(IntSumReducer.**class**);//为job设置Reducer类  job.setOutputKeyClass(Text.**class**);//为job的输出数据设置Key类  job.setOutputValueClass(IntWritable.**class**);//为job输出设置value类  FileInputFormat.*addInputPath*(job, **new** Path(otherArgs[0]));//为job设置输入路径  FileOutputFormat.*setOutputPath*(job, **new** Path(otherArgs[1]));//为job设置输出路径  System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);//运行job  }  }  **测试结果：**    **三、实验小结**  在文件合并操作中，我们需要重载reduce函数，它将所有输入键值对的键设置为同一个常数值，这样，所有的输入记录都会被分配到同一个Reduce中，我们可以将所有相同键的值进行拼接，从而实现多个文件的合并操作。  在去重操作中，我们需要重载map函数，它将输入的文件进行行读取操作，然后将每一行作为键值对的键存入Hadoop Map中。由于键的不可重复性，相同的行只会被存储一次。最后，在输出结果时，我们只需要将键对应的值再次合并即可。  通过本次实验，我对Hadoop分布式系统有了更深入的了解，并掌握了实现文件合并和去重操作的基本方法，这对我今后的工作和学习都将有很大的帮助。 | | | |