Hadoop实战1 WordCount

> 词频统计是hadoop实战中相当于Helloword项目的一个作业，它能够很清楚的表现出整个mapreduce的一个流程和每个流程的作用，对我们来说十分有用。

题目：

> 数据集为：
>
> Hello World Bye World
>
> Hello Hadoop Bye Hadoop
>
> Bye Hadoop Hello Hadoop
>
> 请设置一个mapreduce程序进行词频统计，即计算每个单词出现的个数

理解map和reduce

要写一个mapreduce程序，首先要实现一个map函数和reduce函数。我们看看map的方法：

```
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text,
Text, LongWritable>.Context context)
 /**
     * KEYIN    即K1      表示每一行的起始位置（偏移量offset）
     * VALUEIN  即v1      表示每一行的文本内容
     * KEYOUT   即k2      表示每一行中的每个单词
     * VALUEOUT 即v2      表示每一行中的每个单词的出现次数，固定值1
**/
```

这里有三个参数，前面两个LongWritable key, Text value就是输入的key和value，第三个参数Context context这是可以记录输入的key和value，例如：

> context.write(new Text(word), new LongWritable(1));

接下来是reduce方法：

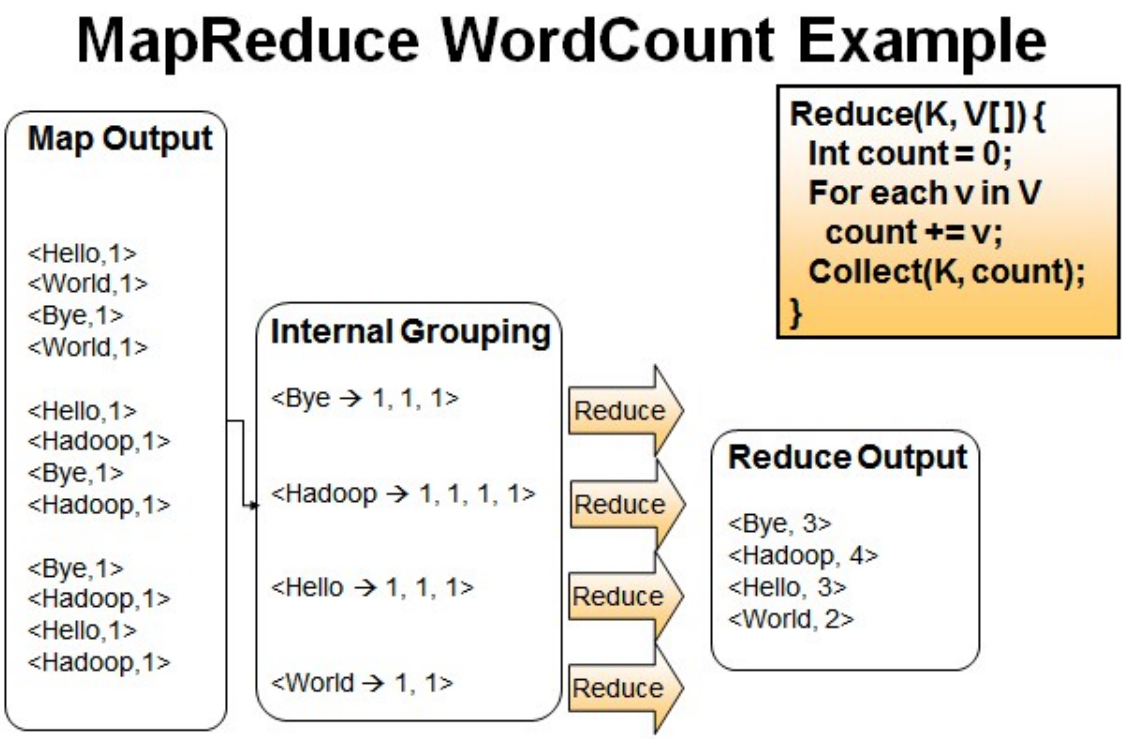
```
protected void reduce(Text k2, Iterable<LongWritable> v2s,Reducer<Text,
LongWritable, Text, LongWritable>.Context context)
/**
     * KEYIN    即k2      表示每一行中的每个单词
     * VALUEIN  即v2      表示每一行中每个单词出现次数，固定值1
     * KEYOUT   即k3      表示整个文件中的不同单词
     * VALUEOUT 即v3      表示整个文件中的不同单词的出现总次数
 **/
```

reduce函数的输入也是一个key/value的形式，不过它的value是一个迭代器的形式Iterable\ values，也就是说reduce的输入是一个key对应一组的值的value，reduce也有context和map的context作用一致。

分配任务：

> 一个好的mapreduce程序最重要的是先理清楚每个部分要做什么，再进行一个程序的编写。
>
> map程序：并行读取文本，对读取的单词进行map操作，每个词都以<key,value>形式生成。
>
> reduce程序：map的结果进行排序，合并，最后得出词频。
>
> main程序：定义job类设置各种参数

程序流程如图：



代码：

```
package org.apache.hadoop.examples;

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;
/**
 *
 * 描述：WordCount explains by York
  * @author Hadoop Dev Group
 */
```

```
publicclass WordCount {
    /**
     * 建立Mapper类TokenizerMapper继承自泛型类Mapper
     * Mapper类:实现了Map功能基类
     * Mapper接口:
     * WritableComparable接口：实现WritableComparable的类可以相互比较。所有被用作key的类应该实现此接口。
     * Reporter 则可用于报告整个应用的运行进度，本例中未使用。
     *
     */
  publicstaticclass TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{
        /**
         * IntWritable, Text 均是 Hadoop 中实现的用于封装 Java 数据类型的类，这些类实现了WritableComparable接口，
         * 都能够被串行化从而便于在分布式环境中进行数据交换，你可以将它们分别视为int,String的替代品。
     * 声明one常量和word用于存放单词的变量
         */
    privatefinalstatic 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
     */
    publicvoid 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);
      }
    }
  }

  publicstaticclass 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
         */
    publicvoid 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);
    }
  }

  publicstaticvoid 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();
    String[] otherArgs =new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length !=2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job =new Job(conf, "word count");    //设置一个用户定义的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
  }
}
```

代码每一步运行结果如下：

Input:

```
Hello World Bye World

Hello Hadoop Bye Hadoop

Bye Hadoop Hello Hadoop
```

Map:

```
<Hello,1>

<World,1>

<Bye,1>

<World,1>

<Hello,1>

<Hadoop,1>

<Bye,1>
```

```
<Hadoop,1>

<Bye,1>

<Hadoop,1>

<Hello,1>

<Hadoop,1>
```

Sort:

```
<Bye,1>

<Bye,1>

<Bye,1>

<Hadoop,1>

<Hadoop,1>

<Hadoop,1>

<Hadoop,1>

<Hello,1>

<Hello,1>

<Hello,1>

<World,1>

<World,1>
```

Combine:

```
<Bye,1,1,1>

<Hadoop,1,1,1,1>

<Hello,1,1,1>

<World,1,1>
```

Reduce:

```
<Bye,3>

<Hadoop,4>

<Hello,3>

<World,2>
```

MergeSort的过程

```
<Hello,1><World,1><Bye,1><World,1><Hello,1><Hadoop,1><Bye,1><Hadoop,1><Bye,1>
<Hadoop,1><Hello,1><Hadoop,1>
```

MergeSort:

> - <Hello,1><World,1><Bye,1><World,1><Hello,1><Hadoop,1> | <Bye,1><Hadoop,1><Bye,1><Hadoop,1><Hello,1><Hadoop,1>
> - <Hello,1><World,1><Bye,1> || <World,1><Hello,1><Hadoop,1> | <Bye,1><Hadoop,1><Bye,1> || <Hadoop,1><Hello,1><Hadoop,1>
> - <Hello,1><World,1> ||| <Bye,1> || <World,1><Hello,1> ||| <Hadoop,1> | <Bye,1><Hadoop,1> ||| <Bye,1> || <Hadoop,1><Hello,1> ||| <Hadoop,1>
> - MergeArray 结果：<Hello,1><World,1> ||| <Bye,1> || <Hello,1><World,1> ||| <Hadoop,1> | <Bye,1><Hadoop,1> ||| <Bye,1> || <Hadoop,1><Hello,1> ||| <Hadoop,1> 在|||这一层级
> - MergeArray 结果：<Bye,1><Hello,1><World,1> || <Hadoop,1><Hello,1><World,1> | <Bye,1><Bye,1><Hadoop,1> || <Hadoop,1><Hadoop,1><Hello,1> 在||这一层级
> - MergeArray 结 果：<Bye,1><Hadoop,1><Hello,1><World,1><Hello,1><World,1> | <Bye,1><Bye,1><Hadoop,1><Hadoop,1><Hello,1><Hadoop,1> 在|这一层级
> - MergeArray结 果：<Bye,1><Bye,1><Bye,1><Hadoop,1><Hadoop,1><Hadoop,1><Hadoop,1><Hello,1><Hello,1><Hello,1><World,1><World,1> 排序完成