院系：数据科学与计算机学院　　专业：计算机科学与技术
姓名：郑康泽　　学号：17341213

云计算项目实践

Hadoop实践

一. 实验要求

1. 理解MapReduce程序的运行原理。

2. 完成InvertedIndex程序，并介绍你的实现过程。

二. 实验过程

1. MapReduce介绍
    1. 概念：
       MapReduce是一个基于集群的计算平台，是一个简化分布式编程的计算框架，是一个将分布式计算抽象成Map和Reduce两个阶段的编程模型。
    2. 工作原理：
        1. Map阶段：
            1. 读取HDFS中的文件，将每一行解析成一个键值对(key: value)，对于每一个键值对(key: value)调用一次map函数；
            2. 通过重写map函数，对上一步产生的键值对(key: value)进行处理，产生新的键值对(key': value')并输出；
            3. 对输出的键值对(key', value')进行分区；
            4. 对于每一个分区的数据，按照key'进行排序、分组，相同key'的value'放到一个集合里。

2. Reduce阶段：
    1. 多个Map任务的输出，按照不同的分区，通过网络复制到不同的Reduce节点上；
    2. 对多个Map的输出进行合并、排序；
    3. 重写reduce函数，对输入的键值对(key: value)处理，产生新的键值对(key': value')并输出；
    4. 将reduce函数的输出保存到文件中。

2. 完成InvertedIndex程序
    1. 什么是InvertedIndex？
    键值对(key: value)格式如下：
    (单词: <出现该单词的文件1的文件名, 出现在文件1中的次数>, ... , <出现该单词的文件n的文件名, 出现在文件n中的次数>)
    2. 实现方法：
        1. Mapper函数：
        输入的键是文本中该行的偏移量，值是该行的数据即该行内容，先将该行内容去掉标点符号，然后通过空格分割成一个个单词，因为还要记录单词所在文件的文件名，所以要获得文件名。可以通过上下文context获得，获取方法如下：

        ```
        FileSplit info = (FileSplit) context.getInputSplit();
        String filename = info.getPath().getName();
        ```

        所以Mapper函数的输出为(单词: 文件名)键值对。因为每行的每个单词都会形成一个键值对输出，所以没必要记录出现次数，都是默认为一次。
        2. Reducer函数：
        Reducer函数的输入是(单词: 文件名1, ... , 文件名n)这样一对键值对（即只有一个单词），在这个函数中我们要统计该单词在哪个文件出现过以及出现的次数。我们可以用数据结构Map帮助统计，遍历输入的键值对的值，对于每一个文件名，首先检查是否在Map中，在则将该文件名对应的数字加一，不在则将该文件名添加进Map中，并将对应的数字置一，显然这个数字就是该单词在该文件名对应的文件中出现的次数。统计完后，将这个Map中的数据转化为一定格式的String字符串，然后将(单词: 一定格式的字符串)作为键值对输出即可。

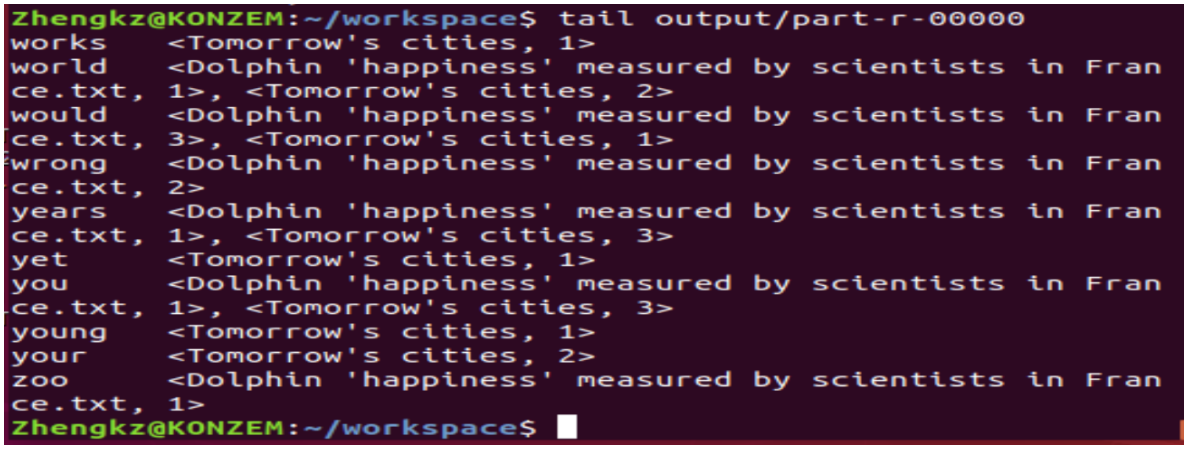
3. 相关代码：

```
// 输入类型是(LongWritable: Text)，输出类型是(Text: Text)
public static class WordCountMapper extends Mapper<LongWritable,
Text, Text, Text> {
    // 统计词频时，需要去掉标点符号等符号，此处定义表达式
    private String pattern = "[^a-zA-Z0-9-]";

    @Override
    protected void map(LongWritable key, Text value, Context
context) throws IOException, InterruptedException {
        // 获取文件名
        FileSplit info = (FileSplit) context.getInputSplit();
        String filename = info.getPath().getName();
        // 将每一行转化为一个String
        String line = value.toString();
        // 将标点符号等字符用空格替换，这样仅剩单词
        line = line.replaceAll(pattern, " ");
        // 将String划分为一个个的单词
        String[] words = line.split("\\s+");
        // 输出
        for (String word : words)
            if (word.length() > 0)
                context.write(new Text(word), new Text(filename));
    }
}
```

```
// 输入类型是(Text: Iterable<Text>)，输出类型是(Text, Text)
public static class WordCountReducer extends Reducer<Text, Text,
Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException {
        String final_value = "";
        Map<String, Integer> map = new HashMap<String, Integer>();
        // 统计
        for (Text value : values) {
            String filename = value.toString();
            int cnt = map.containsKey(filename)?
map.get(filename):0;
            // 添加或加一
            map.put(filename, cnt+1);
        }
        // 将Map中的数据转为一定格式的String
        for (Map.Entry<String, Integer> entry: map.entrySet())
            final_value += "<" + entry.getKey() + ", " +
entry.getValue() + ">, ";
        // 去掉末尾多余的", "
        final_value = final_value.substring(0,
final_value.length()-2);
        // 输出
        context.write(key, new Text(final_value));
    }
}
```

三. 实验结果


Zhengkz@KONZEM:~/workspace$ tail output/part-r-00000
works <Tomorrow's cities, 1>
world <Dolphin 'happiness' measured by scientists in Fran
ce.txt, 1>, <Tomorrow's cities, 2>
would <Dolphin 'happiness' measured by scientists in Fran
ce.txt, 3>, <Tomorrow's cities, 1>
wrong <Dolphin 'happiness' measured by scientists in Fran
ce.txt, 2>
years <Dolphin 'happiness' measured by scientists in Fran
ce.txt, 1>, <Tomorrow's cities, 3>
yet <Tomorrow's cities, 1>
you <Dolphin 'happiness' measured by scientists in Fran
ce.txt, 1>, <Tomorrow's cities, 3>
young <Tomorrow's cities, 1>
your <Tomorrow's cities, 2>
zoo <Dolphin 'happiness' measured by scientists in Fran
ce.txt, 1>
Zhengkz@KONZEM:~/workspace$