大数据综合处理实验

*Lab3 文档倒排索引及词频统计*

**30号小组**

小组成员：

|  |  |  |
| --- | --- | --- |
| 学号 | 姓名 | 邮箱 |
| 161220018 | 陈昕元 | 161220018@smail.nju.edu.cn |
| 161220017 | 陈翔 |  |
| 161220019 | 陈亚栋 |  |

# 1. 实验背景：问题和数据集介绍

倒排索引是被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射，是目前几乎所有支持全文索引的搜索引擎都需要依赖的一个数据结构。我们本次需要对一系列武侠小说进行倒排索引，最后输出这一系列武侠小说中出现的单词、该单词在哪本书中出现多少次、该单词的平均出现次数。

# 2. 实验环境

* + 操作系统版本：Ubuntu 18.04
  + Java版本：1.7.0
  + Hadoop版本：2.7.1

# 3. 使用算法及实现

### （1）分布式倒排索引算法介绍

倒排索引是文档检索系统中最常用的一种数据结构，它根据关键词反过来查找包含该词的文档及文档中出现该词的词频，而不是通过文档来查找内部存在的单词。因此我们称其为倒排索引(Inverted Index)。分布式倒排索引算法将涉及这几个过程：Map过程，Combine过程，Partition过程，Reduce过程。下面将分别介绍这几个过程。

### （2）Map过程实现

当输入目录中需要处理的文档上传到hdfs时，默认的TextInputFormat类会对输入的文件进行处理，得到文件中每一行的偏移量和这一行内容的键值对<内容，偏移量>作为map函数的输入。在重载map函数时，我们需要考虑设计出适合当前问题的key和value的值，传递给后续的reduce等过程，从而得到正确的结果。在map过程中，我们需要获取单词、文档名、当前词频三个信息，并传递给后续过程处理。而我们map可以向后传递的<key, value>只有两个值，因此现在我们有两个选择

1）将单词与文档名合并作为key.

对于这种情况，我们在后续的reduce过程前，会将“单词+文档名”作为key将数据进行Partition，因此我们需要重写Partition函数，仅使用当前key中的“单词”作为Partition的依据。此外，value为一个数值类型，便于我们在Combiner过程中合并value值。

2）只将单词作为key

对于这种情况，我们不需要改写Partition函数，但不易进行Combine操作，需要IO次数较多。对于每个单词，我们在map阶段将单词作为key，出现的文档名作为value，每个单词写入content中。在reduce阶段，对每个单词我们需要建立一个文档名为键的哈希表，将以单词为key的输入全部读取完，才能确认该单词在各个文档出现的总次数。

综合考虑上述两种方案的代价，我们采用方案1），将map阶段的输出设为<单词＋URL，词频>，这样便于我们实现combiner、reduce这些后续操作。

### （3）Combiner过程实现

在Combiner过程中，我们需要将具有相同键的输入的值合并。Combiner其实可以看作是一个Reduce过程，它接受的是Map阶段的输出，Combiner类是继承的Reduce类，并重载了其中的reduce函数。

在该阶段中，我们不改变键的内容，只需要将相同键的输入的值合并，之后输出即可。这个过程可以用如下简单的一段代码表示：

1. **int** sum = 0;
3. **for**(IntWritable value : values)
4. sum += value.get();
6. Value\_int.set(sum);
7. context.write(key, Value\_int);

### （4）Partition过程实现

在MapReduce过程中，框架会自动帮我们将Map阶段的输出进行划分，将具有相同键属性的内容划分到同一个节点上对结果进行Reduce。在此次倒排索引的实验中，我们Map阶段输出的键为“单词+文档名”，因此我们需要对Partition过程进行重载，手动指定使用“单词”为Partition的依据。这是因为如果使用“单词+文档名”作为Key进行划分，有可能同一个单词被划分到不同的节点上进行Reduce，这与我们的预期不符。

### （5）Reduce过程实现

在本次实验中，Reduce的输入为<“单词+文件名”, “词频”>，其中经过Combiner过程的词频应对应于该单词在这个文件中的总出现次数。在Reduce阶段，输入值应已按照键属性排序完毕，我们统计单词的平均出现次数，需要在遍历完这个单词的所有相关输入后，再进行计算。因此，我们需要记住正在处理的单词，在遇到下一个单词时，计算这个单词的平均词频，并输出到content中。这一过程在代码中体现如下，其中appear\_num为当前单词已出现次数、file\_num为当前单词出现在文件总数、ValueText为[“文件名:出现次数;”]\*n形式的字符串：

1. **if**(curWord != **null** && (word.equals(curWord) == **false**)){
2. Key\_word.set(curWord);
3. **float** average = (**float**) appear\_num / file\_num ;
4. textValue.set(average+","+ValueText.toString());
5. context.write(Key\_word, textValue);
7. file\_num = 0;
8. appear\_num = 0;
9. ValueText.delete(0, ValueText.length());
10. }

### （6）项目改进及扩展功能实现

本次项目额外实现了无关词语过滤的功能。

根据多次的实验观察，我们找出了一些无意义的字和词，比如：的、啊、一、网址等单词，这些过滤词都统一在一个配置文件(ignore.txt)中,每个过滤词占一行。管理过滤词的类所在的源文件是:Ignore.java，程序运行中，声明一个Ignore的类，这个类在初始化时便会加载配置文件中的信息，并且它还提供了contains接口，用来判断一个String 是否在过滤词集合中，具体判断的过程在map函数中。在map处理每一组键值对时，都将“键”交给ignore判断，如果是被忽略的词，那么接下来对这个词的处理将会被跳过。

将配置文件随着java代码一起打包,如果配置文件要随着java的字节码一起被打包,需要把它放到src/main/resources目录下,并且为了使这个包运行时依然能找到配置文件,其中重要的代码如下:

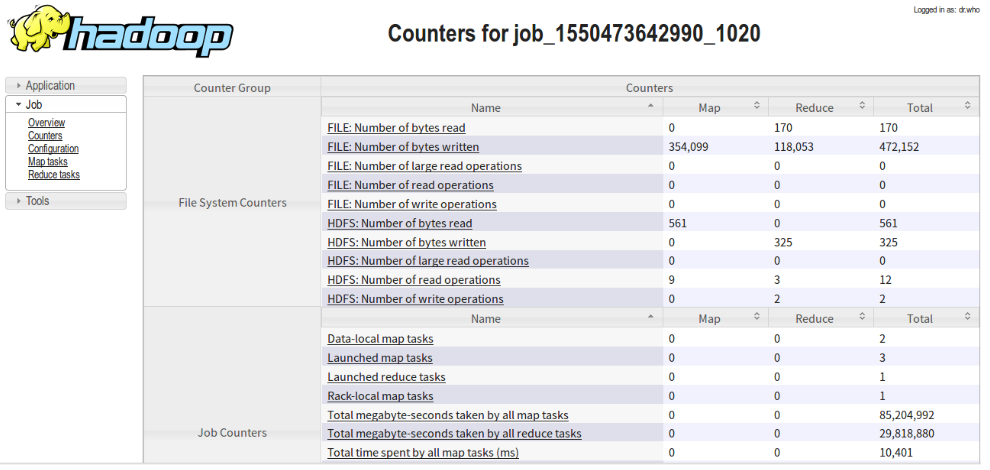
s = this.getClass().getClassLoader().getResourceAsStream("ignore.txt");

这个代码会将src/main/resources/ignore.txt载入到s这个InputStream中,达到获取配置文件的目的。

# 4. 实验总结

本次实验采用MapReduce框架实现了一个能对文档建立倒排索引及词频统计的程序，并进一步实现了无关词语过滤的功能。通过这次实验，我们掌握了在MapReduce框架下进行基础编码的能力，并且对该框架的工作原理也有了进一步的了解。

集群上的执行情况截图如下:

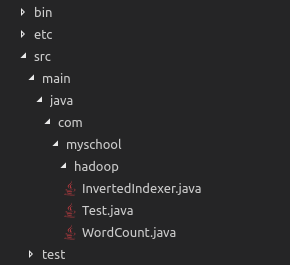


# 5. 代码使用方法（待更新）

首先从Maven的官网下载最新的maven压缩包，版本号为3.6.1，解压到特定文件夹。这里最好更换掉maven的默认镜像，否则下载插件会很慢。接着用指令

mvn archetype:generate -DgroupId=com.myschool.hadoop -DartifactId= hadoop -Dpackage=com.myschool.hadoop -Dversion=1.0-SNAPSHOT

生成我们的java框架，如下：



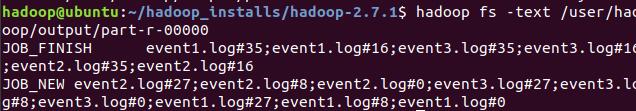
注意，其中的所有java文件都是之后手动添加的。

接着编辑其中的InvertedIndexer文件，上文中已经给出框架，编辑完成后运行如下指令：

mvn package

我们就可以得到一个target文件夹，里面有生成的jar文件，同时还有所有class文件，后面我们会用到其中的InvertedIndexer类。

最后我们在本地使用规模较小的样例测试，其中每个文档都完全相同，只有单词JOB\_NEW和JOB\_FINISH。运行的具体指令与实验2类似，下面为结果截图：



# 6. 实验分工

|  |  |  |
| --- | --- | --- |
| 姓名 | 完成工作 | 备注 |
| 陈昕元 | Map/Combiner/Partition/Reduce代码的部分实现及相应的报告内容。 |  |
| 陈翔 |  |  |
| 陈亚栋 |  |  |