

**课 程 实 验 报 告**

**课程名称： 大数据分析**

**专业班级： 校交1801**

**学 号： U201814132**

**姓 名： 张浚哲**

**指导教师： 王蔚**

**报告日期： 2020年12月30日**

**计算机科学与技术学院**

**目录**

[实验一 wordCount算法及其实现 1](#_Toc60416484)

[**1.1实验目的** 1](#_Toc60416485)

[**1.2 实验内容** 1](#_Toc60416486)

[**1.3 实验过程** 1](#_Toc60416487)

[1.3.1 编程思路 1](#_Toc60416488)

[1.3.2 遇到的问题及解决方式 4](#_Toc60416489)

[1.3.3 实验测试与结果分析 5](#_Toc60416490)

[**1.4 实验总结** 7](#_Toc60416491)

# 实验一 wordCount算法及其实现

## **1.1实验目的**

1、理解map-reduce算法思想与流程；

2、应用map-reduce思想解决wordCount问题；

3、（可选）掌握并应用combine与shuffle过程。

## **1.2 实验内容**

提供9个预处理过的源文件（source01-09）模拟9个分布式节点，每个源文件中包含一百万个由英文、数字和字符（不包括逗号）构成的单词，单词由逗号与换行符分割。

要求应用map-reduce思想，模拟9个map节点与3个reduce节点实现wordCount功能，输出对应的map文件和最终的reduce结果文件。由于源文件较大，要求使用多线程来模拟分布式节点。

学有余力的同学可以在map-reduce的基础上添加combine与shuffle过程，并可以计算线程运行时间来考察这些过程对算法整体的影响。

提示：实现shuffle过程时应保证每个reduce节点的工作量尽量相当，来减少整体运行时间。

## **1.3 实验过程**

### 1.3.1 编程思路

此程序分成三个部分：

1. map，reduce功能的实现。
2. 多线程的同步和互斥实现
3. combine和shuffle的设计和实现
4. map，reduce功能的实现

（1）map函数：

入口参数：filename（传入的待map的文件名称）

返回参数：无。

功能：将源文件中的词语进行map，并进行一个本地combine的过程。

实现：首先将传入文件名对应的文件打开，然后读取文件中的每一行，将里面的每一行使用逗号进行分词，将分出的词存入句子中，将最后一个词的换行符去掉，然后将本地combine后的词典使用pickle模块写入对应文件名的对应文件名加上.pkl后缀的文件中。

（2）reduce函数（实现shuffle版本）：

入口参数：filenames（对应map之后的所有文件名列表），shuffles（对应reduce节点所对应的shuffle规则），id（用于对于分字母开头的单词的分类）

返回参数：返回经过reduce节点reduce后的词典。

功能：对所有的map文件中的词进行reduce，不过只处理对应reduce节点shuffle规则中存在的词。

实现：对于id为0的reduce函数，处理器对应的shuffle规则中的词或者非字母开头的单词，其他id值的reduce函数只需要按照对应的shuffle首字母规则即可，然后根据顺序打开对应的文件列表中对应顺序的文件，统计对应shuffle规则中的词，统计完后将所有统计的单词存入对应id的reduce文件中。

1. 多线程的同步和互斥实现
2. Map线程的实现

Map线程继承python中的threading.Thread类，初始化一个Map线程类需要传入对应Map线程的id，map线程的名字name和对应map线程所负责的文件的文件名filename。

这里使用的是一个Map线程对应一个文件的Map操作，因此9个文件的Map操作使用的是9个Map线程并行处理。

但是，为了保证效率的最大化，给每个Map线程设置一个同步信号灯，每个Map线程完成后释放一个信号灯，这样可以保证每一个Map线程完成后，对应的reduce线程就有机会开始并行执行。

1. Reduce线程的实现

根据实验任务书的要求，实现3个reduce节点，初始化每个reduce节点需要传入thread的id，名称name，对应处理的map文件，以及对应的shuffle规则。

每次reduce节点运行reduce的时候，首先会随机生成一个文件访问顺序列表然后对应的reduce节点会依次访问对应map完成后生成的文件，这样由于三个reduce节点的文件访问列表都是随机生成的（对每个文件的访问等到需要对应信号灯释放之后，表示此文件Map处理完成），所以不光可以使只要有Map线程完成，reduce线程便可以开始进行对应的reduce操作，还能使不同的reduce线程并行处理不同的Map文件（因为每个reduce节点的文件访问顺序不同）。每个reduce节点完成后进行对应信号灯的释放，用于和最后的AllReduce线程进行同步。

1. AllReduce线程的实现

这个AllReduce线程使用来对三个reduce节点的reduce结果做一个最终的reduce，初始化一个AllReduce线程需要传入id，对应的线程名称name，对应处理的所有文件名称列表filenames（这里应该是三个reduce节点结束后产生的reduce文件）。等三个reduce节点对应信号灯释放之后，立即进行最终的reduce。

1. combine和shuffle的设计和实现
2. combine机制的实现

将map出来的对应的<1, 1, 1>结果combine成3，因此这里的combine机制放在map函数中实现，在对一个文件进行map的时候，使用python的字典形式，每次找到一个词，就在其对应的词频后加1，这样就能完成Map节点中的combine过程。

1. shuffle机制的实现

这里，shuffle机制的实现是使用了首字母来分配不同reduce节点的任务量，所有单词可分为27类，分别是26个英文字母开头的字母和非英文字母开头的单词，我们可以在map的时候，统计每种词出现的概率，然后输出出来，得到的结果是{'a': 30850, 'b': 24332, 'c': 38933, 'd': 22778, 'e': 16815, 'f': 15870, 'g': 14466, 'h': 18662, 'i': 15220, 'j': 4158, 'k': 6167, 'l': 14093, 'm': 25194, 'n': 16174, 'o': 15061, 'p': 40956, 'q': 3218, 'r': 21286, 's': 50571, 't': 25223, 'u': 23791, 'v': 6816, 'w': 11672, 'x': 609, 'y': 1740, 'z': 1822}以及非字母开头的单词为73个，所以将shuffle规则分成['a', 'b', 'c', 'd', 'e', 'f']和其他，['g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'q', 'r']和['p', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']三部分。这样能够保证每个节点大概都能够处理150000左右种类的单词。

### 1.3.2 遇到的问题及解决方式

1.map进程和reduce进程和allreduce进程的同步问题

一个Map进程完成之后，会产生一个map文件，但是Reduce进程只有等对应map文件产生后，才能进行操作，而且allReduce进程只有等所有的Reduce进程结束后才能进行，因此，这些进程存在一个同步的关系，并且由于这种同步关系是由map决定的，因此有两种实现方法，第一种是使用信号灯进行同步互斥关系，每次一个Map线程结束后，释放对应的信号灯，然后reduce线程通过其随机访问的文件列表进行信号灯的请求，这样可实现reduce线程的并行和Map的并行，但是存在的问题是可能对应随机访问列表的第一个Map文件执行的比较慢，因此可能会出现很多个reduce线程刚开始启动时会等待较长时间的情况。

第二种实现的方法就是，因为每一次Map之后都会生成对应的Map文件，只有生成Map文件后才能进行对应文件的reduce操作，而且多个reduce线程是可以同时打开一个文件的，所以我们在对应的信号灯的请求操作后立即加上释放，这样，一个Map线程成功进行完后，便会释放此Map线程完成的信号，而第一个收到这个信号的reduce线程，在执行对应文件的reduce之前将此reduce信号传递出去，即请求完后立即释放，这样其他的reduce线程也可对此Map操作完成的文件进行reduce操作，更加节省时间。

2.shuffle规则的设计问题

设计shuffle规则的重点是要将每个reduce节点的工作量尽量分配均匀，最后的结果一共有466550种不同的词汇，因此我的设计是保证三个reduce节点，每个节点大概处理150000种词汇，达到一个粗略的估算负载均衡的效果。首先我在最后的结果中统计出了不同首字母开头的单词总数：{'a': 30850, 'b': 24332, 'c': 38933, 'd': 22778, 'e': 16815, 'f': 15870, 'g': 14466, 'h': 18662, 'i': 15220, 'j': 4158, 'k': 6167, 'l': 14093, 'm': 25194, 'n': 16174, 'o': 15061, 'p': 40956, 'q': 3218, 'r': 21286, 's': 50571, 't': 25223, 'u': 23791, 'v': 6816, 'w': 11672, 'x': 609, 'y': 1740, 'z': 1822}以及非字母开头的单词为73个，因此我按照 ['a', 'b', 'c', 'd', 'e', 'f']和其他，['g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'q', 'r']和['p', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'] 将shuffle规则分成三部分。其中每一个shuffle规则对应的总词数分别为y1:149651 y2:153699 y3:163200。可以看到对应的三个reduce节点shuffle后的任务量应该差不多。

### 1.3.3 实验测试与结果分析

1.执行主程序后可以看到对应9个Map进程和3个Reduce进程和1个AllReduce进程同时开启。如图1.1所示：

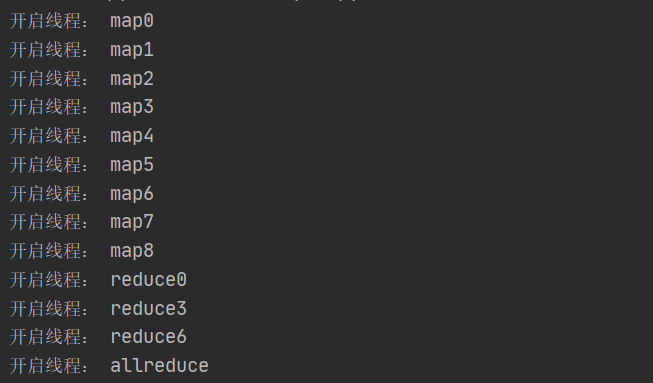


图1.1

13个线程同时开启，然后Map线程开始随机执行，其完成顺序是随机的如图1.2所示：

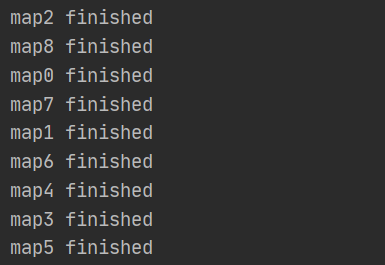


图1.2

Map线程完成后，会得到对应Map线程的pkl文件，source01.pkl，source02.pkl等。

等Map线程中的一部分完成之后，reduce线程就会开始执行，三个reduce线程完成的先后顺序也是随机的，如图1.3所示：

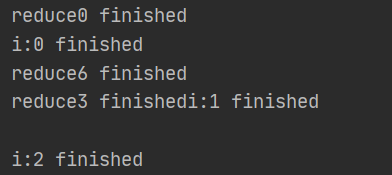


图1.3

reduce线程结束后会得到对应reduce线程的reduce文件，reduce0.reducev2，reduce3.reducev2,reduce6.reducev2这三个二进制文件。

等到三个reduce线程都完成之后，才会执行对应的AllReduce线程，执行AllReduce线程后会得到最后的结果reduce.reducev2\_finaly这个二进制文件。

所有对应的二进制文件可使用转换函数转成对应的txt文件，如map1.txt, map2.txt ……, reduce0\_v2.txt , reduce3\_v2.txt……, reduce0\_v2\_finaly.txt文件目录如图1.4所示：

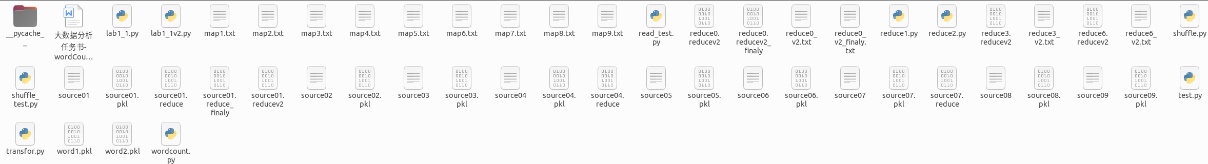


图1.4

## **1.4 实验总结**

这次MapReduce模拟的WordCount还是比较复杂的，首先是对python的多线程类要熟悉，其次是怎样规划Map线程Reduce线程和AllReduce线程的同步关系，怎样设计可以使效率最高，并且不会出错，最主要的同步关系就是只有当对应的Map文件做完了之后，reduce线程才能对其进行reduce操作，只有所有的reduce线程结束之后，AllReduce线程才能对三个reduce线程进行操作。还需要结合对应文件可同时被多个线程读但不可被多个线程写的特点进行进程同步互斥的设计。

然后是对map和reduce函数的设计，具体怎样实现map和reduce的功能。怎样将combine功能加入到对应的map线程中，怎样将shuffle规则结合到reduce线程中。有时候还要根据对应的进程同步的设计进行函数功能的修改。

最后是对combine和shuffle的设计，怎样能够使每个reduce节点所承载的负载较为均衡，怎样通过当前的信息估计和设计每个节点对应的任务量，使得每个节点处理的词数所差不多。

本次实验要求对python多线程和mapreduce的思想和一些tricks有一定的了解，并能够进行模拟和实现。通过本次实验，我学到了怎样在python中合理的使用多线程达到并行加速的目的，怎样使用多节点的MapReduce思想实现wordCount。