

**课 程 实 验 报 告**

**课程名称： 大数据分析**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期：**

**计算机科学与技术学院**

**目录**

[实验一 wordCount算法及其实现 1](#_Toc57053298)

[**1.1实验目的** 1](#_Toc57053299)

[**1.2 实验内容** 1](#_Toc57053300)

[**1.3 实验过程** 1](#_Toc57053301)

[1.3.1 编程思路 1](#_Toc57053302)

[1.3.2 遇到的问题及解决方式 1](#_Toc57053303)

[1.3.3 实验测试与结果分析 1](#_Toc57053304)

[**1.4 实验总结** 2](#_Toc57053305)

# 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。

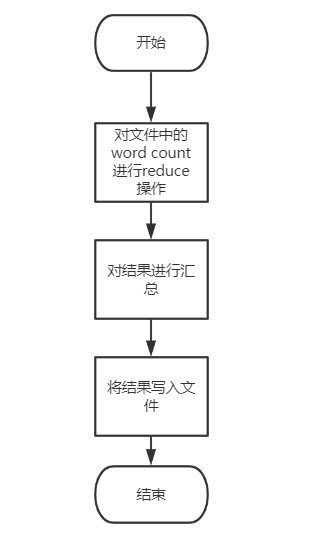
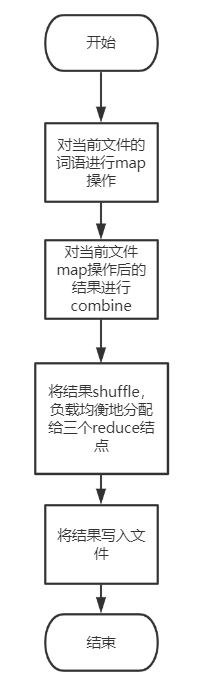
提示：实现shuffle过程时应保证每个reduce节点的工作量尽量相当，来减少整体运行时间。

## **1.3 实验过程**

### 1.3.1 编程思路

根据MapReduce的思想，将流程分为Map和Reduce两个阶段，分别建立两个线程类MapThread和ReduceThread，重写run方法。

对于MapThread，其处理流程如下：



对于Map，即为一个简单地分词+建立字典的操作，combine即为将单个文件中的word\_count进行一个reduce的操作，对于shuffle，这里采取了两种策略：

1. 根据首字母分配给三个reduce结点

2. 首先根据word对字典排序，接着均匀分配给三个reduce结点

对于第一种策略，可以将三个reduce的结点的统计互不重叠，最大限度减少reduce的工作量，但是存在难以决定如何分配的问题，因为单词依据首字母的分布是不均匀的。

对于第二种策略，可以使得三个reduce结点工作量相同，但是各个reduce结点间的词语会有所重叠，并没有显著减少总工作量。

对于Reduce，也是非常简单的遍历，然后对word对应的count值进行增加。

### 1.3.2 遇到的问题及解决方式

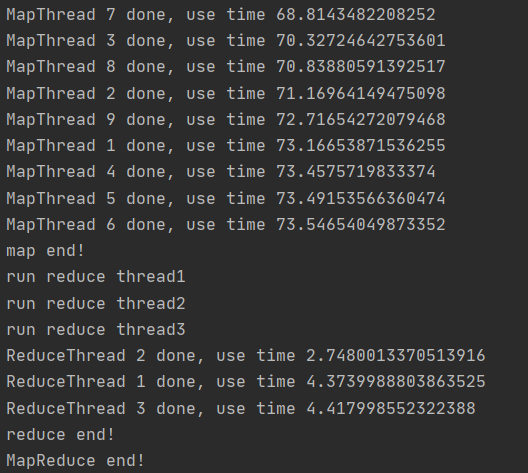
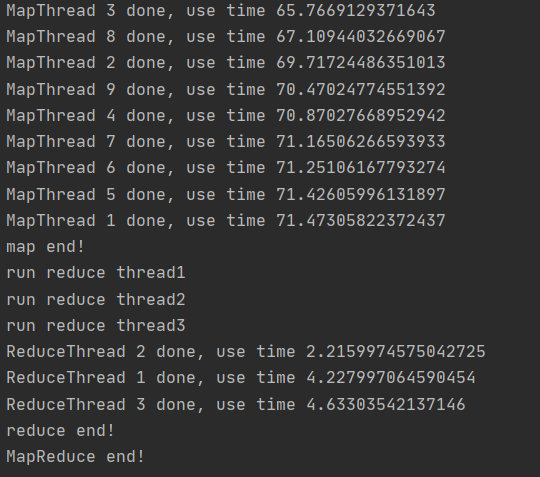
1. 结果错误

第一次跑出来的结果非常错误，属于是一看就错的一种，每一个词的count都有几十万。

解决：通过printf“调试”，以及仔细审阅代码，发现是从中间文件读出来count值没有转换成int类型，还是str类型，导致就结果非常之大

### 1.3.3 实验测试与结果分析

主要是测试了一下两种策略的时间。



可以发现，这两种策略在时间上并没有太大的区别，猜测是因为数据集中，每个文件的单词按照首字母的分布较为类似（注意，并不是说他们是均匀分布），这也是比较好理解的，因为每个文件都是词语库的一个子集，由于本身词语库很大，可以近似认为每个文件都是无偏采样，每个文件中的词语按照首字母的分布相近，所以当排完序后划分，会使得三个reduce结点之间重合度较小。

## **1.4 实验总结**

本次实验有以下收获：

1. 深入理解并学会如何实现MapReduce

2. 学会使用python的多线程