

**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](#_Toc134641467)

[**1.1实验目的** 1](#_Toc134641468)

[**1.2 实验内容** 1](#_Toc134641469)

[**1.3 实验过程** 1](#_Toc134641470)

[1.3.1 系统整体框架 1](#_Toc134641471)

[1.3.2 编程思路与实现 2](#_Toc134641472)

[1.3.2实验测试 7](#_Toc134641473)

[1.3.3 对比试验及结果分析 9](#_Toc134641474)

[**1.4 实验总结** 10](#_Toc134641475)

[1.4.1 实验中遇到的问题及解决 10](#_Toc134641476)

[1.4.2 实验总结 11](#_Toc134641477)

# 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，了解其在大规模数据处理中的优势和不足，在实验中模拟数据的并行处理。

2、应用map-reduce思想解决wordCount问题；

3、掌握并应用combine与shuffle过程，在原有map-reduce的基础上增加combine，shuffle过程以增加数据处理效率。

## **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 系统整体框架

Map-Reduce框架是处理大规模数据任务中使用的编程模型和计算框架，其优势在于高并行性，高扩展性，低成本。

本实验采用分布式处理方式，模拟多个计算节点，解决数词问题（WordCount）。从架构出发，本实验的map-reduce系统由Map，Combine，Shuffle，Reduce四个部分组成，其整体的运行逻辑如图1-1所示：

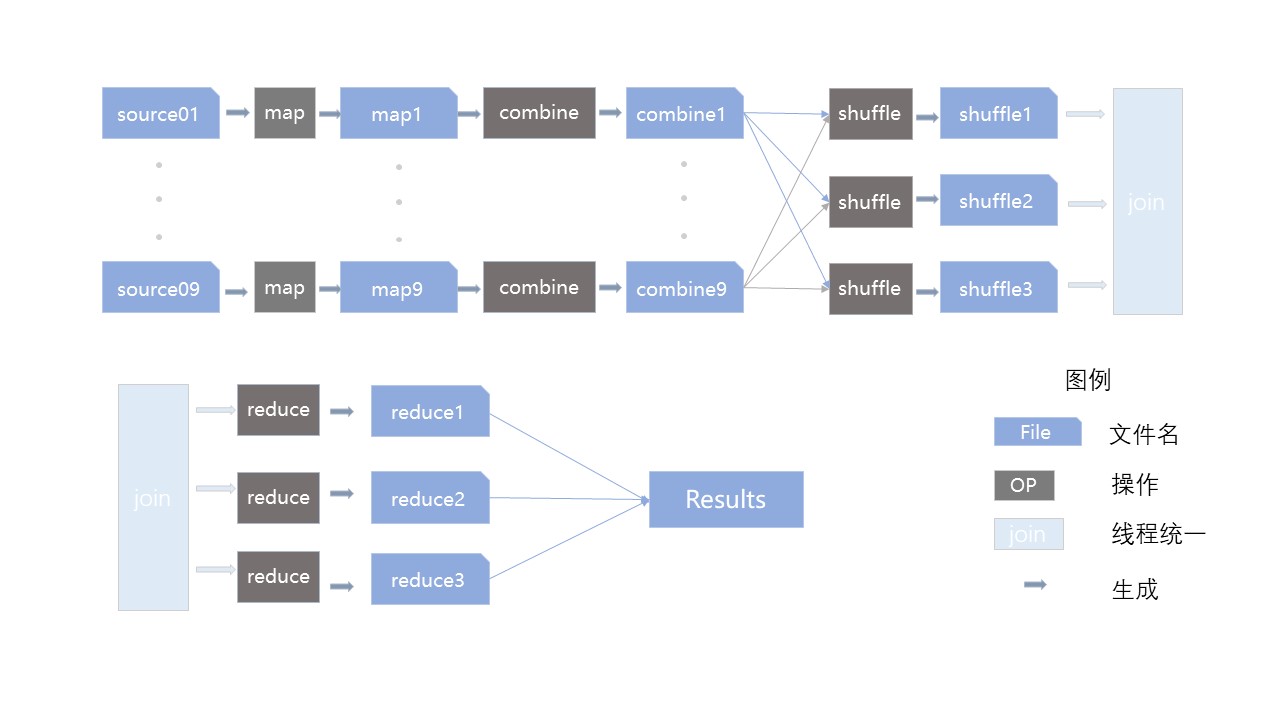
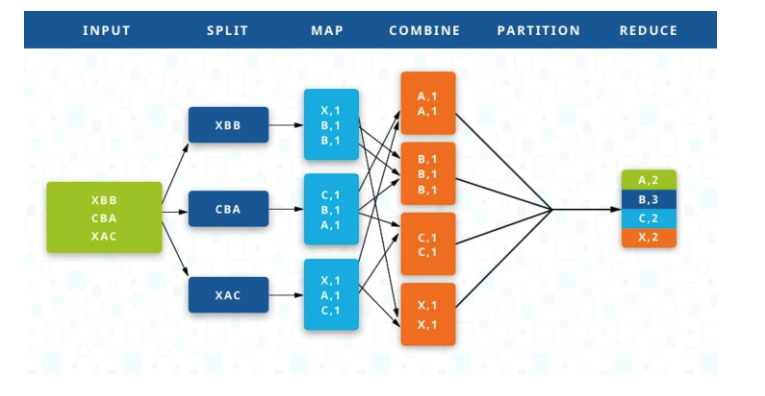


图1-1 Map-Reduce架构流程图

具体到WordCount问题上，图1-2展示了一个简单的分词示例。[1]

****

### 1.3.2 编程思路与实现

**1）数据处理**

在进行map操作之前，通常需要对数据进行预处理。幸运的是，本实验给出的源文件（source0x）已经用逗号与换行符分隔，无需预处理，可以直接进行map操作。

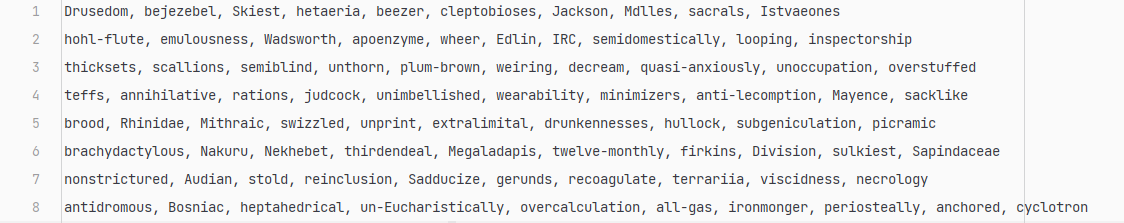


图1-3 source01文件格式

**2）Map**

map过程用于将源数据source0x文件中的信息提取出来，形成<word, count>的键值对。由于源文件中的数据较为规整，故此处的count恒为1。在实际操作中，我根据逗号进行分词，将每个词以<word,1>的格式写入了map文件。由于源文件有9个，我选择建立9个线程对数据进行并行处理，分别进行map操作。



图1-4 map过程代码展示



图1-5 创建线程并执行map操作

**3）Combine**

在Map-Reduce框架中，combine属于可选的中间阶段，负责对map过程中产生的结果（map-x文件）进行局部规约，在本文件内先对重复单词进行合并，这样可以显著减少数据传输量。

具体地，combine过程对map文件中处理好的<word, count>键值对进行统计，累加相同的关键字。在实现上，我设置了一个字典方便累加，遍历map文件中的每一行，若字典中已存在该词，则count累加；若未出现这个单词，则在字典中新增该词。在处理完成后，将字典按照键值的大小进行排序，这是为了方便后续的Shuffle和Reduce操作。最后，将结果写入对应combine文件中。由于map文件为map1-map9，相应地会生成combine1-combine9文件。由于建立线程的过程和map操作基本相似，此处不再赘述。



图1-6 Combine过程代码展示

**4）Shuffle**

在Map-Reduce框架中，shuffle主要负责“传递”中间键值对。由于reduce的节点较map（或map-combine）较少，shuffle操作按照一定的规则将map中的中间结果分区至更少的结果中（数量一般与reduce节点数量相同）。

对于如何“分区”，一般的策略有两种。一是按照单词字母顺序继续划分，在本实验中可分为A-I，J-R，others三个类。另一种策略为使用hash模m的方法，将不同的单词哈希进3个shuffle文件中。在具体的实践中，我选择后一种方法，在shuffle过程中，相对于前一种方法，各个线程的负载更加均衡，耗时也更少。



图1-7 使用单词首字母顺序策略进行shuffle

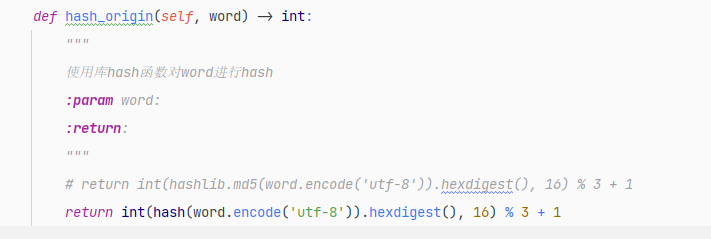


图1-8 使用哈希策略进行shuffle

具体地，shuffle过程对combine文件中的word进行分类，按照哈希结果写入shuffle文件。由于实验并未规定reduce节点的数量，所以我分别尝试了设置3个，4个shuffle节点（在实现上仅仅是mod3或者mod4的微小改动）并创建对应的线程（同map类似）。最后将9个combine文件转换成3个（或4个）shuffle文件（shuffle1-shuffle4）。



图1-9 Shuffle过程代码展示

5）**Reduce**

在Map-Reduce框架中，reduce将shuffle完成的文件再次合并，输出最终的结果。由于仅需要完成计数工作，无需单独设置Reduce函数。

具体地，reduce过程对shuffle文件内的键值对进行统计，其操作与前文的combine操作非常的类似。设置了一个字典，遍历map文件中的每一行，若字典中已存在该词，则count累加；若未出现这个单词，则在字典中新增该词。最后将结果键值对写入对应的reduce文件中即可。其创建线程的过程同combine类似，最后生成了3个reduce文件（reduce1-3）。



图1-10 Reduce过程代码展示

1. **结果生成**

最后生成的结果仅仅需要将3个reduce文件中的内容写入result.csv文件中即可，也可先读入一个字典，先排序后写入。

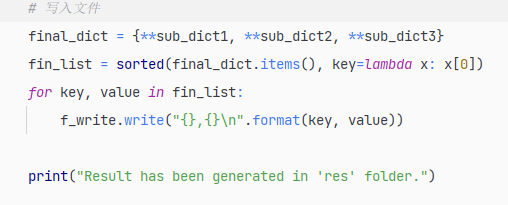


图1-11 生成结果代码展示

1. **并行处理数据**

Map-Reduce的一大优势就在于其并行性，事实上，在map，combine，shuffle操作的过程中，各个文件之间并不涉及交互，因此可以并行处理。在map操作时创建9个线程thread，第i个线程的执行顺序为source0i—map-i—combine-i—shuffle-x。在9个线程全部完成后，再进行reduce操作。

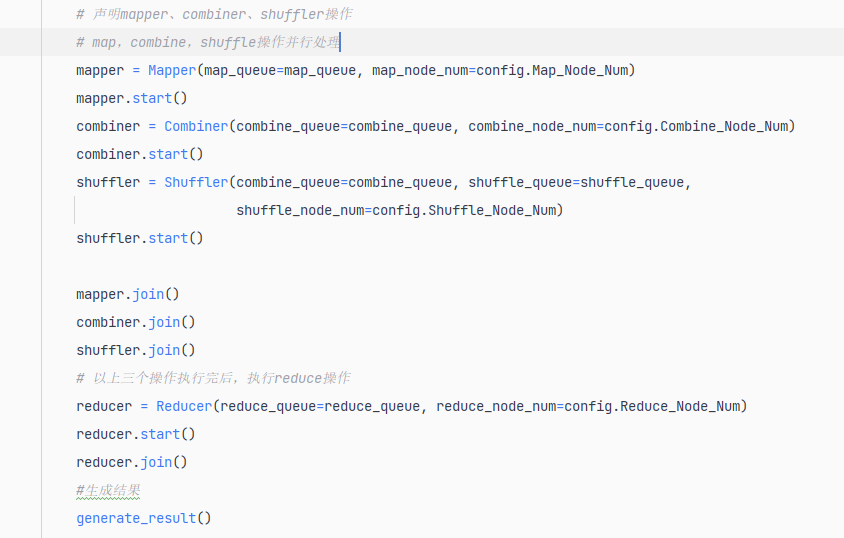


图1-12 并行处理数据代码展示

### 1.3.2实验测试

此部分主要展示Map-Reduce框架中各个环节运行的情况，以及最后的结果。

1. **实验环境**

本实验使用python进行开发，版本为3.11.2。使用的库为threading，os，time等。

实验的目录设置和基本参数如图1-13所示：



图1-13 实验基本参数设置

1. **Map测试**



图1-14 map文件截图展示

1. **Combine测试**

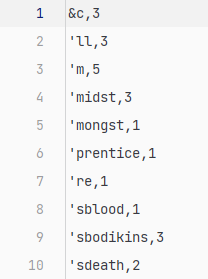


图1-15 combine文件截图展示

1. **Shuffle测试**

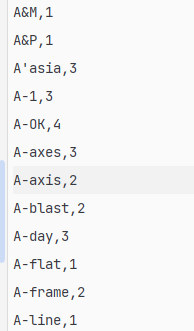


图1-16 shuffle文件截图展示

1. **Reduce测试**

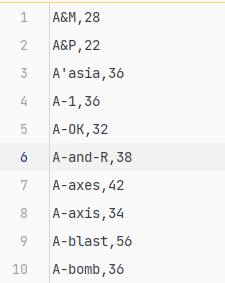


图1-17 shuffle文件截图展示

1. Map-Reduce系统测试

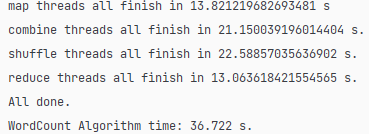


图1-18 总体运行时间展示

总体来说，系统运行正常，各个部分工作正常，生成结果正确且有序。

### 1.3.3 对比实验及结果分析

在设计程序中，我也在关注程序性能的优化问题，希望在保持正确性的基础上，最大程度的提升性能。我关注了以下几个问题：

1. 使用Combine操作是否能提升系统的运行效率？
2. 在Shuffle部分选择不同的分区策略会对该部分的运行时间产生什么影响？
3. 在Reduce过程中选择不同的计算节点数量对系统的性能产生什么影响？
4. 多线程并行能否提升系统运行的效率？

针对这四个问题，我做了四组对比实验，其整合的结果如图1-18所示。对比结果证明，这些操作都对Map-Reduce系统的性能产生了较大的影响。



图1-19 对比实验结果图

下面我将逐一分析这些结果的原因：

1. **Shuffle过程是Map-Reduce系统的性能的瓶颈**

由于shuffle过程需要完成map至reduce的中间数据传输，其中的数据映射，数据排序，甚至是网络传输，磁盘I/O都可能造成长时间耗时，成为map-reduce系统的性能瓶颈。因此，前三个问题均围绕着降低shuffle过程的耗时来提升系统的整体效率。

对于问题一，通过combine操作，能显著减少键值对的数量，方便shuffle的映射，同时combine操作还会对键值对进行排序，这极大的减轻了shuffle过程的工作负担。

对于问题二，使用Hash函数策略进行shuffle操作会使得每个线程的工作量大致一致（而且这个效果会随着数据集规模的增大表现得更加优秀），各个线程负载更加均衡；而首字母划分则可能导致一个线程工作量太大而其他线程“无事可做”，影响整个系统的效率。

对于问题三，reduce节点的数量应该适应数据集的规模和特征，本实验的数据集相对较小，过多的reduce节点并不一定意味着性能的提升。一方面，shuffle过程的映射可能造成负载不均，更重要的是，在最后生成结果时，节点更多意味着数据更加分散，整合的效率会降低。

1. **并行处理是map-reduce的重要优势**

map-combine-shuffle几个操作过程中，由于没有文件数据交互，因此具有非常强的并行性，这也带给程序极大的效能提升，这一点在处理更大规模数据集时会变得更加突出。这也告诉我在学习一个框架时，要发扬他的优势，以达到性能最优的目标。

## **1.4 实验总结**

### 1.4.1 实验中遇到的问题及解决

1）多线程编程

为了模拟map-reduce的分布式节点，我采用了多线程的方式，但是我从前没有接触过并行计算。为了完成实验任务，我首先阅读了《深入理解计算机系统》等资料，了解线程的概念，线程和进程的区别，然后学习了python中thread库的使用，包括join，start，lock等操作，实现了较好的模拟效果。

2）shuffle过程出现某一行只有键（单词），没有值

报错信息如图1-19所示，经排查，报错的原因是在程序运行出现combine文件中有3行只有单词，没有其计数值，我意识到这可能是在读取文件过程中出现了不合法的数据，导致字典无法处理。于是我在combine操作时加上了过滤操作，保证所有的数据均为合法，否则触发exception（代码如图1-20所示），成功解决问题。

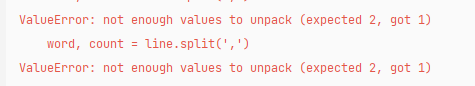


图1-20 shuffle过程报错截图

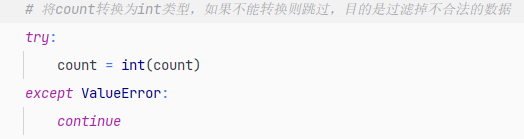


图1-21 解决报错代码

### 1.4.2 实验总结

本次实验让我第一次接触了Map-Reduce的大数据处理框架。在数据规模不断增加的今天，许多习以为常的方法都会发生变化，通过巨大规模数据的处理，我们可以获得更加精准，更加有价值的信息。通过实验，我熟悉了map-reduce的运行原理，挖掘了他的优势和不足，更是对多线程，并行计算，分布式节点等新兴概念有了初步的认识和应用，属实大开眼界。“纸上得来终觉浅，绝知此事要躬行。”我将继续努力。