

**课 程 实 验 报 告**

**课程名称： 大数据分析**

**专业班级： ACM1901**

**学 号： U201914965**

**姓 名： 卫云泽**

**指导教师： 崔金华**

**报告日期： 2021年12月13日**

**计算机科学与技术学院**

**目录**

[实验一 wordCount算法及其实现 1](#_Toc92477693)

[**1.1实验目的** 1](#_Toc92477694)

[**1.2 实验内容** 1](#_Toc92477695)

[**1.3 实验过程** 1](#_Toc92477696)

[1.3.1 编程思路 1](#_Toc92477697)

[1.3.2 遇到的问题及解决方式 4](#_Toc92477698)

[1.3.3 实验测试与结果分析 5](#_Toc92477699)

[**1.4 实验总结** 5](#_Toc92477700)

# 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

**Map过程：**

在本实验中，WordCount的Map过程由9个节点各自独立完成。每个节点对负责处理的原文件进行分词，然后对每个出现的单词附加“1”的计数，形如“Word， 1”的形式，将格式化后的内容输出到新的中间文件中，map过程主要代码如下。

1. fin = open(filename\_in)
2. fout = open(filename\_out, 'w')
3. line = fin.readline()
4. vecline = line.split(', ')
5. **while** line:
6. **for** word **in** vecline:
7. fout.write("{}, {}\n".format(word.strip().lower(), 1))
8. line = fin.readline()
9. vecline = line.split(', ')
10. fin.close()
11. fout.close()

**Combine过程：**

Combine过程也由9个节点各自独立完成。对于Map过程处理后形成的中间文件，Combine读取文件中的每一行内容，使用字典（dict）统计文件中同一单词的出现次数，然后遍历该字典，同样以“Word，Count”的格式输出到新的中间文件中。Combine过程的代码实现如下。

1. res = {}
2. fin = open(filename\_in)
3. fout = open(filename\_out, 'w')
4. line = fin.readline()
5. key, value = line.strip().split(', ')
6. **while** line:
7. value = int(value)
8. res[key] = res.get(key, 0) + value
9. line = fin.readline()
10. **if**(line):
11. key, value = line.strip().split(', ')
12. **for** (key, value) **in** res.items():
13. fout.write("{}, {}\n".format(key, value))
14. fin.close()
15. fout.close()

**Shuffle过程：**

在实验中，Shuffle过程的实现同样采用9个节点独立完成的方式，即对Combine之后的中间结果按照首字母字典序进行分割，目的是为3个节点的分布式reduce过程做准备。Shuffle过程每个节点会产生3个中间结果文件，大致是先前结果以首字母字典序的平均分割，由于本实验中一个Shuffle过程仅采用一个节点的Combine结果作为输入，因此一个单词只会有一个Count Value，即一个单词的计数序列中仅有一个计数值，为简便起见，本实验Shuffle具体实现时同样采用“Word，Count”的形式作为输出结果表示，实现代码如下。

1. **global** key, value
2. fin = open(filename\_in)
3. shuffle1 = open('shuffle0'+ str(i) + '01', 'w')
4. shuffle2 = open('shuffle0'+ str(i) + '02', 'w')
5. shuffle3 = open('shuffle0'+ str(i) + '03', 'w')
6. line = fin.readline()
7. **if** line:
8. key, value = line.strip().split(', ')
9. **while** line:
10. value = int(value)
11. **if** 'a' <= key <= 'g':
12. shuffle1.write("{}, {}\n".format(key, value))
13. **elif** 'h' <= key <= 'r':
14. shuffle2.write("{}, {}\n".format(key, value))
15. **else**:
16. shuffle3.write("{}, {}\n".format(key, value))
17. line = fin.readline()
18. **if** line:
19. key, value = line.strip().split(', ')
20. shuffle1.close()
21. shuffle2.close()
22. shuffle3.close()

**Reduce过程：**

实验中的reduce过程由三个节点完成。由于Shuffle过程一共产生 个中间结果文件，故每个reduce节点会以包含相同首字母的单词的9个文件作为输入，利用字典统计这9个文件中所有单词的Count值之和，产生reduce结果。3个节点的分别形成reduce文件，即最终Word Count结果是由3个reduce文件表示的，每个文件包含若干首字母开头的单词的Count值。Reduce过程实现如下。

1. **global** key, value
2. res = {}
3. **for** i **in** range(1, 10):
4. reduce = open('shuffle0'+str(i)+num)
5. line = reduce.readline()
6. **if** line:
7. key, value = line.strip().split(', ')
8. **while** line:
9. res[key] = res.get(key, 0) + int(value)
10. line = reduce.readline()
11. **if** line:
12. key, value = line.strip().split(', ')
13. reduce.close()
14. fout = open('reduce' + num, 'w')
15. **for** key, value **in** res.items():
16. fout.write("{}, {}\n".format(key, value))
17. fout.close()

### 1.3.2 遇到的问题及解决方式

**分词问题：**

实验中的输入数据是多行以“，”分割的单词，map过程的第一步是分词。

**解决方式：**

采用逐行读取，再对读取的行进行分词的方法，即使用readline()函数读取文件的一行（即按照“\n”分句读入），若读入内容不为空（文件未到达结尾），则对读入的行使用split(“, ”) 以“逗号+空格”作为分隔符进行分词，遍历分词并产生格式化的结果写入输出文件，如此循环直到读取至文件结尾。

**Shuffle过程文件访问冲突问题：**

由于shuffle过程需要将每个文件中的单词按照首字母划分到三个文件中供3个reduce节点使用，但9个shuffle节点同时访问一个目标文件会产生文件访问冲突问题。

**解决方式：**

9个节点每个节点分别产生3个独立的目标文件，即一共27个中间文件，后续只需要每个reduce节点读取9个一组的输入文件即可，该方式更加真实地模拟了分布式情况下的节点工作流程。

**字典中单词存在性判断问题：**

Word Count过程中的Combine和Reduce均使用到字典（dict），但在实际实现过程中需要判断一个单词是否在字典中，若不再，则创建该字典项；否则将给字典项对应的value值加上当前值作为新的value值。问题是如何更好地实现这一过程。

**解决方式：**

使用get() 函数获取字典中某个key对应的value值，如下所示：

1. res[key] = res.get(key, 0) + int(value)

即给定一个默认值（代码中为0），若字典中没有该key，则返回默认值0，否则返回key对应的value值。get函数的使用可以更加便捷地使用dict统计单词出现频次。

**计时问题：**

Python中较难直接对线程的执行过程进行计时。

**解决方式：**

可以采用线程内部自我计时的方式，即在每个过程的函数起止处加入计时开始和停止的语句，每个线程会分别进行自我计时，并打印各自的执行时间。

### 1.3.3 实验测试与结果分析

基于预处理后的九个源文件source01-09进行测试，程序依次执行map、combine、shuffle和reduce过程。map、combine和shuffle过程各节点执行时间和源文件有关，只有reduce过程的执行时间和shuffle策略有关，报告中仅展示reduce过程各节点运行时间，如图1-1所示。

文本, 信件

描述已自动生成

图1-1 三个reduce节点的执行时间

三个reduce节点的执行时间较为均衡，说明shuffle过程较好地将任务平均分派给三个reduce节点。reduce执行完毕后的部分结果如图1-2所示。

图形用户界面, 文本

描述已自动生成

图1-2 reduce执行结果（部分）

reduce结果即为wordCount的最终结果，即对所有出现过的单词的词频统计。经抽样验证，结果具备正确性。实验程序正确地实现了wordCount过程。

## **1.4 实验总结**

在本次实验中，我深入理解了map-reduce思想，并实现了基于9个map节点和3个reduce节点的wordCount程序。此外，我还学习并掌握了combine和shuffle的基本用法，并在实验的进阶部分实现了这两个过程。最终三个reduce节点的工作量大体相当，程序整体运行时间较快。

此外，我还学习并尝试了基于python的多线程编程，在实验中模拟了多个分布式节点执行wordCount过程。