华中科技大学

Huazhong University of Science and Technology

课程实验报告

课程名称：大数据分析

专业班级：校交 1901 班

学　　号：U201910681

姓　　名：骆瑞霖

指导教师：崔金华

报告日期：2021-12-12

计算机科学与技术学院

目录

1 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. 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。

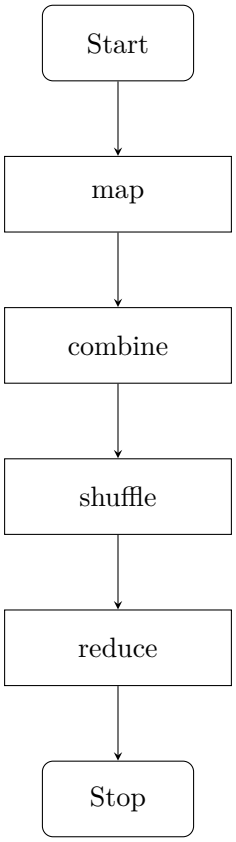
要求应用 map-reduce 思想，模拟 9 个 map 节点与 3 个 reduce 节点实现 wordCount 功能，输出对应的 map 文件和最终的 reduce 结果文件。由于源文件较大，要求使用多线程来模拟分布式节点。

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。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

首先根据 map-reduce 的思想可以得到实验的编程流程图大致如下：



map, combine, reduce 等过程均可以使用多线程实现，下面分别进行编程思路的叙述：

- map

map 过程是将文本中的所有单词全部提取出来，并为每一个单词打上 1 的标签，这里使用多线程实现，定义一个数组 thread_list 来作为线程的列表，每一个线程的执行目标函数为 mapper, 参数就是几号文件;

```
1 //Python code
2 thread_list = []
3 for i in range(1, 10):
4     thread_list.append(threading.Thread(target=mapper, args=(i, )))
```

- combine

combine 过程的目的是将 map 后的每一个文件中的结果，相同词汇的次数累计起来，得到新的词典；同样使用多线程完成，由于 combine 后词汇行数会减小很多，所以不用文件存储了，

直接得到新的词汇到出现次数的哈希，存入一个列表中，由于需要得到线程执行函数的返回值，这里定义类 CombineThread 继承 threading.Thread 类，在其中定义方法 get_result 来获取该线程任务执行完毕后的结果；使用 join 方法使得所有创造的子线程执行结束后主线程再将所有生成的 9 个新词典加入列表。

CombineThread 类定义如下：

```
//Python code
class CombineThread(threading.Thread):
    def __init__(self, func, args=()):
        super(CombineThread, self).__init__()
        self.func = func
        self.args = args

    def run(self):
        self.res = self.func(*self.args)

    def get_result(self):
        try:
            return self.res
        except Exception as e:
            return None
```

程序执行完毕后得到大小为 9 的列表 combine_dict_list, 其中每一个元素都是一个原文本的词汇字典，key 为词汇，value 为在文本中的出现次数。

- shuffle

在 shuffle 过程中，我们需要将 combine 后的 9 个字典中的键值对分配给 3 个 reduce 结点来完成最后的 reduce 过程，在 shuffle 过程中我们应该保证三个 reduce 结点的工作负载是差不多的，我们需要选取一个策略将词汇进行分配，考虑到最后 reduce 的结果应该是按照字典序升序的，不如就按照键值对中词汇键的首字符来分区，经过计数，大写和符号开头的词汇占比相比小写字母开头的词汇要少很多，所以我的策略是将大写开头、符号开头，小写字母 a c 开头的单词分进第一个 reduce 结点中，剩下的字母以首字母'o' 作为划分，shuffle 操作完毕后使用三个文件记录三个 reduce 结点应该处理的数据。

- reduce

在 reduce 过程中，每一个结点分别处理 shuffle 过程输出的三个文件中的数据，将不同分区中相同词汇的出现次数进行累计，然后三个结点分别输出按键升序的排列结果，由于三个分区的结果已经按照首字母顺序划分过了，所以直接按第 1 到第 3 个结点的顺序进行最终结果的写

入就得到了 map-reduce 的最终结果，输出到文件。

其中 reduce 的子线程需要完成的目标函数为 Reducer();

```
//Python code
def Reducer(order):
    sourcefile = './shuffle/shuffle0' + (str)(order)
    dict = {}
    for line in open(sourcefile):
        element = line.split('\t')
        word, num = element[0], (int)(element[1])
        if word in dict:
            dict[word] += num
        else:
            dict.update({word : num})
    dict = sorted(dict.items(), key=lambda d: d[0])
    filename = './reducer/reducer0' + (str)(order)
    w = open(filename, 'w')
    for ind in range(len(dict)):
        w.write(dict[ind][0] + '\t' + str(dict[ind][1]) + '\n')
    w.close()
```

1.3.2 遇到的问题以及解决方式

- 对于 map_reduce 过程记忆不够清晰，通过查找博客的方式对这个过程有了更深的印象，并将其应用到了代码实现中。

- 对于 shuffle 过程，思考如何让所有的 reduce 结点所花处理时间得到合理的分配，通过在网上寻找一些常见的 shuffle 策略以及针对单词数据类型本身特点，最后制定了上述的分配策略。

1.3.3 实验测试与结果分析

实验要求输出对应的 map 文件和最终的 reduce 结果文件，map 部分代码执行完毕后得到 9 个文件命名为 mapper01 mapper09 分别存储 9 个文件的 map 后的结果;

reduce 部分程序执行完毕后得到输出文件 ans，就是最终 map_reduce 的结果，部分输出如下图所示:

```
nonritualistic  1
freedoot    1
coleslaws   1
argalas 1
Essam   1
Vinnie  1
immaculateness  1
Listerised  1
moisturizer 1
dicrotic    1
dingeys 1
Ajanta  1
```

图 1: map 运行结果

```
zygodont    25
zygogenesis 30
zygogenetic 14
zygoid  18
zygolabialis    21
zygoma  22
zygomas 9
zygomata    11
zygomatic   16
zygomaticoauricular 22
zygomaticoauricularis   20
zygomaticofacial    15
zygomaticofrontal   20
zygomaticomaxillary 10
zygomaticoorbital   21
zygomaticosphenoid  23
zygomaticotemporal  14
```

图 2: reduce 输出部分结果

1.4 实验总结

通过本次实验，我体会了一次搭建简易的 map_reduce 数据处理过程，对 map_reduce 的整个流程有了更深的印象，同时也完成了用 Python 语言进行大数据分析的第一步。