華中科技大学

课 程 实 验 报 告

课程名称： 大数据分析

专业班级： **CS2202**

学　　号： **U202215378**

姓　　名： 冯瑞琦

指导教师： 崔金华

报告日期： **2024 年 6 月 7 日**

计算机科学与技术学院

目录

实验一 Map-reduce 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 个预处理过的文件夹（folder_1-9）模拟 9 个分布式节点中的数据，每个源文件夹中包含大约 6 千个文件，每个文件标题为维基百科条目标题，内容为对应的网页内容。提供 words.txt 文件作为待统计的词汇。

要求应用 map-reduce 思想，模拟 9 个 map 节点与 3 个 reduce 节点实现对维基百科条目词汇的词频的统计。

map 节点输出<((title1,key1),1),…,((titlem,keyn),1)>，其中 key 为文件 title.txt 中出现的且在 words.txt 中词。同时，要求最终的 reduce 节点输出出现次数最多的前 1000 个词汇，以及这些词汇的跳转关系。

输出对应的 map 文件和最终的 reduce 结果文件。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。

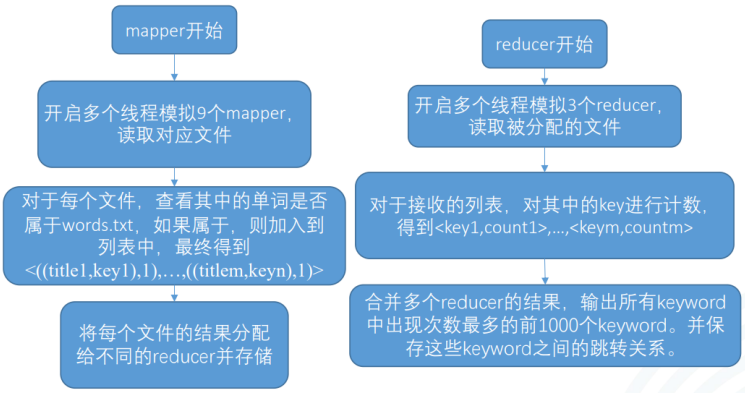
提示：实现 shuffle 过程时应保证每个 reduce 节点的工作量尽量相当，来减少整体运行时间。

1.3 实验过程

1.3.1 编程思路

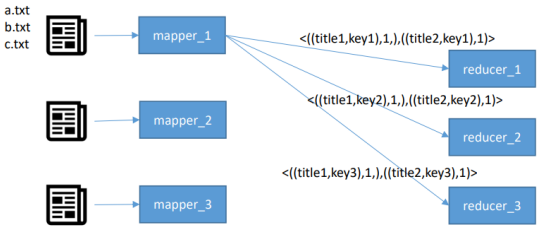
首先，map-reduce 算法的大致流程如下图所示。关于 mapper 结果的分配：最简单的版本可以是每 3 个 mapper 输出的文件作为一个 reducer 的输入文件。

◆ MapReduce参考算法流程

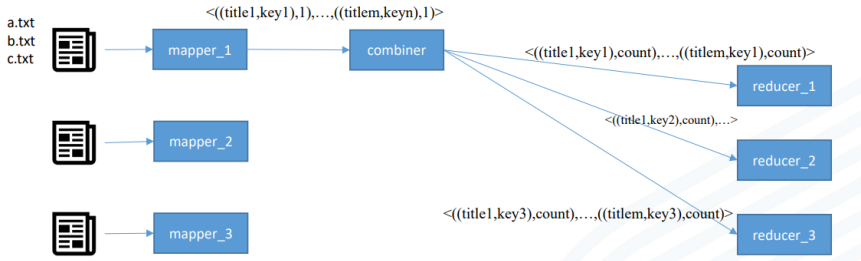


但是，当每个 mapper 输入的文件数量差距很大时不同 reducer 的工作量差异可能会很大。这时就需要使用 shuffle 和 combine 算法进行优化，其过程如下图。

◆ 进阶：使用shuffle过程。



使用 shuffle，一个 mapper 将输出平分为多份，分给多个 reducer，这样每个 reducer 的工作量大致相同。可以通过 hash 来将 keyword 分配到 reducer 上。



注意到在上述方法中，mapper 到 reducer 的传输开销较大，key_list 中可能包含很多重复的关键字，每个 mapper 可以通过 combiner 来压缩传输开销。

具体的编程步骤如下：

1) 读取文件

根据合适路径导入 9 个 folder 文件和 word 文件。

2) 读取词汇表

初始词汇表需经过预处理才能用于后续算法。读取词汇表，并将每行转换为小写，去除空白字符后加入集合。

3) Map 过程

在 Map 阶段，我们需要模拟 9 个 Map 节点，每个节点负责处理一个文件夹中的文件。通过多线程实现，每个线程负责一个文件夹。

读取每个文件的内容，统计其中在 words.txt 中出现的词汇，并将其转换为键值对形式<((title,key),1)>。处理过程为去掉'.txt'扩展名添加到标题列表中，读取文件内容并转换为小写，使用正则表达式找到所有单词，将单词和标题作为键，1 作为值放入队列。函数实现如下图所示。

```
# Map阶段函数
def map_task(folder, queue):
    for file in os.listdir(folder):
            titles.append(file[:-4])   # 去掉'.txt'扩展名添加到标题列表中
            with open(os.path.join(folder, file), 'r', encoding='utf-8') as fi:
                content = fi.read().lower()    # 读取文件内容并转换为小写
                words = re.findall(r'\b\w+\b', content)  # 使用正则表达式找到所有单词
                for word in words:
                    if word in target_words:
                        queue.put(((file[:-4], word), 1))   # 将单词和标题作为键，1作为值放入队列
```

4) Reduce 过程

reduce_task 函数从队列中获取键值对，并更新 Reduce 结果。该函数首先创建一个临时字典 counter 用于累加键值对。在循环中，使用 queue.get_nowait()非阻塞地从队列中获取一个元素，并将其键值对更新到最终结果字典 reduce_results 中。如果队列为空，会引发异常并跳过当前循环。

```
# Reduce阶段函数
def reduce_task(queue, results):
    counter = defaultdict(int)
    while not queue.empty():
        try:
            (key, value) = queue.get_nowait()      # 从队列中获取一个元素
            reduce_results[key] += value  # 更新该键对应的值
        except:
            pass
```

5) 并行实现过程

使用多线程进行并行处理。创建队列 queue 存储 Map 阶段的中间结果，并定义列表 map_threads 存储 Map 阶段的线程。遍历所有文件夹，为每个文件夹创建一个执行 map_task 的线程。启动线程并使用 join()方法等待所有线程完成，以确保所有中间结果生成并存储在队列中。

在 Reduce 阶段，同样使用多线程处理数据。创建并启动 3 个执行 reduce_task 的线程，处理队列中的中间结果并生成最终结果。主线程使用 join()方法等待所有 Reduce 线程完成执行。通过多线程并行处理，提高了 MapReduce 过程的执行效率和性能。

6) 创建跳转关系的字典

创建集合 top_1000_set，包含出现次数最多的前 1000 个单词。使用 defaultdict 创建一个字典 jump_relations，用于存储单词之间的跳转关系。遍历 reduce_results 字典中的所有键值对，生成单词之间的跳转关系。如果单词和标题都在前 1000 个词汇中，并且单词在标题列表中，则将单词添加到标题的跳转关系集中。

```
# 创建跳转关系的字典
jump_relations = defaultdict(set)   # 使用defaultdict创建一个默认值为空集的字典
for ((title, word), count) in reduce_results.items():  #遍历Reduce阶段的结果
    if word in top_1000_set:   # 如果单词在前1000个词汇中
        if title in top_1000_set:   # 如果标题在前1000个词汇中
            if word in titles:   # 如果单词在标题列表中
                jump_relations[title].add(word)   # 将单词添加到标题的跳转关系集中
print(jump_relations)
```

7) 结果文件生成

将 map 阶段的结果输出到文件 map_results.txt。遍历 reduce_results 字典中的所有键值对，将其写入文件。然后将出现次数最多的前 1000 个单词及其计数输出到文件 reduce_results.txt。跳转关系字典输出到文件 jump_relations.txt，每行表示一个单词及其对应的跳转关系。

1.3.2 遇到的问题及解决方式

1) 数据同步问题

问题描述：在多线程环境下，处理共享数据（如 titles 列表）时，可能会出现数据竞争和同步问题，导致数据不一致。

解决方式：使用全局变量 titles 并在每个线程中独立操作，避免复杂的同步问题。尽管没有使用锁机制，但这种方法在一定程度上简化了线程间的同步需求。

2) 多线程处理大数据集

问题描述：处理大量文件需要高效的并行处理机制，否则会导致程序运行缓慢，无法及时处理数据。

解决方式：使用 Python 的 threading 模块创建多个线程来并行处理文件夹中的文件。每个文件夹由一个独立的线程处理，大大提高了处理速度。这种方式充分利用了多核 CPU 的优势，提高了文件处理的并行度。

```
# 启动Map线程
for folder in folders:
    thread = threading.Thread(target=map_task, args=(folder, queue))
    thread.start()   # 启动线程
    map_threads.append(thread)   # 将线程添加到线程列表中
```

3) Map 和 Reduce 过程的数据传输

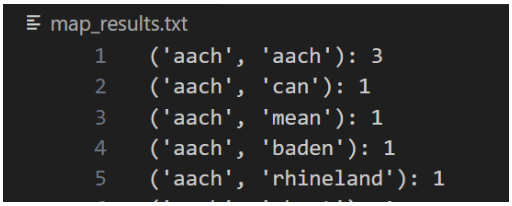
问题描述：Map 阶段生成的大量中间结果需要传输到 Reduce 阶段进行处理，如何高效地传输和处理这些数据是一个挑战。

解决方式：使用 Python 的 Queue 模块创建一个队列，将 Map 阶段生成的中间结果放入队列，Reduce 阶段从队列中获取数据进行处理。队列在多个生产者（Map 线程）和多个消费者（Reduce 线程）之间传递数据，保证了数据传输的高效性和线程安全。

1.3.3 实验测试与结果分析

1) map_result.txt 文件

该文件记录了在 Map 阶段每个单词在每个文档中的出现频次。这些键值对形式的记录可以帮助理解数据在 Map 阶段的处理结果。map_result 文件记录了在 Map 阶段每个单词在每个文档中的出现频次。这些键值对形式的记录，如下图所示。

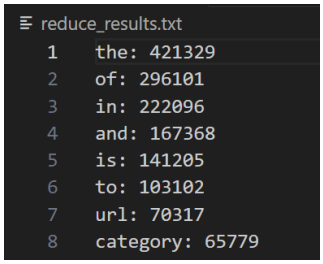


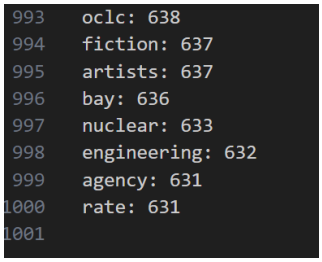
具体解释如下：

('aach', 'aach')是键的结构：键是一个二元组，包含两个元素，第一个元素是文件名，第二个元素是单词。:3 为值，表示这个单词在该文件中出现的次数。在这里，值是 3，表示单词'aach'在文件'aach.txt'中出现了 3 次。

2) reduce_result.txt 文件

reduce_result 文件记录了在 Reduce 阶段每个单词在所有文档中的总出现频次。这些键值对形式的记录可以帮助理解数据在 Reduce 阶段的处理结果，如下图所示。



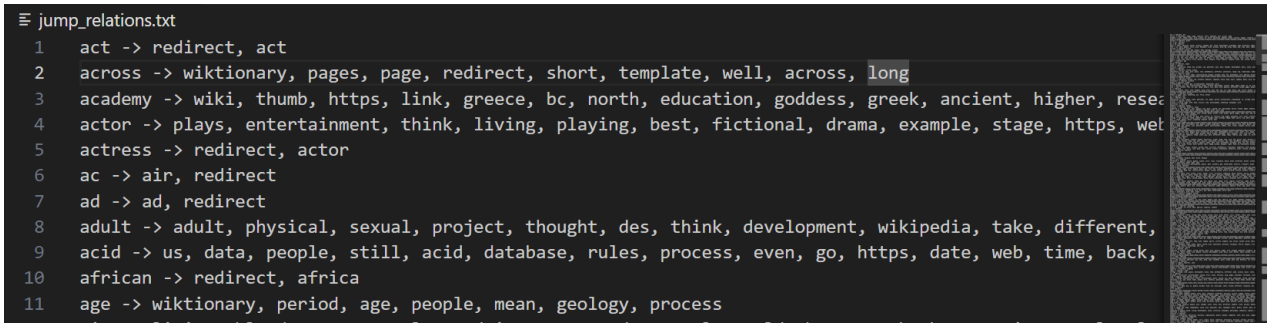


具体解释如下：

the 是键的结构，表示具体的单词。在这个例子中，单词是 the。421329 是值，表示单词 the 在所有文档中出现的总次数。在这里，值是 421329，表示单词 the 在所有处理过的文件中一共出现了 421329 次。该文件中内容是出现次数最多的 1000 个单词，出现次序降序排列。

3) jump_relation.txt 文件

jump_relations 文件记录了每个词汇的跳转关系，这些跳转关系表示词汇之间的关联和引用。在这个文件中，每一行显示一个词汇及其跳转到的目标词汇集合。例如，act -> redirect, act 表示词汇 act 可以跳转到 redirect 和 act。



1.4 实验总结

通过本次实验，我对 map-reduce 算法有了深入的理解和实践。map-reduce 作为一种大数据处理模型，通过 map 阶段将大任务分解成小任务并行处理，通过 reduce 阶段汇总结果，具有很好的扩展性和处理效率。在实际应用中，map-reduce 算法广泛应用于分布式计算、数据挖掘和大数据分析等领域。

实验过程中，我遇到了一些问题，从最开始的软件下载到 python 语法的不熟练，还有例如多线程同步问题和数据分配不均等问题。我深刻意识到了课堂上学习的理论知识和实际变成运用之间还需要很多努力。通过合理的编程技巧和算法优化，我成功解决了这些问题，保证了程序的正确性和效率。同时，我了解了 combine 与 shuffle 过程的算法概念，知道了可以用这些方法进一步提高程序的性能，减少了 map 节点与 reduce 节点之间的通信开销。不过很遗憾，由于时间有限，我的熟练度也还不高，所以没能完成进阶算法。

总体来说，本次实验不仅增强了对 map-reduce 算法的理解，还提高了编程能力和解决实际问题的能力，为后续的大数据分析学习奠定了良好的基础。