# MapReduce实验报告

516030910375 蔡一凡

## 实验目的

1. 了解MapReduce的实现原理；
2. 编写基础的基于MapReduce框架的应用程序；
3. 实现Map和Reduce接口的内部实现；
4. 动态调度执行Map和Reduce的设备，并实现容错机制。

## 实验各部分设计

1. Map/Reduce的输入和输出

* 本部分完成了doMap和doReduce内部的实现。
* 对于doMap函数，首先读取指定文件名的文件内容，并调用用户定义的map函数。该函数将返回一个List<KeyValue>。对List中的每一个元素，调用hashCode函数，并计算出该KeyValue对将被调度到哪一台设备上运行。然后，据此生成对应文件名的文件内容，此处用JSON格式储存，以便编码和解码。doMap函数只生成一个文件。
* 对于doReduce函数，需要从对应文件名的一个或多个由doMap生成的文件中读取内容，并解码后合并存储在一个List<KeyValue>中，然后将该List排序。排序后，对于具有相同的key的一个或多个元素，调用用户定义的reduce函数，将输出的结果存在一个文件中。

1. 单工作节点的单词计数

* 在mapFunc中，调用正则表达式的处理库，对每一个单词word生成一个 (word, “1”) 的KeyValue对。
* 在reduceFunc中，上一级的doReduce函数已经将每个单词的KeyValue对合并。因此，reduce函数只需返回values的长度。

1. 任务的调度和容错机制
   * 由于任务的调度和容错机制的实现两者互相关联，因此放在一个部分中叙述。
   * 创建一个全局的ConcurrentHashMap，来记录每一个任务的完成情况。每当需要一个新的schedule任务时，向ConcurrentHashMap添加所有未完成的任务。
   * 遍历HashMap中的每一个未完成的任务，并通过RegisterChannel获取所有空闲的执行节点，向其分配任务。
   * 对于每一个worker，创建一个worker thread。当任务完成后，将countDownLatch减一，并在HashMap中更新其任务为完成状态。
   * 当schedule发现有任务10s（这个时间可以根据需要进行修改）没有返回（即countDownLatch没有归零），则将未完成的任务重新分配。
2. 倒排索引(Inverted Index)

* 在mapFunc中，调用正则表达式的处理库，对每一个单词word和已知的文件file生成一个(word,file)的keyValue对，并添加到List中。完成所有的单词的处理后，返回该List。
* 在reduceFunc中，对于每一个key和对应的values，找出values中所有不同的文件名，排序后按照规定的格式添加到一个字符串中并返回。

## III. 实验结果及反思

* 完成了实验指导文档中所有的实验要求。对于用Junit测试的部分，成功通过了测试；对于手动对照的部分，该实验的输出与预期相符。
* 本实验中自己设计实现了任务的调度等工作，对于MapReduce的执行模式有了更深刻的认识。
* 实验中用到了大量的java的并发处理，在实践中学习了Java Thread，countDownLatch和各类并发数据结构如阻塞队列、并行哈希表的用法。
* 对于文件的读写，使用了多种的尝试。结果表明，逐个字节读或者逐行读都会大大降低程序的运行效率，而一次性读整个文件能够更好地利用文件系统以及缓存的设计，在保证效率的同时也能满足任务的要求。