实验 2 MapReduce 的实现

1. 实验前提

a) Linux / macOS 系统，并且配置了 Golang 开发环境，掌握 Golang 的基本用法。

b) 了解 Git 的基本使用。

c) 实验源地址：http://nil.csail.mit.edu/6.824/2022/labs/lab-mr.html

2. 实验目的

通过对 MapReduce 的复现，加深对《MapReduce: Simplified Data Processing on Large Clusters》论文思想和原理的理解。

3. 实验步骤

3.1 阅读原文 http://nil.csail.mit.edu/6.824/2022/labs/lab-mr.html，理解实验内容。

3.2 代码实践

a) 克隆项目，项目地址：git://g.csail.mit.edu/6.824-golabs-2022

```
zhuchangzhen@zhucz:~$ git clone git://g.csail.mit.edu/6.824-golabs-2022 6.824
正克隆到 '6.824'...
remote: Enumerating objects: 122, done.
remote: Counting objects: 100% (122/122), done.
remote: Compressing objects: 100% (118/118), done.
remote: Total 122 (delta 38), reused 0 (delta 0)
接收对象中: 100% (122/122), 1.26 MiB | 68.00 KiB/s, 完成.
处理 delta 中: 100% (38/38), 完成.
zhuchangzhen@zhucz:~$ cd 6.824/
zhuchangzhen@zhucz:~/6.824$ ls
Makefile  src
zhuchangzhen@zhucz:~/6.824$
```

其中，在 src/main/mrsequential.go 中提供了一个简单的单线程的 MapReduce 实现，可以参考 mrsequential.go、mrapps/wc.go。如下图，编译源码并执行源码的 main 函数。

b) 实现一个分布式的 MapReduce 系统。

该系统由两个程序组成：coordinator（即 master） 和 worker，在系统运行期间共包含一个 coordinator 进程和多个并行运行的 worker 进程。在实际应用的 MapReduce 系统中，worker 进程运行在很多不同的机器上，而在本实验中，只需要在一台机器上运行它们即可。

worker 进程和 coordinator 进程之间使用 RPC 通信。每个 worker 进程需完成以下工作：向 coordinator 进程请求 task，从若干文件中读取输入数据，执行 task，并将 task 的输出写入到若干文件中。coordinator 进程除了为 worker 进程分配 task 外，还需要检查在一定时间内（本实验中为 10 秒）每个 worker 进程是否完

成了相应的 task，如果未完成的话则将该 task 转交给其他 worker 进程。

main/mrcoordinator.go 和 main/mrworker.go 中分别提供了 coordinator 程序和 worker 程序的 main 函数入口，只需要在 mr/coordinator.go 和 mr/worker.go 、 mr/rpc.go 中添加关键代码来实现 MapReduce。

c) 代码运行，以单词计数为例。

- 首先要确保对应的二进制版本是最新的。

  go build -race -buildmode=plugin ../mrapps/wc.go

- 在 main 目录下，运行 coordinator。

  rm mr-out*

  go run -race mrcoordinator.go pg-*.txt

  其中，pg-*.txt 中保存了程序的输入数据，每个文件可以作为单个 Map task 的输入。

- 在其他若干窗口中，可以运行如下命令来执行一个或多个 worker 进程。

  go run -race mrworker.go wc.so

- 当 coordinator 和 worker 程序运行完成，mr-out-*中则保存了程序输出，经过排序后的输出结果应与 mrsequential.go 的输出相同。

d) 测试。

main/test-mr.sh 为测试脚本，该脚本会检查程序是否产生了正确的输出，worker 是否并行运行，以及是否能够正确处理发生崩溃的 worker 进程。测试脚本期望对于每个 reduce task，都会生成一个被命名为 mr-out-x 的输出文件。

当测试成功时，脚本输出应该如下所示：

$ bash test-mr.sh

*** Starting wc test.

--- wc test: PASS

*** Starting indexer test.

--- indexer test: PASS

*** Starting map parallelism test.

--- map parallelism test: PASS

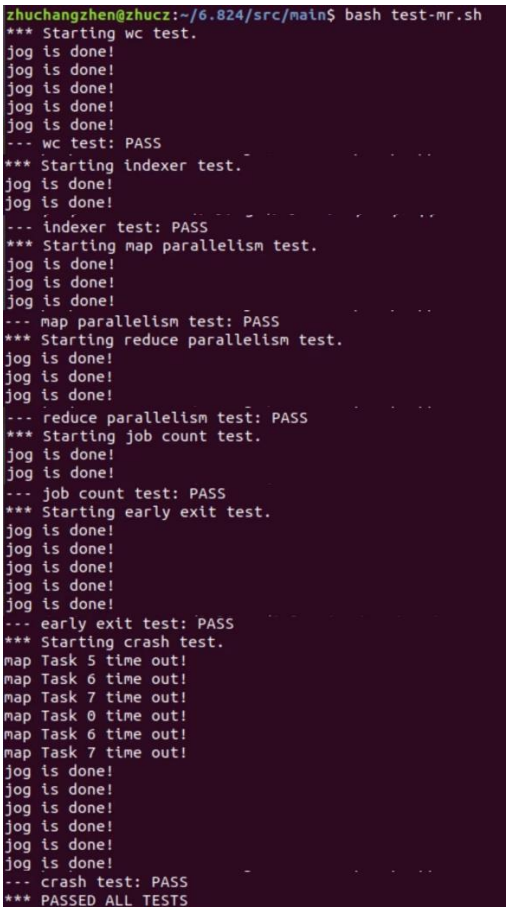
*** Starting reduce parallelism test.

--- reduce parallelism test: PASS

*** Starting crash test.

--- crash test: PASS

*** PASSED ALL TESTS



e) 其他。

- worker 进程应把第 x 个 reduce task 的输出保存到文件 mr-out-x 中。
- mr-out-X 中每行都应该是调用一次 Reduce 函数的输出，应该按照 Go 语言的"%v %v"的格式生成，也即 key 和 value。
- worker 进程应该将 Map 函数的输出（intermediate key）保存在当前目录的文件中，使得后续 worker 进程可以读取它们并将其作为 Reduce task 的输入。
- 当 MapReduce Job 被计算完毕后，main/mrcoordinator.go 希望您实现的 mr/coordinator.go 中的 Done()方法返回 true。这样 mrcoordinator.go 就能知道 Job 已经顺利完成，进程即可退出。
- 当 MapReduce job 被做完后，worker 进程应该可以正常退出。

4. 实验要求

基于项目提供的框架实现 MapReduce，并通过所有测试用例。

5. 实验报告

通过流程图的形式，结合代码实现，分析 MapReduce 执行过程，并给出结果截图。

6. 提交方式

将实验代码与实验报告一同打包，并以学号_姓名__DSC_Lab2 格式命名，提交到研究生信息系统的课程平台。