# 目标成绩： B

# 大数据实验报告（实验3）

 班级 21计算机 4 班  学号 2021329600006 姓名 陈昊天

实验时间： 2024.04.30

一、实验名称：MapReduce

实现一个和MapReduce论文类似的机制，也就是数单词个数Word Count。

二、实验设计

### 1 Goland配置

GoLand 2024.1.1

Build #GO-241.15989.157, built on April 30, 2024

Runtime version: 17.0.10+1-b1207.14 aarch64

VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

macOS 14.5

GC: G1 Young Generation, G1 Old Generation

Memory: 2048M

Cores: 8

Metal Rendering is ON

Registry:

ide.completion.variant.limit=500

suggest.all.run.configurations.from.context=true

Non-Bundled Plugins:

com.intellij.zh (241.230)

SDK: go1.22.2 darwin/arm64

### 2 申请任务

worker通过getTask()向Coordinator请求任务。函数执行成功后，会返回一个包含五个元素的元组：任务所需的文件名列表、任务类型、任务ID、Reduce任务的数量以及Reduce任务的文件。

func getTask() ([]string, string, int, int, int) {

args := GetArgs{} // 请求参数

reply := GetReply{} // 响应结构

ok := call("Coordinator.GetTask", &args, &reply)

if !ok {

fmt.Printf("Worker Request Failed\n")

}

return reply.Filenames, reply.TaskName, reply.TaskId, reply.NReduce, reply.ReduceFile

}

GetTask 方法负责协调任务分配给工作器，包括 Map 和 Reduce 任务，并处理任务超时重新分配的情况。

// [coordinator]GetTask 分配文件给工作器

func (c \*Coordinator) GetTask(args \*GetArgs, reply \*GetReply) error {

c.mu.Lock()

defer c.mu.Unlock()

// 检查所有Map任务是否已经分配但尚未全部完成

if len(c.todoFile) == 0 && c.numFileRemain > 0 {

for len(c.todoFile) == 0 && c.numFileRemain > 0 {

c.cond.Wait()

}

}

// 如果还有待处理的文件

if len(c.todoFile) != 0 {

filename := c.todoFile[0]

c.todoFile = c.todoFile[1:]

reply = c.makeReply(filename, "Map", reply)

c.fileManager[filename] = c.mapId

c.mapId += 1

// 设置超时，如果任务10秒内未被处理，则重新标记为待处理

// 匿名函数的声明和立即调用

go func(filename string) {

time.Sleep(10 \* time.Second)

c.mu.Lock()

defer c.mu.Unlock()

if c.fileManager[filename] > -2 {

c.fileManager[filename] = -1

c.todoFile = append(c.todoFile, filename)

c.cond.Broadcast()

}

}(filename)

} else if c.numFileRemain == 0 {

// 若所有Map任务已完成，处理Reduce任务

if len(c.reduceFile) == 0 && c.numReduceFileRemain > 0 {

for len(c.reduceFile) == 0 && c.numReduceFileRemain > 0 {

c.cond.Wait()

}

}

// 若有Reduce任务失败，重新分配

if len(c.reduceFile) != 0 {

reduceFile := c.reduceFile[0]

c.reduceFile = c.reduceFile[1:]

reply.ReduceFile = reduceFile

reply.TaskName = "Reduce"

reply.TaskId = c.reduceId

c.intermediateFileManager[reduceFile] = c.reduceId

c.reduceId++

// 设置超时

go func(reduceFile int) {

time.Sleep(10 \* time.Second)

c.mu.Lock()

defer c.mu.Unlock()

if c.intermediateFileManager[reduceFile] > -2 {

c.intermediateFileManager[reduceFile] = -1

c.reduceFile = append(c.reduceFile, reduceFile)

c.cond.Broadcast()

}

}(reduceFile)

}

}

// 检查所有任务是否完成

if c.numFileRemain == 0 && c.numReduceFileRemain == 0 {

//fmt.Printf("Assignining Termination to Worker\n")

reply.TaskName = "Terminate"

}

return nil

}

### 3 任务完成

worker通知coordinator一个Map任务已完成

func submitTask(filename string, taskId int) bool {

args := SubmitArgs{Filename: filename, TaskId: taskId} // 提交参数

reply := SubmitReply{} // 回复结构

ok := call("Coordinator.SubmitTask", &args, &reply)

if !ok || !reply.Ok {

fmt.Printf("Task submit failed\n")

return false

}

return true

}

worker通知coordinator一个Reduce任务已完成

func submitReduceTask(reduceFile int, taskId int) bool {

args := SubmitArgs{ReduceFile: reduceFile, TaskId: taskId} // 提交参数

reply := SubmitReply{} // 回复结构

ok := call("Coordinator.SubmitReduceTask", &args, &reply)

if !ok || !reply.Ok {

fmt.Printf("Task submit failed\n")

return false

}

return true

}

coordinator提交Map任务结果

func (c \*Coordinator) SubmitTask(args \*SubmitArgs, reply \*SubmitReply) error {

submitFilename := args.Filename

c.mu.Lock()

defer c.mu.Unlock()

// 首次提交，标记任务为已写入

if c.fileManager[submitFilename] == args.TaskId {

c.fileManager[submitFilename] = -2

reply.Ok = true

return nil

} else if c.fileManager[submitFilename] == -2 {

// 二次提交，标记任务为成功完成

c.fileManager[submitFilename] = -3

c.numFileRemain-- // 减少剩余文件数量

reply.Ok = true

c.cond.Broadcast() // 通知等待的线程

return nil

}

reply.Ok = false // 任务ID不匹配，返回失败

return nil

}

coordinator提交Reduce任务结果

func (c \*Coordinator) SubmitReduceTask(args \*SubmitArgs, reply \*SubmitReply) error {

reduceNum := args.ReduceFile

taskId := args.TaskId

c.mu.Lock()

defer c.mu.Unlock()

// 首次提交Reduce任务，标记为已写入

if c.intermediateFileManager[reduceNum] == taskId {

c.intermediateFileManager[reduceNum] = -2

reply.Ok = true

return nil

} else if c.intermediateFileManager[reduceNum] == -2 {

// 二次提交，标记为成功完成

c.intermediateFileManager[reduceNum] = -3

reply.Ok = true

c.numReduceFileRemain-- // 减少剩余Reduce文件数量

c.cond.Broadcast() // 通知等待的线程

return nil

}

reply.Ok = false // 任务ID不匹配，返回失败

return nil

}

### 4 定义任务和信息交换含义

// GetArgs 请求任务 无需字段

type GetArgs struct {

}

// GetReply 响应worker请求任务

type GetReply struct {

Filenames []string //文件名列表

TaskName string // Map/Reduce/Terminate

TaskId int //任务ID

NReduce int //Reduce总任务数

ReduceFile int //如果是Reduce,它的中间结果文件

}

// SubmitArgs 提交任务

type SubmitArgs struct {

Filename string

TaskId int

ReduceFile int //如果是Reduce,它的中间结果文件

}

// SubmitReply 响应任务提交

type SubmitReply struct {

Ok bool // 任务成功接受

}

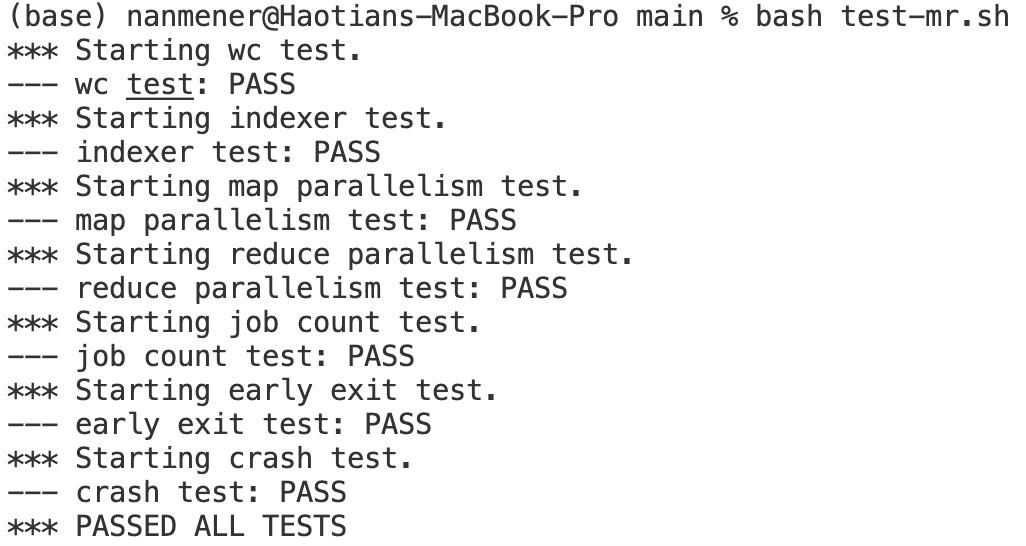
### 5 测试

为了在MacOS上运行测试脚本，需要先安装 GNU coreutils。

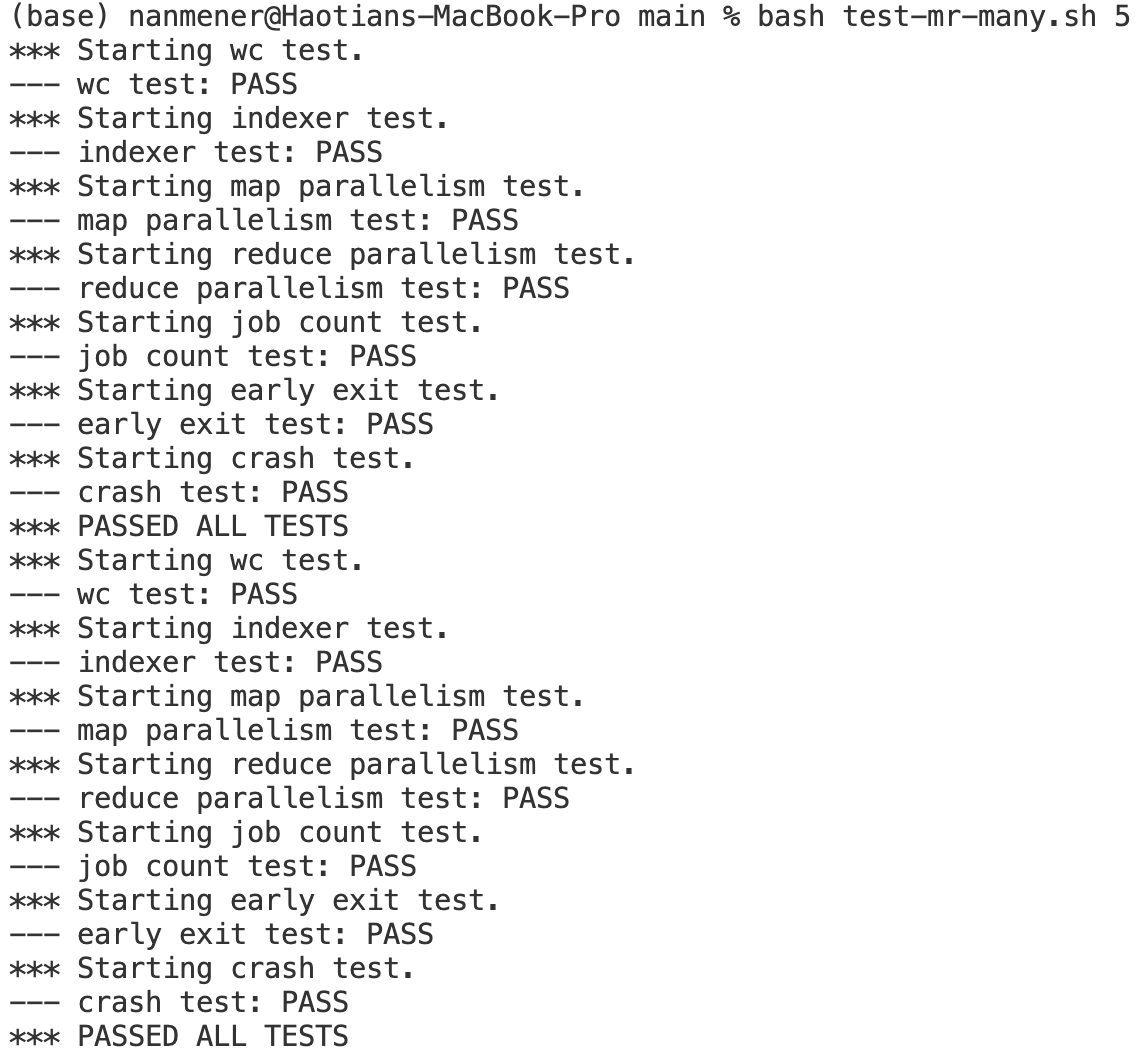
brew install coreutils

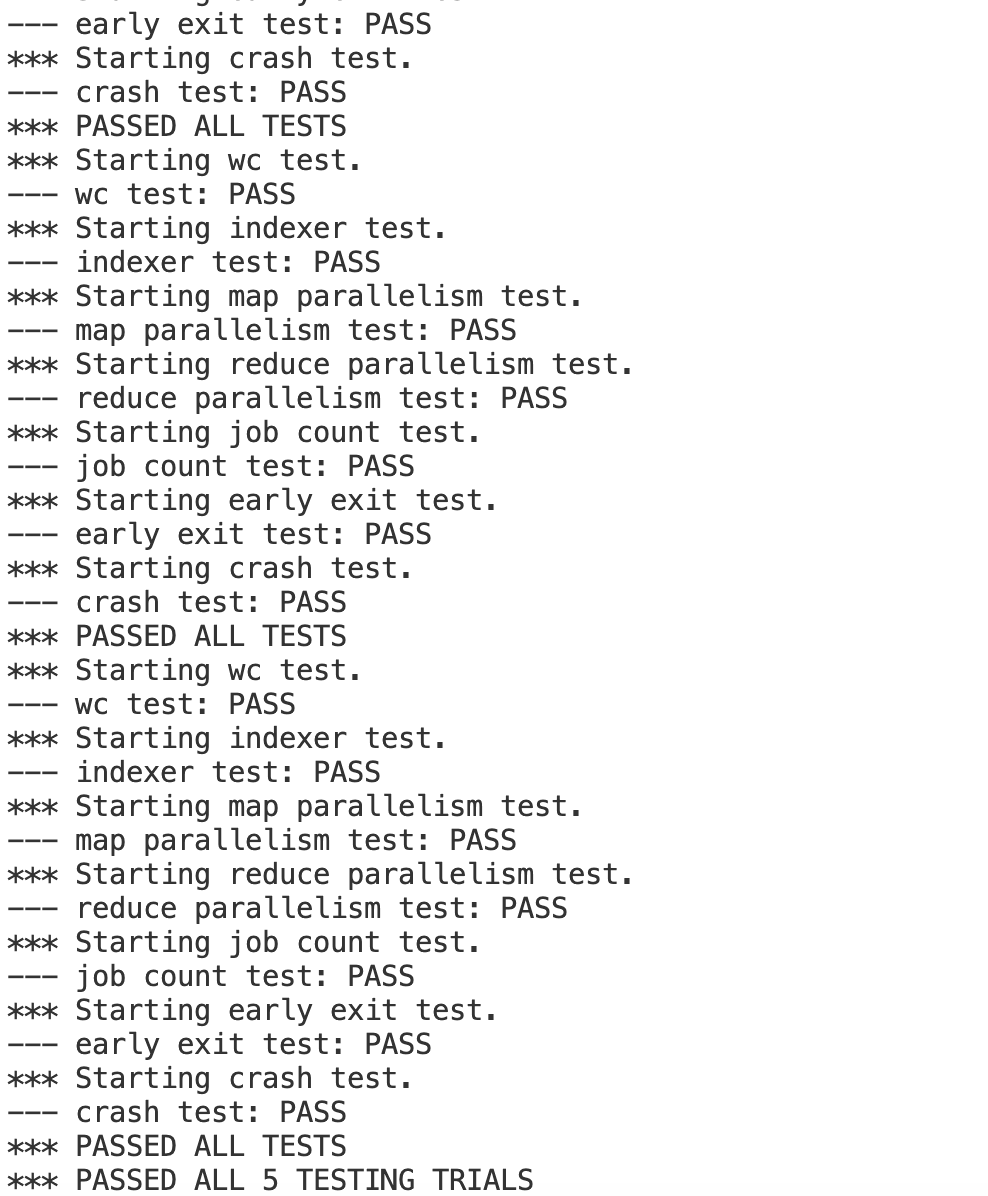
安装完成后，timeout 命令可作为 gtimeout 使用

运行测试脚本test-mr.sh，发现可以通过测试。



随后进行多次测试，均成功通过。





### 6 附录

其他重要函数如下：

// MakeCoordinator 创建一个协调器实例

// main/mrcoordinator.go调用此函数

// nReduce 是使用的Reduce任务数量

func MakeCoordinator(files []string, nReduce int) \*Coordinator {

c := Coordinator{}

c.fileManager = make(map[string]int) // 文件管理器

c.intermediateFileManager = make(map[int]int) // 中间文件管理器

c.todoFile = files // 待处理文件列表

c.mapId = 0 // Map任务ID从0开始

c.nReduce = nReduce // Reduce任务数量

c.numFileRemain = len(files) // 剩余文件数量

c.cond = \*sync.NewCond(&c.mu) // 条件变量

c.reduceFile = make([]int, nReduce) // Reduce文件列表

for i := 0; i < nReduce; i++ {

c.reduceFile[i] = i // 分配Reduce任务

}

c.numReduceFileRemain = nReduce

c.server()

return &c

}

// [worker]中间文件写入

func writeToIntermediate(intermediateBuckets map[int][]KeyValue, taskId int) error {

for k, v := range intermediateBuckets {

sort.Sort(ByKey(v))

tmpFile, err := ioutil.TempFile("", "temp-\*")

if err != nil { log.Fatal(err) }

defer tmpFile.Close()

oname := fmt.Sprintf("mr-%v-%v", taskId, k)

enc := json.NewEncoder(tmpFile)

for \_, kv := range v {

err := enc.Encode(&kv)

if err != nil {

fmt.Printf("error in encoding kv, %v , %v\n", taskId, k)

os.Remove(tmpFile.Name())

return err

}

}

os.Rename(tmpFile.Name(), oname)

}

return nil

}

// [worker]Worker 循环请求任务、执行任务、提交结果

func Worker(mapf func(string, string) []KeyValue,

reducef func(string, []string) string) {

// 不断请求和执行任务

for {

filenames, taskName, taskId, nReduce, reduceFile := getTask() //请求一个任务

if taskName == "Map" {

filename := filenames[0]

intermediate := []KeyValue{}

kva := mapf(filename, mapFile(filename))

intermediate = append(intermediate, kva...) // kva 追加到 intermediate

intermediateBuckets := make(map[int][]KeyValue)

for \_, v := range intermediate {

reduceId := ihash(v.Key) % nReduce

intermediateBuckets[reduceId] =

append(intermediateBuckets[reduceId], v)

}

// 提交Map任务

if submitTask(filename, taskId) {

// 中间结果写入

err := writeToIntermediate(intermediateBuckets, taskId)

if err != nil {

fmt.Printf("Failure: %v \n", err)

} else {

submitTask(filename, taskId)

}

}

} else if taskName == "Reduce" {

KVCollect := []KeyValue{}

oname := fmt.Sprintf("mr-out-%v", taskId)

tempOfile, \_ := ioutil.TempFile("", "mr-out-\*")

defer tempOfile.Close()

pattern := fmt.Sprintf("mr-\*-%v", reduceFile)

files, err := filepath.Glob(pattern)//pattern匹配的所有文件的路径

if err != nil {

fmt.Println("Error:", err)

return

}

for \_, file := range files {

f, err := os.Open(file)

if err != nil { break }

dec := json.NewDecoder(f)

for {

var kv KeyValue

// 解码键值对

if err := dec.Decode(&kv); err != nil {

break

}

KVCollect = append(KVCollect, kv)

}

}

sort.Sort(ByKey(KVCollect))

// 组织成值的列表 from ../mrsequentail.go

i := 0

for i < len(KVCollect) {

j := i + 1

for j < len(KVCollect) && KVCollect[j].Key == KVCollect[i].Key {

j++

}

values := []string{}

for k := i; k < j; k++ {

values = append(values, KVCollect[k].Value)

}

output := reducef(KVCollect[i].Key, values)

fmt.Fprintf(tempOfile, "%v %v\n", KVCollect[i].Key, output)

i = j

}

// 提交Reduce任务

if submitReduceTask(reduceFile, taskId) {

// 临时文件重命名

os.Rename(tempOfile.Name(), oname)

submitReduceTask(reduceFile, taskId)

} else {

os.Remove(tempOfile.Name())

}

} else if taskName == "Terminate" {

os.Exit(0)

}

}

}