

Relatório da Atividade 3:

Map Reduce

Isabelle Ferreira de Oliveira
CES-27 - Engenharia da Computação 2020
Instituto Tecnológico de Aeronáutica (ITA)
São José dos Campos, Brasil
isabelle.ferreira3000@gmail.com

Resumo—Esse relatório documenta um trabalho com o modelo de programação MapReduce de forma sequencial e distribuída.

Index Terms—Map Reduce, Golang, sistema sequencial, sistema distribuído

I. IMPLEMENTAÇÃO

A. Parte 1: Trabalhando em modo sequencial

1) *Tarefa 1.1:* Bastou percorrer a lista de palavras presentes em *words* e, para cada palavra, adicionar a um array de *mapreduce.KeyValue* um novo item, contendo a palavra em questão como atributo *Key* e "1" como atributo *Value*.

```
func mapFunc(input [][]byte) (result
    []mapreduce.KeyValue) {

    // [...]

    var item mapreduce.KeyValue

    for _, word := range words {
        word = strings.ToLower(word)
        item.Key = word
        item.Value = "1"
        result = append(result, item)
    }

    return result
}
```

2) *Tarefa 1.2:* Bastou percorrer o array *input* de *mapreduce.KeyValue*, atualizando um mapa a fim de agrupar as palavras repetidas nesse *input*. O atributo *Key* de cada item desse *input* era a palavra que serviria de chave nesse mapa. Caso a palavra já estivesse no mapa, acrescentava-se o valor contido no atributo *Value* do item ao valor já presente no mapa. Caso contrário, esse novo item era adicionado no mapa, com o atributo *Value* sendo setado como valor inicial dessa chave. Após isso, percorreu-se esse mapa, adicionando os elementos no array *result* de *mapreduce.KeyValue*. Era necessário, também, fazer as conversões de inteiro para string, e vice-versa, para adicionar corretamente nos mapas e no array *result* de *mapreduce.KeyValue*.

```
func reduceFunc(input []mapreduce.KeyValue)
    (result []mapreduce.KeyValue) {
```

```
    var mapAux map[string]int =
        make(map[string]int)
    var value int

    for _, item := range input {
        value, _ = strconv.Atoi(item.Value)
        _, ok := mapAux[item.Key]
        if ok {
            mapAux[item.Key] += value
        } else {
            mapAux[item.Key] = value
        }
    }

    var itemAux mapreduce.KeyValue

    for key, value := range mapAux {
        itemAux.Key = key
        itemAux.Value = strconv.Itoa(value)
        result = append(result, itemAux)
    }

    return result
}
```

3) *Tarefa 1.3:* Executou-se o comando *./wordcount -mode sequential -file files/teste.txt -chunksize 100 -reducejobs 2*, conforme sugerido no roteiro.

4) *Tarefa 1.4:* Executou-se o comando da Tarefa 1.3, dessa vez para o arquivo *files/musica.txt*. Os valores de *chunksize* e *reducejob* foram alterados para: 100 e 2; 100 e 1; e 102400 e 2. O arquivo *musica.txt* pode ser observado no código enviado em anexo ao relatório.

B. Parte 2: Trabalhando em modo distribuído

1) *Tarefa 2.1:* Bastou, para cada elemento presente no canal *master.failedWorkerChan*, deletar esse worker que falhou da lista de workers *master.workers* e decrescer o total de workers disponível *master.totalWorkers*. Antes e depois dessa alteração, foi utilizado o *Lock()* e o *Unlock()*, respectivamente, do *master.workersMutex*, a fim de evitar acesso simultâneo à região crítica.

```
func (master *Master) handleFailingWorkers() {

    for elem := range master.failedWorkerChan {
        master.workersMutex.Lock()
        delete(master.workers, elem.id)
```

```

master.totalWorkers--
master.workersMutex.Unlock()

fmt.Println("Removing worker", elem.id,
"from master list")
}
}

```

2) *Tarefa 2.2:* Primeiramente foi necessário criar um novo canal (*failedOperationChan*) na *Master*, para transmitir a operação que falhou. É importante salientar também a inicialização do canal *master.failedOperationChan*.

```

type Master struct {
    // [...]

    // Fault Tolerance
    failedOperationChan chan *Operation
}

```

Era necessário também adicionar a operação falhada ao canal quando essa operação falhasse. Também foi removido o *wg.Done()* desse momento, uma vez que essa operação não foi realmente concluída.

```

func (master *Master)
    runOperation(remoteWorker *RemoteWorker,
        operation *Operation, wg *sync.WaitGroup)
{
    // [...]

    if err != nil {
        log.Printf("Operation %v '%v' Failed.
            Error: %v\n", operation.proc,
            operation.id, err)

        master.failedWorkerChan <- remoteWorker
        master.failedOperationChan <- operation
    } else {
        wg.Done()
        master.idleWorkerChan <- remoteWorker
    }
}

```

Foi criada uma função para ser rodada em thread que lidasse com as operações falhadas (função *handleFailingOperations*). Essa função ficava a espera de alguma operação ser colocada no canal *master.failedOperationChan* e, cada vez que isso acontecia, esperava-se o próximo worker ser adicionado ao canal *master.idleWorkerChan*, a fim de essa operação falhada ser atribuída a esse worker. Essa nova atribuição era feita através da chamada de uma nova thread executando *master.runOperation*.

```

func (master *Master)
    handleFailingOperations(wg
        *sync.WaitGroup) {

    var operation *Operation
    var worker *RemoteWorker
    var ok bool

```

```

for {
    operation, ok =
        <-master.failedOperationChan
    if !ok { break }

    worker, _ = <-master.idleWorkerChan

    go master.runOperation(worker, operation,
        wg)
}
}

```

Essa função *master.handleFailingOperations* foi chamada em thread na função *schedule*.

```

func (master *Master) schedule(task *Task,
    proc string, filePathChan chan string)
    int {

    var (
        wg      sync.WaitGroup
        filePath string
        worker   *RemoteWorker
        operation *Operation
        counter int
    )

    master.failedOperationChan = make(chan
        *Operation, RETRY_OPERATION_BUFFER)

    go master.handleFailingOperations(&wg)

    // [...]
}

```

3) *Tarefa 2.3:* Executou-se o comando *./wordcount -mode sequential -file files/teste.txt -chunksize 100 -reducejobs 2*, além de dois workers. Na primeira situação, um dos workers falhou na operação 2 (map operation). Já na segunda situação, um dos workers falhou na operação 3 (reduce operation).

4) *Tarefa 2.4:* Executou-se o comando *./wordcount -mode sequential -file files/pg1342.txt -chunksize 102400 -reducejobs 5* em um primeiro momento; e o mesmo comando com a diferença de ter apenas 1 reducejob em um segundo momento. Em ambas as situações, dois workers falharam e um terceiro não falhou.

II. RESULTADOS E CONCLUSÕES

A. Teste 1

Este foi o caso com um processo solicitando a CS e, depois que ele liberasse, outro processo solicitando a CS, sugerido no roteiro do laboratório. O esquema do resultado esperado foi apresentado na Figura 1.

Na Figura 1, para P1, as setas azuis representam requests e as verdes, replies; para P3, essas setas são rosas e cinzas, respectivamente. Nos requests, a mensagem enviada é da forma *"relógio lógico, < timestamp, id >"*; já no reply, a forma é *"relógio lógico, 'reply'"*. Além disso, a linha amarela representa o processo no estado WANTED; e a linha vermelha, no estado HELD, ou seja, na CS.

```
isabelle@isabelle-Inspiron-5448 ~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce/wordcount
$ ./wordcount -mode sequential -file files/teste.txt -chunksi
2019/09/13 18:16:46 Running in sequential mode.
2019/09/13 18:16:46 Running RunSequential...
2019/09/13 18:16:46 Fanning in file map/map-0
2019/09/13 18:16:46 Fanning in file map/map-1
2019/09/13 18:16:46 Fanning out file result/result-0
2019/09/13 18:16:46 Fanning out file result/result-1
```

Figura 1. Teste

Foi simulada essa situação acima co
mentado no laboratório, tendo os resulta
Figuras de 2 a 5. Na simulação, ao in
requisição, é o P2 a faz, mas isso não p

```
isabelle@isabelle-Inspiron-5448 ~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce/wordcount
$ ./wordcount -mode sequential -file files/musica.txt
2019/09/13 18:18:27 Running in sequential mode.
2019/09/13 18:18:27 Running RunSequential...
2019/09/13 18:18:27 Fanning in file map/map-0
2019/09/13 18:18:27 Fanning in file map/map-1
2019/09/13 18:18:27 Fanning in file map/map-2
2019/09/13 18:18:27 Fanning in file map/map-3
2019/09/13 18:18:27 Fanning in file map/map-4
2019/09/13 18:18:27 Fanning in file map/map-5
2019/09/13 18:18:27 Fanning in file map/map-6
2019/09/13 18:18:27 Fanning in file map/map-7
2019/09/13 18:18:27 Fanning in file map/map-8
2019/09/13 18:18:27 Fanning in file map/map-9
2019/09/13 18:18:27 Fanning in file map/map-10
2019/09/13 18:18:28 Fanning out file result/result-0
2019/09/13 18:18:28 Fanning out file result/result-1

isabelle@isabelle-Inspiron-5448 ~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce/wordcount
$ ./wordcount -mode distributed -type master -file files/teste.txt -chunksi 100 -reducejobs 2
2019/09/13 18:48:49 Running in distributed mode.
2019/09/13 18:48:49 NodeType: master
2019/09/13 18:48:49 Reduce Jobs: 2
2019/09/13 18:48:49 Address: localhost
2019/09/13 18:48:49 Port: 5000
2019/09/13 18:48:49 File: files/teste.txt
2019/09/13 18:48:49 Chunk Size: 100
2019/09/13 18:48:49 Running Master on localhost:5000
2019/09/13 18:48:49 Scheduling Worker.RunMap operations
2019/09/13 18:48:49 Accepting connections on 127.0.0.1:5000
2019/09/13 18:48:53 Registering worker '0' with hostname 'localhost:50001'
2019/09/13 18:48:53 Running Worker.RunMap (ID: '0' File: 'map/map-0' Worker: '0')
2019/09/13 18:48:53 Running Worker.RunMap (ID: '1' File: 'map/map-1' Worker: '0')
2019/09/13 18:48:54 Operation Worker.RunMap '1' Failed. Error: dial tcp 127.0.0.1:50001: connect: c
fused
Removing worker 0 from master list
2019/09/13 18:49:00 Registering worker '0' with hostname 'localhost:50001'
2019/09/13 18:49:00 Running Worker.RunMap (ID: '1' File: 'map/map-1' Worker: '0')
2019/09/13 18:49:00 2x Worker.RunMap operations completed
2019/09/13 18:49:00 Scheduling Worker.RunReduce operations
2019/09/13 18:49:00 Running Worker.RunReduce (ID: '0' File: 'reduce/reduce-0' Worker: '0')
2019/09/13 18:49:00 Running Worker.RunReduce (ID: '1' File: 'reduce/reduce-1' Worker: '0')
2019/09/13 18:49:00 2x Worker.RunReduce operations completed
2019/09/13 18:49:00 Closing Remote Workers.
2019/09/13 18:49:00 Done.

isabelle@isabelle-Inspiron-5448 ~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce/wordcount
$ ./wordcount -mode distributed -type master -file files/musica.txt -chunksi 100 -reducejobs 3
2019/09/13 18:49:00 Running reduce id: 0, p
reduce-0
2019/09/13 18:49:00 Running reduce id: 1, p
reduce-1
2019/09/13 18:49:00 Done.
2019/09/13 18:49:00 Failed to accept connec
accept tcp 127.0.0.1:50001: use of closed
nection
2019/09/13 18:49:00 Stopped accepting connec
isabelle@isabelle-Inspiron-5448 ~/Graduac
liana-CES-27/Lab3/src/labMapReduce/wordcount
$
```

Figura 2. Exemplo do funcionamento da tarefa c
processo 1.

```
isabelle@isabelle-Inspiron-5448 ~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce
$ ./wordcount -mode sequential -file files/musica.txt -chunksi 100 -reducejobs 3
2019/09/13 18:20:03 Running in sequential mode.
2019/09/13 18:20:03 Running RunSequential...
2019/09/13 18:20:03 Fanning in file map/map-0
2019/09/13 18:20:03 Fanning in file map/map-1
2019/09/13 18:20:03 Fanning in file map/map-2
2019/09/13 18:20:03 Fanning in file map/map-3
2019/09/13 18:20:03 Fanning in file map/map-4
2019/09/13 18:20:03 Fanning in file map/map-5
2019/09/13 18:20:03 Fanning in file map/map-6
2019/09/13 18:20:03 Fanning in file map/map-7
2019/09/13 18:20:03 Fanning in file map/map-8
2019/09/13 18:20:03 Fanning in file map/map-9
2019/09/13 18:20:03 Fanning in file map/map-10
2019/09/13 18:20:04 Fanning out file result/result-0
```

Figura 3. Exemplo do funcionamento da tarefa com 3 processos. Tela do
processo 2.

```
isabelle@isabelle-Inspiron-5448 ~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduc
$ ./wordcount -mode sequential -file files/musica.txt -chunksi 102400 -reducejobs 2
2019/09/13 18:27:54 Running in sequential mode.
2019/09/13 18:27:54 Running RunSequential...
2019/09/13 18:27:54 Fanning in file map/map-0
2019/09/13 18:27:54 Fanning out file result/result-0
2019/09/13 18:27:54 Fanning out file result/result-1
```

Figura 4. Exemplo do funcionamento da tarefa com 3 processos. Tela do
processo 3.

Figura 5. Exemplo do funcionamento da tarefa com 3 processos. Tela do
SharedResource.

A fim de entender melhor cada estágio do funcionamento,
foram realizados os mesmos testes novamente, agora com
prints de debug. Dessa forma, esses resultados estão apresen-
tados nas Figuras de 6 a ??.


```
on-5448: ~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce/wordcount para essas mensagens, foi dado o feedback de mensagem
~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce/wordcount
panic: Induced failure.

goroutine 26 [running]:
labMapReduce/mapreduce.(*Worker).RunMap(0xc000066420, 0xc00000ec40, 0xb28978, 0x0, 0x0)
/home/isabelle/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce/mapreduce/worker_rpc.go:25 +0x27b
reflect.Value.call(0xc0000664e0, 0xc000013578, 0x13, 0x7f0b76, 0x4, 0xc000050f18, 0x3, 0x3, 0xc00010e280, 0xb9cb80, ...)
/usr/local/go/src/reflect/value.go:447 +0x461
reflect.Value.Call(0xc0000664e0, 0xc000013578, 0x13, 0xc000063a718, 0x3, 0x3, 0xc000088801, 0xc000079080)
/usr/local/go/src/reflect/value.go:308 +0xa4
net/rpc.(*service).call(0xc000065b80, 0xc000082320, 0xc00010aed0, 0xc00010ae0, 0xc0000e6180, 0xc00000f2e0, 0x75dd80, 0xc00000ec40, 0x16, 0x763a40, ...)
/usr/local/go/src/net/rpc/server.go:384 +0x14e
created by net/rpc.(*Server).ServeCodec
/usr/local/go/src/net/rpc/server.go:481 +0x42b

isabelle@isabelle-Inspiron-5448 ~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce/wordcount <master*>
$

~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce/wordcount
2019/09/13 19:03:44 NodeType: worker
2019/09/13 19:03:44 Address: localhost
2019/09/13 19:03:44 Port: 50003
2019/09/13 19:03:44 Master: localhost:5000
2019/09/13 19:03:44 Running Worker on localhost:50003
2019/09/13 19:03:44 Registering with Master
2019/09/13 19:03:44 Registered. WorkerId: 0 (Settings = (ReduceJobs: 5))
2019/09/13 19:03:44 Accepting connections on 127.0.0.1:50003
2019/09/13 19:03:44 Running reduce id: 1, path: reduce/reduce-1
2019/09/13 19:03:44 Running reduce id: 0, path: reduce/reduce-0
2019/09/13 19:03:44 Running reduce id: 2, path: reduce/reduce-2
2019/09/13 19:03:44 Running reduce id: 3, path: reduce/reduce-3
2019/09/13 19:03:44 Running reduce id: 4, path: reduce/reduce-4
2019/09/13 19:03:44 Done.
2019/09/13 19:03:44 Failed to accept connection. Error: accept tcp 127.0.0.1:50003: use of closed network connection
2019/09/13 19:03:44 Stopped accepting connections.
isabelle@isabelle-Inspiron-5448 ~/Graduacao/hirata-juliana-CES-27/Lab3/src/labMapReduce/wordcount <master*>
$
```

Figura 8. Exemplo do funcionamento da tarefa com 3 processos. Tela do processo 3.

Como esses resultados foram condizentes com os resultados esperados, leva-se a perceber que a implementação da Tarefa foi feita corretamente. Mas, antes de concluir-se algo, foi-se realizado um segundo teste.

B. Teste 2

Este foi o caso com processos solicitando a CS "simultaneamente", sugerido no roteiro do laboratório. O esquema do resultado esperado foi apresentado na Figura ??.

Análogo ao teste 1, na Figura ??, para P1, as setas azuis representam requests e as verdes, replies; e para P4, essa setas são rosas e cinzas, respectivamente. Nos requests, a mensagem enviada também é da forma "*relógio lógico*, < *timestamp*, *id* >"; assim como no reply, que a forma também é "*relógio lógico*, '*reply*'". Além disso, nesse caso também a linha amarela representa o processo no estado WANTED; e a linha vermelha, no estado HELD, ou seja, na CS.

Foi simulada essa situação acima com o código implementado no laboratório, tendo os resultados apresentados nas Figuras de ?? a ??.

Durante o período na CS, também foi digitado mensagens no terminal dos processos, para que se verificasse o funcionamento da validação da mensagem. Assim,