Instituto Tecnológico de Aeronáutica

Departamento de Engenharia de Software
Divisão de Ciência da Computação
Laboratório de CES-27

Relatório da atividade

Laboratório 3:
Map-reduce

Lucas França de Oliveira, lucas.fra.oli18@gmail.com
Professor: Celso Massaki Hirata , hiratacm@gmail.com

30 de outubro de 2017

1 Introdução

Este laboratório tem como objetivo a familiarização com a comunicação via UDP em linguagem Go e implementação do método map-reduce para contagem de palavras em paralelo. A entrada é um arquivo texto e a saída, uma array de pares em que o primeiro valor é uma palavra e o segundo, o número de aparições dela no texto. Este processo é feito de forma distribuída, com a função map servindo para a leitura de arquivos para um resultado intermediário, e a reduce, para a junção de resultados intermediários em um possível resultado final.

O laboratório é dividido em três tarefas, cada uma com um passo na implementação e sendo um requerimento dos passos seguintes. A tarefa 1 é a divisão do arquivo de texto em arquivos menores, de forma a se dividir o trabalho e permitir sua distribuição. A 2 e 3, de implementação das funções *map* e *reduce*, respectivamente.

A maior parte da estrutura da prática já está pronta, sendo implementada no github do ex-aluno Paulo Aguiar: https://github.com/PauloAguiar/ces27-lab1. Já estão codificadas as funções de funcionamento geral do laboratório, sendo necessário apenas a implementação das atividades citadas acima. Também foram fornecidos testes de unidade e um arquivo de teste grande.

2 Implementação

2.1 Função Split

A função de separação do arquivo de entrada, que faz parte do preprocessamento necessário para o map-reduce paralelizado, foi implementada da seguinte maneira: o arquivo é lido palavra por palavra e tudo é armazenado em um buffer com o tamanho máximo de arquivo permitido. Quando o buffer estoura, ele é descarregado em um arquivo.

2.2 Função Map

A função realiza o parsing de uma string por caracteres que não são letras ou números e constrói uma array de pares, em que o primeiro elemento é a palavra e o segundo é a string "1". O par é definido como a struct KeyValue no arquivo common.go. Este processo é feito em $O(n)$, em que n é o número de palavras.

2.3 Função Reduce

A função recebe uma array de pares palavra-frequência e, iterando sobre ela, constrói uma estrutura map da linguagem Go em que a chave é a palavra e o valor é o número de aparições. No final ele itera sobre o map para construir uma nova array em que cada chave aparece uma única vez. Este processo é feito em $O(nlogn)$, em que n é o número de palavras.

O fator logarítmico da complexidade vem do uso da estrutura map, implementada internamente como uma árvore de busca binária, que é a grande vantagem de se usar este método. Uma busca por repetições implementada de forma bruta adicionaria um fator de complexidade linear sobre cada palavra a ser analisada, totalizando a complexidade $O(n^2)$. Utilizando uma estrutura de dados de divisão-e-conquista, consegue-se fazer esse processo em tempo logarítmico, resultando na complexidade $O(nlogn)$, muito mais rápida.

3 Testes

Como pode ser demonstrado a seguir, os testes de unidade fornecidos passaram:

```
C:\Users\Lucas Franca\Documents\CES-27\Lab3\ces27-lab1\wordcount>go test -v
=== RUN   TestSplitData
--- PASS: TestSplitData (0.01s)
        wordcount_test.go:67: Description: text file empty
        wordcount_test.go:67: Description: text files bigger than chunk size
        wordcount_test.go:67: Description: text file smaller than chunk size
        wordcount_test.go:67: Description: text file has exact chunk size
=== RUN   TestMapFunc
--- PASS: TestMapFunc (0.00s)
        wordcount_test.go:133: Description: empty
        wordcount_test.go:133: Description: one word
        wordcount_test.go:133: Description: two words
        wordcount_test.go:133: Description: repeated word
        wordcount_test.go:133: Description: invalid character
        wordcount_test.go:133: Description: newline character
        wordcount_test.go:133: Description: multiple whitespaces
        wordcount_test.go:133: Description: special characters
        wordcount_test.go:133: Description: uppercase characters
=== RUN   TestReduceFunc
--- PASS: TestReduceFunc (0.00s)
        wordcount_test.go:223: Description: no entry
        wordcount_test.go:223: Description: one entry
        wordcount_test.go:223: Description: two entries with same keys
        wordcount_test.go:223: Description: two entries with different keys
        wordcount_test.go:223: Description: non-numeric counter
PASS
ok      _/C_/Users/Lucas_Franca/Documents/CES-27/Lab3/ces27-lab1/wordcount       0.338
    s
```

A execução da função main e com o arquivo fornecido apresentaram a resposta correta:

```
C:\Users\Lucas Franca\Documents\CES-27\Lab3\ces27-lab1\wordcount>wordcount.exe -file
    files/pg1342.txt
2017/10/30 00:35:25 Running in local mode.
2017/10/30 00:35:25 File: files/pg1342.txt
2017/10/30 00:35:25 Reduce Jobs: 1
2017/10/30 00:35:25 Chunk Size: 102400
2017/10/30 00:35:25 Running RunSequential...
2017/10/30 00:35:25 Fanning in file map\map-0
2017/10/30 00:35:25 Fanning in file map\map-1
2017/10/30 00:35:25 Fanning in file map\map-2
2017/10/30 00:35:25 Fanning in file map\map-3
2017/10/30 00:35:25 Fanning in file map\map-4
2017/10/30 00:35:25 Fanning in file map\map-5
2017/10/30 00:35:25 Fanning in file map\map-6
2017/10/30 00:35:25 Storing locally.    MapId: 0        Len: 18544
2017/10/30 00:35:25 Storing locally.    MapId: 1        Len: 18321
2017/10/30 00:35:25 Storing locally.    MapId: 2        Len: 18317
2017/10/30 00:35:25 Storing locally.    MapId: 3        Len: 18307
2017/10/30 00:35:25 Storing locally.    MapId: 4        Len: 18315
2017/10/30 00:35:25 Storing locally.    MapId: 5        Len: 18775
2017/10/30 00:35:25 Storing locally.    MapId: 6        Len: 15493
2017/10/30 00:35:26 Fanning out file result\result-0
```

4 Conclusão

Todos os testes apresentaram sucesso segundo os logs apresentados acima, e o map-reduce para o caso de teste grande executou com sucesso. Com o auxílio dos testes de unidade, pode-se fazer o processo de desenvolvimento orientado a testes, o que permitiu rápido e eficiente debug.

O map-reduce se mostrou simples de implementar e eficiente no cálculo de frequência de palavras, executando com complexidade $O(nlogn)$, em que n é o número de palavras. Com a maior parte do código já pronto, o laboratório proporcionou um aprendizado eficiente e não cansativo. Este laboratório foi bom para o aprendizado e familiarização da linguagem Go e suas API's de IO e de Map. A API se mostrou de fácil uso, sendo um método simples fazer o mapeamento.