Pontifícia Universidade católica do rio de janeiro

Uma infraestrutura para ensino da arquitetura MapReduce em disciplinas de sistemas distribuídos

Elaine Cristina Ferreira dos Santos

Relatório de Projeto Final de Graduação

Centro Técnico Científico - CTC

Departamento de Informática

Curso de Graduação em Engenharia da Computação

Rio de Janeiro, julho de 2022

Elaine Cristina Ferreira dos Santos

Uma infraestrutura para ensino da arquitetura MapReduce em disciplinas de sistemas distribuídos

Relatório de Projeto Final, apresentado ao programa **Engenharia da Computação** da PUC-Rio como requisito parcial para a obtenção do título de Engenheira de Computação.

Orientador: Noemi de la Rocque Rodriguez

Rio de Janeiro
julho de 2022

Ao verme que primeiro roeu as frias carnes do meu cadáver dedico como saudosa lembrança estas memórias póstumas. Machado de Assis(Memórias Póstumas de Brás Cubas)

Resumo

Santos, Elaine Cristina Ferreira dos. Rodriguez, Noemi de la Rocque. Uma infraestrutura para ensino da arquitetura MapReduce em disciplinas de sistemas distribuídos. Rio de Janeiro, 2022. Número de páginas p. Relatório de Projeto Final II – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

Neste trabalho criamos dois frameworks, um local e um distribuído, para a familiarização de alunos de disciplinas de sistemas distribuídos com o MapReduce e a linguagem de programação Go. O framework supõe o uso de goroutines para processamento paralelo local e de chamadas remotas de procedimento (RPC) para o processamento paralelo distribuído. Como aplicações exemplo, usamos contagem de palavras, índice inverso, e estatísticas sobre filmes, usando como bases de dados os textos de Machado de Assis e uma tabela com dados de filmes e séries da plataforma de streaming Netflix. Para o caso local deixamos o MapReduce sequencial pronto para o caso contagem de palavras, e para o caso distribuído (RPC) deixamos uma chamada simples ao servidor prontos, para servir como referência.

Palavras-chave

MapReduce, go, RPC, goroutines, waitgroups, channels, Big Data

Abstract

Santos, Elaine Cristina Ferreira dos. Rodriguez, Noemi de la Rocque.An infrastructure for teaching the MapReduce architecture in distributed systems disciplines. Rio de Janeiro, 2022. Número de páginas p. Relatório de Projeto Final II – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

This project aims to create two frameworks, a local and a distributed, to familiarize students of courses on distributed systems with MapReduce and the Go programming language. The framework assumes the use of goroutines for the parallel local processing and the remote procedure calls (RPC) for the distributed parallel process. As examples of applications we used word counting, inverted index and movie statistics, with databases using texts from Machado de Assis and a table of data about movies and series from the streaming platform Netflix. For the local case we prepared a MapReduce sequential for the case of word counting, and for the distributed (RPC) we prepared a simple call of servers ready, to use as reference.

Keywords

MapReduce, go, RPC, goroutines, waitgroups, channels, Netflix, Machado de Assis, Big Data

Agradecimentos

A minha avó Bernadete Belarmino(in memoriam) por tudo, eu não seria quem eu sou sem ela. A minha mãe Rosangela Almeida por ter sido parte essencial da minha jornada até aqui. Ao meu irmão Luis Paulo dos Santos por sempre apoiar e entender minhas escolhas, a minha irmã do coração Ana Carolina Teixeira por estar do meu lado sempre, até menos nos momentos mais difíceis. Ao meu namorado Ricardo Aranha por sempre me escutar falando da faculdade e por tentar ficar acordado enquanto eu virava a noite pra terminar um trabalho. E aos meu amigos que a PUC-Rio me trouxe: Ana Carolina Esteves, Alice Candeias, Arthus James Erthal, Eduardo Junqueira Wichrowski, Gabriel Andrade, Leonardo Gomes, Lucas Gomes, Rafael Canário, Rafael Gama, Sarakali Nobre por fazerem parte da minha história e não me deixarem desistir. E a Noemi Rodriguez por ter aceitado me orientar, e sempre ser uma solução para quando eu ficava presa em alguma tarefa.

Sumário:

1. Introdução

O estilo de programação MapReduce começou a ganhar popularidade em 2003 depois de proposto por Dean e Ghemawat da Google [1], com o objetivo de processar e criar grandes volumes de dados. Atualmente a versão mais popular é o Apache Hadoop que é usado por grandes empresas como IBM, Oracle, Facebook, Yahoo!. As soluções atuais são em Java (Hadoop) e em C++(Google).

A linguagem de programação GO, também criada pela Google (Robert Griesemer, Rob Pike, e Ken Thompson iniciaram e aos poucos outros funcionários começaram a contribuir com o projeto [12]), surgiu pela necessidade de resolver os problemas da empresa, e também aproveitar as melhoras que tivemos nos computadores, porque os computadores foram evoluindo e as linguagens nem tanto, então eles queriam uma linguagem que tivesse uma forma mais simples de usar paralelismo e concorrência, uma linguagem que fosse mais rápida de compilar, uma linguagem que controlasse as dependências de forma fácil, entre outras [13].

O objetivo deste trabalho é criar uma infraestrutura que pudesse ser utilizada para o aprendizado sobre MapReduce, utilizando a linguagem Go, com uso de RPC e processamento concorrente.

2. Situação Atual

O MapReduce foi criado com o intuito de melhorar as buscas na Google, já que para fazer as buscas rápidas pela qual a empresa ficou conhecida era necessário analisar e processar uma quantidade muito grande de dados.

Em artigo de 2004 chamado “MapReduce: Simplified Data Processing on Large Clusters” [1], Dean e Ghemawat explicaram como esse framework funciona e mostraram diversos exemplos de uso.

A ideia por trás da fase Map é transformar os dados que nós queremos mapear em tuplas de chave e valor, e depois na fase Reduce, reduzir e unir todas as tuplas com a mesma chave e produzir uma saída no final do processo. E para atingir o processamento em um tempo reduzido é usado o paralelismo para dividir a carga de dados, então cada componente é responsável por processar uma parte pequena dos dados, ao invés de dividir por etapas de processamento.

O exemplo mais simples é o de contagem de palavras que sempre aparece em qualquer pesquisa sobre MapReduce. O input para o Map é um texto: a função mapper vai separar as palavras em tuplas chave e valor, a chave

vai ser a palavra e o valor vai ser um. O o input para o Reduce serão essas tuplas de chave e valor, as tuplas com o mesmo hash vão estar no mesmo arquivo, assim conseguimos juntas as palavras iguais, essa contagem será o output, que é a contagem de quantas vezes aquela palavra aparece no texto.

Atualmente existem muitas infra estruturas prontas para o uso comercial, mas não temos muitas infra estruturas prontas para alunos interessados em aprender como o MapReduce funciona. Encontramos um material oferecido pela Universidade de Boston no curso CS451/651 Distributed Systems [3] (vamos nos referir a esse curso como CS-451-UB).

O objetivo do CS-451-UB é criar uma biblioteca MapReduce com a meta de auxiliar alunos a se familiarizarem com a linguagem de programação Go. Todo o código deve ser feito pelo aluno, com exceção do que já foi dado. Só que o framework disponibilizado para o curso não contém apenas as tarefas sobre MapReduce, que é apenas um dos assuntos abordados. E o nosso trabalho será apenas sobre o MapReduce, que diminuirá a quantidade de pacotes e facilitará a navegação pelo código, além de deixar claro como cada coisa está funcionando.

Além disso queremos dar as ferramentas para os alunos se familiarizarem com Go, que é um linguagem bem recente, com menos de 20 anos, (a título de comparação, a linguagem C tem por volta de 50 anos [14]) e está sendo usada em empresas como a Nokia, SoundCloud, Heroku entre outras [13]. Também quisemos criar a infraestrutura necessária para que os alunos possam explorar o conceito de RPC na criação de aplicações distribuídas.

3. Objetivos

Vamos criar dois frameworks, um para o uso de MapReduce completamente local e outro para o uso com RPC com o cliente e servidor, com algumas partes faltando para serem completadas por alunos interessados em se familiarizarem com MapReduce, linguagem Go e aplicações distribuídas. Teremos no framework três casos de uso: contagem de palavras, índice inverso e dados Netflix.

O MapReduce local é onde vamos rodar a aplicação sequencialmente depois de forma concorrente com o uso de goroutines. O MapReduce com RPC vai ser usado para simular o processamento em várias máquinas, todos os servidores estarão na mesma máquina assim não precisando de um gerenciador de arquivos distribuídos.

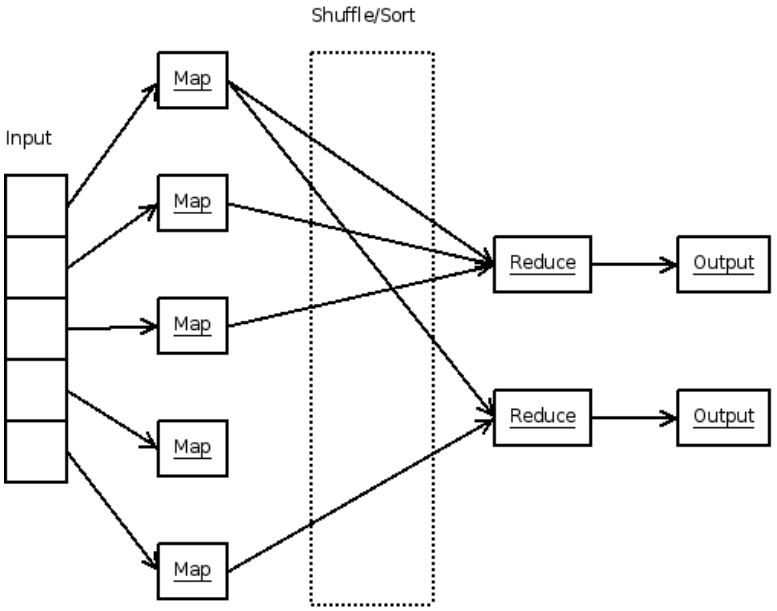
Para auxiliar os alunos na infraestrutura do MapReduce local deixaremos

como referência o MapReduce sequencial para caso de uso contagem de palavras completo, e teremos uma comunicação bem simples entre o cliente e o servidor pronto no framework do MapReduce com RPC.

4. Atividades realizadas

4.1. Estudos preliminares sobre MapReduce

Essa é a parte que menos aparece no trabalho e a parte que durou menos tempo, mas foi a base para tudo, necessária para entender o que é o MapReduce, como funciona e quais são os requisitos necessários para o funcionamento desse estilo de programação.



A forma mais simples de entender como funciona é por meio da imagem anterior[18]. Todos os arquivos usados, sendo input ou output, sempre são manipulados por um gerenciador de arquivos, o que não criamos nesse trabalho. Achamos melhor focar em fazer MapReduce com métodos diferentes e inputs diferentes, já que achamos que seria a menor curva de aprendizado: Usar o Hadoop Distributed File System(HDFS) [5] seria acrescentar uma complexidade a mais ao nosso trabalho final, e criar um sistemas para arquivos distribuídos simplificado já seria um projeto diferente.

O módulo shuffle/sort vai ser explicado com mais detalhe quando falarmos sobre ela no nosso framework final.

Na parte de desenvolvimento começamos fazendo os exercícios no arcabouço dado no CS-451-UB sobre MapReduce em GO. Inicialmente iríamos partir do trabalho da Universidade de Boston para desenvolver nosso projeto, mas como o repositório do trabalho original fazia parte de um curso maior sobre

sistemas distribuídos encontramos alguns desafios. Não podíamos usar o gerenciador de pacotes do Go e o repositório do CS-451-UB tem muitos pacotes e arquivos, o que dificultou a nossa compreensão.

Então mudamos a nossa abordagem, construindo do zero nosso código. Montamos uma infraestrutura que poderá ser usada futuramente em aulas, ou em cursos sobre o assunto, e que, assim como o CS-451-UB, será incremental, com testes para verificar se o algoritmo está funcionando com o esperado.

4.2. Estudo sobre RPC

Para a comunicação entre as partes envolvidas na execução paralela do programa, optamos por usar o modelo de chamada remota de procedimento (RPC). Assim, uma etapa inicial do trabalho foi estudar sobre o uso de RPC em Go.

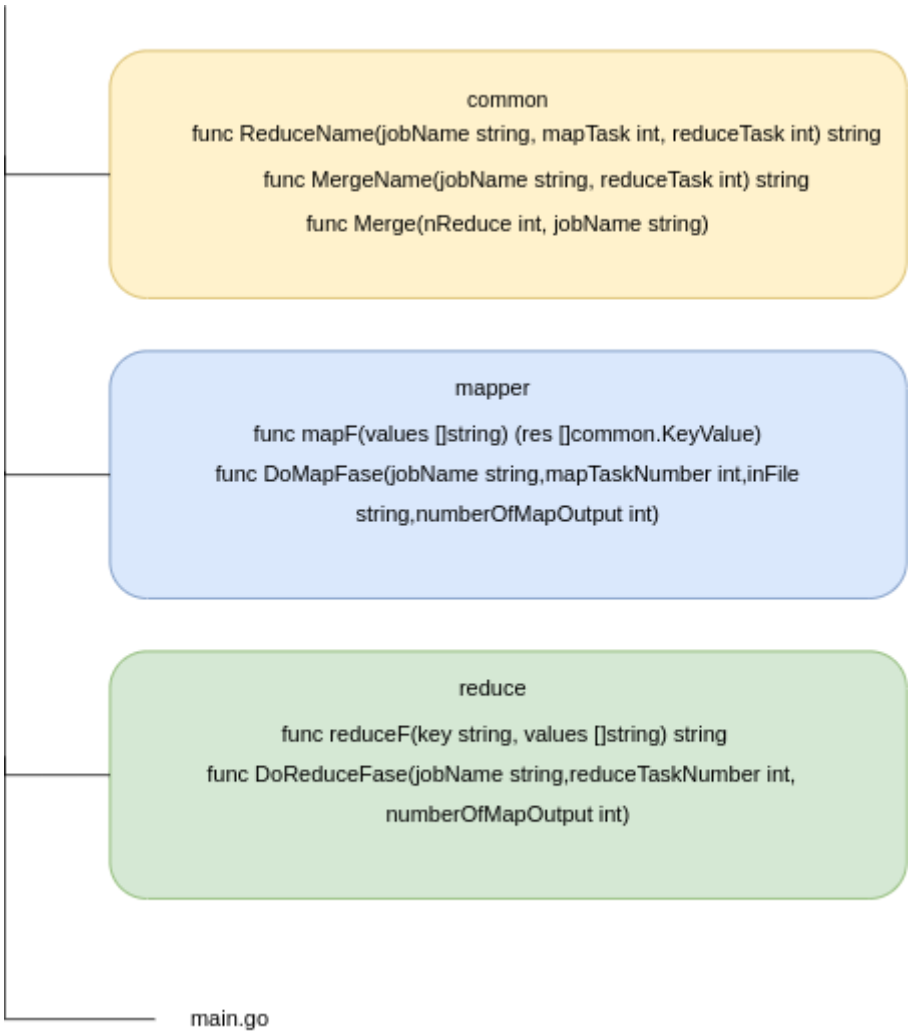
Seguindo um tutorial no Youtube [15] criamos um servidor RPC exemplo, ainda sem relação com o nosso projeto, com métodos que simulam um CRUD sobre uma lista, assim não adicionando a complexidade de um acesso a banco de dados.

Depois seguimos a regra da linguagem para chamadas remotas: precisamos ter duas variáveis, sendo a segunda um ponteiro, e o retorno ser um tipo error. Então pudemos criar a função main que rodava o servidor.

Para chamarmos os métodos de forma remota usamos o Register, função da biblioteca RPC de Go. Colocamos um print, para exibir uma mensagem no servidor cada vez que um método fosse chamado, assim criando uma forma simples de log. E também criamos um cliente e o deixamos encapsulado em um módulo separado da função main.

4.3. MapReduce Sequencial

Dividimos em três módulos, o mapper (map é uma palavra reservada), o reduce e o common, e também temos uma main que só faz as chamadas das funções.



O módulo common é onde definimos o tipo usado durante o programa,o KeyValue, também tem outras funções utilizadas em várias partes do código. O mapper e reduce é onde fazemos todo processamento necessário para cada fase e temos as funções que devem ser alteradas para cada caso de uso. Esse framework é reutilizável, só tendo a necessidade de criar novas mapF e reduceF.

4.4. MapReduce com Goroutines

Essa foi a tarefa que não estava no nosso plano inicial, mas vejo como parte essencial da evolução do trabalho. Afinal, uma grande qualidade divulgada pelos criadores do Go é que é simples criar threads para o processamento concorrente. Goroutines são threads leves que são executadas de forma assíncrona. Disparar gourotines para executar uma função, ou apenas parte dela, é um padrão comum para o aproveitamento de muitos núcleos, mas temos que esperar essas threads terminarem o processamento para seguir com o

programa, senão a thread principal pode terminar antes menos do fim do processando da goroutine [16].

A única coisa que temos que fazer para criar uma tarefa é usar a palavra chave “go”. Coloquei os loops do DoMap e do DoReduce em funções privadas e cada vez que essas funções iriam para a próxima posição no loop uma nova goroutine seria criada.

A primeira dificuldade é que eu não tinha criado um canal para poder esperar o processamento acabar antes de seguir. Fizemos isso e colocamos uma mensagem simples no canal, o que fez com que o programa funcionasse como o esperado.

Colocamos uma contagem de tempo, para podermos ver o quanto tempo de processamento diminuiria. Colocando os contadores na main tivemos como resultado valores muito próximos e coerentes.

Para todos os testes descritos neste relatório, utilizamos uma máquina Linux Ubuntu 18.0 de 64-bit, com 15.5 GiB de memória e processador Intel® Core™ i7-8550U CPU @ 1.80GHz × 8(8 núcleos).

	Mapper	Reducer
Sequencial	981.4 ms	695.8 ms
Concorrente	878.8 ms	636.0 ms

Quando colocamos o contador dentro das funções tivemos valores muito pequenos para cada fazer, o código estava funcionando corretamente, o problema era que não estávamos esperando os goroutines terminarem o processamento antes de medir o tempo. Então usamos um waitgroup [17], que tem o método Wait(), assim não precisando criar um loop para esperar o processamento das threads.

Então os números finais ficaram para as fases ficaram muito próximos do processamento sequencial, o que não era o objetivo, porque isso significa que não estamos aproveitando os núcleos da máquina.

Pesquisamos extensivamente qual seria o motivo de não termos visto melhora no tempo, vimos várias outras pessoas com o mesmo problema no Stack OverFlow [23] e até mesmo no Reddit [25], mas nada me fazia chegar em uma conclusão concreta. Então como nosso volume de dados ainda estava pequeno, era apenas um texto, achamos que por isso que a diferença não tinha

sido significativa, mas a causa raiz desse problema só descobrimos depois(falaremos sobre isso no item 4.7).

4.5. Netflix

Para testarmos o funcionamento no nosso framework, por enquanto só sequencial e com goroutines, decidimos usar um conjunto de dados diferente. Pegamos um grande .csv disponível de forma gratuita no site Kaggle [19] onde, entre outros conjuntos de dados, há dados sobre filmes e séries na plataforma de streaming Netflix. Esses dados foram baixados apenas uma vez e não refletem a realidade da plataforma no momento.

Logo percebemos que não havia tanta mudança da contagem de palavras para esse caso. A principal mudança foi no tratamento do arquivo. Não tínhamos mais um .txt e sim o .csv e uma coluna específica - usamos a coluna de países(country). Criamos uma nova versão do mapF de forma que recebesse agora só um array de string. Nosso reduceF continuou o mesmo código que o de contagem de palavras, com a diferença de que as palavras agora são nomes de países.

	Mapper	Reducer
Sequencial	50.1 ms	11.0 ms
Concorrente	32.5 ms	5.1 ms

Nossos resultados foram muito mais satisfatórios, o tempo de processamento da fase reduce caiu pouco mais que a metade e a fase mapper foi processada em aproximadamente um quinto do tempo do sequencial. Com esses resultados achamos que poderíamos avançar para o próximo passo.

4.6. MapReduce com RPC

Decidimos usar o nosso código do MapReduce com goroutines como base para o nosso servidor RPC. Começamos criando funções que são o loop de cada fase no servidor (SaveMapResults para a fase map e DoReduceStep para a fase reduce), e chamamos o servidor em goroutines, mas para isso fizemos um contrato de comunicação entre as partes, usando um tipo de dados para cada fase

Usamos argumentos na linha de comando para definir em qual porta o servidor iria subir, e do lado do cliente para dizer o número de servidores que temos e também em quais portas os servidores se encontram. Nossos resultados com o uso de RPC foram quase os mesmos do resultado da versão

concorrente, o que faz sentido porque estamos fazendo uma chamada de um procedimento remoto que pode aumentar o tempo, e esse teste foi feito com os dados do Netflix, então ainda não tínhamos um boa quantidade de dados para testar.

4.7. Textos do Machado de Assis

A principal característica do MapReduce é a capacidade de melhorar o tempo de processamento de uma quantidade de dados muito grande (Big Data) [7]. Como o processamento de um texto muito grande não foi o suficiente para obter benefícios com a concorrência, decidimos usar toda a bibliografia de Machado de Assis para o nosso trabalho. Pegamos 116 textos, que estão sob domínio público [30].

Com essa massa de dados conseguimos ver que estávamos lançando uma fase reduce para cada arquivo, e como consequência estávamos criando um resultado para cada arquivo, o que não era o nosso objetivo. Queríamos simular uma grande quantidade de dados dividida em n partes, mas gerando um único resultado.

Para cada texto na fase map, geramos a saída com arrays de chave e valor em 8 arquivos numerados de 0 a 7. Agora rodamos 8 fases reduce, uma para cada grupo de arquivos onde as tuplas de chaves e valores tem o mesmo hash.

Com os textos fizemos a implementação para dois casos de uso, o índice inverso e a contagem de palavras, e foi com esses dados que vimos uma diferença de tempo maior entre os métodos de implementação que usamos.

4.8. Framework para o MapReduce local

Criamos um único framework [26] para o MapReduce local, e nessa implementação já temos sequencial e concorrente já pronto para três casos de uso diferentes: contagem de palavras, índice inverso e dados Netflix. Usamos os argumentos na linha de comando para definir o caso de uso, e o método(sequencial ou concorrente). Como no curso CS-451-UB todos os métodos estão em um único framework, decidimos deixar os métodos locais juntos.

Para todos os casos de uso funcionarem, tivemos que deixar os cabeçalho das funções mapF e reduceF iguais, porque vamos passar as funções como variável. Temos três casos de uso e vamos falar aqui de todos.

Todas funções mapF devem receber um string com o caminho para o

documento e um string com o texto a ser manipulado e retornar um array de chave e valor. Para o caso do dados Netflix a mapF retorna um array onde a chave é o dado (o país, por exemplo) e o valor é 1.

```
func netflixDataMapF(document string, value string) (res
[]common.KeyValue) {
    values := strings.Split(value, ",")
    for i := 0; i < len(values); i++ {
        column := values[i]
        res = append(res, common.KeyValue{Key: column, Value: "1"})
    }
    return res}
```

Para índice inverso nosso objetivo é mostrar em quantos arquivos uma determinada palavra aparece.Neste a mapF retorna um array onde as chaves são as palavras e o valor é o nome do arquivo com a extensão.

```
func invertedIndexMapF(document string, value string) (res
[]common.KeyValue) {
    words := strings.FieldsFunc(value, func(r rune) bool {
        return !unicode.IsLetter(r)
    })
    for _, word := range words {
            res = append(res, common.KeyValue{Key: word, Value:
path.Base(document)})
    }
    return res
}
```

Para a contagem de palavras, nosso objetivo é mostrar em quantas vezes uma determinada palavra aparece.Neste a mapF retorna um array onde as chaves são as palavras e o valor é o nome do arquivo com a extensão.

```
func wordCountMapF(document string, value string) (res
[]common.KeyValue) {
    words := strings.FieldsFunc(value, func(r rune) bool {
        return !unicode.IsLetter(r)
    })
    for _, word := range words {
            res = append(res, common.KeyValue{Key: word,
Value: "1"})
    }
```

```
        return res
    }
```

Todas as funções reduceF devem receber uma chave e um array de valores e retornar um string. O do índice inverso a chave é uma palavra e o array contém a lista de nomes de arquivos onde essa palavra aparece e o retorno é um string com a quantidade de arquivos onde essa palavra aparece e todos os nomes dos arquivos separados por vírgula.

```
func invertedIndexReduceF(key string, values []string) string {
        var fileNames []string

        for _, v := range values {
            if !contains(fileNames, v) {
                fileNames = append(fileNames, v)
            }
        }
    return strconv.Itoa(len(fileNames)) + " " + strings.Join(fileNames, ",")}
```

Para os dados netflix e a contagem de palavras temos o mesmo reduceF. Neste a reduceF só soma todos os valores no array de valores. Cada valor é 1, e estamos contabilizando o número de ocorrências.

```
func netflixDataReduceF(key string, values []string) string {
        total := 0
        for _, v := range values {
            val, _ := strconv.Atoi(v)
            total += val
        }
        return strconv.Itoa(total)
    }
```

Como podemos ver os tempos de processamento são coerentes com a quantidade de dados que temos para processar.

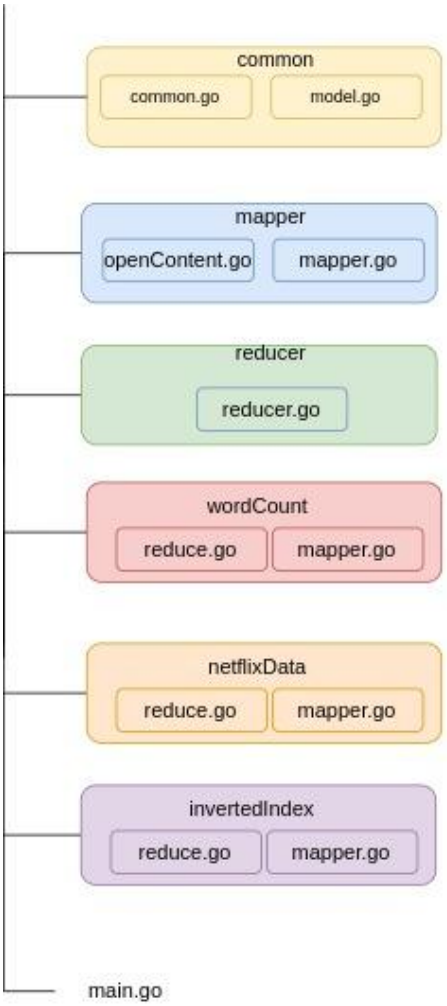
Contagem de Palavras		
	Mapper	Reducer
Sequencial	6.8 s	19.5 s

Concorrente	2.2 s	6.2 s
Índice Inverso		
	Mapper	Reducer
Sequencial	6.5 s	29.0 s
Concorrente	2.4 s	10.0 s
Dados Netflix(países)		
	Mapper	Reducer
Sequencial	43.4 ms	8.5 ms
Concorrente	33.7 ms	5.7 ms

4.9. Framework para o MapReduce com RPC

O framework de MapReduce com RPC [27] é composto por duas aplicações, o servidor e o cliente, e usamos os argumentos na linha de comando da mesma forma que usamos quando começamos a testar o uso do RPC(vide tópico 4.6).

A nossa estratégia para o processamento em vários servidores mudou. Voltamos para a teoria para entender como deveríamos fazer o uso do RPC para o processamento em paralelo. Então agora todo o processamento de cada iteração das fases é feito nos servidores, e ainda usamos goroutines para os loops de cada fase. O framework completo, na parte do servidor, terá a seguinte estrutura:



Os resultados com 3 servidores para cada caso de uso foram os seguintes

	Contagem de Palavras	Índice Inverso	Dados Netflix(países)
Mapper	2.4 s	2.5 s	55.9 ms
Reducer	8.6 s	15.4 s	9.9 ms

5. Projeto e especificação do sistema

A cada avanço nós fazíamos uma infraestrutura nova, então até a montagem dos frameworks, nós tínhamos uma estrutura para o sequencial, uma para o concorrente e uma para o RPC(processamento em paralelo).

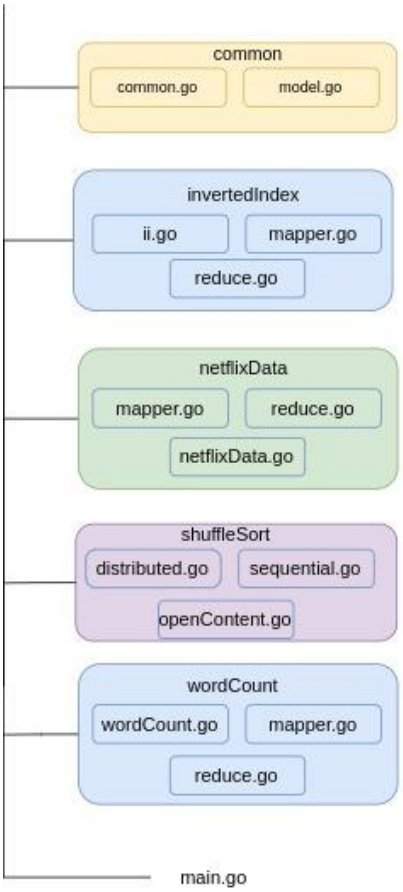
5.1. MapReduce Local Incompleto

Fizemos um MapReduce Local completo com todos os casos de uso implementado, junto com a versão distribuída. A ideia seria poder usar essa

versão completa como um possível gabarito.

Deixamos toda a implementação do MapReduce sequencial para o caso de uso de palavras, para poder servir como uma referência para o aluno.

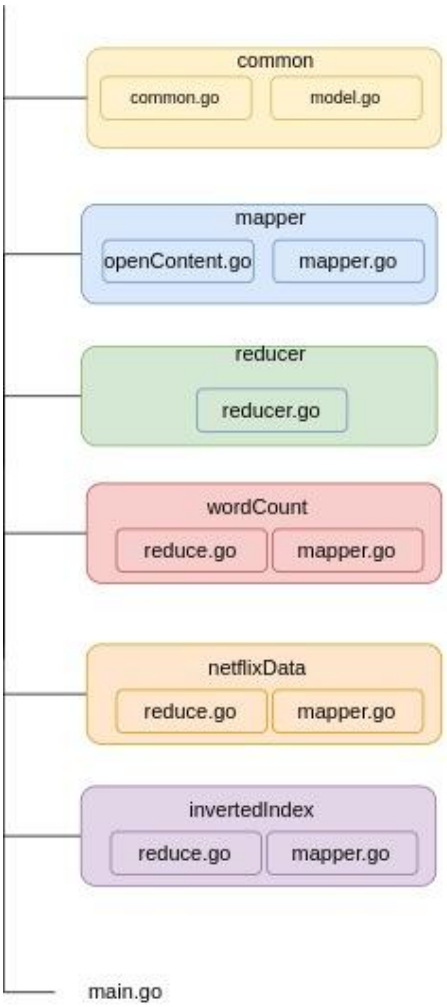
A figura a seguir ilustra a arquitetura. Os pontos a serem completados estão indicados no código para deixar a experiência dos alunos mais simples.



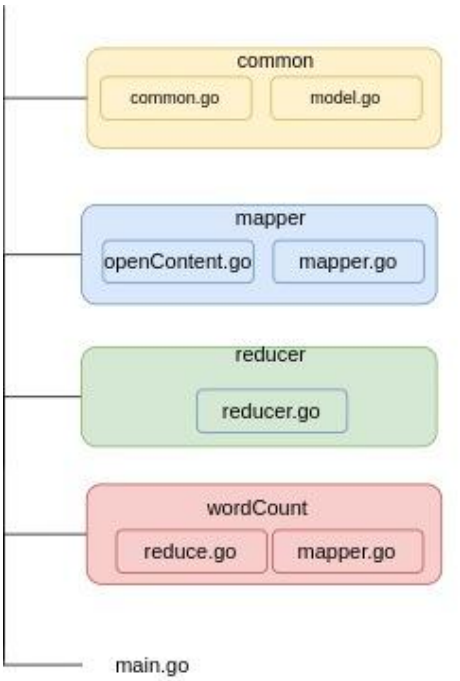
5.2. MapReduce com RPC incompleto

Para essa fase MapReduce local é um pré requisito, porque esse código deverá ser usado no MapReduce com RPC incompleto [29]. Nessa parte não deixamos claro que tem como fazer a implementação para todos os casos de uso, já que acreditamos que é opcional.

O lado do cliente é bastante simples, e já deixamos uma chamada simples para o server implementada.. A lógica para cada chamada para o servidor é a mesma usada para o MapReduce Local concorrente, e para facilitar nós damos os tipos que vão ser usados na comunicação e o cabeçario da função que irá chamar a comunicação pronto, a arquitetura ficou da seguinte forma :



No lado do servidor só indicamos o que deve ser completado, todo o código do MapReduce estará no servidor, simplificamos um pouco a arquitetura da versão incompleta, mas pode ser facilmente alterada para cobrir todos os casos de uso. A arquitetura do lado do servidor ficou a seguinte:



6. Implementação e avaliação

Nosso principal desafio foi os programas estarem rodando e não apresentarem nenhum erro aparente. O que vimos primeiro foi o problema da requisição não chegar no servidor, a resolvemos de uma forma simples: colocando logs em tudo. Colocamos logs na parte do cliente e do servidor para acompanhar todo o processo, e só com os logs percebemos que as requisições não estavam chegando ao servidor porque as structs que estávamos usando para a transmissão de dados estavam como privadas no lado do cliente.

E o nosso principal erro só foi visto quando começamos a usar os textos de Machado de Assis, só quando tivemos uma grande quantidade de dados, com mais de um arquivo que vimos que o nosso reduce não estava reduzindo da forma certa (foi discutido no item 4.7). A forma que resolvemos isso foi voltando para a primeira parte, a parte teórica, só assim que percebemos que tínhamos que reduzir os arquivos com o mesmo hash e não reduzir cada arquivo e depois juntar tudo no merge.

A avaliação da nossa implementação foi discutida ao longo do item 4 de acordo com cada avanço que fizemos. A avaliação da versão para aprendizado é mais complexa porque podemos apenas comentar sobre o código e como foi dividido. Nenhum aluno usou o nosso framework até a data de escrita deste relatório então não podemos avaliar o quanto é de fácil uso para quem não esteve envolvido com o desenvolvimento desses sistemas.

Analisando o tempo de processamento de cada método de

implementação para o MapReduce podemos ver que a grande diferença de tempo de processamento foi da versão sequencial para a versão concorrente, essa melhora pode ser explicada pelo aproveitamento dos núcleos da máquina, mas não vemos melhora entre a versão concorrente com a versão com RPC, a concorrente. Isso porque de fato ainda estamos utilizando os mesmos núcleos e ainda introduzimos o overhead da chamada remota, então efetivamente não estamos fazendo nada que possa reduzir o tempo de processamento.

7. Considerações finais

Esse trabalho mostra o quanto é simples usar threads e RPC em Go, desde que a pessoa tenha uma noção de programação anterior. Em nossos programas ficou simples de entender o que está acontecendo. Podemos navegar pelo código e acompanhar o funcionamento de cada programa sem dificuldades.

Por isso que foi tão importante termos feitos duas infraestruturas, e foi a nossa principal crítica ao material do curso CS-451-UB: A parte RPC utilizava uma chamada de função que abstraía todo o processo, sem nenhum real apredizado sobre o RPC. O RPC do curso não é um chamada remota e sim um simulação de RPC entre goroutines e dessa forma não é possível migrar o programa para várias maquinas.

Tive algumas dificuldades já que não cursei a disciplina de Sistemas Distribuídos, então tive que estudar sobre sistemas distribuídos sem o auxílio de uma aula e por isso não consegui evoluir tão rápido quanto podia nessa parte, uma coisa que eu mudaria é que eu faria essa matéria como eletiva.

Acredito também que deveria ter passado menos tempo com o curso CS-451-UB, deveria ter usado apenas para estudo de como fazer o MapReduce sequencialmente e então começar com a minha própria estrutura, já que na parte distribuída não montamos um servidor RPC, não criamos um cliente e não fazemos uma chamada explícita para o servidor.

Esse projeto está bem completo e cumpri o objetivo: ajudar na familiarização sobre a linguagem Go, sobre RPC e o processamento em threads. O que ficou faltando para ser um versão simplificada e completamente funcional de MapReduce foi o sistema de arquivo distribuídos, algum aluno poderia fazer como projeto final esse sistema e usar a nossa versão de MapReduce como um caso de uso, ou eu poderia continuar com esse projeto para um pós graduação.

8. Referências bibliográficas

[1]Dean, Jeffrey; Ghemawat, Sanjay. **MapReduce: Simplified Data Processing**

on Large Clusters. Commun. ACM 51, 1 (January 2008), 107−113. Disponível em: https://www.cs.amherst.edu/~ccmcgeoch/cs34/papers/p107-dean.pdf Acesso em 20 julho.2022

[2]**Frequently Asked Questions (FAQ).** Disponível em .https://golang.org/doc/faq Acesso em 20 novembro.2021

[3]CS451/651 - **Lab 1 :MapReduce.** Disponível em: https://cs-web.bu.edu/~jappavoo/jappavoo.github.com/451/labs/lab-1.html Acesso em 20 novembro.2021

[4]Andrade, Tiago Pedroso da Cruz. **MapReduce - Conceitos e Aplicações.** Disponível em: https://www.ic.unicamp.br/~cortes/mo601/trabalho_mo601/tiago_cruz_map_reduce/relatorio.pdf Acesso em 20 novembro.2021

[5]GOLDMAN, Alfredo et al. **Apache hadoop: conceitos teóricos e práticos, evolução e novas possibilidades.** 2012, Anais do Congresso da Sociedade Brasileira de Computação. Porto Alegre: SBC, 2012. Disponível em: http://www.lbd.dcc.ufmg.br/colecoes/jai/2012/003.pdf. Acesso em: 20 jul. 2022.

[6]INE5645 - **Programação Paralela e Distribuída, Unidade V: Um exemplo didático de aplicação Hadoop.** Disponível em: http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/Exemplo_MapReduce.pdf Acesso em 20 novembro.2021

[7]**Hadoop MapReduce: Introdução a Big Data.** Disponível em: https://www.devmedia.com.br/hadoop-mapreduce-introducao-a-big-data/30034 Acesso em 20 novembro.2021

[8] **Package rpc**.Disponível em https://golang.org/pkg/net/rpc/ Acesso em 20 novembro.2021

[9]**Concurrency**. Disponível em : https://golang.org/doc/effective_go#concurrency Acesso em 20 novembro.2021

[10]**Building a Basic RPC Server and Client with Go**. Disponível em : https://www.youtube.com/watch?v=1MPWPq2N768 Acesso em 20 novembro.2021

[11]**Using Go Modules**. Disponível em : https://go.dev/blog/using-go-modules Acesso em 20 novembro.2021

[12]**Go: Ten years and climbing** Disponível em : https://commandcenter.blogspot.com/2017/09/go-ten-years-and-climbing.html Acesso em 20 novembro.2021

[13]**Go at Google: Language Design in the Service of Software Engineering** Disponível em : https://go.dev/talks/2012/splash.article Acesso em 20

novembro.2021

[14] **C (linguagem de programação)** Disponível em : https://pt.wikipedia.org/wiki/C_(linguagem_de_programa%C3%A7%C3%A3o) Acesso em 22 novembro.2021

[15] **Building a Basic RPC Server and Client with Go** Disponível em :https://www.youtube.com/watch?v=1MPWPq2N768 Acesso em 23 novembro.2021

[16] **Goroutines e go channels** Disponível em: https://medium.com/trainingcenter/goroutines-e-go-channels-f019784d6855 Acesso em 24 novembro.2021

[17]**Go by Example: WaitGroups** Disponível em https://gobyexample.com/waitgroups Acesso em 24 novembro.2021

[18] **Processamento distribuído de dados com MapReduce utilizando Python, MRJob e EMR** Disponível em :https://medium.com/data-hackers/processamento-distribu%C3%ADdo-de-dados-com-mapreduce-utilizando-python-mrjob-e-emr-c826a617f8b3 Acesso em 16 fevereiro.2022

[19] **Netflix Movies and TV Shows** Disponível em https://www.kaggle.com/datasets/shivamb/netflix-shows Acesso em 17 fevereiro.2022

[20] **How to Wait for All Goroutines to Finish Executing Before Continuing** Disponível em https://nathanleclaire.com/blog/2014/02/15/how-to-wait-for-all-goroutines-to-finish-executing-before-continuing/ Acesso em 16 dezembro.2021

[21] **How to Wait for All Goroutines to Finish Executing Before Continuing, Part Two: Fixing My Oops** Disponível em https://nathanleclaire.com/blog/2014/02/21/how-to-wait-for-all-goroutines-to-finish-executing-before-continuing-part-two-fixing-my-ooops/ Acesso em 16 dezembro.2021

[22] **Chapter 30: Concurrency** Disponível em https://www.practical-go-lessons.com/chap-30-concurrency Acesso em 16 dezembro.2021

[23] **Best approach to getting results out of goroutines** Disponível em https://stackoverflow.com/questions/66818346/best-approach-to-getting-results-out-of-goroutines Acesso em 16 dezembro.2021

[24] **Can someone explain why this program isn't faster with goroutines?** Disponível em

https://www.reddit.com/r/golang/comments/a1wznn/can_someone_explain_why_this_program_isnt_faster/ Acesso em 16 dezembro.2021

[25] **A first look at Goroutines** Disponível em https://medium.com/golicious/a-first-look-at-goroutines-c3608b7e8c40 Acesso em 16 dezembro.2021

[26] **elaines0104/mapReduceLocal** Disponível em https://github.com/elaines0104/mapReduceLocal Acesso em 13 julho.2022

[27] **elaines0104/mapReduceWithRPC** Disponível em https://github.com/elaines0104/mapReduceWithRPC Acesso em 13 julho.2022

[28] **elaines0104/mapReduceLocalIncomplete** Disponível em https://github.com/elaines0104/mapReduceLocalIncomplete Acesso em 13 julho.2022

[29] **elaines0104/mapReduceWithRPCIncomplete** Disponível em https://github.com/elaines0104/mapReduceWithRPCIncomplete Acesso em 13 julho.2022

[30] **MACHADO DE ASSIS - OBRA COMPLETA** Disponível em: http://machado.mec.gov.br/obra-completa-lista?start=12 Acesso em 17 julho.2022