Instituto Tecnológico de Aeronáutica - ITA
Sistemas distribuídos - CSC-27
Alunos:
Bernardo Hoffmann da Silva
Marcos Vinicius Pereira Veloso
Victor Maciel

Implementação de Hash tables distribuídas para sistemas de compartilhamento de arquivos P2P: Kademlia e Chord

1 Motivação

1.1 Introdução

Tabelas Hash são estruturas de dados que implementam uma coleção de pares chave-valor. Elas são eficientes para a busca, inserção e exclusão de elementos com base em uma chave única. O conceito central por trás de uma hash table é o uso de uma função de hash para mapear chaves para índices em uma tabela. Essa função de hash transforma a chave em um valor numérico, que é usado como índice para acessar a posição correspondente na tabela.

Hash tables distribuídas são uma extensão do conceito de hash tables para ambientes distribuídos, onde os dados são armazenados em vários nós ou computadores. Em um sistema distribuído, uma única tabela de hash pode se tornar um gargalo de desempenho ou um ponto único de falha. Portanto, as hash tables distribuídas dividem os dados entre diferentes nós para distribuir a carga e melhorar a escalabilidade.

Entre as principais características das hash tables distribuídas estão o particionamento dos dados entres os diversos nós, garantindo uma certa tolerância a falhas e um balanceamento na carga. Assim, há menos gargalos devido a essa descentralização e uma maior consistência dos dados, que podem ser replicados em diferentes servidores.

As primeiras implementações dessas estruturas de dados distribuídas tinham o objetivo de realizar o armazenamento e distribuição de arquivos em uma arquitetura descentralizada (P2P). Assim, os arquivos ficam salvos em uma rede de computadores, e os usuários (nodes) podem entrar na rede e fazer o download de um arquivo, que é compartilhado pelos outros usuários. Assim, cria-se um sistema de distribuição que não depende de um único servidor, o que geraria gargalo e concentraria o ponto de falha. Esse tipo de serviço ainda é muito utilizado em aplicações como BitTorrent e Freenet.

Entre as principais implementações possíveis de hash tables distribuídas, estão o Kademlia e o Chord, que realizam a construção da tabela com diversos nós, porém com estruturas de comunicação diferentes.

1.2 Fundamentação Teórica: Kademlia

O kademlia é um protocolo de hash table distribuída P2P que utiliza como métrica a "distância"entre os nós. Essa distância é feita a partir da função XOR entre a chave (hash) de cada um desses servidores. Seu objetivo era a criação de uma rede de arquivos distribuída que garantisse uma inserção e busca escalável e eficiente de dados.

Cada computador dessa rede pode se comunicar com os outros que estão salvos em sua tabela de roteamento, a qual é o grande diferencial da comunicação dessa implementação. Dada uma função de hash e um nó dentro de uma rede, define-se um k-bucket como uma lista de hashs dentro de um intervalo de distâncias para o hash do nó em questão. Esses intervalos são definidos de forma binária, sendo que esses aumentam em tamanho de forma exponencial conforme a distância para o hash do nó em que se trabalha aumenta. Por exemplo, em um hash de tamanho 64, um nó terá k-buckets com distâncias 0, 1, 2 a 3, 4 a 7, 8 a 15, 16 a 31 e 32 a 63. Assim, escolhe-se um número k que será o número de nós que esse computador salvará em cada um de seus k-buckets, com suas respectivas distâncias.

Portanto, um computador terá mais informação sobre nós próximos de si e menos informação sobre nós distantes, pois a probabilidade de guardar nós próximos é maior, já que há mais k-buckets em distâncias curtas. Caso se deseje acessar um nó distante que não está salvo na tabela de roteamento, transfere-se a mensagem para um nó próximo ao desejado, que terá uma probabilidade maior de ter acesso a esse nó.

Para a inserção de um arquivo, calcula-se o hash desse arquivo e esse é armazenado em k nós mais próximos desse hash. Assim, dado que um cliente envia uma mensagem de Store, procura-se os nós mais próximos do hash do arquivo e esse é armazenado nesse. De forma parecida, a busca por um arquivo na rede é feita pela consulta aos nós com hash mais próximo do hash procurado. Assim, os saltos de comunicação e passagem de mensagem da tabela são feitos de acordo.

1.3 Fundamentação Teórica: Chord

O Chord, por sua vez, também é um protocolo em que implementa a hash table distribuída em uma arquitetura específica. A estrutura básica do Chord envolve uma rede circular, onde cada nó é atribuído um identificador único em um espaço de identificação circular. Cada recurso também é associado a um identificador único, e a correspondência entre recursos e nós é estabelecida com base nesses identificadores. Para a implementação a ser apresentada, faz-se o uso dos endereços IPs e nome de arquivos.

Dado um nó que queira entrar na topologia, o novo nó deve escolher um identificador que o representará no anel, no qual é feito por meio de uma função de hash aplicada ao endereço IP de tal nó. Para efetivamente entrar, ele configura uma solicitação de junção de nó, contendo seu endereço e hash. Para fazer isso, ele pode envia tal mensagem de solicitação TCP de junção para qualquer nó existente na rede, pedindo informações sobre o nó que deverá ser seu sucessor. Esta mensagem é encaminhada pela topologia até chegar no nó que deverá ser seu antecessor; este, por sua vez, entra em contato via TCP para o nó solicitante e envia mensagens que rodam métodos de atualização de seus vizinhos.

Para o envio de um recurso de dado, cada nó, ao receber a tal solicitação, a encaminha para o próximo nó mais próximo no anel, com base nas informações armazenadas em suas estruturas de

dados que identificam seus vizinhos, em um algoritmo análogo ao de inserção de nó a topologia.

Para localizar um recurso, por sua vez, um nó inicia o processo de roteamento, passando a mensagem para o nó mais próximo do identificador do recurso. Isso é feito de maneira eficiente, aproveitando a estrutura do anel, caso seja usado uma tabela de roteamento maior, e não apenas o próximo vizinho.

2 Trabalhos relacionados

Os trabalhos relacionados ao tema normalmente são estudos de implementações ou de avanços possíveis em cada um desses algoritmos. Entre eles, podem ser listados os seguintes.

- A formal specification of the Kademlia distributed hash table

  Isabel Pita, Dept. Sistemas Informáticos y Computación, Universidad Complutense de Madrid

- Kademlia: A Peer-to-Peer Information System Based on the XOR Metric

  Petar Maymounkov and David Mazières, New York University

- Building peer-to-peer systems with chord, a distributed lookup service

  F. Dabek et al., Proceedings Eighth Workshop on Hot Topics in Operating Systems, Elmau, Germany, 2001, pp. 81-86

- A review of recent advancement in Kademlia and Chord algorithm

  Mohammad Asyraff; Mohamad Ariff; Suraya Mohamad

- A performance comparison of Chord and Kademlia DHTs in high churn scenarios

  Adán G. Medrano-Chávez, Elizabeth Pérez-Cortés and Miguel Lopez-Guerrero

3 Objetivo

O objetivo desse trabalho consiste no estudo e implementação dos algoritmos de Hash Tables Distribuídas Chord e Kademlia. A abordagem adotada inclui o desenvolvimento dos algoritmos em linguagem de programação *Go*, alinhadas com a utilização de contêineres *Docker* para simular ambientes distribuídos, assegurando portabilidade e eficiência dos mesmos.

Além disso, a intenção é que durante o desenvolvimento sejam identificadas e analisadas as diferenças de implementação, buscando compreender como cada algoritmo aborda desafios específicos relacionados à distribuição de arquivos em redes *peer-to-peer*. Ademais, espera-se comparar as dificuldades de implementação e de manutenção dos algoritmos, considerando os pontos positivos e negativos de cada um.

4 Proposta

A proposta deste projeto engloba a implementação prática e comparação em alto nível dos algoritmos de Hash Tables Distribuídas, nomeadamente Chord e Kademlia, utilizando a linguagem de programação Go e contêineres Docker. A ênfase está na compreensão das divergências fundamentais de implementação desses algoritmos em ambientes distribuídos.

4.1 Requisitos Funcionais

As implementações de Chord e Kademlia atenderão aos seguintes requisitos funcionais:

1. **Armazenamento de Arquivos:**
   - Permitir que o usuário realize o armazenamento de arquivos no sistema distribuído.
   - O *hash* do arquivo será calculado a partir de seu nome.
2. **Busca de Arquivos:**
   - Possibilitar a busca de arquivos dentro do sistema distribuído, explorando as funcionalidades de pesquisa oferecidas pelos algoritmos.
3. **Tratamento Distribuído de Arquivos:**
   - Garantir que os arquivos na aplicação sejam tratados de maneira distribuída e balanceada, refletindo a natureza *peer-to-peer* da infraestrutura.

4.2 Requisitos Não Funcionais

Os requisitos não funcionais guiarão a implementação visando a eficiência, confiabilidade e segurança do sistema:

1. **Eficiência na Busca:**
   - Priorizar a eficiência nas operações de busca, otimizando os algoritmos para proporcionar respostas rápidas e eficazes.
2. **Tolerância a Falhas:**
   - Implementar mecanismos de tolerância a falhas, assegurando que o sistema continue operacional mesmo diante de potenciais falhas em nós da rede.
3. **Consistência de Dados:**
   - Garantir a consistência dos dados distribuídos, promovendo uma visão coerente do sistema mesmo em cenários dinâmicos.

4.3 Tecnologias Utilizadas

A implementação dos algoritmos Chord e Kademlia será realizada na linguagem de programação *Go*, conhecida por sua eficiência e concorrência. Para garantir a portabilidade e eficiência, as implementações serão encapsuladas em contêineres *Docker*.

Destaca-se o enfoque não está em criar uma interface gráfica para que o usuário possa lidar com os arquivos e interagir com os sistemas distribuídos, mas sim na implementação dos algoritmos em si. Dessa forma, a interação do usuário será realizada por linha de comando (CLI) ou diretamente no código das aplicações.

5 Protótipo

Nesta seção, será apresentada a implementação realizada dos algoritmos. A aplicação está organizada em duas pastas, Chord e Kademlia, cada uma contendo o algoritmo desenvolvido correspondente. Serão discutidos os testes realizados, a instalação da aplicação e como executar a implementação.

5.1 Testes Realizados

5.1.1 Teste para o Kademlia:

Para verificar a robustez da implementação Kademlia, foi realizado o seguinte teste: Foram adicionados 10 nós na arquitetura, criando-se as tabelas de roteamento de cada um. Em seguida, foi solicitado o armazenamento do arquivo `text.txt`. O comportamento esperado é que esse arquivo seja salvo nos dois nós com a menor distância binária para o *hash* desse arquivo. Posteriormente, foi realizada a busca desse arquivo. A Figura 1 mostra os resultados desse teste.

Primeiramente, observa-se as tabelas de roteamento obtidas (*Distance Tables*). Elas foram printadas no decorrer no código e algumas estão desatualizadas, mas a do ultimo nó, `c10`, está atualizada e segue o esperado, apresentando no máximo dois nós em cada bucket.

Ademais, observa-se que o arquivo foi salvo em dois nós, nesse caso `c3` e `c6`, que são os nós da arquitetura com a menor distância ao *hash* do arquivo. Qundo foi buscado por esse arquivo, ele também foi encontrado nesses mesmos dois nós, de acordo com o esperado.

Outra observação pertinente é que os nós que não tinham o arquivo salvo se comunicaram com aqueles com quem eles tem conexão e estão com uma distância menor do *hash* do arquivo, solicitando o mesmo. Todo esse comportamento seguiu o que era esperado do algoritmo.

5.1.2 Testes para o Chord:

No caso do Chord, são colocados 7 nós candidatos a serem elementos do anel, seguido por requisições de *joins*, *send* e *search*. Para configurá-lo, basta alterar o conteúdos de arquivos de entrada *inputX.txt* e colocá-los como comandos no *docker-compose.yml*.

No caso especificado na Figura 2, c1 solicita a entrada na topologia, o qual altera os precessores e sucessores de cada um dos dois pelos métodos implementados. De modo análogo, c2 pede para entrar na topologia e faz com que c2 e c1 tenham que alterar seus sucessores, bem como c2 configura tais elementos em sua estrutura.

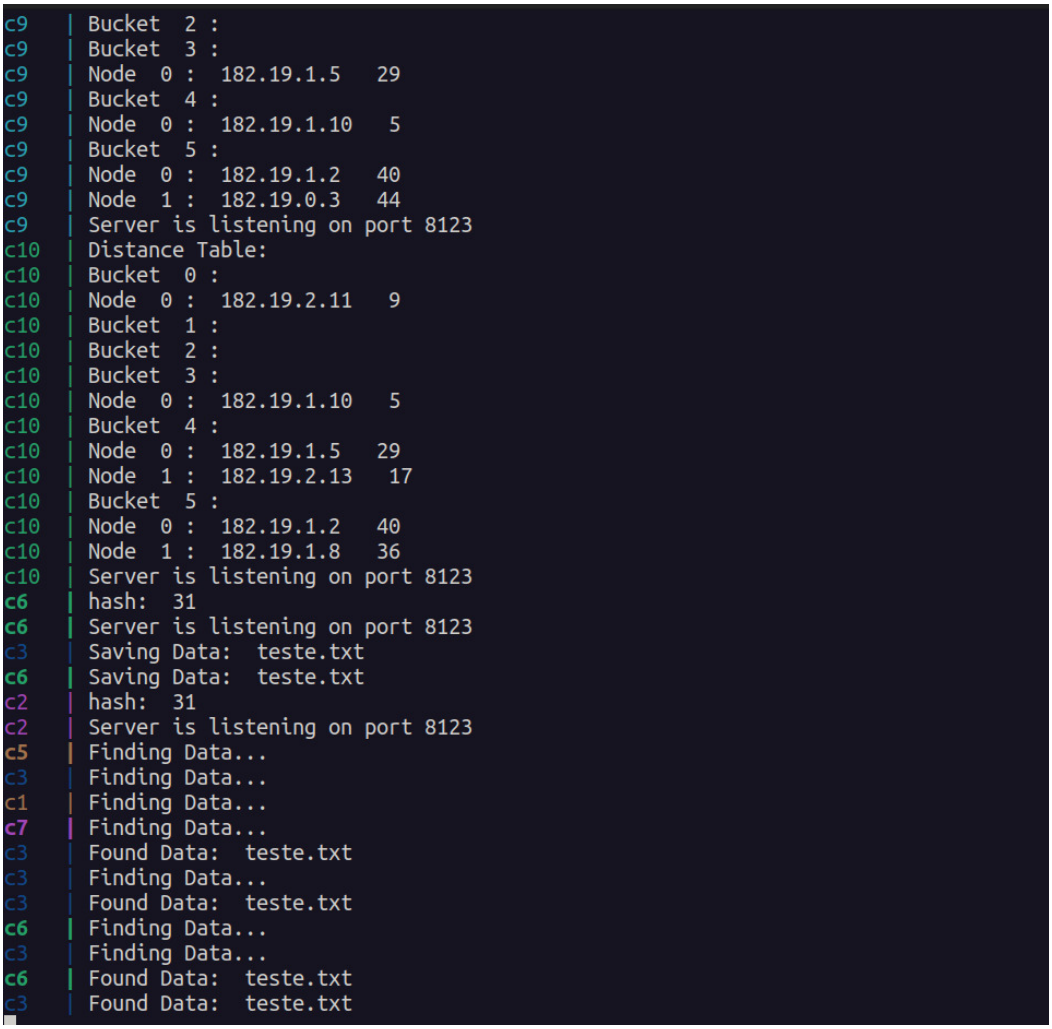

Figura 1: Resultado do teste realizado para a implementação Kademlia.

Feito isso, prossegue-se com o envio de arquivos: c2 envia um arquivo de nome file.txt no tempo 5 (veja script2.txt como referência) e c1 envia text.txt no tempo 8, com c2 fazendo uma pesquisa de text.txt no tempo 6. Como é esperado, e de acordo com a Figura, a pesquisa de c2 não houve resultados, já que c1 ainda não havia submetido seu arquivo até então. Com tempo suficiente, pode-se denotar que c0 e c2 armazenam então o conteúdo que foi enviado pelos dois nós especificados aqui.

Por fim, prosseguiu-se para mais um teste de armazenamento e busca de alguns arquivos. Testes foram realizados em casos em que o arquivo buscado não estava armazenado em nenhum computador, e verificou-se que o sistema analisou corretamente e acusou: "Arquivo não encontrado", conforme mostram as últimas linhas quando c2 tenta buscar um arquivo de nome tal que ainda não foi submetido por ninguém.

A Figura 2 mostra os resultados desse teste.

Como resultado, pode-se denotar que a busca e envio de arquivos, uma vez feita a topologia de inserção de nós, ocorre de forma satisfatória. Vale ressaltar que, usando o comando

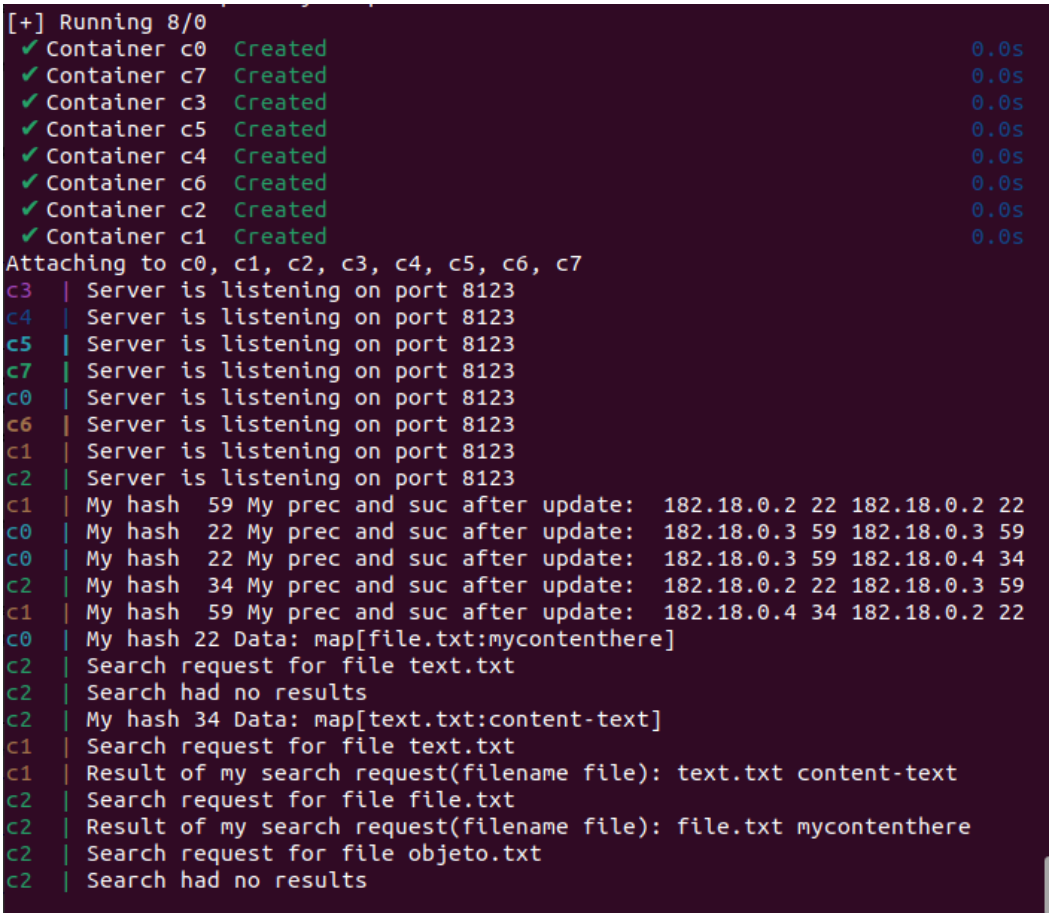

Figura 2: Resultado do teste realizado para a implementação Chord.

`docker compose attach ci`, podemos ter acesso ao terminal de qualquer nó que foi dado inicialmente para realizar os comandos manualmente.

5.2 Instalação e Execução da Aplicação

Para instalar a aplicação, siga os passos abaixo:

1. Clone o repositório do Git:

```
git clone https://github.com/BerHoffmann/DHT-Kademlia-Chord
```

2. Navegue até a pasta `Kademlia` ou `Chord`, conforme a implementação desejada.
3. Execute os comandos para realizar o build e up do Docker:

```
docker compose -f nome-do-DockerCompose build
docker compose -f nome-do-DockerCompose up
```

4. Para rodar a aplicação, execute o comando Docker Compose:

```
docker compose up
```

Este comando inicializará a aplicação, permitindo a execução dos algoritmos Kademlia ou Chord conforme a pasta selecionada.

6 Conclusões

O desenvolvimento e implementação dos algoritmos Kademlia e Chord proporcionaram uma compreensão aprofundada das complexidades inerentes às Hash Tables Distribuídas. Os requisitos funcionais foram cumpridos com sucesso, evidenciando a capacidade de ambos os algoritmos de realizar operações de upload, download, busca e tratamento distribuído de arquivos.

Entretanto, os requisitos de tolerância a falhas e consistência de dados apresentaram desafios significativos devido à grande complexidade da implementação de ambos os algoritmos.

A implementação bem-sucedida e os testes realizados confirmaram as expectativas em relação ao desempenho dos algoritmos. O Kademlia, embora mais complexo devido à necessidade de armazenar tabelas de roteamento e à recursão na busca dos nós mais próximos, demonstrou eficiência na distribuição de arquivos, incluindo replicação de dados. Por outro lado, o Chord, embora um pouco mais simples, carece de replicação de dados.

Ambos os algoritmos exibiram escalabilidade notável e abordagens inteligentes para armazenar tabelas hash de tamanhos consideráveis. A complexidade de implementação ligeiramente maior do Kademlia, atribuída à tabela de roteamento, garante operações de pesquisa e inserção em O(log n), em contraste com a implementação O(n) do Chord. Há uma clara tendência para o balanceamento e tolerância a falhas dos nós, destacando a diferença marcante em relação à distribuição de arquivos por servidores centralizados.

Diante desses resultados, identificam-se possíveis melhorias e direções para trabalhos futuros. Entre elas destacam-se a gestão adequada de casos em que nós são adicionados após o armazenamento de um arquivo, o tratamento eficaz de falhas ou saídas de nós das topologias, aplicação de melhorias existentes no Chord (a fim de existir replicação nos dados) e a realização de testes quantitativos comparativos entre os algoritmos.

Outros aspectos a serem explorados incluem a implementação de operações de input/output de arquivos e busca em tempo real, visando enriquecer a interatividade e a usabilidade da aplicação.

Essas perspectivas de desenvolvimento refletem a contínua evolução e refinamento necessários para alcançar sistemas distribuídos mais robustos e eficientes.

7 Referências

[1] KSHEMKALYANI, Ajay. Chapter 18: Peer-to-peer computing and overlay graphs. *In*: KSHEMKALYANI, Ajay. **Distributed Computing: Principles, ALgorithms and Systems**.

[2] MAYMOUNKOV, PETAR; MAZIÈRES, DAVID **Kademlia: A Peer-to-Peer Information System Based on the XOR Metric**, New York University

[3] DABEK, F. et al **Building peer-to-peer systems with chord, a distributed lookup service** Proceedings Eighth Workshop on Hot Topics in Operating Systems, Elmau, Germany, 2001, pp. 81-86

8 Organização do Grupo

Segue a tabela contendo a distribuição de atividades de cada um dos membros do grupo.

Aluno	**Atividades**	**Percentual do Trabalho**
Bernardo	Estudos e implementação do Kademlia	33.3%
Marcos	Estudos e implementação do Chord	33.3%
Victor	Estudos e implementação de funções específicas	33.3%

Tabela 1: Distribuição de Atividades no Grupo

9 Apêndice

O repositório com os algoritmos desenvolvidos está presente neste link.
`https://github.com/BerHoffmann/DHT-Kademlia-Chord`