Universidade Federal do ABC
MCTA025-13 - Sistemas Distribuídos - 2024.Q2

Projeto 02 - Tabela de Hash Distribuída (DHT)

Emilio Francesquini
e.francesquini@ufabc.edu.br

Entrega: 15/09/2024

1 Descrição geral

Neste projeto vamos implementar uma tabela de hash distribuída (DHT) que apesar da sua semelhança com o Chord é bem mais simples. A sua implementação de DHT deverá ser capaz de manter a estrutura em anel e dar suporte a:

- **Entrada e saída de nós da rede**. Você pode assumir que os nós nunca falham e que sempre avisam os outros nós sobre sua partida para que a estrutura da rede possa ser mantida.
- **Inclusão, remoção e pesquisa de pares de chave-valor**.

Reforçando que *não é necessário* implementar o protocolo completo do Chord! A sua implementação deve ser capaz apenas de garantir que os pontos descritos acima funcionam utilizando uma topologia em anel.

Para verificar o funcionamento de sua implementação de DHT, cada grupo deverá, também, implementar uma aplicação distribuída que a utilize e demonstre o seu funcionamento.

2 O projeto

Este projeto compreende duas tarefas principais:

- Implementar uma DHT cuja topologia de conexão seja um anel
- Implementar uma aplicação de exemplo que utilize a sua DHT

Em seguida detalhamos cada uma destas tarefas.

2.1 DHT

Ao contrário do Chord que mantém uma *finger table* para diversos nós participantes da rede, cada nó participante da nossa DHT só vai ter dois apontadores: um para o nó sucessor e outro para ou seu antecessor. Isso quer dizer que nossa DHT não vai ter as mesmas garantias relativas ao número de mensagens e nós afetados por inserções e buscas que Chord oferece ($O(\log n)$).

Assim como no Chord, em nossa DHT:

- A cada nó é atribuído um identificador aleatório de m-bits
- A cada entidade armazenada na DHT é atribuída uma chave de m-bits
- Entidades com chave `k` estão sob responsabilidade do nó cujo identificador `id` seja o menor possível tal que id $\geq k$
- Chamamos de nó `p` o nó cujo identificador é `p`

Para que a DHT seja utilizável pelas mais diversas aplicações é necessário que ela tenha uma API bem definida. As seguintes operações são sugestões das operações mínimas que a API da sua DHT deve oferecer:

- `join(KnownHostList)` Esta operação será utilizada pelas aplicações no momento que desejarem se conectar à DHT. Como parâmetro a operação recebe uma lista de nós conhecidos participantes da rede. Note que é uma lista de possíveis participantes já que eles podem não estar mais em execução. Neste caso a sua operação deve testar os elementos da lista até encontrar um participante ativo e se conectar a ele. Caso nenhum dos elementos da lista possa ser alcançado, a sua implementação deve assumir que é o primeiro nó da rede e iniciar uma nova DHT. Ao final desta chamada devem ter sido atualizados:
  - Os campos `sucessor` e `predecessor` do nó ingressante
  - O campo `predecessor` do sucessor do nó ingressante
  - O campo `sucessor` do predecessor do nó ingressante

  Também devem ter sido transferidos do sucessor do nó ingressante os dados armazenados na DHT que agora são responsabilidade do novo nó.
- `leave()` Esta operação é chamada pela aplicação quando deseja se desconectar da rede. Ao final desta chamada, o anel da sua DHT deve ter sido atualizado para estar consistente, ou seja, os ponteiros dos vizinhos devem ter sido atualizados e os e os eventuais valores que estivessem sendo armazenados pelo nó que está partindo devem ter sido transferidos para o seu sucessor.

- `store(key,value)` Armazena o valor `value` na DHT utilizando a chave `key`. Mais precisamente, armazena o valor `value` no nó cujo identificador seja `sucessor(hash(key))`. Note que a função de hash utilizada deve produzir um valor com m-bits, o mesmo número dos identificadores utilizados pelos nós.

- `retrieve(key)` Efetua uma busca na DHT e devolve o valor `value` (caso presente na DHT) que havia sido previamente armazenado através de uma chamada à operação `store(key,value)`.

Você pode assumir que as chaves utilizadas nas funções `store` e `retrieve` são simples strings (ou qualquer valor cujo hash possa ser calculado facilmente). Já para os valores, você vai ter que determinar que tipo de dados deverá ser armazenado para que a sua aplicação de exemplo funcione corretamente. Para que sua API seja a mais genérica possível, a sugestão é que o tipo dos valores seja um vetor de bytes. Desta maneira qualquer tipo de objeto serializado seria facilmente armazenado.

2.2 Aplicação

Você deverá implementar uma aplicação simples que demonstre o funcionamento da sua DHT. Você pode escolher uma das sugestões abaixo ou inventar a sua própria.

- Sistema de arquivos distribuído
- Álbum de fotos
- Chat
- ...

Importante: Todas as operações da sua API devem ser utilizadas pela sua aplicação de exemplo.

3 Implementação

Você é livre para escolher a linguagem de programação que desejar para este projeto. Isto só torna mais importante o fato de que o seu relatório (1 página no máximo) deverá trazer obrigatoriamente instruções detalhadas sobre como compilar e executar a sua aplicação. Também tenha em mente que o ambiente e utilizado para correção será uma máquina rodando Linux e que deve ser possível utilizar sua aplicação neste SO. Sua nota será drasticamente reduzida caso o professor não consiga compilar e executar o seu projeto utilizando as instruções fornecidas no seu relatório.

Para implementar a API descrita acima será necessário definir um protocolo de comunicação a ser seguido pelos nós participantes da DHT. O protocolo abaixo é uma sugestão de como fazê-lo.

3.1 Protocolo

Comunicações entre nós participantes da DHT devem ser obrigatoriamente feitas utilizando gRPC sobre Protocol Buffers. Você é responsável por definir o seu próprio protocolo de comunicação (usando um ou mais arquivos `.proto`).

Cada um dos nós participantes da rede poderá ter mais de uma conexão aberta simultaneamente, por exemplo, uma com o sucessor e outra com o predecessor. Quando há apenas um nó presente na DHT há zero conexão aberta.

As seguintes mensagens são necessárias para o funcionamento do protocolo:

- `JOIN` Que contém como valores o identificador, o endereço IP e a porta do nó ingressante. O identificador é um número de `m` bits sem sinal. Já o IP é enviado pode ser enviado como uma string no formato decimal separado por pontos. Por exemplo: `"127.0.0.1"` e a porta como um número inteiro decimal (ex. `12345`). A mensagem de `JOIN` é roteada ao nó atualmente responsável pelo identificador do nó ingressando na rede que, segundo as regras definidas na Seção DHT, será o seu sucessor.

- `JOIN_OK` Esta mensagem é enviada a um nó ingressante em resposta a uma mensagem do tipo `JOIN` enviada por este. Como argumentos a mensagem `JOIN_OK` tem o identificador do nó, o endereço IP e a porta do predecessor e sucessor a serem utilizados pelo novo nó. O sucessor é, na verdade, o próprio nó que está enviando a mensagem `JOIN_OK` e o predecessor é o seu atual predecessor. A mensagem `JOIN_OK` vai ser seguida imediatamente por 0 ou mais mensagens do tipo `TRANSFER` que conterão as chaves e valores que são responsabilidade do novo nó. Note que a mensagem `JOIN_OK` será enviada por um nó que não necessariamente será o nó que recebeu a mensagem original de `JOIN`.

- `NEW_NODE` Quando um novo nó ingressa na DHT, a sua mensagem `JOIN` é enviada a um nó arbirário na rede contendo o identificador do novo nó. Este nó ingressante recebe então o seu novo sucessor e predecessor via mensagem `JOIN_OK` do seu novo sucessor. O nó ingressante deve, então, enviar uma mensagem `NEW_NODE` ao seu predecessor notificando-o que ele deve atualizar o seu sucessor para apontar para o nó ingressante. `NEW_NODE` recebe como parâmetros o endereço IP e porta do nó ingressante.

- `LEAVE` Esta mensagem é enviada por um nó ao seu sucessor quando ele deseja deixar a DHT. Os argumentos para esta mensagem são o identificador, endereço IP e porta do predecessor do nó que está partindo. Uma mensagem `LEAVE` é seguida imediatamente de zero ou mais mensagens `TRANSFER` que se encarregam de transferir os dados armazenados no nó que está partindo para o seu sucessor.

- `NODE_GONE` Quando um nó deixa a rede, ele envia a mensagem `LEAVE` ao seu sucessor. O nó que está partindo também avisa ao seu predecessor que ele deve atualizar o seu sucessor para que aponte para o sucessor do nó que está saindo. Esta mensagem recebe 3 argumentos: o identificador, IP e porta do novo sucessor do destinatário.

- `STORE` Esta mensagem é utilizada para armazenar um valor sob uma determinada chave. Como primeiro argumento ela recebe a chave do valor a ser armazenada com `m` bits. Em seguida, ela recebe um número `n` que indica o número de bytes do objeto e finalmente os `n` bytes do valor a ser armazenado. Atente-se para utilizar os tipos apropriados para valores binários de tamanho arbitrário disponíveis no gRPC/Protocol Buffer.

- `RETRIEVE` Esta mensagem é usada para recuperar valores armazenados na DHT. Ela recebe como argumentos a chave sendo buscada representada por um hash de `m` bits (hash da chave do objeto sendo buscado), o identificador do nó que está fazendo a busca, seu endereço IP e porta.

- `OK` Esta mensagem é enviada em resposta a uma mensagem `RETRIEVE` pelo nó responsável pela chave sendo buscada quando ela está presente na DHT. Ela recebe como argumentos a chave do objeto, `n` que é o seu tamanho em bytes e uma sequência de `n` bytes que é o valor armazenado (mesmo formato utilizado pela mensagem `STORE`). Note novamente que o remetente desta mensagem não é, necessariamente, o nó que recebeu a mensagem de `RETRIVE`.

- `NOT_FOUND` Esta mensagem é enviada em resposta a uma mensagem `RETRIEVE` pelo nó responsável pela chave buscada quando não há nenhum dado armazenado com a chave procurada. Esta mensagem não tem nenhum argumento.

- `TRANSFER` Esta mensagem transfere a responsabilidade pelo armazenamento de um par chave valor entre os nós da DHT. Uma mensagem `TRANSFER` recebe como primeiro argumento a chave do valor armazenado no formato já descrito acima, seguido do próprio valor no mesmo formato utilizado pelas mensagens `STORE` e `OK`. Tantas mensagens `TRANSFER` quanto forem necessárias podem ser enviadas em

sequência durante o processo de ingresso ou partida de um determinado nó na DHT. Você também pode escolher mandar vários objetos em uma única mensagem `TRANSFER`, por exemplo, utilizando uma lista ou um RPC do tipo `stream`.

As mensagens `JOIN`, `STORE` e `RETRIEVE` **devem ser roteadas através do anel** até alcançarem o nó onde serão efetivamente processadas.

As mensagens `JOIN_OK`, `OK`, `LEAVE`, `NEW_NODE`, `NODE_GONE`, `TRANSFER` e `NOT_FOUND` **devem ser enviadas diretamente ao destinatário, sem serem roteadas pelo anel**.

Atente-se para o fato de que, (i) excetuado-se a função `join(KnownHostList)` da API, o protocolo não tem nenhum tratamento especial para o caso de falhas de nós. Caso um nó saia da rede sem que o processo iniciado com o envio da mensagem `LEAVE` seja terminado a rede ficará em um estado inconsistente. E (ii) também poderia ocorrer de dois ou mais nós tentarem se conectar simultaneamente à DHT. Neste caso, se pelo menos dois dos nós ingressantes compartilharem o mesmo sucessor o protocolo de conexão descrito acima falharia. Para este projeto específico **não será necessário** incluir o código para o tratamento destes casos, contudo é importante estar ciente que uma DHT real precisaria necessariamente lidar com estes e outros casos de concorrência e falhas.

3.2 Roteamento de mensagens e manutenção de conexões

O roteamento de mensagens `JOIN`, `STORE` e `RETRIEVE` no anel da DHT deve ser feito de maneira recursiva[1]. Isto significa que quando um nó recebe uma mensagem cujo destinatário é outro nó, ele deve encaminhá-la para o seu sucessor que repetirá o processo caso necessário. Quando a mensagem finalmente alcançar o seu destinatário, ele a processa e a responde enviando a resposta ao seu predecessor até que ela alcance o nó que originou a comunicação, seguindo a sequência reversa dos nós envolvidos no envio da mensagem original.

Já as demais mensagens ({`JOIN_OK`, `OK`, `LEAVE`, `NEW_NODE`, `NODE_GONE`, `TRANSFER` e `NOT_FOUND`) podem ser enviadas diretamente ao destinatário via chamada gRPC.

Note que você não precisa necessariamente manter uma conexão aberta entre os nós e seus sucessores e predecessores o tempo todo: os endereços IP e portas são conhecidos e você pode sempre abrir uma nova conexão quando necessário. Você também pode querer considerar um canal de stream para manter uma conexão permanente. A depender da sua implementação, cada escolha terá as suas vantagens e desvantagens.

[1] [ST] Capítulo 6, Seção 6.2.3

3.3 Funções de hash

Para obter os identificadores dos nós e dos objetos você pode utilizar uma função de hash pronta dentre as disponíveis na linguagem de programação da sua escolha. Favoreça funções de hash criptograficamente seguras (disponíveis na maior parte das linguagens de programação) pois elas garantem que os hashes serão uniformemente distribuídos. Como sugestão para calcular o identificador de um nó, utilize o endereço IP concatenado com a porta que ele está utilizando. Durante os testes você pode querer utilizar funções mais simples de hash (ou até mesmo valores baixos escolhidos manualmente) para poder avaliar o funcionamento da sua implementação.

3.4 Dicas para testar o seu código

Quando estiver testando o seu código utilize pelo menos 4 nós. Não há problema algum caso todos os nós estejam no mesmo computador, apenas preste atenção para que cada um deles esteja usando um número de porta diferente. Também deixe de uma maneira relativamente fácil no seu código uma funcionalidade para especificar o identificador do nó e de um dado a ser armazenado na DHT manualmente. Como os identificadores gerados pelas funções de hash são longos isso pode facilitar a depuração de eventuais bugs.

Cuidado também com concorrência por acesso às estruturas de dados compartilhadas. Lembrem-se que possivelmente milhares de clientes simultâneos podem estar enviando e recebendo mensagens ao mesmo tempo e pode causa concorrência por acesso a estruturas compartilhadas.

4 Milestones

A seguir descrevemos uma série de *milestones* cuja ordem sugerimos que seja seguida pois ela facilitará o desenvolvimento incremental do projeto.

4.1 Milestone 1

Escreva uma aplicação simples que usa a API definida na Seção DHT. A aplicação deverá prover uma interface (texto ou GUI, são aceitáveis). A aplicação deverá utilizar todas as operações descritas na API da DHT.

4.2 Milestone 2

Implemente o anel da DHT de modo que ele possa ser criado e mantido. Mais especificamente, este *milestone* requer que seu código seja capaz de responder corretamente às mensagens `JOIN`, `JOIN_OK`, `NEW_NODE`, `LEAVE` e `NODE_GONE`. Com o tratamento dessas mensagens implementado, as operações `join()` e `leave()` da DHT já devem funcionar corretamente. Como

neste ponto do campeonato não haverá nenhuma chave armazenada nos nós da DHT, não haverá o envio de mensagens `TRANSFER` entre os nós.

4.3 Milestone 3

Neste *milestone* você deverá escrever o código para dar suporte ao armazenamento e recuperação de objetos da DHT. Para isto o resto das mensagens definidas pelo protocolo deverão ser implementadas. Não há necessidade de persistir as chaves/valores armazenadas pelos nós, podendo-se considerá-las efêmeras.

5 Instruções

O projeto tem como **prazo de entrega 15/09/2024**. A entrega deverá ser feita via GitHub Classroom. Não serão aceitas entregas por outros meios. O link para entrega será disponibilizado na página da disciplina.

5.1 Linguagem

Você pode implementar o seu código na sua linguagem de programação favorita, seja ela qual for. Contudo tenha em mente que:

- O ambiente de correção é uma máquina Linux, logo é importante que você se assegure que o seu código funciona neste ambiente ainda que você o desenvolva no Windows, MacOS, etc.
- Códigos que não compilam ou que não contiverem instruções para sua compilação receberão nota ZERO.
- Junto com o seu projeto você deverá entregar uma aplicação que exemplifiquem o uso da sua DHT. Esta aplicação devem estar descrita no relatório e mostrada no vídeo.

5.2 Equipe

O projeto pode ser feito em grupos de no máximo 3 pessoas

- Apenas um dos integrantes do grupo deve fazer o upload do projeto para o GitHub e deve estar bem claro quais são os integrantes da equipe no relatório.
- Todos os *commits* e o *push* final devem ter sido feitos até no máximo às **23h59 do dia 15/09/2024**.

– Atenção, caso ocorram/commits/pushes após o prazo, será considerada apenas a última versão para avaliação. Caso o último esteja com atraso > 3 dias, então será considerado o último commit até 3 dias (valendo no máximo 5).
– Entregas em atraso sofrerão descontos conforme a tabela abaixo.

Dias em atraso	Nota máxima
1 dia	7
2 dias	6
3 dias	5
>3 dias	0

5.3 Entregáveis

Juntamente com o código do seu projeto, você deverá entregar o código da aplicação que demonstre o funcionamento da sua DHT.

No mesmo repositório do seu código, deverá haver um relatório de no máximo 1 páginas (em PDF! nada de `.md`, `.docx`, `.odt`, ou qualquer outro formato da moda…) descrevendo:

- O seu projeto
- Como utilizar o seu código
- Dificuldades, surpresas e destaques do seu código

O seu relatório também deverá conter **obrigatoriamente** um link para um vídeo que:

- Esteja abrigado no Youtube (ou qualquer serviço semelhante)
- Seja privado/não listado e disponível apenas àqueles com o link
- Tenha no máximo 3 minutos (ou seja, <= 180 segundos). **Sem exceções**.
- Mostre a compilação, o uso e uma apresentação do código do projeto (pode acelerar/cortar a espera pela compilação deixando apenas as partes relevantes).
- Durante a correção o professor pode pedir por esclarecimentos adicionais.

5.4 Dúvidas

Todas as dúvidas devem ser sanadas diretamente com o professor através do Discord da disciplina.