Projeto 2
Tabela de Hash Distribuída (DHT)

Profs. Emilio Francesquini e Fernando Teubl Ferreira
{e.francesquini,fernando.teubl}@ufabc.edu.br

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

Entrega: **27 de agosto de 2018**

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. 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.

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

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 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 podem ser feitas utilizando *sockets* TCP/IP. Se você desejar, poderá também utilizar JRMI (ou o equivalente na linguagem de programação que você escolher) para implementar a comunicação entre os nós. Cada um dos métodos tem as suas vantagens, porém o uso de TCP/IP acaba deixando mais claro o fluxo de mensagens entre os participantes da rede para o programador.

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.

No protocolo que sugerimos, os nós farão a comunicação usando um protocolo baseado em texto puro[1] que seguirá o seguinte formato:

OP	**Argumentos**	**\n**

Onde **OP** identifica o tipo de mensagem e **Argumentos** contém todos os argumentos necessários para que aquela mensagem possa ser processada pelo destinatário. Caso o tipo da mensagem exija mais de um argumento, eles são enviados um seguido do outro, todos na mesma linha separados por espaços. Cada mensagem termina quando o caractere de quebra de linha (`\n`) for encontrado.

As seguintes mensagens são necessárias para o funcionamento do protocolo:

- **JOIN** Recebe como argumentos o identificador, o endereço IP e a porta do nó ingressante. O identificador é um número de m bits sem sinal que pode

[1] Uma implementação de DHT para uso real provavelmente privilegiaria um protocolo binário em vez de um protocolo baseado em texto puro. Neste projeto, por simplicidade, abrimos mão do possível ganho de desempenho que um protocolo binário propiciaria.

ser enviado utilizando a sua representação em decimal. Já o IP é enviado 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 2.1, 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.

- **NEW_NODE** Quando um novo nó ingressa na DHT, a sua mensagem `JOIN` é enviada ao antigo responsável pelo identificador do novo nó, ou seja, seu sucessor. O no ingressante então recebe o seu predecessor via mensagem `JOIN_OK` do seu 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 no formato de um inteiro decimal sem sinal. Em seguida, ela recebe um número `n` que indica o número de bytes do objeto a ser transferido. Em seguida, são passados `n` inteiros com valores entre 0 a 255 (inclusive) representando cada um dos bytes do objeto a ser armazenado. A

mensagem, como as demais, acaba no final da linha que é demarcado por um `\n`.

Note que esta maneira de representar os dados binários a serem transferidos não é nada eficiente. Para cada byte do dado sendo enviado serão transferidos até 3 bytes em sua representação como decimal em texto puro. Existem outras maneiras bem mais eficientes, utilizando apenas texto puro, que poderiam ser utilizadas para transferir uma sequência de bytes (por exemplo, Base64[2]). Caso deseje, fique à vontade para alterar o protocolo para utilizá-las. Isto será bem recompensado durante a correção.

- **RETRIEVE** Esta mensagem é usada para recuperar valores armazenados na DHT. Ela recebe como argumentos a chave sendo buscada representada por um inteiro sem sinal (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` inteiros entre `0` e `255` (inclusive), no mesmo formato utilizado pela mensagem `STORE`. A mensagem como as demais, acaba no final da linha que é demarcado por um `\n`.

- **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.

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

[2] `https://en.wikipedia.org/wiki/Base64`

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[3]. 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 TCP/IP (ou RMI caso este seja o método de comunicação escolhido).

Caso você tenha escolhido TCP/IP para o envio das suas mensagens, 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. Isso vai simplificar muito a sua implementação.

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 utilizar funções mais simples de hash (ou até mesmo valores baixos escolhidos manualmente) para poder avaliar o funcionamento da sua implementação.

[3] [ST] Capítulo 5, Seção 5.3.

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.

4 Milestones

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

Milestone 1 Escreva uma aplicação simples que usa a API definida na Seção 2.1. A aplicação deverá prover uma interface (texto ou GUI). A aplicação deverá utilizar todas as operações descritas na API da DHT.

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 as 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 não haverá nenhuma chave armazenada nos nós da DHT, não haverá o envio de mensagens `TRANSFER` entre os nós.

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.

5 Instruções

As instruções abaixo devem ser seguidas à risca:

GRUPOS O projeto deve ser feito em *equipes de até 4 pessoas*. A lista com os integrantes de cada grupo deve ser enviada ao professor da sua turma de prática pelo TIDIA em tarefa criada especificamente com este propósito até **no máximo dia 30/07/2018**.

- Grupos que não informarem os seus integrantes no prazo acima terão **1 ponto deduzido na nota final do projeto**.

DÚVIDAS Questões e discussões relativas ao enunciado do projeto ou sua implementação devem ser discutidas no fórum do TIDIA. Todos são **fortemente encorajados** a participar das discussões e ajudar seus colegas.

RELATÓRIO Entregue junto com o projeto um relatório detalhado descrevendo a solução implementada e alguns exemplos de uso da interface do projeto. Por exemplo, se você implementar um *prompt* de comando, cada exemplo consistiria de um sessão com os comandos necessários e suas respectivas saídas. Os cenários serão testados durante a correção, portanto assegure-se que eles estão completos e autoexplicativos. Seu relatório deve ser enviado no formato PDF e deve ter **no máximo 4 páginas**.

- Relatórios que não descrevem detalhadamente os passos necessários para compilar e executar o projeto poderão sofrer uma redução significativa na nota final.
- Projetos com erros de compilação receberão nota 0

ENTREGA A entrega deve ser feita no prazo indicado no início deste documento em um único arquivo contendo todo o código fonte, relatório e arquivos relevantes via TIDIA em uma atividade que será criada exclusivamente para este fim. Lembre-se que seu código também será avaliado pela organização, clareza e comentários.

- Entregas de projetos com atraso serão aceitas. Contudo a cada dia de atraso a sua nota máxima será atualizada conforme abaixo:

Dias em Atraso	**Nota Máxima**
0	10
1	8
2	7
3	6
4	5
5 ou mais	0

6 Datas Importantes

- **30/07/2018** - Informar ao professor da prática (via TIDIA) os RAs e Nomes dos participantes de cada grupo.
- **27/08/2018** - Entrega via TIDIA do projeto final.

Esse projeto é uma adaptação daquele proposto pelo Prof. Jussi Kangasharju da Universidade de Helsinki[4] *para o curso de Sistemas Distribuídos.*

[4] `https://www.cs.helsinki.fi/en/courses/582665/2012/K/K/1`