Documentação do projeto DHT

Sumário

Instalação

Para a instalação do projeto é necessário ter o ambiente Node.js instalado no computador. As instruções de instalação para o rutime estão no site oficial: http://nodejs.org

> Ao instalar, certifique-se de que os comandos `node -v` e `npm -v` estejam presentes no seu terminal.

Clonar o repositório

Primeiro clone o repositório do projeto usando `git clone http://github.com/khaosdoctor/node-dht`

Instalar dependências

Após a clonagem, entre na pasta do projeto, na raiz (aonde se encontra o arquivo `package.json`) e execute o comando `npm install`. Aguarde a finalização e certifique-se de que, no final, há uma pasta chamada `node_modules` nesta mesma raiz.

Execução do projeto

Para executar o projeto, na raiz do mesmo, execute o comando `node index.js <nodeList>`, onde `<nodeList>` é uma lista de hosts conhecidos de uma DHT já existente separados por vírgula, exemplo:

```
$ node index.js #Vai criar uma nova rede, já que não há nenhum nó conhecido
```

```
$ node index.js localhost:4344 #Tentará se conectar ao nó enviado, se não conseguir, irá criar uma nova rede
```

```
$ node index.js localhost:4344,localhost:5656 #Tentará se conectar a cada um dos nós enviados, se nenhum responder, criará uma nova rede
```

Interação

Ao se conectar ou criar uma nova rede, o nó será conectado ao próximo nó e então um prompt será exibido na tela, então o usuário poderá digitar os comandos:

- `help`: Mostra a mensagem de ajuda com todos os comandos disponíveis
- `info`: Mostra informações sobre o nó, seu sucessor e predecessor, bem como informações de si próprio, também de sua lista de arquivos local e as estatísticas de contagem de quantas mensagens foram recebidas pelo nó
- `debug`: Ativa/Desativa o modo debug, uma vez ativado, o nó irá printar **todas as mensagens TCP recebidas** na sua forma crua.
- `store <nome> <caminho-do-arquivo>`: Armazena um arquivo na rede. O primeiro parâmetro é o nome da chave que este arquivo terá na rede, o segundo, separado por espaço, é o caminho para o arquivo que contém o conteúdo que será armazenado, por exemplo, `./package.json`.
  - Exemplo: `store informações ../informações.txt` irá armazenar o conteúdo do arquivo presente em `../informações.txt` em uma chave `informações` na rede
- `retrieve <chave> <local-para-salvar>`: Busca um arquivo na rede e salva o seu conteúdo no local especificado. Se o local for um arquivo existente ele será **sobrescrito**, caso contrário será criado
  - Exemplo: `retrieve informações ./myInfo.txt` vai buscar a chave `informações` na rede e, se encontrar, irá armazenar seu conteúdo em um arquivo `./myInfo.txt`
- `leave`: Informa que o nó sairá da rede, transfere todos os seus arquivos para seu sucessor e informa o mesmo sobre as atualizações de ponteiros que deverão ser feitas para manter o anel.

> Em alguns casos é um bug conhecido que o prompt **não irá printar o caracter** `>`, mas isto não significa que ele não está ouvindo por comandos, basta digitar `help` ou `info` que ele irá printar o caracter de entrada novamente.

Arquivos

O projeto segue uma organização simples de arquivos:

- `src`: Contém todos os arquivos do projeto

- `config`: Contém o arquivo que dita quais são as strings enviadas pelas mensagens TCP
- `consoleCommands`: Contém os *handlers* de ativação do comandos dados pelo usuário no terminal. Quando um usuário digitar algo, o comando será avaliado e, se for válido, terá um arquivo com seu nome chamado para execução
- `messages`: Contém os *handlers* de mensagens recebidas, estes arquivos cuidam apenas das mensagens que chegam de outros nós, não de mensagens enviadas pelos próprios
- `utils`: Utilidades de desenvolvimento
  - `getRandomInt`: Obtém um inteiro aleatório entre um *range* dado pelo desenvolvedor, atualmente é utilizado para sortear portas entre 3000 e 65000
  - `hashFactory`: Contém as funções que geram o hash SHA256 utilizado para os ID's dos nós e também para os hashes das chaves dos arquivos armazenados na DHT, bem como *encoding* e *decoding* de base64
  - `socketClient`: Arquivo *wrapper* da biblioteca `net` do Node.js, responsável por criar e enviar as mensagens do nó atual para outros nós. Também é responsável por padronizar os envios e mandar sempre o mesmo conteúdo de mensagem
  - `socketServer`: O mesmo que o `socketClient` mas inicia o servidor TCP local que fica ouvindo novas conexões para receber mensagens de outros nós

Protocolo

O protocolo seguido é basicamente o mesmo descrito no arquivo de enunciado do projeto. Algumas adições foram feitas para facilitar a comunicação e o roteamento das mensagens no anel:

LEAVE_ACK

Mensagem de confirmação enviada após um nó receber o comando `LEAVE` de outro nó. Este arquivo basicamente só serve para que possamos desconectar da DHT com sucesso dada a natureza assíncrona dos eventos TCP no Node.js.

> Basicamente esta foi uma maneira mais rápida do que criar *binds* para os eventos do socket, que incluiriam uma alteração na biblioteca client principal. Então, para ficar mais simples, coloquei uma resposta informando que o nó pode ser desconectado com sucesso.

TRANSFER_ACK

Esta mensagem é apenas uma confirmação de recebimento de transferência para evitar um problema: Se uma mensagem `TRANSFER` for enviada mas não recebida, o nó que a enviou não pode remover o arquivo da sua lista de arquivos ou isto causaria uma inconsistência e perda de arquivos.

Para isto, sempre que uma mensagem `TRANSFER` é enviada, o nó que a recebe envia uma nova mensagem ao remetente do tipo `TRANSFER_ACK`, informando que o arquivo pode ser deletado da lista de arquivos com segurança.

Fluxo

O fluxo desta DHT funciona da seguinte maneira

1. O nó é inicializado com 0 ou mais nós conhecidos
    1. Se o número de nós conhecidos for 0 então uma nova DHT é criada
    2. Se o número de nós conhecidos for maior que 0 então, um a um, o nó que está iniciando tentará a conexão. O primeiro que obtiver sucesso será tido como o nó "pai" do nó ingressante e será responsável por rotear as mensagens de `JOIN` para os demais nós
    3. Se nenhum dos nós de uma lista de nós responder, caímos no estado 1
2. Ao se conectar a um nó conhecido, o ingressante manda uma mensagem `JOIN`, que será recebida pelo nó conhecido e avaliada.
    1. Se ele for o nó com o ID mais próximo do ID do nó ingressante então ele será o responsável pelo nó, se tornando seu sucessor (e, consequentemente, atualizando seu próprio predecessor para o novo nó) e enviando a mensagem `JOIN_ACK` diretamente para o ingressante informando qual será o seu nó sucessor e predecessor.
    2. Caso contrário a mensagem será roteada para o próximo nó, que fará a mesma coisa.
    3. Uma vez que o nó avalie com sucesso a posição do ingressante, ele então irá voltar para o caso 1, enviando a mensagem `JOIN_ACK` para o nó ingressante.
    4. Após o envio da mensagem `JOIN_ACK` também será enviado a mensagem `NEW_NODE` para o nó predecessor informando que ele deve atualizar seu sucessor para o nó que acabou de entrar.
    5. A partir daí, se o nó que recebeu a mensagem `JOIN` possuir arquivos, ele passara por cada arquivo verificando se o nó que ingressou na rede tem seu ID mais próximo de alguma das chaves do que ele próprio, se sim, enviará uma mensagem `TRANSFER` para cada arquivo que se encaixe nessa regra.
3. O nó ingressante recebe a mensagem `JOIN_ACK` e atualiza seus ponteiros sucessor e predecessor.
4. O nó predecessor do nó que recebeu a mensagem `JOIN` do nó ingressante irá receber a mensagem `NEW_NODE` com os dados do novo nó. Ele então atualizará seu ponteiro sucessor para o novo nó
5. Após atualizar seu predecessor e informar o antigo predecessor sobre o novo nó, o nó que recebeu a mensagem `JOIN` irá enviar 0 ou mais mensagens `TRANSFER` para o nó ingressante, somente se sua lista de arquivos contiver algum arquivo cuja chave seja mais próxima do ID do novo nó do que do seu próprio
    1. Ao receber uma mensagem `TRANSFER` o novo nó irá responder com uma mensagem `TRANSFER_ACK` informando o nó que pediu a transferência, que a mesma foi concluída e ele pode remover o arquivo da sua lista
6. Após este fluxo o nó está pronto para receber os comandos `STORE` e `RETRIEVE`.
    1. Ao receber um comando de usuário `STORE` o nó irá ler o arquivo descrito no caminho e transformar seu conteúdo em base64

2. Após esta verificação, ele checa se o seu próprio ID é mais próximo do hash da chave do que o ID do seu sucessor (as comunicações sempre acontecem de maneira horária)
    1. Se sim, armazenará o arquivo nele mesmo
    2. Caso contrário irá enviar uma mensagem `STORE` contendo o arquivo para o nó sucessor, que repetirá o processo a partir de 6.1
3. Ao receber um comando de usuário `RETRIEVE`, o nó atual irá transformar a chave de busca em um hash.
4. Se o arquivo existir em sua própria lista, ele irá decodificar o conteúdo em base64 para um Buffer e salvá-lo no caminho especificado
5. Caso contrário a mensagem `RETRIEVE` será enviada ao próximo nó, que irá checar se seu ID é mais próximo do hash do que seu sucessor
    1. Se sim, enviará a mensagem `FOUND`, juntamente com o arquivo, diretamente para o nó que o pediu
    2. Se não, roteará a mensagem para seu sucessor
6. Se houver um ponto onde o nó é o escolhido por cuidar do arquivo com determinado hash, mas o arquivo não existir, então este nó irá enviar a mensagem `NOT_FOUND` diretamente ao nó requisitante do arquivo, informando que o arquivo não foi encontrado

7. Ao receber o comando `LEAVE` do usuário, o nó irá enviar uma mensagem `LEAVE` a seu sucessor, enviando juntamente a ela o endereço do seu ponteiro predecessor
    1. Uma vez que o sucessor recebe a mensagem `LEAVE` ele atualiza seu predecessor para que seja o nó enviado pelo nó que está saindo da rede
    2. O nó que recebeu a mensagem `LEAVE` então encaminha outra mensagem `NODE_GONE` para o seu novo predecessor já atualizado, informando que ele é o seu novo sucessor
        1. O nó que receber a mensagem `NODE_GONE` irá atualizar seu sucessor com o novo sucessor recebido
    3. Após enviar a mensagem leave, o nó que está saindo irá enviar 0 ou mais mensagens `TRANSFER` para seu sucessor, informando que está passando a responsabilidade dos arquivos sob sua tutela para ele
        1. O nó sucessor que receber a mensagem `TRANSFER` irá armazenar os arquivos em sua lista de arquivos
        2. Após o recebimento de todos os arquivos, se houverem, o nó sucessor do nó que está saindo enviará uma mensagem `LEAVE_ACK`, que permitirá que o nó se desconecte da rede

Outras informações

Outras informações sobre o fluxo podem ser encontradas no repositório do projeto https://github.com/khaosdoctor/node-dht, porém, em inglês.