

Computação Distribuída – Guião 2

Diogo Gomes & Nuno Lau & Mário Antunes & Alfredo Matos
Março 2023

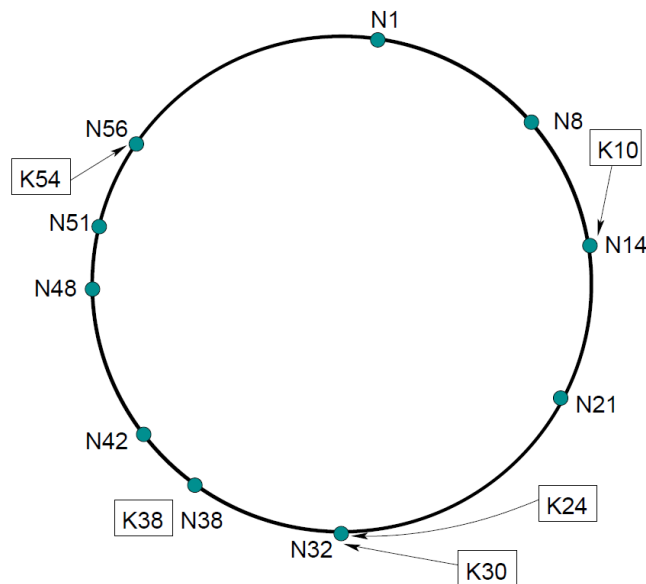
Introdução

O objectivo deste guião é desenvolver uma DHT, nomeadamente CHORD. Chord é um protocolo e um algoritmo que implementam uma DHT (distributed hash table). Uma DHT armazena pares de chaves-valor distribuídos entre diferentes computadores (chamados nós). Um nó só armazena os valores pelos quais é responsável. O protocolo Chord define como as chaves e os valores devem ser distribuídos por nós.

Tanto os nós como os objectos são identificados por um ID, gerado através de hash consistente. Tipicamente a SHA-1 é usada como algoritmo de hash. A consistência do algoritmo de hash é crucial para maximizar a performance da DHT.

Baseados no valor de ID os nós são organizados em anel ordenado. De forma simplificada, a pesquisa no CHORD é efectuada da seguinte forma:

- Cliente contacta um nó do anel e requisita uma determinada chave
- O nó verifica se possui a chave em questão
- Se possui, retorna o objeto ao cliente
- Se não possui, propaga a mensagem para o próximo nó no anel



Anel Chord com 10 nós e 5 chaves. Crédito [1]

Objectivos

- Completar uma implementação adaptada do protocolo Chord.
- Comunicação através de sockets UDP.
- Trabalho em equipa usando repositório GIT

Requisitos da solução

- O código já implementa os principais métodos usados para a criação do anel e garante a sua correção, mas falta a implementação válida da função de `utils.py: contains()`.
- As funções que implementam a inserção e pesquisa de objectos não estão completas.
 - falta verificar se os objectos existem (`get()`) ou devem ser armazenados localmente (`put()`)
 - falta propagar para o resto do anel no caso da responsabilidade ser de outro nó:
 - Implementar a pesquisa remota (performance $O(n)$)
 - Expandir o código para suportar Finger Tables (em vez de apenas conhecer um único sucessor)
 - Implementar a pesquisa remota que utiliza as Finger Tables (performance $O(\log(n))$) para `get()` e `put()`
 - **Extra/Opcional:** Suportar a remoção de nós em runtime.
- Comunicação através de socket UDP.
- As mensagens deverão ser codificadas em Pickle.
- Ajustar o número de nós e a precisão da hash como necessário.

Prazo

29 de Março de 2023 pelas 23h00

Entrega através do Github Classroom (automática)

GitHub Classroom

- Este projecto é realizado em **grupos de 2** alunos.
- Para resolver este guião deverá começar por aceitar o mesmo em <https://classroom.github.com/a/s5vvB0P1>
- Ao aceitar o guião será criado um repositório online a partir do qual deve fazer um clone local (no seu computador).
- Deverá enviar as suas alterações periodicamente para o repositório e manter-se atento ao canal `#cd` em <https://detiuaveiro.slack.com>
- Antes do prazo, o seu código deverá passar os testes automáticos (*tab "Actions"* no seu repositório github)

Notas

- Devido a restrições na rede Eduroam, o sistema apenas funcionará no seu computador local (o código já está preparado para isto).

Protocolo

Todas as mensagens devem ser codificadas em Pickle e ser encaminhadas usando sockets UDP.

Objetivo	Destino	Mensagem
Inserir (<i>key</i> , <i>value</i>)	Qualquer nó	cliente: {"method": "PUT", "args": {"key": <i>key</i> , "value": <i>value</i> }} entre nós: {"method": "PUT", "args": {"key": <i>key</i> , "value": <i>value</i> , "from": <i>client_addr</i> }}
Resposta	Cliente	{'method': 'ACK'}
Obter <i>value</i>	Qualquer nó	cliente: {'method': 'GET', 'args': {'key': <i>key</i> }} entre nós: {"method": "GET", "args": {"key": <i>key</i> , "from": <i>client_addr</i> }}
Resposta	Cliente	{'method': 'ACK', "args": <i>value</i> }
Inserir nó <i>id</i> com endereço <i>addr</i>	Qualquer nó	{'method': 'JOIN_REQ', 'args': {'addr': <i>addr</i> , 'id': <i>id</i> }}
Resposta	Novo nó	{'method': 'JOIN_REP', 'args': {'successor_id': <i>succ_id</i> , 'successor_addr': <i>succ_addr</i> }}
Notificar sucessor de existência	Sucessor	{'method': 'NOTIFY', 'args': {'predecessor_id': <i>id</i> , 'predecessor_addr': <i>addr</i> }}
Resposta	n.a.	n.a.
pedir predecessor do sucessor	Sucessor	{'method': 'PREDECESSOR'}
Resposta	Nó que fez o pedido	{'method': 'STABILIZE', 'args': <i>pred_id</i> } ao receber esta mensagem o receptor inicia stabilize()
nó <i>addr</i> pede sucessor de <i>id</i>	Predecessor mais próximo de <i>id</i> ou sucessor de <i>id</i>	{"method": "SUCCESSOR", "args": {"id": <i>id</i> , "from": <i>addr</i> }}
Resposta	Nó que fez o pedido	{"method": "SUCCESSOR_REP", "args": {"req_id": <i>req_id</i> , "successor_id": <i>succ_id</i> , "successor_addr": <i>succ_addr</i> }}

2

Referências

1. "Chord: A scalable peer-to-peer lookup protocol for Internet applications", Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.,

IEEE/ACM Transactions on Networking, 11 (1), pp. 17-32. (2003)

DOI: 10.1109/TNET.2002.808407

<https://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf>