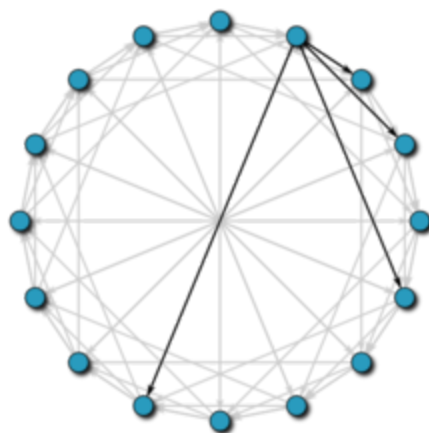


## Opção A

*Publish-Subscribe using Structured*



Trabalho realizado por:

**Luís Duarte Oliveira, nº 41894**    **Daniel Pimenta, nº 45404**  
ld.oliveira@campus.fct.unl.pt    d.pimenta@campus.fct.unl.pt

**Luís Martins, nº 45640**  
lg.martins@campus.fct.unl.pt

Para a cadeira de:  
Algoritmos e Sistemas Distribuídos (ASD)

Professor regente: João Leitão  
Professor responsável: Nuno Preguiça

Departamento de Informática  
Faculdade de Ciências e Tecnologia  
Universidade Nova de Lisboa  
17 de novembro de 2018

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>4</b>
<b>2</b>	<b>Visão global</b>	<b>5</b>
<b>3</b>	<b>Visão global detalhada</b>	<b>6</b>
3.1	Chord . . . . .	7
3.2	Publish-Subscribe . . . . .	8
3.3	Aplicação de teste . . . . .	9
<b>4</b>	<b>Pseudo-código e arquitetura</b>	<b>10</b>
4.1	Chord . . . . .	11
4.2	Publish-Subscribe . . . . .	12
4.3	Aplicação de teste . . . . .	13
<b>5</b>	<b>Avaliação dos resultados</b>	<b>14</b>
5.1	Experiências . . . . .	14
5.2	Resultados com observações . . . . .	14
<b>6</b>	<b>Conclusão</b>	<b>15</b>

# Lista de Figuras

1	Pseudo-código do algoritmo Chord. . . . .	11
---	---	----

## Lista de Tabelas

# 1 Introdução

No âmbito da cadeira de Algoritmos e Sistemas Distribuídos foi-nos lançado o desafio de implementar um sistema de *Publish-Subscribe* em cima de uma *distributed hash table* (DHT). Uma DHT é um modelo de rede estruturada com capacidade para guardar dados de forma distribuída em cada nó da mesma.

Com a implementação deste sistema podemos estudar a complexidade desta classe de algoritmos, aprender a realizar testes sobre uma rede deste tipo e aprender a identificar as propriedades do sistema.

## 2 Visão global

Este sistema pode ser decomposto em 3 camadas. A primeira é a camada que dá suporte ao sistema. Esta no nosso caso corresponde a implementação do algoritmo Chord [2]. A segunda é a camada relacionada com o sistema de *Publish-Subscribe*. Esta é responsável pela criação e subscrição dos respetivos tópicos de interesse a cada individuo na rede. Com estas subscrições será possível a troca de mensagens dentro de cada tópico. Por ultimo é a camada de teste. Esta é responsável pela recolha e processamento dos dados de teste para futura análise.

As primeiras duas camadas foram implementadas tendo como guia os materiais da cadeira e ASD [1].

A implementação deste sistema foi feita com a linguagem computacional de Scala [3] e utilizando o conjunto de ferramentas do Akka [4].

### 3 Visão global detalhada

Nesta secção iremos explicar textualmente ao pormenor os detalhes de cada uma das camadas anteriormente mencionadas.

### 3.1 Chord

O Chord quando arranca o primeiro nó inicializa o predecessor a nulo e o sucessor como ele mesmo. No caso de não ser o primeiro, inicializa o predecessor a nulo e o sucessor com o resultado da função de encontrar o sucessor.

A função de encontrar o sucessor funciona tendo um nó  $n$  que procura um nó  $n'$ .

As restantes funções do algoritmo têm como objectivo manter o sistema num estado estável e correcto. Estas são chamadas regularmente do início ao fim da execução.

A função de estabilização verifica se o nó contém o sucessor correcto e notifica o sucessor sobre quem é o seu predecessor.

É importante verificar se os *fingers* estão a apontar para os valores correctos. Para isso temos a função de arranjar os *fingers*. Esta função a cada execução verifica um finger diferente para poder analisar gradualmente todos os fingers.

Por ultimo temos a função de verificar o predecessor que confirma se o predecessor é um nó activo.



## 3.2 Publish-Subscribe

O Publish-Subscribe tem como função permitir os clientes fazerem a subscrição em determinados topicos e submeterem mensagens nos mesmos.

Cada nó de cliente por subscrição submete uma mensagem utilizando o comando *route*.

Regularmente os clientes verificam se os topicos ainda estão ativos, caso nao estejam, então é criados o novo topico.

Os nós de subscrição verificando os interessados em cada topico e caso nao existam o topico é apagado.

### 3.3 Aplicação de teste

TO DO...

## 4 Pseudo-código e arquitetura

TO DO...

## 4.1 Chord

O pseudo-código do Chord segue rigorosamente o algoritmo apresentado pelo professor nos slides da cadeira de ASD [1].

```
// Ask node n to find the successor of id.
// The condition of the if ( $id \in (n, \text{successor}]$ ) must
// have in account the many valid intervals.
n.find_successor(id)
    if ( $id \in (n, \text{successor}]$ )
        return successor;
    else
         $n' = \text{closest\_preceding\_node}(id)$ ;
        return  $n'.\text{find\_successor}(id)$ ;

// Search the local table for the highest predecessor of id.
n.closest_preceding_node(id)
    for  $i = m$  downto 1
        if ( $\text{finger}[i] \in (n, id)$ )
            return  $\text{finger}[i]$ ;
    return  $n$ ;

// Create a new Chord ring.
n.create()
    predecessor = nil;
    successor =  $n$ ;

// Join a Chord ring containing node  $n'$ .
n.join( $n'$ )
    predecessor = nil;
    successor =  $n'.\text{find\_successor}(n)$ ;

// Called periodically. Verifies  $n$ 's immediate
// successor, and tells the successor about  $n$ .
n.stabilize()
     $x = \text{successor.predecessor}$ ;
    if ( $x \in (n, \text{successor})$ )
        successor =  $x$ ;
    successor.notify( $n$ );

//  $n'$  thinks it might be our predecessor.
n.notify( $n'$ )
    if (predecessor is nil or  $n' \in (\text{predecessor}, n)$ )
        predecessor =  $n'$ ;

// Called periodically. Refreshes finger table entries.
// next stores the index of the next finger to fix.
n.fix_fingers()
    next = next + 1;
    if (next >  $m$ )
        next = 1;
     $\text{finger}[\text{next}] = \text{find\_successor}(n + 2^{\text{next}-1})$ ;

// Called periodically. Checks whether predecessor has failed.
n.check_predecessor()
    if (predecessor has failed)
        predecessor = nil;
```

Figura 1: Pseudo-código do algoritmo Chord.

## 4.2 Publish-Subscribe

TO DO...

### 4.3 Aplicação de teste

TO DO...

## 5 Avaliação dos resultados

TO DO...

### 5.1 Experiências

TO DO...

### 5.2 Resultados com observações

TO DO...

## 6 Conclusão

TO DO...



## Referências

- [1] Leitão, J. (2018). Materiais da cadeira de ASD. FCT/UNL.
- [2] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., & Balakrishnan, H. (2003). Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11(1), 17–32. <https://doi.org/10.1109/TNET.2002.808407>
- [3] Fédérale, É. P., & (EPFL), L. (n.d.). Scala. Retrieved November 17, 2018, from <https://www.scala-lang.org/>
- [4] Lightbend, I. (n.d.). Akka. Retrieved November 17, 2018, from <https://akka.io/>
- [5] Penman, T. (n.d.). Implementing a Distributed Hash Table with Scala and Akka. Retrieved November 17, 2018, from <http://tristanpenman.com/blog/posts/2015/11/26/implementing-a-dht-with-scala-and-akka/>