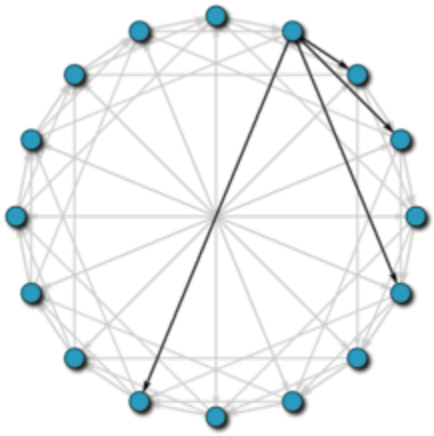
Opção A

Publish-Subscribe ***utilizando redes estruturadas***



Trabalho realizado por:

Luís Duarte Oliveira, nº 41894 `ld.oliveira@campus.fct.unl.pt`

Daniel Pimenta, nº 45404 `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

Lista de Figuras

Lista de Tabelas

1 Introdução

No âmbito da cadeira de Algoritmos e Sistemas Distribuídos, foi-os proposto o projeto, de implementar um sistema de *Publish-Subscribe*, em cima de uma *distributed hash table* (DHT).

DHT é um modelo de rede estruturada, com capacidade para guardar dados de forma distribuída, em cada nó da mesma.

Na implementação deste sistema, constatamos a complexidade desta classe de algoritmos. Ao realizar testes sobre uma rede deste tipo, aprendemos 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.

A terceira e última, 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 de ASD [1].

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

3 Visão global detalhada

Nesta secção, iremos explicar por escrito, em pormenor, os detalhes de cada uma das camadas anteriormente mencionadas.

3.1 Chord

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

A função de encontrar o sucessor, funciona com a seguinte condição, "determinado nó, pede ajuda ao nó anterior, para encontrar o nó posterior".

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

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

A função de arranjar os *fingers*. É importante verificar se os *fingers* estão a apontar para os valores corretos, caso contrário torna a execução do algoritmo impossível. Esta função, analisa gradualmente, *finger* a *finger*, se cada *finger* de cada nó tem os valores certos.

Por último, temos a função de verifica se o predecessor é um nó ativo, ou não.

3.2 Publish-Subscribe

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

Cada cliente pode publicar mensagens em cada tópico, sendo as mesmas disseminadas por todos os subscritores. As mensagens são publicadas, utilizando o comando *route*.

Regularmente os clientes verificam se os tópicos, ainda estão ativos. Caso não estejam, então é criado um novo tópico.

Periodicamente, os nós que têm tópicos, verificam se o tópico é procurado. Caso não seja, o tópico 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 ∈ (n, successor]) must
// have in account the many valid intervals.
n.find_successor(id)
   if (id ∈ (n, successor])
      return successor;
   else
      n' = closest_preceding_node(id);
      return n'.find_successor(id);

// Search the local table for the highest predecessor of id.
n.closest_preceding_node(id)
   for i = m downto 1
      if (finger[i] ∈ (n, id))
         return 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'.find_successor(n);

// Called periodically. Verifies n's immediate
// successor, and tells the successor about n.
n.stabilize()
   x = successor.predecessor;
   if (x ∈ (n, successor))
      successor = x;
   successor.notify(n);

// n' thinks it might be our predecessor.
n.notify(n')
   if (predecessor is nil or n' ∈ (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;
   finger[next] = find_successor(n + 2^(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/