Instituto Superior de Engenharia de Lisboa

Computação Distribuída

**Chat Distribuído (Anel de Servidores) com Recurso ao Middleware gRPC**

Mestrado em Engenharia Informática de Multimédia

Pedro Gonçalves, 45890

Rodrigo Dias, 45881

Rúben Santos, 49063

Semestre de Inverno, 2021/2022

# Introdução

O objetivo deste projeto consiste na implementação de um sistema de ***Chat*** com múltiplos servidores ligados em anel para suportar distribuição de carga nos pedidos dos múltiplos clientes. Tirar-se-á partido da ***framework*** **gRPC** (***Google’s Remote Procedure Call***) para conectar serviços e clientes através de ***Protocol Buffers***.

Assim, o contrato dos diversos serviços envolvidos será escrito na linguagem ***Protocol Buffers*** que através do compilador ***Proto Compiler*** gerará o código em **Java** contendo os **stubs** e classes de serialização.

# Análise de Requisitos

De modo a implementar o sistema descrito no capítulo precedente, será necessário definir o problema em questão, os serviços que estarão envolvidos e uma possível solução. A análise de requisitos é o primeiro passo no desenvolvimento de qualquer tipo de software, e deve apontar o programador na direção correta.

## Definição do Problema

Este projeto almeja a implementação de um sistema de Chat, com servidores organizados em anel, o que permitirá a distribuição de carga nos pedidos dos clientes. Este sistema implica a interação entre dois objetos essenciais: O **Cliente** e o **Servidor**.

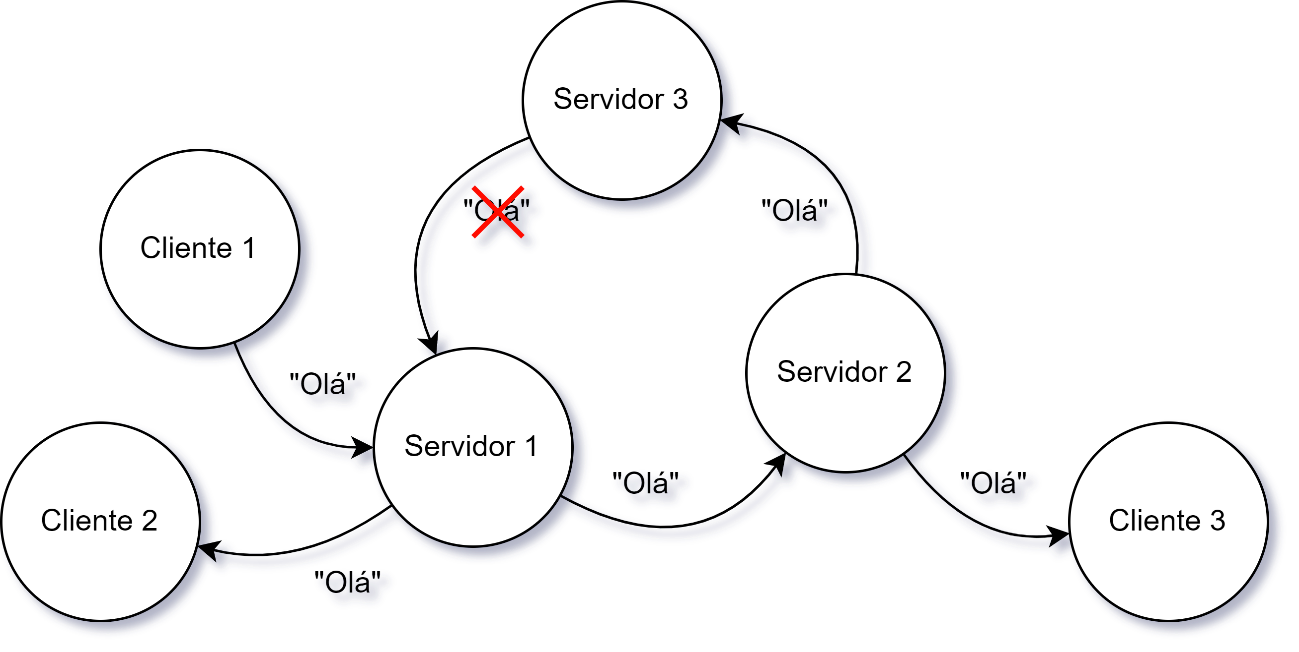
* O lado do **Cliente** deverá ser capaz de, inicialmente, pedir ao sistema o endereço ***IP*** e o porto de um dos servidores do anel. Apenas depois de obter uma resposta, deverá ser capaz de se registar no servidor com o endereço ***IP*** e porto recebido, tornando-se um dos clientes locais do servidor em questão (os servidores deverão ter acesso apenas aos clientes que se encontram em contato diretamente com eles). Finalmente, deverá também ser capaz de enviar mensagens para esse mesmo servidor.
* O lado do **Servidor** deverá ser capaz de registar os clientes e de distribuir uma mensagem de um cliente por todos os clientes locais (ou seja, os clientes registados no servidor em causa). Além disso, deverá ser capaz de passar essa mesma mensagem ao próximo servidor do anel, bem como receber mensagens do mesmo teor, provenientes do servidor precedente no anel. A mensagem deverá ter alguma identificação adicional (indicando o servidor que recebeu a mensagem em primeiro lugar), de forma a travar a viagem da mensagem no anel, como indica a cruz vermelha da **figura 1**.

Figura - Exemplo de uma mensagem inserida no anel de servidores

No exemplo da **figura 1**, o **Cliente 1** enviou a mensagem “Olá”, pelo que o **Servidor 1** distribuiu essa mesmo mensagem por todos os clientes locais, no caso, o **Cliente 2**, e passou-a ao próximo servidor do anel, o **Servidor 2**, que deverá fazer o mesmo.

## Requisitos da Solução

De modo a gerir os servidores presentes no anel, será introduzido um terceiro objeto que atuará como servidor, mas terá um papel especial. Será denominado por **Gestor do Anel**. Este Gestor do Anel terá de ser capaz de registar cada servidor, devolvendo-lhes o próximo servidor do anel. Nesta implementação, o número de servidores presentes no anel é previamente fornecido ao Gestor do Anel, de modo a saber com antecedência quantos servidores irão ingressar no anel. Além disso, o Gestor do Anel deverá ser capaz de fornecer aos clientes o endereço ***IP*** e porto de um dos servidores registados no anel.

Assim, as interações entre as diferentes entidades do sistema classificam-se da seguinte maneira:

* **Do Cliente para o Servidor:**
  + Registo, fornecendo um nome de utilizador;
  + Envio de mensagens, fornecendo uma mensagem composta pelo nome do utilizador que enviou a mensagem e o texto da mensagem.
* **Do Cliente para o Gestor do Anel:**
  + Pedido de Servidor, recebendo o endereço IP e o porto do servidor atribuído.
* **Do Servidor para o Servidor:**
  + Partilha de mensagens, fornecendo uma mensagem composta pelo nome do utilizador que enviou a mensagem, o texto da mensagem e o endereço **IP** e porto do servidor de origem (o servidor que recebeu a mensagem diretamente do cliente).
* **Do Servidor para o Gestor do Anel:**
  + Ingresso no anel, fornecendo o próprio endereço **IP** e porto, recebendo o endereço **IP** e porto do próximo servidor no anel (se o anel for composto por um único servidor, será ele mesmo).

Uma imagem com texto, eletrónica

Descrição gerada automaticamente Analisando novamente o exemplo da **figura 1**, em que o **Cliente 1** envia uma mensagem para o **Servidor 1**, e aplicando a solução discutida neste capítulo, obtém-se o diagrama da **figura 2**.

Figura - Solução do sistema

A **figura** **2** analisa o comportamento do sistema passo a passo, desde o lançamento do Servidor que manuseia o anel (**Gestor do Anel**) até ao momento em que uma mensagem enviada por um cliente está a navegar no anel.

O **Gestor do Anel** entra em execução e fica à espera que três servidores ingressem no anel (número previamente definido, a que o **Gestor do Anel** tem acesso). O **Servidor 1** entra em execução e ingressa no anel **(1)**, ficando à espera de uma resposta do **Gestor do Anel**. Quando todos os servidores já tiverem ingressado no anel, o **Gestor do Anel** retorna o próximo servidor do anel a cada servidor ingressado **(2)**. Depois, o **Servidor 1** fica à escuta de pedidos de clientes.

Admitindo a sucedida configuração do anel e que o **Servidor 1** já está à escuta, o **Cliente 1** entra em execução e faz um pedido ao **Gestor do Anel (3)**, recebendo o ***Endpoint*** do **Servidor 1** como resposta **(4)**. Se o **Cliente 2** fizer o mesmo posteriormente, receberá o ***Endpoint*** do **Servidor 2**, e assim sucessivamente (***Round* *Robin***). Depois, o **Cliente 1** deverá registar-se no **Servidor** **1** **(5)**.

Esporadicamente, o **Cliente** **1** irá enviar mensagens para o **Servidor** **1** **(6)**, que deverão ser distribuídas por todos os restantes clientes. A mensagem não deverá ser enviada apenas para o **Cliente** **4** (que está também associado com o **Servidor** **1**), mas também para o **Cliente** **2** e **Cliente** **3** (que estão associados com outros servidores no anel).

Primeiramente, o **Servidor** **1** irá distribuir a mensagem pelos clientes locais **(7)**, neste caso, o **Cliente** **4**, partilhando também com o próximo servidor no anel **(8)**, previamente atribuído pelo **Gestor** **do** **Anel**. Por sua vez, o **Servidor** **2** irá distribuir a mensagem pelos clientes locais **(9)**, neste caso, o **Cliente** **2**, e partilhando também com o **Servidor** **3**. Assim sucessivamente. Quando a mensagem chegar novamente ao **Servidor** **1**, deverá ser ignorada.

# Implementação

**Nota prévia:**

No código utilizar-se-ão os termos ingleses para designar as entidades do sistema, isto é, ***Client*** (cliente), ***Server*** (servidor) e ***RingManager*** (gestor do anel).

## Contratos de Serviços

O primeiro aspeto a implementar serão os contratos dos serviços. Existirão quatro contratos, como já foi parcialmente abordado no capítulo precedente. Os contratos serão escritos em linguagem ***Protocol Buffer***.

Uma imagem com texto

Descrição gerada automaticamenteO primeiro contrato abrange o serviço disponibilizado pelo servidor ao cliente:

Uma imagem com texto

Descrição gerada automaticamenteO segundo contrato abrange os serviços disponibilizados pelo gestor do anel ao cliente:

Uma imagem com texto

Descrição gerada automaticamenteO terceiro contrato abrange os serviços disponibilizados pelo servidor ao servidor precedente:

Uma imagem com texto

Descrição gerada automaticamenteO terceiro contrato abrange os serviços disponibilizados pelo gestor do anel ao servidor:

## Implementação em Java

Com os contratos implementados e a arquitetura pensada, resta implementar o código em Java. Para programar as funções definidas nos contratos, ter-se-á de hereditar das classes Java geradas pelo **Protocol Buffer**.

### Cliente

No caso da aplicação **ChatClient** que executará do lado do cliente, não há necessidade de hereditar nada, visto que não disponibilizará serviço nenhum às outras entidades do sistema.

Contudo, terá de abrir um Stub de forma a utilizar os serviços disponibilizados pelo RingManager, mais precisamente o serviço **ServerProvider**. Após obter uma respota, poderá fechar esse canal de comunicação, visto que não precisará mais dele. Além disso, será também necessário um **Stub** para o **ChatServer** com o ***endpoint*** retornado pelo **RingManager**. Com recurso a esse **Stub**, o cliente tirará partido dos serviços disponibilizados pelo ChatServer, mais concretamente o serviço de Chat, com as funções **register()** e **sendMessage()**.

### Servidor

No caso do **ChatServer**, implementaram-se três classes. **ChatServer**, que se encarrega da função **main()**, **ChatServerClient** que extende o contrato entre cliente e servidor e **ChatServerServer** que extende o cotnrato entre servidores.

Na classe **ChatServer**, serão abertos dosi stubs. Um será para o **RingManager**, de forma a que o servidor possa tirar partido do serviço **Enroll** para ingressar no sistema, recebendo o ***endpoint*** do próximo servidor do anel. Após a receção (visto que fica bloqueado até todos os servidores espectáveis ingressarem no anel), deverá fechar o canal de comunicação com o **RingManager**. O outro será para usufruir do serviço **Share**, disponibilizado pelo próximo servidor com ***endpoint*** retornado pelo **RingManager**.

A classe **ChatServerClient** responsabilizar-se-á de receber registos e mensagens dos clientes. Ao receber uma mensagem, deverá partilhá-la com o próximo servidor do anel, recorrendo ao **Stub** aberto em **ChatServer**.

A class **ChatServerServer** responsabilizar-se-á de receber mensagens partilhadas pelo servidor precedente no anel. Deve verificar se o servidor de origem associado á mensagem corresponde a si mesmo. Se for o caso, deverá ignorar a mensagem, visto que já deu a volta toda ao anel.

### Gestor do Anel

No caso do **ChatRingManager**, implementaram-se também três classes. **ChatRingManager**, com o intuito de executar a função **main()**, **ChatRingManagerClient** que extende o contrato entre o gestor do anel e o cliente e **ChatRingManagerServer** que extende o contrato entre o gestor de anel e o servidor.

O **ChatRingManager** não abrirá quaisquer **Stubs**, visto que não tirará partido de quaisquer serviços disponibilizados pelas outras entidades no sistema.

A classe **ChatRingManagerClient** responsabilizar-se-á de receber pedidos de servidores por parte dos clientes. Ao receber um pedido, deve retornar o próximo servidor do anel. Deve recorrer a uma distribuição **Round** **Robin**.

A classe **ChatRingManagerServer** almeja permitir que os servidores ingressem no anel. O **RingManager** deverá saber com antecência o número de servidores espectados, e, quando todos ingressarem, retorna o ***‘next’*** respetivo.

## Diagrama de Stubs

# Teste

Para testar o sistema

# Conclusão

A distribuição de carga recorrendo a um anel de servidores é um sistema muito eficaz, que abre portas para a concretização de um sistema ainda mais elaborado. Por exemplo, quando o RingManager retorna o ‘next’ a todos os servidores que ingressaram no anel, passa a ficar sem objetivo, podendo ser utilizado para prestar outros serviços.