Computação Distribuída com gRPC

Prof Alcides/Prof Mario

Laboratório

Objetivo da atividade:

O objetivo é projetar, implementar e testar um microserviço simples usando o framework gRPC. O foco principal será na definição de um contrato de serviço, na implementação da lógica do servidor, do cliente e na demonstração da interoperabilidade entre duas linguagens de programação diferentes (Python e C++).

Descrição do problema:

Você deverá construir um serviço de encurtamento de URL. Este serviço deve expor uma API que permite aos clientes realizarem duas operações principais:

1. Submeter uma URL longa e receber em troca uma URL curta e única.
2. Submeter um código curto e receber de volta a URL longa original associada a ele.

Requisitos técnicos:

1. **Framework:** A comunicação entre cliente e servidor deve ser feita exclusivamente via gRPC.
2. **Interoperabilidade:** O cliente e o servidor devem ser implementados em linguagens diferentes. A escolha é sua:
    - Servidor em Python e Cliente em C++.

      OU

    - Servidor em C++ e Cliente em Python.
3. **Armazenamento de dados:** Para simplificar, o serviço não usará um banco de dados. Todas as associações entre URLs curtas e longas devem ser armazenadas em uma estrutura de dados em memória no servidor (ex: um dicionário ou mapa de hash).
    - **Atenção:** Isso significa que todos os dados serão perdidos quando o processo do servidor for encerrado. Isso é esperado e aceitável para esta atividade.

Roteiro

Passo 1: Definir o contrato (.proto)

- Crie um arquivo chamado urls.proto.
- Dentro deste arquivo, defina o serviço EncurtadorURL.
- Este serviço deve conter dois métodos RPC:
    1. EncurtarURL: Deve aceitar uma mensagem contendo a URL longa e retornar uma mensagem com a URL curta.
    2. ObterURLLonga: Deve aceitar uma mensagem contendo o código curto e retornar uma mensagem com a URL longa original.
- Defina as mensagens de requisição e resposta necessárias para esses RPCs. Pense em nomes claros como RequisicaoEncurtar, RespostaEncurtar, etc.

Passo 2: Gerar o código base (Stubs e Skeletons)

- Use o compilador do Protocol Buffers (protoc) com os plugins gRPC para gerar o código base para ambas as linguagens.
    - **Para Python:** python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. urls.proto
    - **Para C++:** (geralmente automatizado via CMake) O comando base é protoc --grpc_out=. --cpp_out=. -I. urls.proto.

Passo 3: Implementar o Servidor

- Escolha uma das linguagens (Python ou C++) para o seu servidor.
- Crie o arquivo do servidor (ex: servidor.py ou servidor.cpp).
- Implemente a classe de serviço que herda do "skeleton" gerado no Passo 2.
- Dentro da classe, implemente a lógica para os dois métodos RPC:
    - **EncurtarURL:**
        1. Receba a URL longa da requisição.
        2. Gere um código curto e único (veja dicas abaixo).
        3. Armazene a associação codigo_curto -> url_longa no seu dicionário/mapa em memória.
        4. Retorne a URL curta completa (ex: http://localhost:50051/CODIGO_GERADO).
    - **ObterURLLonga:**
        1. Receba o código curto da requisição.
        2. Busque por este código no seu dicionário/mapa.
        3. Se encontrar, retorne a URL longa. Caso contrário, retorne um erro ou uma string vazia.

- Adicione o código para iniciar o servidor gRPC e fazê-lo escutar em uma porta (ex: 50051).

Passo 4: Implementar o Cliente

- Use a outra linguagem para implementar o cliente.
- Crie o arquivo do cliente (ex: cliente.cpp ou cliente.py).
- No seu código, crie um canal para se conectar ao endereço e porta do servidor.
- Crie um "stub" gRPC usando o canal.
- Use o stub para chamar os dois métodos RPC:
    1. Chame EncurtarURL com uma URL longa de exemplo e imprima o resultado.
    2. Use o código curto recebido na resposta anterior para chamar ObterURLLonga e imprima a URL original para verificar se o sistema funciona.

Passo 5: Testar a Interoperabilidade

- Abra dois terminais.
- Em um terminal, execute o seu servidor.
- No outro terminal, execute o seu cliente.
- Verifique se a comunicação ocorre sem erros e se os resultados impressos pelo cliente estão corretos.

Dicas para a lógica de encurtamento: Para gerar o código curto, você pode usar uma abordagem simples:

- **Python:** Use a biblioteca hashlib para gerar um hash da URL longa somada ao tempo atual e pegue os primeiros 7 caracteres.
- **C++:** Gere uma string aleatória de 7 caracteres usando as bibliotecas <random> e <chrono>.

Rubricas de Avaliação (0 a 10 pontos)

Critério	Pontuação	Descrição da Avaliação

1. Definição do Contrato (.proto)	2.0	O arquivo .proto está sintaticamente correto, define claramente o serviço, os dois RPCs e todas as mensagens necessárias.
2. Implementação do Servidor	3.0	O servidor inicia sem erros, implementa a lógica dos dois RPCs, armazena os dados corretamente em memória e gera os códigos curtos de forma funcional.
3. Implementação do Cliente	2.0	O cliente conecta-se ao servidor, chama com sucesso os dois métodos RPC, envia os dados corretamente e processa as respostas recebidas.
4. Interoperabilidade	2.0	O sistema funciona de ponta a ponta com o cliente e o servidor rodando em linguagens diferentes, provando a interoperabilidade.
5. Qualidade do Código	1.0	O código está limpo, bem organizado, comentado onde necessário e segue as boas práticas da linguagem escolhida.