Atividade Extraclasse 01

Programação para Sistemas Paralelos e Distribuídos

Turma 01

Engenharia de Software - FGA

Integrantes:
Felipe Aguiar Hansen - 222032810
Felipe Direito Corrieri de Macedo- 190086971

Introdução

Este relatório apresenta o desenvolvimento de um sistema distribuído utilizando o framework gRPC como parte da Atividade Extraclasse 01 da disciplina de Programação de Sistemas Paralelos e Distribuídos. O trabalho consiste na implementação de uma biblioteca distribuída, demonstrando diferentes tipos de comunicação entre componentes distribuídos através do gRPC.

O relatório está organizado da seguinte forma: primeiramente, apresentamos uma visão geral sobre o framework gRPC, seus componentes e tipos de comunicação; em seguida, detalhamos a implementação da aplicação de biblioteca distribuída, incluindo sua arquitetura e funcionalidades; posteriormente, abordamos conceitos de virtualização com KVM, QEMU e a API Libvirt; por fim, apresentamos as conclusões e aprendizados obtidos com o desenvolvimento deste trabalho.

Framework gRPC

O gRPC (gRPC Remote Procedure Calls) é um framework de código aberto desenvolvido pelo Google para comunicação entre serviços distribuídos. Lançado em 2015, o gRPC foi projetado para ser eficiente, rápido e independente de linguagem, permitindo a comunicação entre serviços escritos em diferentes linguagens de programação. O framework é amplamente utilizado em ambientes de microsserviços, sistemas distribuídos e aplicações que requerem comunicação de baixa latência e alta performance.

Neste projeto, implementamos exemplos dos quatro tipos de comunicação suportados pelo gRPC, demonstrando sua versatilidade e capacidade de atender diferentes requisitos de comunicação entre serviços. Cada tipo de comunicação foi implementado com um cliente e servidor correspondentes, permitindo a visualização prática de como o gRPC funciona em diferentes cenários.

Componentes Principais do gRPC

Protocol Buffers (protobuf)

Protocol Buffers é a linguagem de definição de interface (IDL) e o formato de serialização de dados utilizado pelo gRPC. Desenvolvido pelo Google, o Protocol Buffers oferece vantagens em relação a outros formatos de serialização como JSON e XML. Este arquivo é a base para a comunicação entre cliente e servidor, estabelecendo um contrato claro entre as partes. O arquivo que ser

No projeto, definimos o protobuf como o arquivo `*service.proto*` e definimos nele:
- Um serviço `*CommunicationDemo*` com quatro métodos RPC
- Estruturas de mensagens para requisições e respostas
- Anotações para indicar streaming de cliente, servidor ou bidirecional

HTTP/2 como Protocolo de Transporte

O gRPC utiliza HTTP/2 como seu protocolo de transporte, o que representa uma evolução significativa em relação ao HTTP/1.1. As principais características do HTTP/2 que beneficiam o gRPC são:

1. **Multiplexação**: Permite múltiplas requisições e respostas simultâneas em uma única conexão TCP, eliminando o problema de "head-of-line blocking" presente no HTTP/1.1.

2. **Compressão de cabeçalhos**: Reduz o overhead de rede ao comprimir cabeçalhos HTTP, especialmente importante em ambientes com muitas requisições.

3. **Streaming bidirecional**: Permite que cliente e servidor enviem dados simultaneamente, possibilitando comunicação em tempo real.

4. **Priorização de fluxos**: Permite que recursos críticos sejam entregues primeiro, otimizando a experiência do usuário.

5. **Server push**: Permite que o servidor envie recursos proativamente ao cliente, reduzindo a latência.

O HTTP/2 é fundamental para a implementação dos diferentes padrões de comunicação do gRPC, especialmente os modos de streaming que seriam difíceis ou ineficientes de implementar com HTTP/1.1.

Tipos de Comunicação no gRPC

O gRPC suporta quatro padrões de comunicação, cada um adequado para diferentes cenários de uso:

Unary RPC (Comunicação Unária)
Na comunicação unária, o cliente envia uma única requisição e recebe uma única resposta do servidor. Este é o padrão mais simples e se assemelha a uma chamada de função tradicional.

- Implementação: No nosso projeto, o método `UnaryCall` demonstra este padrão, onde o cliente envia uma mensagem simples e o servidor responde com uma única mensagem.

Server Streaming RPC (Streaming de Servidor)
Neste padrão, o cliente envia uma única requisição e o servidor responde com um fluxo de mensagens. O cliente processa cada mensagem à medida que ela chega.

- Implementação: O método `ServerStreamingCall` em nosso projeto demonstra este padrão, onde o servidor envia múltiplas respostas para uma única requisição do cliente.

Client Streaming RPC (Streaming de Cliente)
No streaming de cliente, o cliente envia um fluxo de mensagens ao servidor e recebe uma única resposta após o servidor processar todas as mensagens.

- Implementação: O método `ClientStreamingCall` demonstra este padrão, onde o cliente envia múltiplas mensagens e o servidor responde com uma única mensagem resumindo o processamento.

Bidirectional Streaming RPC (Streaming Bidirecional)
Este é o padrão mais flexível, onde tanto o cliente quanto o servidor podem enviar fluxos de mensagens independentes. Os fluxos operam de forma assíncrona, permitindo comunicação em tempo real.

- Implementação: O método `BidirectionalStreamingCall` demonstra este padrão, onde cliente e servidor trocam mensagens de forma independente.

gRPC vs. Outras Alternativas para Aplicações Distribuídas

O gRPC se destaca em cenários onde a performance, a tipagem forte e o suporte a streaming são requisitos importantes, especialmente em arquiteturas de microserviços e sistemas distribuídos internos. No entanto, para APIs públicas ou casos onde a interoperabilidade com navegadores é crucial, REST ou GraphQL podem ser alternativas mais adequadas.

Característica	**gRPC**	**REST API**
Protocolo	HTTP/2	HTTP/1.1 (principalmente)
Formato de dados	Protocol Buffers (binário)	JSON/XML (texto)
Contrato de API	Estrito (arquivo .proto)	Flexível (OpenAPI/Swagger)
Geração de código	Automática em várias linguagens	Geralmente manual ou com ferramentas adicionais
Streaming	Suporte nativo	Limitado (WebSockets como alternativa)
Performance	Alta (serialização binária eficiente)	Média (overhead de texto JSON/XML)
Navegadores	Suporte limitado (gRPC-Web)	Suporte nativo
Curva de aprendizado	Moderada a alta	Baixa a moderada

Característica	**gRPC**	**SOAP**
Protocolo	HTTP/2	HTTP, SMTP, etc.
Formato de dados	Protocol Buffers (binário)	XML (texto)
Tamanho das mensagens	Compacto	Verboso
Performance	Alta	Baixa
Complexidade	Moderada	Alta
Recursos enterprise	Em desenvolvimento	Maduros (WS-*)
Compatibilidade	Multiplataforma	Multiplataforma

3. Aplicação Desenvolvida: Biblioteca Distribuída

A aplicação desenvolvida consiste em um sistema distribuído de gerenciamento de biblioteca virtual, que permite o cadastro e gerenciamento de usuários e livros. O sistema implementa os seguintes componentes:

Módulo P (Python): Web Server + gRPC Stub
- Implementa uma API REST para clientes web
- Atua como stub gRPC para se comunicar com os servidores A e B
- Fornece uma interface web para interação com o sistema

Módulo A (Go): gRPC Server para gerenciamento de livros
- Implementa serviços para gerenciar o catálogo de livros
- Oferece operações CRUD para livros
- Implementa streaming de servidor para busca por categoria

Módulo B (Node.js): gRPC Server para gerenciamento de usuários e empréstimos
- Implementa serviços para gerenciar usuários
- Implementa serviços para gerenciar empréstimos de livros
- Oferece streaming bidirecional para atualizações de status

A arquitetura distribuída permite a separação de responsabilidades entre os diferentes componentes, facilitando a manutenção e escalabilidade do sistema. Para garantir a portabilidade e isolamento dos serviços, utilizamos a tecnologia Docker para *conteinerização* dos sistemas distribuídos. Cada módulo é executado em seu próprio contêiner Docker, com suas dependências específicas isoladas, e a comunicação entre eles é orquestrada através do Docker Compose. Esta abordagem de *conteinerização* oferece diversas vantagens.

Aprendizados Individuais

Felipe Direito C. Macedo (190086971)
O desenvolvimento deste projeto me permitiu aprofundar meus conhecimentos em comunicação entre sistemas distribuídos, especialmente utilizando gRPC. A implementação de diferentes padrões de streaming foi particularmente desafiadora e enriquecedora. Auto-avaliação: 9/10.

Felipe Aguiar Hansen (222032810)
O projeto proporcionou um aprendizado sobre os quatro tipos de comunicação em gRPC: Unary (requisição única, resposta única), Server Streaming (requisição única, múltiplas respostas), Client Streaming (múltiplas requisições, resposta única) e Bidirectional Streaming (múltiplas requisições, múltiplas respostas). A definição do Protocol Buffers (.proto) mostrou a necessidade de estabelecer o contrato de comunicação entre cliente e servidor, especificando serviços, métodos e estruturas de dados de forma independente de linguagem. A estrutura geral das comunicações no projeto seguiu um padrão consistente: definição dos serviços no protobuf, geração de código para cliente e servidor, implementação da lógica do servidor para processar as requisições e implementação do cliente para invocar os métodos remotos. Auto-avaliação 9/10

Conclusão

O gRPC representa uma evolução significativa na comunicação entre serviços distribuídos, combinando a eficiência do Protocol Buffers com as capacidades avançadas do HTTP/2. Seus quatro padrões de comunicação oferecem flexibilidade para atender diversos requisitos, desde simples chamadas de função até comunicação bidirecional em tempo real.

Embora não seja a solução ideal para todos os cenários (especialmente para comunicação direta com navegadores), o gRPC se destaca em ambientes de microsserviços, sistemas distribuídos de alto desempenho e aplicações que necessitam de comunicação eficiente entre serviços.

A escolha entre gRPC e outras alternativas deve considerar fatores como requisitos de desempenho, necessidade de streaming, suporte a navegadores, familiaridade da equipe e o ecossistema tecnológico existente. Em muitos casos, uma abordagem híbrida pode ser ideal, utilizando gRPC para comunicação interna entre serviços e REST ou GraphQL para APIs públicas.

O projeto implementado demonstra na prática como o gRPC pode ser utilizado para diferentes padrões de comunicação, fornecendo uma base sólida para explorar e compreender este poderoso framework.