Universidade de São Paulo

Remote Procedure Calls (RPC)

Docente: Prof. Dr. Daniel de Angelis Cordeiro

Luan Zanetich Acquafreda
12717120

Sumário

1. Introdução

1.1 Objetivo

Inicialmente, o objetivo principal do trabalho é colocar em prática e avaliar os resultados obtidos em torno dos métodos RPC (Remote Procedure Calls). O foco principal é medir a partir de uma conexão cliente servidor local e remota o desempenho de diferentes ferramentas RPC, assim como mensurar os desempenhos na execução de uma sequência de operações específicas como caso de testes. Por conseguinte, uma análise póstuma dos dados coletados e a apresentação dos mesmos neste documento.

1.2 RPC

O Remote Procedure Call (RPC) é um protocolo de comunicação utilizado em sistemas distribuídos para permitir que um programa em um sistema remoto invoque procedimentos em um sistema local de forma transparente. Em outras palavras, o RPC permite que um programa chame funções ou métodos localizados em outros sistemas como se estivessem sendo executados na mesma máquina. Por exemplo, em uma aplicação cliente-servidor, o cliente pode usar o RPC para solicitar serviços ao servidor, como realizar consultas a um banco de dados remoto ou enviar uma requisição a um serviço web. O RPC simplifica a comunicação entre os sistemas distribuídos, tornando possível o desenvolvimento de sistemas escaláveis e modulares.

2. Procedimentos e Hardware

2.1 RPCs utilizados

2.1.1 gRPC

Primeiro utilizado, o gRPC é um framework de comunicação de alto desempenho, desenvolvido pelo Google, que permite a comunicação eficiente entre diferentes sistemas distribuídos. Essa tecnologia utiliza o Protocol Buffers, uma linguagem de serialização de dados eficiente e escalável, para definir a estrutura das mensagens e a interface dos serviços. Sucintamente, o gRPC utiliza HTTP/2 como protocolo de transporte, o que possibilita a transferência assíncrona de dados e multiplexação de várias solicitações em uma única conexão, resultando em um desempenho superior e menor consumo de recursos de rede.

Por conseguinte, a arquitetura do gRPC é baseada em contratos de serviço definidos em arquivos de IDL (Interface Description Language), que especificam os métodos disponíveis, os tipos de dados das mensagens e outros detalhes da interface. A partir desses arquivos de IDL, o gRPC gera automaticamente o código cliente e servidor em diferentes linguagens de programação, facilitando a implementação e garantindo a consistência entre os sistemas.

2.1.2 Golang RPC

Em sequência, o Golang RPC é um framework de comunicação que utiliza um sistema de serialização de dados eficiente, para facilitar a troca de informações entre sistemas distribuídos. O Golang oferece suporte a chamadas de procedimento remoto, permitindo que os clientes invoquem métodos em servidores remotos de forma transparente. Notoriamente, uma das principais características do Golang RPC é sua simplicidade, a própria biblioteca padrão do Go fornece um conjunto de ferramentas para criar um servidor RPC de maneira simples, além dos desenvolvedores poderem definir interfaces que descrevem as funções ou métodos disponíveis no servidor, tornando a comunicação mais estruturada e fácil de usar.

Por fim, o Golang utiliza a serialização de dados em formato binário para minimizar o tamanho dos pacotes de rede, assim como usa sockets TCP/IP para o transporte dos dados, garantindo uma transferência rápida e confiável das informações entre os programas.

2.2 Operações implementadas

O projeto foi baseado na análise de 5 tipos de mensagens que foram formuladas e enviadas, sendo elas:

1. operação sem argumentos e sem valor de retorno (void);
2. operação com um argumento long e valor de retorno long;
3. operação com oito argumentos long e valor de retorno long;
4. operação com um argumento String e valor de retorno String, para strings com diferentes tamanhos (2, 4, 8, 16, 32, 64, 128, 256, 512, 1024);
5. operação mais complexa, que envia uma classe *Movie* com os atributos de um filme (título, gênero, diretor, atores, duração e um boolean confirmando se foi enviado com sucesso) e recebe de volta o filme, com o boolean atualizado, como demonstra na estrutura a seguir:

```
49  message Movie {
50    int64 id = 1;
51    string title = 2;
52    string genre = 3;
53    string director = 4;
54    repeated Actor actors = 5;
55    int32 duration = 6;
56    bool foiEnviado = 7;
57  }
58
59  message Actor {
60    int64 id = 1;
61    string name = 2;
62    string surname = 3;
63    repeated string character_name = 4;
64  }
```

Figura 1: imagem da estrutura da classe *Movie*

2.3 Bibliotecas e linguagens de programação

A linguagem de programação utilizada na construção do modelo gRPC foi Python, com as estruturas proto definidas e compiladas para a linguagem. Já para o Golang, foi utilizada a linguagem de programação go, que fornece toda estrutura necessária. Para ambos projetos foi utilizada a IDE Visual Studio Code. A seguir estão as bibliotecas que foram utilizadas nos projetos:

gRPC:

- Java JDK 1.8 (apenas para classes do Protobuf, geradas por outras bibliotecas)
- Python3 versão 3.11.3
- Plugin Gradle - Google Protobuf versão 0.8.6
- Google API GRPC Common Protos versão 0.1.9
- GRPC-Netty (Servidor) versão 1.5.0
- GRPC Protobuf versão 1.5.0
- GRPC Stub versão 1.5.0
- Protoc (Compilador de Arquivos Proto) versão 3.3.0

Golang:

- go versão 1.20

2.4 Hardware e Software

Para os experimentos utilizados, assim como a implementação de todos os códigos, foram utilizados duas máquinas distintas, sendo elas e suas características:

Notebook Acer Nitro 5:

CPU:

- Intel Intel Core i5 - 7300 7ª geração – Quad Core

Memória:

- 2x8 DDR4 2666 MHz (16GB RAM)
- 256GB SSD M2

Sistema Operacional:
- Windows 10 - Versão 22H2

Lenovo ideapad 3:
CPU:
- AMD Ryzen 5 5500U with Radeon Graphics × 6

Memória:
- 8GB RAM DDR4 2666 MHz
- 256GB SSD M2

Sistema Operacional:
- Linux Mint 21.1 Cinnamon

Por fim, vale ressaltar que para os casos de testes, foi utilizada uma conexão sem fio de 300 Mb utilizando a tecnologia de fibra óptica, a uma distância intermediária do roteador.

3. Resultados e observações

3.1 Procedimentos

Para análise dos dados obtidos, foram coletados 10 casos de testes na execução de cada operação listada no tópico 2.2 com o descarte dos casos de teste que retornaram em um tempo excessivo, ou fora do convencional. Com os tempos registrados, foi calculada a média, assim como o desvio padrão para o resultado de cada uma das operaçẽs apresentadas na plataforma do google planilhas.

3.2 Resultados

3.2.1 gRPC

Esses foram os dados coletados nos casos de testes utilizando o gRPC:

Tabela 1: dados gRPC cliente e servidor mesma máquina (Acer Nitro 5)

operações	Média (ms)	Desvio padrão
void	7,688	1,062
long	7,416	0,5
eightlongs	7,7	0,946
string (2)	7,247	0,546
string (4)	7,1	0,563
string (8)	7,302	0,482
string (16)	7,41	1,089
string (32)	7,301	0,486
string (64)	7,5	0,706
string (128)	7,403	0,513
string (256)	7,094	0,559
string (512)	7,47	0,73
string (1024)	7,002	0,662
movie	7,296	0,493

Tabela 2: dados gRPC cliente e servidor em máquinas diferentes (servidor - ideapad 3, cliente - Acer Nitro 5)

operações	Média (ms)	Desvio padrão
void	198,231	162,646
long	214,251	196,25
eightlongs	159,108	149,632
string (2)	290,413	147,169
string (4)	356,237	249,538
string (8)	336,098	187,917
string (16)	222,444	195,852

operações	Média (ms)	Desvio padrão
string (32)	360,389	194,991
string (64)	271,228	315,662
string (128)	282,16	328,18
string (256)	198,02	152,593
string (512)	266,925	251,565
string (1024)	246,278	225,535
movie	192,399	184,136

3.2.2 Golang RPC

Esses foram os casos de teste utilizando o Golang RPC:

Tabela 3: dados Golang cliente e servidor mesma máquina (Acer Nitro 5)

operações	Média (ms)	Desvio padrão
void	0,774	0,270
long	0,735	0,219
eightlongs	0,890	0,292
string (2)	0,754	0,259
string (4)	0,966	0,224
string (8)	0,816	0,285
string (16)	0,693	0,264
string (32)	0,706	0,194
string (64)	0,782	0,247
string (128)	0,810	0,309
string (256)	0,708	0,215
string (512)	0,774	0,245
string (1024)	0,730	0,263
movie	1,074	0,431

Tabela 4: dados Golang cliente e servidor em máquinas diferentes (servidor - ideapad 3, cliente - Acer Nitro 5)

operações	Média (ms)	Desvio padrão
void	8,662	1,213
long	10,139	4,231
eightlongs	11,228	6,824
string (2)	0,706	0,223
string (4)	0,659	0,203
string (8)	0,679	0,200
string (16)	0,700	0,196
string (32)	0,722	0,250
string (64)	0,640	0,158
string (128)	0,705	0,188
string (256)	0,757	0,247
string (512)	0,729	0,243
string (1024)	0,676	0,206
movie	11,285	6,333

3.3 Análise dos resultados

3.3.1 Mesma máquina

A partir dos dados coletados na mesma máquina (acer nitro 5), podemos observar que os resultados dos testes executados mantiveram um padrão de tempo para todos os casos de testes impostos, indiferente da complexidade das operações. No primeiro caso, utilizando o gRPC a média ficou girando em torno de 7 milissegundos, com um desvio padrão não passando de 1,1 ms. Já utilizando o Golang, podemos observar uma drástica diferença nas médias, que giraram em torno de 0,7 ms - cerca de 10% do valor registrado utilizando o gRPC, com os valores do desvio padrão não passando de 0,5. Um fato curioso nos casos de testes implementados na mesma máquina utilizando gRPC foi a redução pela metade do

tempo de execução e retorno quando o ip local foi fornecido diretamente, ao invés de *localhost:50051*.

No geral, podemos observar a diferença de desempenho em uma mesma máquina com o gráfico a seguir, que representa as médias de tempos em cada operação:

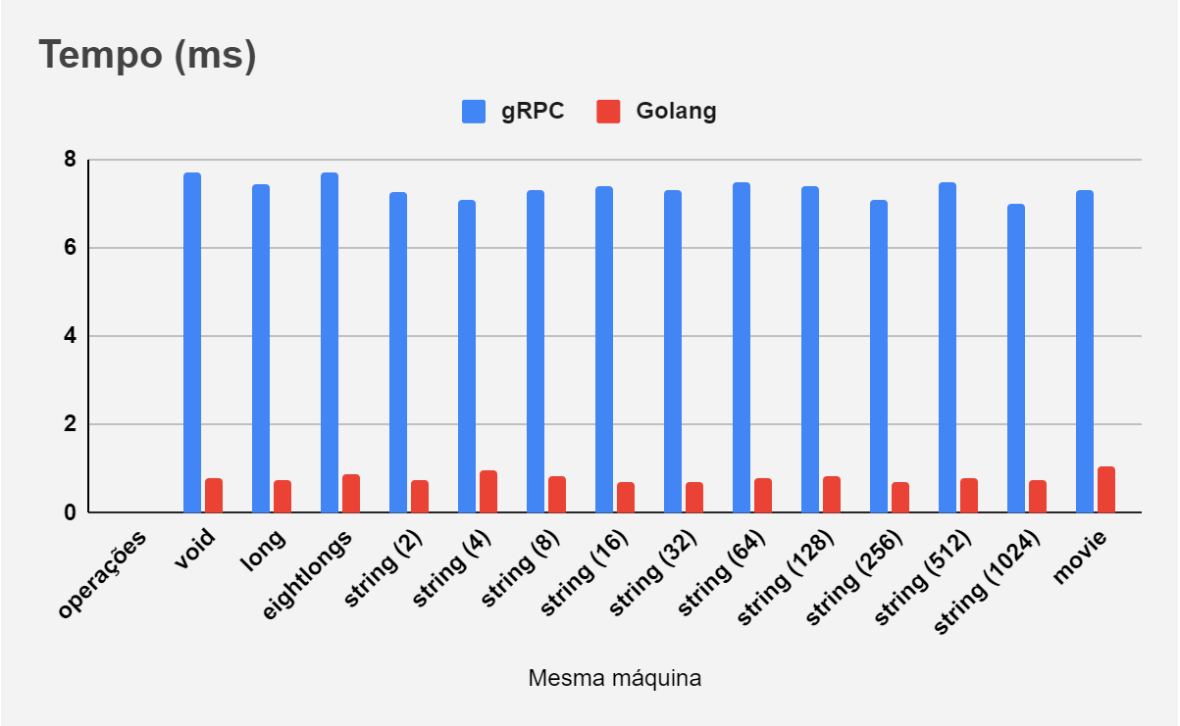

Figura 2: Gráfico de desempenho mesma máquina

Com as médias comparadas, podemos analisar também o resultado do desvio padrão dos testes. Ao impor lado a lado, podemos observar um maior desvio padrão nos testes executados pelo gRPC que, mesmo de uma grandeza muito baixa, apresenta uma maior variação quando comparado com a consistência do Golang, como podemos também observar no gráfico a seguir:

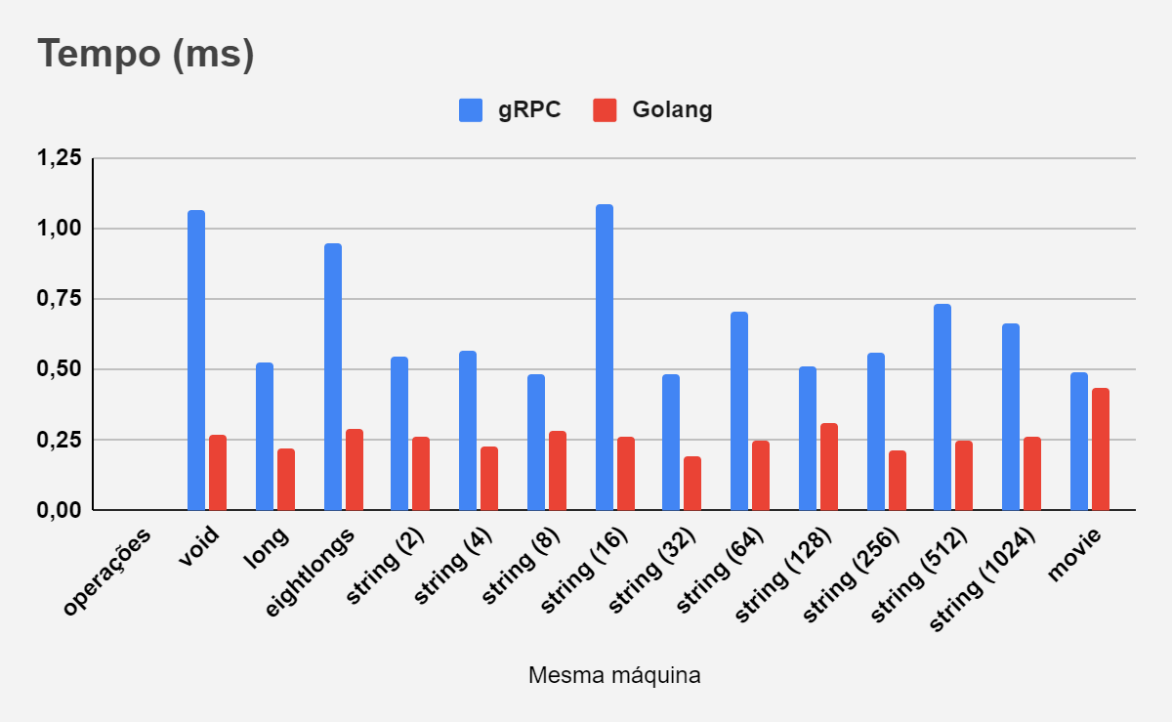

Figura 2: Gráfico de desvio padrão mesma máquina

3.3.2 Máquinas diferentes

Os testes executados em diferentes máquinas relataram uma bruta diferença entre os tempos de execução com cada método. Enquanto o gRPC houve um aumento de 2600% no tempo em alguns casos o Golang apresentou um aumento de 1100%, demonstrando uma disparidade dos dois métodos remotos não somente na mesma máquina, mas também com chamadas em máquinas diferentes, como podemos analisar o gráfico do desempenho de ambos a seguir e sua diferença de dados, com o Golang quase não aparecendo nos resultados:

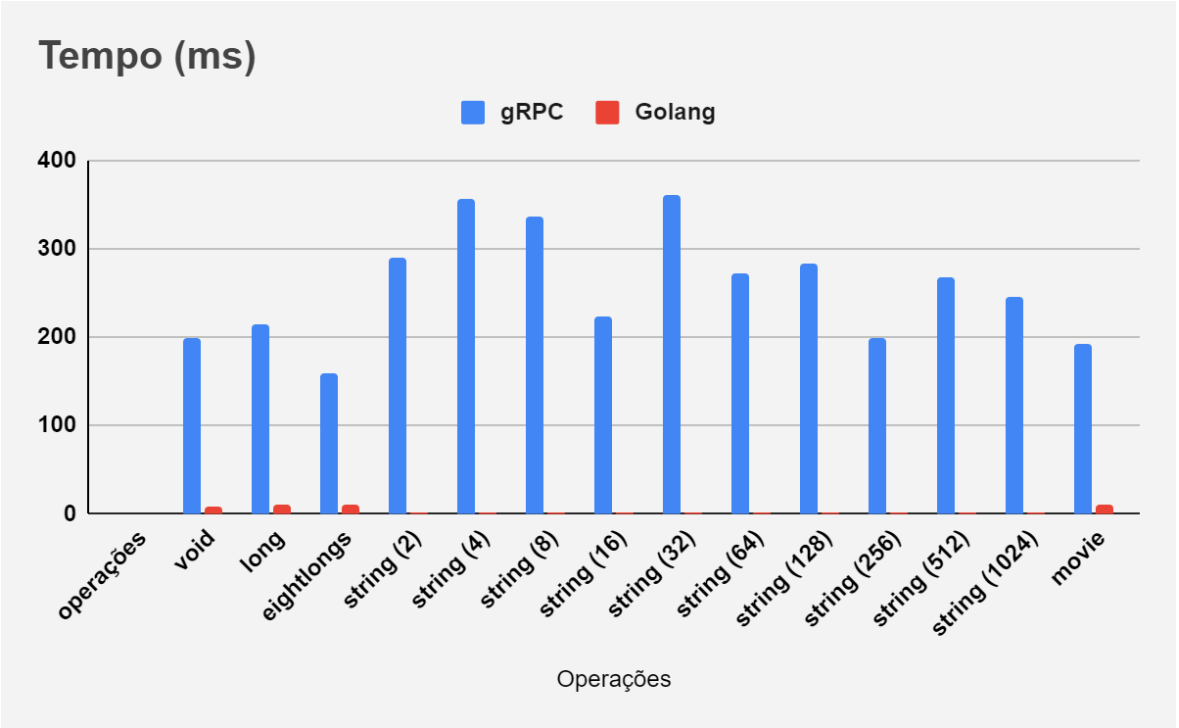

Figura 3: Gráfico de desempenho diferentes máquinas

Primeiramente, a comparação de desempenho quando os testes foram executados em diferentes máquinas foi possível observar alguns fatos curiosos. As operações envolvendo strings representaram a maior instabilidade nos casos de teste, tendo as 10 maiores médias de tempo utilizando o gRPC que, em contrapartida, dominam as 10 menores médias utilizando o Golang - surpreendentemente as strings mantiveram o mesmo tempo de resposta tanto na mesma máquina, quanto em máquinas diferentes. Vale ressaltar que foram testadas operações de string com 1000000 de caracteres para comparação, retornando um tempo de 19ms no Golang.

Em um panorama geral, podemos observar uma maior dispersão dos dados ao todo no gRPC com seu desvio padrão com alto um grau elevado, não estabelecendo um padrão referente a complexidade da operação imposta, enquanto o Golang, com exceção das operações de string, apresenta uma consistência nas médias como nos casos de testes na mesma máquina mantendo um desvio padrão praticamente nivelado com o colhido nos testes na mesma máquina, contendo um atraso maior devido a conexão remota.

4. Conclusão

Com base nos resultados obtidos, concluímos que o Golang apresentou um desempenho superior em comparação com o gRPC. Durante os testes realizados, o Golang demonstrou ser mais eficiente e rápido, superando as expectativas em termos de tempo de execução e desempenho geral.

Além disso, o Golang, como uma linguagem de programação altamente otimizada e de baixo nível, ofereceu benefícios significativos em relação ao gRPC em relação à velocidade de execução e capacidade de processamento de dados. Sua simplicidade, concorrência nativa e capacidade de aproveitar ao máximo os recursos do sistema contribuíram para um desempenho excepcional.

Em suma, os experimentos realizados foram executados em situações muito semelhantes entre os dois ambientes, tentando aproximar ao máximo as mensagens que foram enviadas e as respostas do servidor. Por fim, vale ressaltar que ambos os sistemas de conexão foram simples de implementar, com uma abrangente biblioteca para poder manejar todas as operações dos casos de testes com facilidade.

5. Referências Bibliográficas

gRCP - A high performance, open-source universal RPC framework - Website disponível em: https://grpc.io/ - Acesso em: 06-Jun-2023.

Golang RPC Documentation - Website disponível em https://pkg.go.dev/net/rpc - Acesso em: 06-Jun-2023

Steen, M. V., & Tanenbaum, A. S. (2023). Distributed Systems (4th ed.).