46527-04 - Programação Distribuída - Turma 031 - 2021/2 - Prof. Sergio Johann Filho

Alunos : André Marques, Guilherme Carvalho, José Goulart

Programação Distribuída - Trabalho 1

O presente artigo tem como objetivo explicar o desenvolvimento e o funcionamento de um um programa distribuído que permite operações concorrentes em um banco de dados no qual os processos distribuídos devem realizar de modo síncrono a utilização dos recursos disponíveis.

1. Introdução

O exercício deste trabalho instrui o desenvolvimento de um programa que pode realizar três operações sobre um mesmo recurso compartilhado, sendo essas as operações insere elimina e busca.

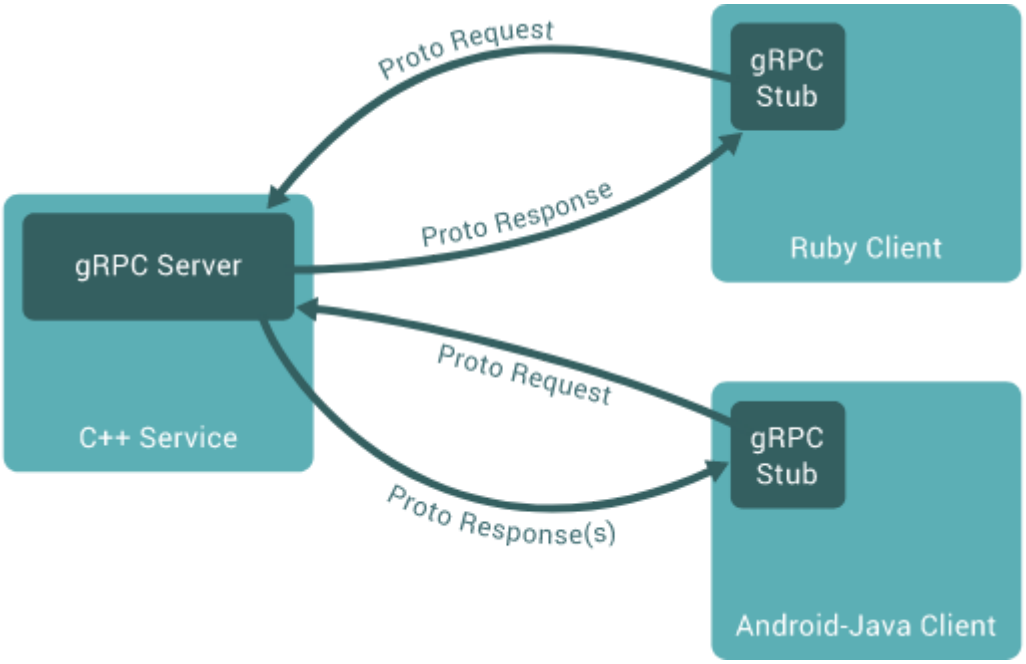
- Busca: pode ser executado de modo concorrente;
- Insere: mutuamente exclusiva, mas podendo ser executada em paralelo com a Busca;
- Elimina: mutualmente exclusiva, mas não podendo ser executada em paralelo com eliminação e busca.

O programa irá executar em um processo de servidores (servidor de semáforos) em loop atendendo requisições de clientes. As requisições dos clientes serão executadas aleatóriamente em portas diferentes e realizando operações diferentes a cada 2 segundos.

A aplicação foi desenvolvida em linguagem Java utilizando gRPC ao invés de Java RMI como visto em aula, com o objetivo de aprender sobre uma tecnologia mais recente e que já está sendo adotada no mercado. No próximo tópico será explicado a tecnologia API gRPC, seus benefícios.

2. gRPC

No gRPC, um aplicativo cliente pode chamar diretamente um método em um aplicativo de servidor em uma máquina diferente como se fosse um objeto local, facilitando a criação de aplicativos e serviços distribuídos. Como em muitos sistemas RPC, o gRPC é baseado na ideia de definir um serviço, especificando os métodos que podem ser chamados remotamente com seus parâmetros e tipos de retorno. No lado do servidor, o servidor implementa essa interface e executa um servidor gRPC para lidar com as chamadas do cliente. No lado do cliente, o cliente tem um stub (referido apenas como um cliente em alguns idiomas) que fornece os mesmos métodos que o servidor.



Os clientes e servidores gRPC podem ser executados e se comunicar entre si em uma variedade de ambientes - de servidores dentro do Google a sua própria área de trabalho - e podem ser escritos em qualquer uma das linguagens com suporte do gRPC. Portanto, por exemplo, você pode criar facilmente um servidor gRPC em Java com clientes em Go, Python ou Ruby. Além disso, as APIs do Google mais recentes terão versões gRPC de suas interfaces, permitindo que você construa facilmente a funcionalidade do Google em seus aplicativos.

2.1. Beneficios

Os principais benefícios do gRPC são:

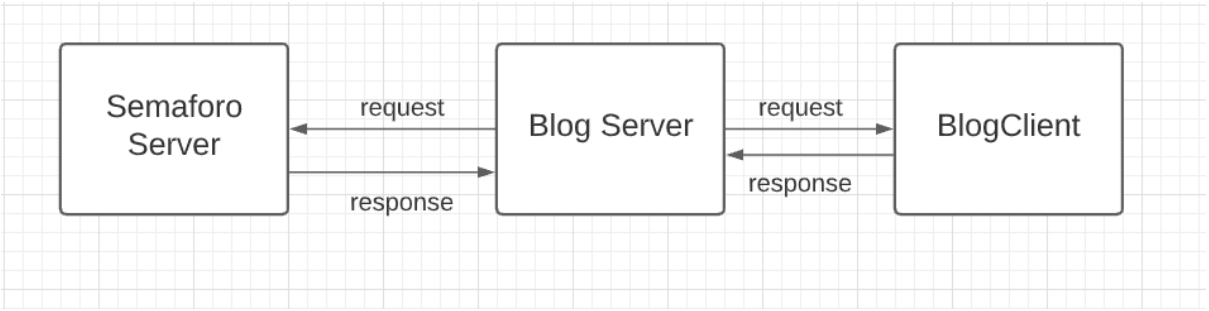
- Estrutura RPC leve e de alto desempenho.
- Desenvolvimento de API Contract-first, usando Protocol Buffers por padrão.
- Ferramentas disponíveis para várias linguagens de programação.
- Suporta chamadas streaming do client, server e bidirecionais.
- Redução do uso de rede através da serialização do Protobuf.

Esses benefícios tornam o gRPC ideal para:

- Ambientes com Microsserviços, pois torna a comunicação leve. Onde a eficiência é crítica.
- Por estar disponível em diversas linguagens torna os sistemas poliglotas. Novamente beneficiando ambientes com Microsserviços.
- Serviços real time ponta-a-ponta que precisam lidar com solicitações ou respostas de streaming.

3. Solução

Nosso domínio é um serviço de blogs, onde os clientes irão criar, ler e excluir blogs. Cada blog possui um autor, título e conteúdo.



Na definição de arquitetura, o cliente irá fazer chamadas remotas para o servidor (ou servidores) e o servidor por sua vez irá se comunicar com um servidor de semáforos que é responsável por liberar ou não as operações no banco de dados. Nós utilizamos um banco de dados MongoDB hospedado em nuvem na plataforma Atlas, podendo ser facilmente conectado e monitorado.

Servidor: Na classe servidor foram definidas três threads, cada uma rodando em uma porta diferente.

Cliente: Em cliente, nós temos 5 threads (especificamente schedulers) que executam a cada dois segundos em uma porta aleatória do servidor, com uma operação aleatória (sendo inserção, leitura e remoção).

Servidor Semafóro: É composta por três semáforos, um para cada operação. Executa toda vez que o BlogServer recebe uma operação de inserção ou deleção. Quando acontece uma inserção, o SemaphoreServer realiza um acquire antes da sessão crítica (sendo no código `collection.insertOne(document)`) e libera o mesmo após a execução. Quando ocorre uma deleção não é muito diferente, porém a deleção bloqueia todos três semáforos para garantir que nenhuma operação seja executada concorrentemente. A função de leitura é livre.

O trabalho foi desenvolvido na IDE Intellij, sendo os arquivos executáveis criados com o Gradle. A equipe acabou não utilizando a ferramenta VirtualBox devido a complicações de instalação. Então para executar utilizamos apenas a máquina local no IntelliJ e o WSL Ubuntu. Distribuindo um cliente, semáforo e um servidor na máquina original e outro cliente e servidor na máquina virtual. A comunicação do cliente da máquina local é feita com o servidor na VM e vice-versa: o cliente da VM se comunica com o servidor da máquina local.

Para facilitar a execução de múltiplos executáveis em um mesmo projeto, o Gradle nos permite dividir “tasks” para cada execução:

```
task runClient1(type: JavaExec) {
    group = 'Run' // <-- change the name as per your need
    classpath sourceSets.main.runtimeClasspath // <-- Don't change this
    main = "grpc.blog.client.BlogClient"
    args "192.168.1.103"
}

task runClient2(type: JavaExec) {
    group = 'Run' // <-- change the name as per your need
    classpath sourceSets.main.runtimeClasspath // <-- Don't change this
    main = "grpc.blog.client.BlogClient"
    args "172.28.155.115"
}

task runServer(type: JavaExec) {
    group = 'Run' // <-- change the name as per your need
    classpath sourceSets.main.runtimeClasspath // <-- Don't change this
    main = "grpc.blog.server.BlogServer"
    args "8000,8001,8002"
}

task runSemaphore(type: JavaExec) {
    group = 'Run' // <-- change the name as per your need
    classpath sourceSets.main.runtimeClasspath // <-- Don't change this
    main = "grpc.blog.semaphoreserver.SemaphoreServer"
}
```

Aqui podemos ver que cada task executa um main Java e opcionalmente um argumento de entrada. No servidor, executamos a mesma task (com as mesmas portas) em ambas máquinas, o semáforo rodamos apenas localmente. O cliente foi dividido em duas tasks, a primeira passa o IP da máquina local e a segunda o IP da VM. Comandos para execução:

- gradle runClient1
- gradle runClient2
- gradle runServer
- gradle runSemaphore

Para execução em outros ambientes e até mesmo com mais máquinas para execução, os argumentos das tasks clientes devem ser adaptados conforme a estrutura desejada.

4. Conclusão

A descrição do trabalho deu a impressão de que fosse relativamente fácil de fazer, porém alguns empecilhos surgiram no caminho, primeiramente era aprender a utilizar um framework novo e até então nenhum colega do grupo tinha conhecimento, depois foi a dificuldade em testar e validar todos os cenários envolvendo a sincronização com semáforos. Como o grupo acabou não utilizando VirtualBox, também houve muitas hipóteses sobre como utilizar mais de duas máquinas (Docker, AWS, etc…) mas não tivemos sucesso nessas tentativas então resolvemos apenas usar uma instância local e duas instâncias de VM Ubuntu. Consideramos que este trabalho foi uma excelente oportunidade para se aprofundar em como criar sistemas distribuídos de uma forma mais "baixo nível" do que estamos acostumados (basicamente APIs REST e HTTP).

5. Referências

gRPC ou REST: qual utilizar?. Disponível em:< https://vertigo.com.br/grpc-ou-rest-qual-utilizar/ >. Acesso em: 12 setembro 2021.

Introduction to gRPC. Disponível em:< https://grpc.io/docs/what-is-grpc/introduction/ >. Acesso em: 12 setembro 2021.

O que é gRPC. Disponível em:< https://www.brunobrito.net.br/grpc/ >. Acesso em: 12 setembro 2021.

Semaphores in multicore computing. Disponível em <https://cs.appstate.edu/~blk/distributedComputing/MulticoreComputing/L02-Semaphores.pdf> Acesso em: 13 setembro 2021.