Laboratório II – RPC

Programação Paralela e Distribuída (PPD)

Marcel Santana - 2213291
Arthur Manenti - 2212320

10 de novembro de 2025

Sumário

1 Objetivo

O objetivo deste laboratório é implementar um sistema distribuído cliente/servidor utilizando **Remote Procedure Call (RPC)**, utilizando a biblioteca **gRPC** em Python. O cliente realiza chamadas remotas ao servidor, que processa operações matemáticas básicas, enquanto implementa **Circuit Breaker** para lidar com falhas de comunicação.

2 Ferramentas e Tecnologias

- **Linguagem:** Python 3.x
- **Bibliotecas:**
  - grpcio
  - grpcio-tools
  - pybreaker
- **Estrutura do projeto:**
  - `proto/` - Definição dos serviços e mensagens gRPC (`calculator.proto`)
  - `server/` - Implementação do servidor e arquivos gerados pelo `protoc`
  - `client/` - Implementação do cliente e arquivos gerados pelo `protoc`

3 Definição do Serviço RPC

O serviço **Calculator** possui quatro métodos:

- **Add(Numbers)** → **Result** - Soma de dois números
- **Subtract(Numbers)** → **Result** - Subtração
- **Multiply(Numbers)** → **Result** - Multiplicação
- **Divide(Numbers)** → **Result** - Divisão

As mensagens são definidas da seguinte forma:

```
message Numbers {
  double numOne = 1;
  double numTwo = 2;
}

message Result {
  double value = 1;
  string message = 2;
}
```

4 Implementação do Servidor

O servidor implementa todas as operações e retorna o resultado com uma mensagem de status. Para divisão, trata o caso de divisão por zero.

```
class CalculatorService(calculator_pb2_grpc.CalculatorServicer):
    def Add(self, request, context):
        result = request.numOne + request.numTwo
        return calculator_pb2.Result(value=result, message="Operao␣realizada␣
            com␣sucesso")

    def Subtract(self, request, context):
        result = request.numOne - request.numTwo
        return calculator_pb2.Result(value=result, message="Operao␣realizada␣
            com␣sucesso")

    def Multiply(self, request, context):
        result = request.numOne * request.numTwo
        return calculator_pb2.Result(value=result, message="Operao␣realizada␣
            com␣sucesso")

    def Divide(self, request, context):
        if request.numTwo == 0:
            return calculator_pb2.Result(value=0, message="Erro:␣diviso␣por␣zero
                ")
        result = request.numOne / request.numTwo
        return calculator_pb2.Result(value=result, message="Operao␣realizada␣
            com␣sucesso")
```

5 Implementação do Cliente

O cliente conecta-se ao servidor, oferece um menu interativo para operações matemáticas e implementa Circuit Breaker para lidar com falhas.

```
breaker = pybreaker.CircuitBreaker(fail_max=2, reset_timeout=5)

@breaker
def do_operation(client, op, x, y):
    if op == 1:
        return client.Add(calculator_pb2.Numbers(numOne=x, numTwo=y))
    elif op == 2:
        return client.Subtract(calculator_pb2.Numbers(numOne=x, numTwo=y))
    elif op == 3:
        return client.Multiply(calculator_pb2.Numbers(numOne=x, numTwo=y))
    elif op == 4:
        return client.Divide(calculator_pb2.Numbers(numOne=x, numTwo=y))
```

6 Testes Realizados

1. **Execução normal:** Operações com números válidos.

2. **Divisão por zero:** Teste de tratamento de erro.
3. **Falha do servidor:** Verificação do Circuit Breaker.
4. **Recuperação:** Servidor religado após reset_timeout.

7 Estrutura de Pastas

```
grpc_calculator_lab/

 proto/
  calculator.proto
 server/
  server.py
  calculator_pb2.py
  calculator_pb2_grpc.py
 client/
  client.py
  calculator_pb2.py
  calculator_pb2_grpc.py
 requirements.txt
 README.txt
```

8 Conclusão

O laboratório demonstrou a implementação de um sistema distribuído com RPC usando gRPC em Python, cobrindo definição de serviços, implementação de servidor e cliente, tratamento de falhas com Circuit Breaker e testes de operação e falha. Todos os objetivos do laboratório foram atingidos.

9 Passo a Passo para Demonstração

1. Instalar dependências:

```
pip install -r requirements.txt
```

2. Gerar arquivos gRPC:

```
python -m grpc_tools.protoc -Iproto --python_out=server --grpc_python_out=
    server proto/calculator.proto
python -m grpc_tools.protoc -Iproto --python_out=client --grpc_python_out=
    client proto/calculator.proto
```

3. Iniciar servidor:

```
cd server
python server.py
```

4. Iniciar cliente em outro terminal:

```
cd client
python client.py
```

5. Testar todas as operações e simular falha do servidor para demonstrar Circuit Breaker.