Relatório Técnico: Laboratório

Disciplina: Programação Paralela e Distribuída 2025/2

Professor: Breno Krohling

Alunos:

- Kauã Pereira Porto (2313485)
- Lorena Silva Zanelato (2113314)
- Thayna Pereira Muniz (2310482)

Introdução

Este relatório detalha a metodologia de implementação, os testes realizados e os resultados obtidos no desenvolvimento do Laboratório II, focado em Chamada de Procedimento Remoto (RPC). O objetivo principal do trabalho foi experimentar a implementação de sistemas distribuídos cliente-servidor utilizando o conceito de RPC , com foco na biblioteca gRPC para a linguagem Python.

O trabalho foi dividido em duas atividades:

1. **Atividade 1:** A criação de uma calculadora distribuída interativa (soma, subtração, multiplicação e divisão).
2. **Atividade 2:** O desenvolvimento de um protótipo de mineração de criptomoedas, também baseado no modelo cliente-servidor com gRPC.

Metodologia de Implementação

Para ambas as atividades, foi utilizada a linguagem Python em conjunto com o framework gRPC. A definição dos serviços, métodos e mensagens foi realizada utilizando **Protocol Buffers** (arquivos .proto).

Ambiente e Ferramentas

- **Linguagem:** Python
- **Framework RPC:** gRPC
- **Bibliotecas Principais:** grpcio, grpcio-tools (para geração dos stubs), pybreaker (para implementação do padrão Circuit Breaker no cliente da calculadora) e inquirer (para a criação de menus interativos no cliente).

O fluxo de desenvolvimento consistiu em:

1. Definir a interface de serviço no arquivo .proto.
2. Compilar o arquivo .proto usando grpc_tools.protoc para gerar os arquivos de stub do cliente e do servidor (_pb2.py e _pb2_grpc.py).
3. Implementar a lógica do servidor (o "Servicer") com os métodos definidos.
4. Implementar a lógica do cliente para chamar os métodos remotos.

Atividade 1: Calculadora RPC (gRPC)

Definição do Serviço (grpcCalc.proto): Foi definido um serviço CalculadoraService contendo quatro métodos unários: Somar, Subtrair, Multiplicar e Dividir. Todos os métodos recebem uma mensagem OperacaoDecimal (contendo dois números float) e retornam uma mensagem ResultadoDecimal (contendo um único float).

Implementação do Servidor (grpcCalc_server.py): O servidor implementa a classe CalculadoraServiceServicer, fornecendo a lógica de negócio para as quatro operações matemáticas. Uma atenção especial foi dada à operação de divisão, que retorna 0 caso o divisor seja zero. O servidor foi configurado para escutar na porta 8080 usando um pool de threads.

Implementação do Cliente (grpcCalc_client.py): O cliente gRPC se conecta ao servidor na localhost:8080. Foi implementado um menu interativo (via biblioteca inquirer) que permite ao usuário escolher a operação desejada. O cliente então solicita os dois operandos, chama o procedimento remoto correspondente no servidor e exibe o resultado.

Adicionalmente, o cliente utiliza a biblioteca pybreaker para implementar o padrão **Circuit Breaker**. Isso aumenta a resiliência da aplicação, de modo que, após um número definido de falhas consecutivas (ex: servidor offline), o circuito "abre" e o cliente para de tentar a conexão por um tempo, informando ao usuário que o servidor está indisponível.

Atividade 2: Mineração de Criptomoedas (gRPC)

Definição do Serviço (grpc_mine.proto): Seguindo os requisitos, foi definido o serviço Mine. Este serviço expõe diversos métodos RPC, como getTransactionId, getChallenge, getTransactionStatus, submitChallenge, getWinner, getSolution e um método adicional registerClient para registrar novos mineradores. Mensagens customizadas (NumResult, TaskResult, TxnId, TaskRequest) foram criadas para lidar com as diferentes solicitações e respostas.

Implementação do Servidor (grpcMine_server.py): O servidor implementa a lógica central do sistema de mineração:

- **Gerenciamento de Estado:** Mantém um registro (dicionário) das transações, seus desafios (nível de dificuldade), soluções e vencedores.
- **Novos Desafios:** Ao ser iniciado e a cada vez que um desafio é resolvido, o servidor gera uma nova transação (_current_tx) com um novo desafio (um número aleatório de 1 a 5, representando a quantidade de zeros exigida no hash).
- **Concorrência:** Utiliza threading.Lock para garantir que o acesso ao estado compartilhado (o dicionário de transações) seja thread-safe, evitando condições de corrida.
- **Validação:** Ao receber uma submissão (submitChallenge), o servidor verifica se a transação é válida e se já não foi resolvida. Em seguida, calcula o hash **SHA-1** da solução proposta e verifica se a quantidade de zeros no início do hash é maior ou igual ao desafio proposto.
- **Registro:** Se a solução for válida, o servidor armazena a solução, o ID do cliente vencedor (Winner) e cria a próxima transação.

Implementação do Cliente (grpcMine_client.py): O cliente implementa a interface do minerador:

- **Registro:** Ao iniciar, o cliente se registra no servidor para obter um USER_ID único, gerado de forma aleatória um número até 9999, que é usado em submissões.
- **Menu Interativo:** Assim como na Atividade 1, um menu interativo oferece ao usuário acesso a todas as funções RPC disponibilizadas pelo servidor (como getChallenge, getWinner, etc.).
- **Mineração (Local):** A opção "Mine" (implementada como submitChallenge) é a mais complexa:
    1. O cliente primeiro busca a transactionID e o challenge atuais do servidor.
    2. Inicia um processo de mineração local (minerar_desafio) para encontrar uma solução.
    3. Para otimizar a busca (conforme sugerido na especificação), a mineração local utiliza múltiplas threads (4, no caso) que competem para encontrar aleatoriamente uma string cujo hash SHA-1 satisfaça o desafio.
    4. Ao encontrar uma solução, o cliente a submete ao servidor usando submitChallenge.

5. O cliente exibe a resposta do servidor (ex: "Válida", "Inválida", "Já resolvida").

Testes e Resultados

Os testes foram executados iniciando o servidor e, em seguida, um ou mais clientes em terminais separados.

Resultados da Atividade 1 (Calculadora): O sistema funcionou como esperado.

- **Operações:** Todas as quatro operações foram testadas com sucesso (ex: 10 + 5 = 15; 10 - 5 = 5; 10 * 5 = 50; 10 / 5 = 2).
- **Divisão por Zero:** O teste de divisão por zero (ex: 10 / 0) retornou corretamente 0, conforme implementado no servidor.
- **Circuit Breaker:** Ao desligar o servidor, o cliente tentou se conectar fail_max (2) vezes. Nas tentativas subsequentes, o circuito abriu e o cliente imediatamente exibiu a mensagem "Servidor temporariamente indisponível", sem tentar a conexão de rede, validando a funcionalidade do pybreaker.

Resultados da Atividade 2 (Mineração): O protótipo de mineração operou conforme os requisitos.

- **Funções de Consulta:** Todos os métodos get (getTransactionId, getChallenge, getStatus, getWinner, getSolution) retornaram as informações corretas do estado do servidor.
- **Mineração e Submissão:** O teste principal envolveu o processo de mineração. O cliente obteve o desafio (ex: 3), e as threads de mineração locais foram executadas até encontrar uma solução (uma string cujo hash SHA-1 começava com 3 ou mais zeros). A solução foi submetida e o servidor respondeu "Válida".
- **Competição (Simulada):** Ao submeter a mesma solução novamente (ou uma solução para uma transação já resolvida), o servidor corretamente respondeu "Já resolvida". Ao consultar o getWinner para aquela transação, o ID do primeiro cliente a submeter foi exibido. Imediatamente após a submissão válida, o getTransactionId do servidor foi incrementado, e um novo desafio foi disponibilizado.

Conclusão

O laboratório permitiu aplicar na prática os conceitos de Programação Distribuída e Chamada de Procedimento Remoto, cumprindo os objetivos propostos.

A utilização do gRPC e Protocol Buffers demonstrou ser uma abordagem eficiente para o desenvolvimento de sistemas cliente-servidor, abstraindo grande parte da complexidade de serialização de dados e comunicação de rede. A Atividade 1 consolidou o fluxo básico de implementação do gRPC, enquanto a Atividade 2 introduziu desafios de gerenciamento de estado, concorrência (com threading.Lock no servidor e múltiplas threads no cliente) e a implementação de uma lógica de aplicação mais complexa sobre o paradigma RPC.