MULTIVIX

RELATÓRIO TÉCNICO – Implementação de RPC e Mineração de Criptomoedas via gRPC

Membros: Kayky Salvador, Vinicius Tozato Marques, Enzo Rubim e Ingrid do Santos Gomez
Curso: Engenharia da Computação
Disciplina: Programação Distribuída e Paralela

2025/2

1. Introdução

Este relatório apresenta a implementação de duas atividades propostas na disciplina de Programação Paralela e Distribuída:
(1) adaptação de uma aplicação RPC para criar uma calculadora interativa;
(2) Desenvolvimento de um sistema de mineração de criptomoedas utilizando a arquitetura Cliente/Servidor baseada em gRPC.

O objetivo foi aplicar os conceitos de chamadas de procedimento remoto, geração de stubs, comunicação via mensagens estruturadas (protobuf) e desenvolvimento de aplicações distribuídas, incluindo operações sequenciais, concorrentes e controle de múltiplos clientes.

2. Metodologia de Implementação

2.1 Tecnologias Utilizadas

- **Linguagem:** Python
- **Framework de RPC:** gRPC
- **Estruturação das mensagens:** Protocol Buffers (.proto)
- **Hash criptográfico utilizado:** SHA-1
- **Execução paralela no cliente:** uso de múltiplas threads para busca da solução
- **Ambiente executado no vídeo:** execução local com cliente e servidor separados

2.2 Arquitetura Geral

A solução foi estruturada no modelo Cliente/Servidor:

- **Servidor gRPC**
    - Mantém uma tabela de transações contendo: *TransactionID, Challenge, Solution* e *Winner*.

    - Gera automaticamente um novo desafio ao iniciar a execução.
    - Permite múltiplas conexões simultâneas.
    - Responde a todas as funções RPC exigidas no enunciado: *getTransactionID, getChallenge, getTransactionStatus, submitChallenge, getWinner, getSolution.*

- **Cliente gRPC**
    - Conecta-se ao servidor através do stub gerado pelo protobuf.
    - Oferece menu interativo com todas as funções solicitadas.
    - Implementa o processo de mineração:
        1. Obtém TransactionID atual
        2. Obtém o desafio
        3. Executa busca da solução usando threads
        4. Exibe solução encontrada
        5. Submete ao servidor
        6. Exibe resultado da mineração

2.3 Estrutura do Arquivo .proto

O arquivo .proto define:

- Estruturas de dados para troca entre cliente e servidor
- Mensagens contendo inteiros e strings
- O serviço com todos os métodos exigidos no enunciado

Esse arquivo foi posteriormente compilado usando o comando *grpc_tools.protoc*, gerando automaticamente os stubs em Python.

2.4 Implementação do Servidor

O servidor permanece em execução contínua, gerando desafios e armazenando o histórico das transações. As principais responsabilidades foram:

- Criar challenges aleatórios ou sequenciais de dificuldade entre **1 e 20**
- Controlar o status de cada transação
- Verificar se a solução submetida corresponde ao hash SHA-1 esperado
- Registrar o **ClientID** vencedor
- Retornar os códigos solicitados:
    - **1**: solução válida
    - **0**: solução inválida
    - **2**: desafio já foi solucionado
    - **-1**: transactionID inválido

Toda conversão, envio e recebimento de dados é realizada automaticamente pelo stub do gRPC.

2.5 Implementação do Cliente

O cliente apresenta um menu com todas as operações RPC. A operação **Mine** utiliza múltiplas threads para tentar encontrar a solução do desafio, reduzindo o tempo total de busca.

Passos realizados pelo cliente na mineração:

1. Solicitar transactionID corrente ao servidor
2. Solicitar o challenge correspondente
3. Gerar tentativas de solução usando threads

4. Imprimir a solução encontrada
5. Submeter a solução ao servidor
6. Interpretar a resposta do servidor

3. Testes Realizados

Os testes foram executados conforme apresentado no vídeo enviado, com o servidor e o cliente rodando em terminais separados.

3.1 Teste das Funções Individuais

Foram testadas todas as operações do menu:

- **getTransactionID:** retornou corretamente o ID atual em aberto
- **getChallenge:** mostrou o valor do desafio associado ao ID
- **getTransactionStatus:** indicou se a transação estava resolvida ou pendente
- **getWinner:** retornou o clientID vencedor após a mineração
- **getSolution:** retornou o hash esperado e o challenge correspondente

Todas as funções responderam corretamente, indicando comunicação adequada via gRPC.

3.2 Teste da Mineração

O teste de mineração seguiu as etapas:

- O cliente recebeu o ID da transação ativa
- Obteve o desafio (nível de dificuldade)
- Executou o processo de busca usando múltiplas threads
- Encontrou uma solução válida

- Submeteu ao servidor
- Recebeu a confirmação de sucesso

O servidor registrou corretamente:

- A **solution**
- O **ClientID vencedor**
- O **status da transação** como resolvido

Após isso, o servidor criou automaticamente uma nova transação.

3.3 Comportamento no Vídeo

Com base no vídeo fornecido:

- O servidor inicializa e exibe que está escutando conexões.
- O cliente se conecta e o menu é apresentado.
- Cada função é usada corretamente, exibindo os resultados esperados.
- Na etapa de mineração, o cliente encontra a solução, imprime, e recebe o retorno positivo do servidor.
- O servidor registra o vencedor e gera novo challenge.

O comportamento observado confirma que a implementação atende às exigências funcionais do laboratório.

4. Resultados

Os resultados obtidos comprovam o funcionamento correto do sistema distribuído:

- Todas as chamadas RPC funcionaram sem erros.
- A comunicação cliente/servidor via gRPC ocorreu de forma eficiente.
- A mineração foi executada com sucesso utilizando concorrência.
- O servidor conseguiu armazenar e atualizar corretamente a tabela de transações.
- O vídeo demonstra claramente a execução das operações e o fluxo completo da mineração.

Os objetivos propostos foram alcançados utilizando boas práticas de RPC, gRPC, programação distribuída e paralelismo com threads.

5. Conclusão

O projeto permitiu explorar na prática os conceitos fundamentais de RPC, stubs, estruturas protobuf e comunicação distribuída. A implementação demonstrou que o gRPC simplifica a construção de sistemas cliente/servidor complexos, mantendo alto desempenho e interoperabilidade.

Além disso, a atividade de mineração agregou o uso de hashing e paralelismo, mostrando como múltiplas threads podem acelerar tarefas computacionais intensivas. Os testes em vídeo confirmam a corretude e estabilidade do sistema implementado.