Documento de Análise de Resultados

Disciplina: Sistemas Distribuídos

Projeto: Paralelismo, Comunicação e Sistemas de notificação

Autores:

- Miguel Melo

- Sílvio Moreira

- Diogo Novaes

**1. Introdução**

Este documento apresenta a análise dos resultados obtidos no miniprojeto da disciplina de Sistemas Distribuídos. O objetivo do trabalho foi explorar e implementar na prática conceitos fundamentais da área, demonstrando o funcionamento e as características de diferentes arquiteturas e tecnologias.

O projeto foi dividido em três partes sequenciais e complementares: a aplicação de paralelismo com threads para otimização de tarefas; a implementação e comparação de modelos de comunicação em rede (Cliente-Servidor e Peer-to-Peer); e, por fim, a construção de um sistema distribuído com múltiplos serviços, orquestrado por meio de comunicação síncrona (RPC) e assíncrona (Filas de Mensagens).

**2. Análise da Parte 1: Filtro de Imagens com Paralelismo**

Nesta etapa, o objetivo era demonstrar a eficácia do paralelismo na redução do tempo total de execução para tarefas independentes, como o processamento de imagens.

**2.1. Metodologia**

Utilizou-se a biblioteca threading do Python para executar a função de aplicação de filtro em duas imagens distintas de forma concorrente. Para fins de comparação, uma versão puramente sequencial do mesmo script foi criada, onde as imagens são processadas uma após a outra em um laço de repetição simples. Os tempos de execução foram medidos e os logs de cada thread foram capturados com timestamps para análise.

**2.2. Resultados**

A execução dos dois modos (sequencial e paralelo) em uma máquina de teste resultou nos seguintes tempos:

Sequencial : 11.42 segundos

Paralelo (Com as Threads): 5.89 segundos.

**2.3. Análise**

Os resultados demonstram uma redução de aproximadamente **48%** no tempo total de execução ao utilizar a abordagem paralela. O log confirma que as duas threads iniciaram suas tarefas quase que simultaneamente e trabalharam em paralelo. A vantagem de desempenho surge da capacidade do sistema operacional de otimizar o uso de recursos: enquanto uma thread aguardava operações de I/O (leitura ou escrita da imagem no disco), a outra podia utilizar o tempo de CPU para o processamento do filtro, resultando em uma execução muito mais eficiente do que o modelo sequencial, que é forçado a esperar a conclusão total de uma tarefa antes de iniciar a próxima.

**3. Análise da Parte 2: Modelos de Comunicação**

Esta etapa teve foco na implementação e análise de duas arquiteturas de comunicação fundamentais.

**3.1. Modelo Cliente-Servidor**

A arquitetura implementada consistiu em um servidor central capaz de aceitar múltiplas conexões simultaneamente, delegando cada cliente a uma thread dedicada.

* **Resultados:** O servidor demonstrou ser robusto, gerenciando um contador de conexões ativas de forma segura (utilizando threading.Lock para evitar condições de corrida) e respondendo a cada cliente de forma independente, sem que um afetasse o outro.
* **Análise:** É uma arquitetura centralizada, o que a torna mais simples de gerenciar e proteger, com papéis claramente definidos. No entanto, sua principal desvantagem é ser um **ponto único de falha**; se o servidor cair, todo o sistema se torna inoperante. Além disso, pode se tornar um gargalo de desempenho com um número muito elevado de clientes.

**3.2. Modelo Peer-to-Peer (P2P)**

Neste modelo, cada instância do programa (no\_p2p.py) atuou simultaneamente como cliente e servidor.

* **Resultados:** A implementação permitiu que cada nó estabelecesse conexões de saída enquanto, ao mesmo tempo, aceitava conexões de entrada através de uma thread de escuta dedicada. Múltiplos nós puderam formar uma rede e trocar mensagens em um modelo de chat distribuído.
* **Análise:** A arquitetura P2P é inerentemente **descentralizada e resiliente**. A falha de um nó não afeta a capacidade de comunicação dos outros. Sua principal desvantagem reside na maior complexidade da lógica de cada nó e nos desafios de descoberta de outros pares e gerenciamento de estado em uma rede maior.

**4. Análise da Parte 3: Sistema de Notificação Distribuída**

A etapa final integrou os conceitos anteriores para construir um sistema com múltiplos serviços, focando em diferentes padrões de comunicação para alcançar objetivos distintos.

**4.1. Arquitetura do Sistema**

O sistema foi composto por três serviços independentes: um Servidor de Pedidos (a "cozinha"), um Serviço de Notificação (o "painel de senhas e garçom"), e o Cliente. A comunicação entre os dois serviços de backend foi o foco da análise.

**4.2. Análise das Tecnologias de Comunicação**

* **RPC (Remote Procedure Call):**
  + **Uso:** Foi utilizado para a notificação **final e síncrona** de que um pedido foi concluído. O servidor\_pedidos invocou diretamente a função notificar\_conclusao no servico\_notificacao.
  + **Análise:** O RPC se mostrou ideal para ações de comando direto, onde uma confirmação imediata é necessária. A chamada é **síncrona**, o que significa que o serviço de pedidos aguarda a confirmação do serviço de notificação antes de continuar. Isso cria um **acoplamento forte** entre os serviços, pois o chamador precisa conhecer o endereço e a assinatura exata da função remota.
* **Fila de Mensagens (RabbitMQ):**
  + **Uso:** Foi utilizado para as atualizações de **status** durante o processamento ("pedido em processamento..."). O servidor\_pedidos publicou uma mensagem na fila fila\_de\_status sem se preocupar com quem a consumiria.
  + **Análise:** Este padrão de comunicação é **assíncrono**, permitindo que o servidor de pedidos envie a notificação e continue seu trabalho imediatamente, sem esperar. Ele promove um **baixo acoplamento**, pois os serviços não se conhecem; eles conhecem apenas o "canal" (a fila). Isso aumenta a **resiliência e a escalabilidade** do sistema: se o serviço de notificação estivesse offline, as mensagens de status aguardariam na fila até que ele voltasse, sem perda de informação.

**5. Conclusão Geral**

A realização deste projeto permitiu a aplicação prática de conceitos fundamentais de sistemas distribuídos. Foi possível observar empiricamente as vantagens de desempenho do paralelismo em tarefas independentes, contrastar as características, vantagens e desvantagens de arquiteturas centralizadas (Cliente-Servidor) e descentralizadas (P2P), e, finalmente, evidenciar os diferentes casos de uso e os trade-offs entre comunicação síncrona (RPC) e assíncrona (Filas de Mensagens) na construção de sistemas de software mais robustos, resilientes e escaláveis.