UNIVERSIDADE CATÓLICA DE BRASÍLIA

PROGRAMAÇÃO CONCORRENTE E DISTRIBUÍDA

Relatório de Análise Experimental sobre o Impacto da Concorrência na Performance de Algoritmos I/O-Bound

Autor: Camila Mendes Osterno

Matrícula: UC23421910

1. Introdução

Este relatório apresenta os resultados de um experimento desenvolvido na disciplina de Programação Concorrente e Distribuída. O objetivo foi analisar o impacto do uso de múltiplas *threads* na performance de um algoritmo com operações intensivas de Entrada e Saída (I/O-bound).

Para isso, foi implementado um programa em Java 17 que realiza requisições HTTP a uma API de dados climáticos, coletando informações das 27 capitais brasileiras durante o mês de janeiro de 2024. Foram testadas quatro versões do programa: uma sequencial (apenas a *thread* principal) e outras três versões concorrentes, utilizando 3, 9 e 27 *threads*.

Este documento apresenta os conceitos teóricos que fundamentam o experimento, a metodologia aplicada, os resultados obtidos e uma análise comparativa sobre o desempenho entre as diferentes abordagens.

2. Fundamentação Teórica

2.1 O que são *Threads*

Uma *thread* é a menor unidade de processamento que pode ser executada de forma independente dentro de um processo. Em um ambiente multithread, várias *threads* compartilham o mesmo espaço de memória do processo, mas cada uma possui seu próprio fluxo de execução. Isso permite que tarefas distintas sejam realizadas simultaneamente, aumentando a eficiência do sistema (TANENBAUM; BOS, 2015).

O uso de *threads* é especialmente vantajoso quando se deseja dividir uma tarefa em partes menores que possam ser executadas em paralelo, aproveitando múltiplos núcleos de um processador moderno (SILBERSCHATZ; GALVIN; GAGNE, 2018).

2.2 *Threads* em Operações I/O-Bound

Algoritmos I/O-bound são aqueles cujo desempenho é limitado não pela CPU, mas por operações de entrada e saída, como acesso à internet, leitura de arquivos ou comunicação com banco de dados. Durante uma requisição HTTP, por exemplo, o programa precisa aguardar uma resposta do servidor remoto. Em uma execução sequencial, esse tempo é desperdiçado.

Com o uso de *threads*, enquanto uma requisição está aguardando a resposta, outras *threads* podem continuar executando. Isso reduz significativamente o tempo total de execução, pois sobrepõe os períodos de espera (GOETZ et al., 2006).

2.3 Concorrência x Paralelismo

Concorrência e paralelismo são conceitos diferentes. Concorrência refere-se à capacidade de lidar com múltiplas tarefas ao mesmo tempo, mesmo em um sistema com apenas um núcleo — por meio de alternância entre tarefas. Já o paralelismo exige múltiplos núcleos, permitindo a execução simultânea de múltiplas tarefas (TANENBAUM; BOS, 2015).

No experimento realizado, o algoritmo foi projetado para executar de forma concorrente. Quando executado em sistemas com múltiplos núcleos, ele também se beneficia do paralelismo, acelerando ainda mais sua execução.

> “Concorrência trata da estruturação do software para lidar com múltiplas tarefas. Paralelismo trata da execução real simultânea dessas tarefas.” (GOETZ et al., 2006, p. 4).

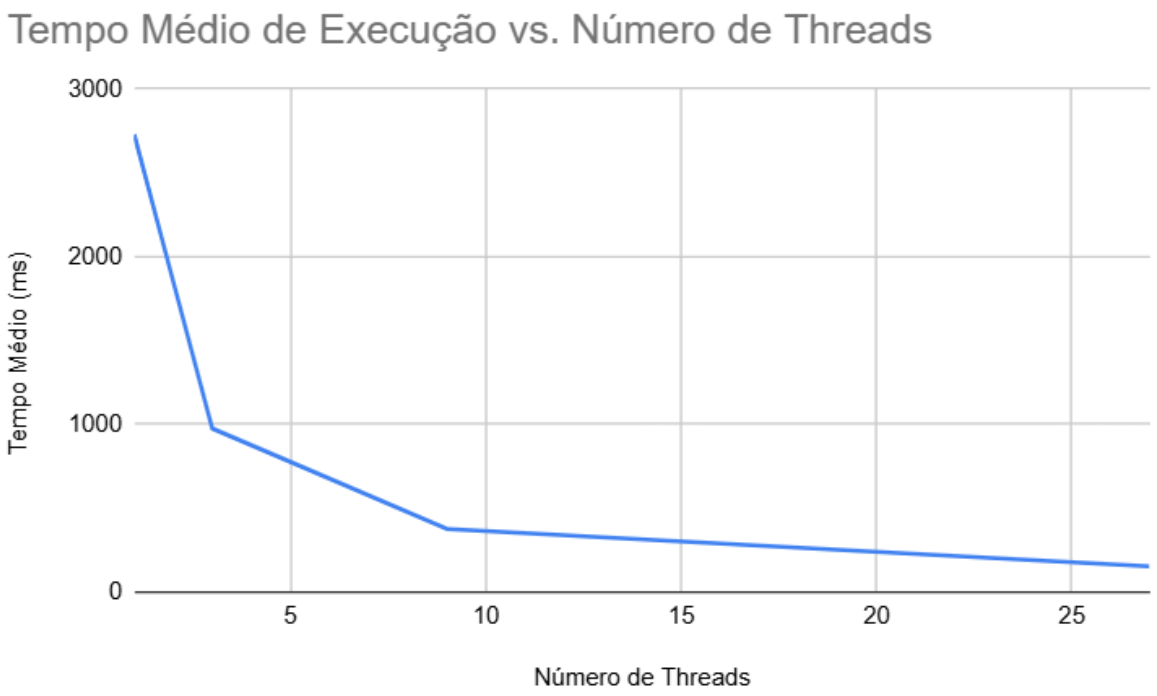
3. Metodologia e Resultados

O experimento coletou dados de temperatura das 27 capitais brasileiras durante janeiro de 2024, utilizando a API da Open-Meteo. Foram testadas quatro versões do programa com diferentes quantidades de *threads* (1, 3, 9 e 27). Cada versão foi executada 10 vezes e, ao final, foi calculada a média do tempo de execução.

3.1 Tabela de Resultados

Versão do Experimento	Nº de Threads	Tempo Médio (ms)	Ganho de Performance (Speedup)
Sequencial (Referência)	1	2.728,80	1,00x
Concorrente	3	975	2,80x
Concorrente	9	376,4	7,25x
Concorrente	27	152,9	17,85x

3.2 Gráfico Comparativo



3.3 Análise dos Resultados

Os dados demonstram que o uso de *threads* trouxe ganhos expressivos de desempenho. A versão sequencial teve o maior tempo médio de execução (2728,80 ms). Com 3 *threads*, o tempo foi reduzido para 975 ms. A versão com 9 *threads* teve uma média de 376,40 ms, e a versão com 27 *threads* atingiu 152,90 ms.

Essa diferença mostra como o paralelismo pode reduzir o tempo ocioso causado por operações I/O-bound. No entanto, é importante destacar que os ganhos não seguem uma

proporção linear: de 9 para 27 *threads*, o ganho percentual foi menor, o que pode indicar limitações relacionadas à rede ou à própria sobrecarga de gerenciar muitas *threads*.

4. Conclusão

O experimento comprovou, na prática, que a programação concorrente com múltiplas *threads* é uma estratégia eficiente para lidar com tarefas que envolvem chamadas externas, como requisições HTTP. A execução concorrente permitiu sobrepor os tempos de espera de rede, otimizando o uso da CPU e reduzindo drasticamente o tempo total.

Ao comparar a versão sequencial com a de 27 *threads*, houve uma melhora de desempenho de mais de 94%, mostrando que a adoção de abordagens concorrentes é essencial no desenvolvimento de sistemas modernos, especialmente aqueles que dependem de I/O intensivo.

5. Referências Bibliográficas

- GOETZ, Brian et al. *Java Concurrency in Practice*. Addison-Wesley, 2006.

- SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. *Sistemas Operacionais: Conceitos*. 9. ed. Rio de Janeiro: LTC, 2018.

- TANENBAUM, Andrew S.; BOS, Herbert. *Modern Operating Systems*. 4. ed. Boston: Pearson, 2015.