Conceito e utilização das Threads

Threads são partes de um todo, são etapas para que o objetivo final seja alcançado. Um programa inteiro funciona para que determinado objetivo seja alcançado, mas até que chegue nesse ponto, cada thread é responsável pelo desenvolvimento de uma tarefa, e ao final fica a sensação de que tudo ocorreu ao mesmo tempo.

Segundo Cordeiro (2004, p. 2)

> Threading é um artifício que permite a coexistência de m´múltiplas atividades dentro de um ´único processo. Java é a primeira linguagem de programação a incluir explicitamente o conceito de Threads na própria linguagem. Threads s˜ao também chamados de "processos leves" (lightweight processes) pois, da mesma forma que processos, threads são independentes, possuem sua própria pilha de execução, seu próprio program counter e suas próprias variáveis locais. Porém, threads de um mesmo processo compartilham memória, descritores de arquivos (file handles) e outros atributos que são específicos daquele processo.

Ou seja, de acordo com Cordeiro (2004) threads são pequenos processos que fazem parte de um processo maior, são independentes, mas trabalham juntos em prol de um objetivo em comum. O autor também já nos traz muito sobre o funcionamento da thread, que é como um subprocesso, onde cada thread tem sua própria pilha de execução, seu contador de programa, mas a memória e recursos são iguais a do processo principal, e é dessa forma que funciona uma thread.

Segundo Guardia & Crestana (2015, p.7)

> Para resolver os problemas de lentidão na criação de processos e também na troca de contexto, além de tornar uso de memória mais eficiente e simplificar o compartilhamento de dados, foi desenvolvido o conceito de processos "leves", com menos informações de contexto, que são chamados de threads, ou linhas de execução. Com um contexto mais reduzido, a criação de threads e a troca de contexto entre elas pode ser muito mais rápida. Com isso, o processador terá mais tempo para executar o trabalho realmente útil das aplicações. Aplicações que executam múltiplas atividades simultaneamente poderão executar mais rápido se utilizarem threads em vez de processos. Além disso, o compartilhamento de memória e a comunicação entre as trheads de um processo é mais simples. Do ponto de vista dos programas, essa facilidade de compartilhamento de dados entre diferentes linhas de execução desse programa talvez seja o maior incentivo para o uso de threads.

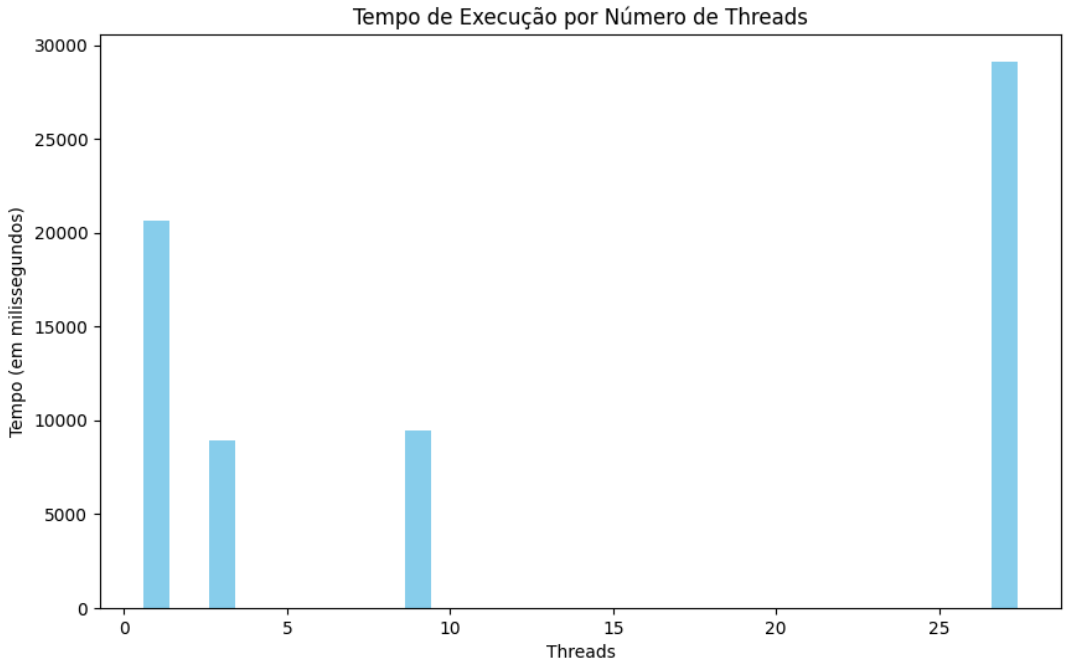
Ou seja, os ‘’processos leves’ surgiram para justamente solucionar problemas de lentidão de criação de processos, sendo assim, as threads colaboram para a eficiência e rapidez de um

programa, pois são várias tarefas executadas ao mesmo tempo, mas de núcleos diferentes, o que faz com que uma não atrase a outra.

Quanto a relação entre modelos de computação concorrente e paralelo e desempenho dos algoritmos, nos dois modelos são utilizados threads com fim de melhorar a eficiência dos mesmos, mas é importante ressaltar que o uso por si só não garante melhora, é preciso saber direcionar e gerir o uso das threads de forma adequada.

Análise dos Resultados Obtidos

A comparação dos tempos de execução das quatro versões do experimento, cada uma utilizando um número diferente de threads (1, 3, 9 e 27), revela algumas observações importantes sobre o comportamento dos programas concorrentes. Os resultados são apresentados no gráfico abaixo:



1. Versão de Referência (1 Thread)

Na versão de referência, todas as requisições e processamento de dados foram realizados em uma única thread principal. O tempo de execução médio registrado foi de 20.651 milissegundos. Este tempo elevado se deve ao fato de que todas as operações são executadas de forma sequencial, sem qualquer paralelismo. Cada requisição HTTP e o processamento subsequente dos dados precisam esperar a conclusão das operações anteriores, o que maximiza o tempo total necessário.

2. Versão com 3 Threads

Na versão com 3 threads, o trabalho foi dividido em três threads, cada uma responsável por 9 capitais. O tempo de execução médio registrado foi de 8.920 milissegundos. Este resultado mostra uma significativa redução no tempo de execução comparado à versão de referência. Utilizando três threads, as requisições e processamentos puderam ser executados em paralelo, aproveitando melhor os recursos disponíveis e diminuindo o tempo de espera ocioso entre operações.

3. Versão com 9 Threads

Na versão com 9 threads, o trabalho foi distribuído em nove threads, cada uma responsável por 3 capitais. O tempo de execução médio registrado foi de 9.477 milissegundos. Embora haja um aumento no paralelismo comparado à versão de 3 threads, o tempo de execução médio não diminuiu significativamente. Isso pode ser devido ao overhead associado à criação e gerenciamento de mais threads, além da possível saturação dos recursos do sistema, como a largura de banda da rede ou a capacidade de processamento da CPU.

4. Versão com 27 Threads

Na versão com 27 threads, cada thread foi responsável por uma única capital. O tempo de execução médio registrado foi de 29.133 milissegundos, o maior entre todas as versões testadas. Este resultado surpreendente pode ser explicado por vários fatores:

- **Overhead de Gerenciamento de Threads**: A criação e o gerenciamento de 27 threads podem introduzir um overhead significativo. O sistema operacional precisa alocar recursos e gerenciar o contexto de cada thread, o que pode diminuir a eficiência global.
- **Concorrência por Recursos**: Com muitas threads ativas ao mesmo tempo, há uma maior concorrência pelos recursos do sistema, como CPU, memória e largura de banda de rede. Isso pode levar a contenções e a um desempenho geral mais lento.
- **Limitações da API e da Rede**: A API utilizada pode ter limitações em termos de número de requisições simultâneas ou na velocidade de resposta quando muitas requisições são feitas ao mesmo tempo.

Comentários Finais

Os resultados mostram que, embora o uso de threads possa acelerar significativamente o tempo de execução de um programa concorrente, há um ponto em que o aumento no número de threads não é mais benefício e pode até prejudicar o desempenho. A versão com 3 threads demonstrou a maior eficiência, reduzindo o tempo de execução em mais de 50% em comparação com a versão de referência de uma única thread. A versão com 9 threads mostrou uma leve piora no desempenho devido ao overhead adicional. Finalmente, a versão com 27 threads teve o pior desempenho, mostrando que a sobrecarga de gerenciamento de muitas threads e a concorrência por recursos podem anular os benefícios do paralelismo.

Para maximizar a eficiência em programas concorrentes, é essencial encontrar um equilíbrio no número de threads utilizadas, levando em consideração o overhead de gerenciamento e os recursos disponíveis do sistema.

REFERÊNCIAS

CORDEIRO, Daniel de Angelis. **Introdução ao uso de Threads em Java.** 2004.

GUARDIA, Hélio Crestana. **Políticas de escalonamento de processos e threads.** 2015.