Universidade Católica de Brasília

Pró-Reitoria Acadêmica

Curso de Ciência da Computação

Trabalho de Programação Concorrente e Distribuída

AT-2/ N1 THREADS

Autor: Isabella Cristina Santos Silva

Orientador: João Robson

BRASÍLIA – DF
2024

INTRODUÇÃO

O presente relatório tem como objetivo apresentar a análise de um código desenvolvido em linguagem Java para processamento de dados climáticos armazenados em arquivos CSV (Comma-Separated Values). A aplicação utiliza a técnica de paralelismo por meio de múltiplas threads para otimizar o processamento de grandes volumes de dados, calculando estatísticas, como a temperatura mínima, média e máxima para diferentes meses.

A abordagem de Threads foi escolhida visando reduzir o tempo de processamento, principalmente em sistemas com múltiplos núcleos de processamento, explorando de maneira eficiente os recursos computacionais disponíveis.

O objetivo deste trabalho é apresentar a implementação de um sistema em Java capaz de processar arquivos CSV com dados de temperatura, dividindo a carga de processamento entre múltiplas threads. O programa deve calcular as estatísticas de temperatura mínima, média e máxima para cada mês do ano e, ao final, medir o tempo de execução de múltiplas rodadas de processamento, exibindo o tempo médio.

SOBRE O CÓDIGO

A aplicação foi desenvolvida em Java utilizando bibliotecas de concorrência da linguagem, como java.util.concurrent.ExecutorService, para criação e gerenciamento de um pool de threads. Abaixo, detalha-se o funcionamento das principais partes do código.

O código é dividido nas seguintes classes e métodos:

- **Classe Principal (ProcessadorTemperaturas)**: Coordena o fluxo de execução do programa, controlando as rodadas de processamento, a distribuição de tarefas entre as threads e a medição de tempo.
- **Método obterArquivosCSV**: Carrega os arquivos CSV presentes em um diretório específico, filtrando apenas aqueles que contêm dados climáticos.
- **Classe TarefaProcessamento**: Implementa a interface Runnable e é responsável por processar os arquivos CSV atribuídos à thread. Cada thread processa seus arquivos de forma independente, calculando as temperaturas mínima, média e máxima para cada mês do ano.
- **Método salvarTemposExecucao**: Ao final do processamento, os tempos de execução de cada rodada e o tempo médio são gravados em um arquivo de texto.

Funcionamento do Código

A execução do código segue as etapas abaixo:

1. O método main inicia a execução, dividindo os arquivos CSV em grupos que serão processados por threads diferentes. São realizadas múltiplas rodadas de processamento (10 no total), e em cada rodada, o tempo de execução é medido.
2. Para cada rodada, o método obterArquivosCSV é chamado para carregar os arquivos CSV, que são divididos entre as threads, utilizando um ExecutorService com um número fixo de threads (320).
3. A classe TarefaProcessamento é responsável por processar cada arquivo. Ela lê os dados de temperatura do arquivo e organiza as temperaturas por mês, calculando em seguida a temperatura mínima, média e máxima de cada mês.
4. Ao final de cada rodada, o tempo de execução é armazenado. O tempo médio de execução é calculado após a última rodada e todos os dados são salvos em um arquivo de texto.

RESULTADOS

Durante a execução, o código realizou 10 rodadas de processamento, cada uma com 320 threads trabalhando em paralelo. A medição de tempo de execução para cada rodada foi realizada, e o tempo médio foi calculado ao final.

Os resultados indicaram que o uso de threading proporcionou uma melhora significativa na performance em comparação com uma abordagem sequencial. Em ambientes com múltiplos núcleos, o tempo de execução foi drasticamente reduzido.

Em um cenário típico, a redução no tempo de execução foi notável, com a divisão das tarefas entre threads resultando em tempos de processamento mais curtos. Porém, o número de threads deve ser ajustado cuidadosamente conforme os recursos disponíveis na máquina, evitando sobrecarga e perda de eficiência.

CONCLUSÃO

O uso de Threads no processamento de dados climáticos demonstrou ser uma solução eficaz para otimizar o tempo de execução em aplicações que processam grandes volumes de dados. A implementação permitiu a distribuição da carga de trabalho de forma eficiente entre threads, resultando em uma diminuição significativa no tempo total de processamento.

No entanto, o ajuste do número de threads em relação à capacidade de processamento da máquina é crucial para maximizar os ganhos de performance. Futuras melhorias no código podem incluir a implementação de técnicas de balanceamento dinâmico de carga e otimizações na leitura e escrita de arquivos.

REFERÊNCIAS

https://cursos.alura.com.br/forum/topico-filewriter-ou-bufferedwriter-68649

https://sentry.io/answers/arraylist-from-array/#:~:text=The%20easiest%20way%20to%20convert,that%20implement%20the%20List%20interface

https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html

https://www.baeldung.com/thread-pool-java-and-guava/