Avaliação do ganho de desempenho usando paralelismo em dados IoT de Cidades Inteligentes

Matheus Tregnago[1], **Ricardo Bregalda**[1] **e Samuel Francisco Ferrigo**[1]

[1]Campus Universitário da Região dos Vinhedos
Universidade de Caxias do Sul (UCS) – Bento Gonçalves – RS – Brasil

{matregnago,rmbregalda,sfferrig}@ucs.br

Resumo. *Este artigo avalia o uso de diferentes técnicas de paralelismo para processamento de dados IoT em cidades inteligentes. Para tal, foram utilizadas as técnicas de paralelismo OpenMP, MPI, Multiprocessing e Dask para identificar falhas em sensores e avaliar o ganho de desempenho, alcançando speedup de até 5,16 com Dask. Os resultados reforçam as vantagens do uso de paralelismo no processamento de grandes volumes de dados.*

1. Introdução

Com o avanço das tecnologias de Internet das Coisas (IoT), o monitoramento em tempo real de ambientes urbanos tornou-se uma realidade nas cidades inteligentes [Bekkai et al. 2021]. Sensores espalhados por diferentes regiões capturam variáveis como temperatura, umidade e luminosidade, gerando uma grande quantidade de dados que precisam ser processados de forma eficiente para extrair informações úteis. No entanto, a complexidade e o volume de dados gerados apresentam desafios significativos de processamento, seja para a extração de informações relevantes para o desenvolvimento das cidades quanto para identificação de possíveis problemas envolvendo os dispositivos que realizam essas leituras, especialmente em situações que demandam análise rápida. Nesse sentido, o uso de tecnologias de processamento paralelo e distribuído são uma alternativa real para essas situações.

2. Referencial Teórico

A computação paralela é uma alternativa para o aumento de desempenho em atividades que requerem alto nível de processamento. Diversas são as tecnologias que podem ser implementadas para tal finalidade, como OpenMP, MPI e bibliotecas Python específicas, como Multiprocessing e Dask.

O OpenMP é uma Interface de Programação de Aplicações (API) que pode ser usada para direcionar explicitamente o paralelismo *multi-threaded* em memória compartilhada em programas C/C++. Ela é projetada de forma não intrusiva ao código original, uma vez que as instruções do OpenMP são inseridas em *pragmas*, que são interpretadas pelo compilador [Carleton 2024]. O MPI é uma especificação de interface para bibliotecas de passagem de mensagens, voltada para o modelo de programação paralela baseado em passagem de mensagens. Nesse modelo, os dados são transferidos entre os espaços de memória de diferentes processos por meio de operações cooperativas [MPI 2024] . O Multiprocessing é um pacote nativo do Python que suporta a criação de processos utilizando uma API semelhante à do módulo *threading*. O pacote multiprocessing oferece concorrência tanto local quanto remota, contornando efetivamente o Global Interpreter

Lock (GIL) ao usar subprocessos em vez de threads [Python 2024]. Finalmente, o Dask é uma biblioteca de computação paralela e distribuída que permite processar grandes volumes de dados de forma eficiente. Ele é ideal para trabalhar com dados que não cabem na memória, pois divide as tarefas em pequenos blocos de trabalho que podem ser executados em paralelo em múltiplos núcleos ou máquinas, com suporte para execução tanto local quanto em *clusters* [Dask 2024].

3. Metodologia

Como cenário de avaliação das tecnologias de computação paralela, buscou-se identificar os 50 maiores períodos de travamento em sensores de temperatura, umidade e luminosidade que transmitiam os dados para uma base, de forma a avaliar a confiabilidade dos sensores. A base de dados utilizada faz parte do projeto de sensoriamento IoT desenvolvido pelo grupo de pesquisa City Living Lab[1], desenvolvido com recursos do programa InovaRS[2]. Os dados estavam organizados em um arquivo *CSV*, contendo doze colunas e mais de quatro milhões de linhas, sendo cada linha uma medição de sensor com informações como identificador, nome do sensor, data e valores registrados. O algoritmo desenvolvido analisa as colunas referentes aos valores de temperatura, umidade e luminosidade; outros valores, como ruído e parâmetros de rede, não foram considerados pois não faziam parte do escopo da análise [Carvalho et al. 2024].

A solução desenvolvida[3] para realizar essa análise consistia em uma aplicação cliente-servidor TCP. O cliente enviava a base de dados para o servidor, que realizava a análise a partir de seis métodos de processamento: dois deles sequenciais, um escrito em C e outro escrito em Python, como referência no cálculo do speedup das aplicações paralelas; e quatro paralelizados, utilizando OpenMP em C, MPI em Python, e as bibliotecas Python Multiprocessing e Dash, conforme mostra a Figura 1. No MPI, ainda, havia a distribuição da carga entre diferentes nós interligados numa mesma rede local. Para cada uma das técnicas de processamento paralelo, foram feitos testes considerando cenários com 1, 2, 4, 8 e 16 threads/processos, com três testes em cada cenário e cálculo do tempo médio das execuções. O servidor responsável pelos testes possuía processador Intel 14600k, 32GB de RAM DDR5 em uma máquina virtual WSL com um Sistema Operacional Ubuntu 22.04.

O servidor processava os dados conforme a requisição solicitada pelo cliente, retornando os 50 maiores intervalos de travamento, juntamente com o tempo total de execução do processamento da tarefa.

4. Resultados e discussões

Primeiramente, ao executar-se os códigos sequenciais, verifica-se que o tempo de execução do código sequencial em C (3,76s) foi 6,7 vezes mais rápido que o processamento realizado em Python (25,22s), devido principalmente ao overhead causado pelo interpretador Python.

A Figura 2 apresenta os resultados obtidos com os diferentes métodos de paralelização utilizados. A paralelização em OpenMP trouxe uma pequena melhora no

[1]https://www.citylivinglab.com/iot-inova-rs

[2]https://programainova.rs.gov.br/inicial

[3]Código-fonte disponível em https://github.com/matregnago/distributed-iot-analytics

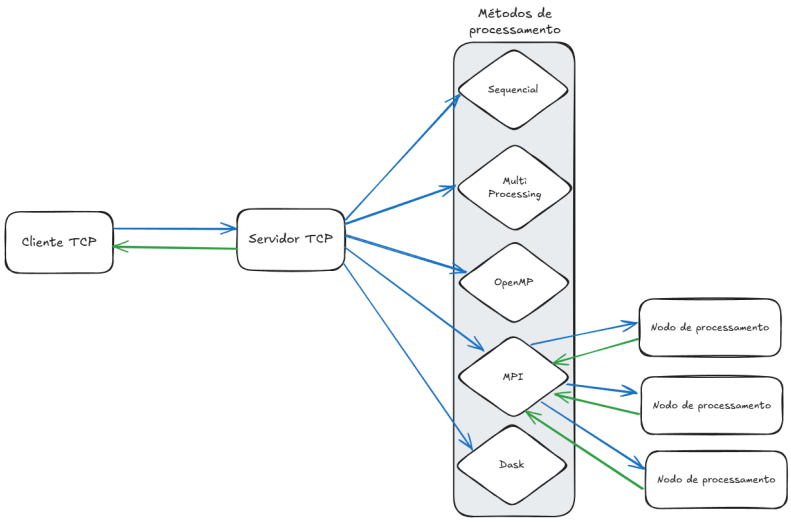

Figura 1. Diagrama de funcionamento da aplicação. Fonte: os autores.

desempenho, chegando a um speedup de 1,24 (3,03s) quando utilizadas 16 threads. A paralelização utilizando MPI apresentou inconsistência no desempenho, apresentando ganhos apenas nos cenários com quatro ou mais threads. Esse comportamento inesperado, principalmente no cenário de duas threads, requer uma análise mais aprofundada, porém pode estar relacionado ao fato de se utilizar o Python, uma linguagem interpretada.

O paralelismo utilizando a biblioteca multiprocessing trouxe um speedup máximo de 2,77 (9,08s) se comparado à execução sequencial em Python. Finalmente, o paralelismo usando Dash apresentou o maior speedup, atingindo 5,16 (4,88s) no cenário com 16 threads. Nele, o tempo de execução aproximou-se dos cenários avaliados em linguagem C, algo também inesperado e que requer uma melhor análise. Entre as possíveis explicações para tal desempenho pode estar relacionado ao fato desta biblioteca ser voltada para o processamento de grandes volumes de dados, conforme relata a documentação oficial.

5. Conclusões

A partir dos resultados obtidos, conclui-se que a paralelização do processamento de dados pode proporcionar uma melhoria significativa no tempo de execução. Entre os métodos avaliados, o Dask se destacou como o mais eficiente, oferecendo o melhor desempenho, especialmente com o aumento do número de processos/threads.

Além disso, as soluções propostas otimizam o monitoramento de cidades inteligentes no contexto de grandes volumes de dados IoT, integrando aplicações de feedback em tempo real e estratégias de manutenção preventiva, o que resulta em uma gestão urbana mais eficiente e resiliente.

Em contrapartida, o OpenMP não apresentou melhora significativa no tempo de processamento, exibindo apenas pequenas variações no tempo de execução com o aumento do número de threads.

Como trabalhos futuros, sugere-se analisar os motivos que levaram o MPI a reduzir o desempenho com o uso de dois processos, bem como avaliar os motivos pelo

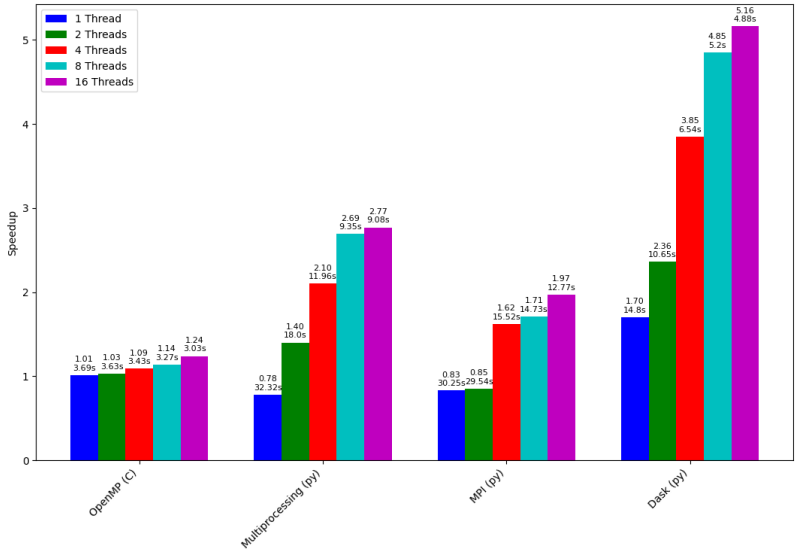

Figura 2. Speedup obtido nos diferentes cenários de paralelização, tendo como referências a execução sequencial em C (3,76s) para o OpenMP e a execução sequencial em Python (25,22s) para MPI, Multithreading e Dash. Fonte: os autores.

qual o Dask apresentou melhora significativa no speedup. Além disso, seria interessante apresentar uma solução utilizando unidades de processamento gráfico.

Referências

[Bekkai et al. 2021] Bekkai, B., Bendjenna, H., and Kitouni, I. (2021). Internet of things: A recent survey. In *2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI)*, page 1–9, Tebessa, Algeria. IEEE.

[Carleton 2024] Carleton (2024). Introduction to openmp. Disponível em: <https://carleton.ca/rcs/rcdc/introduction-to-openmp/>. Acesso em: 24 nov. 2024.

[Carvalho et al. 2024] Carvalho, M. D. S. d., Ferrigo, S. F., Dal Bó, G., Fachinelli, A. C., Perini, R. d. L., and Mosche, S. d. A. (2024). Iot para cidades inteligentes: habilitação tecnológica em uma cidade da serra gaúcha. Disponível em: <https://eventos.anpad.org.br/pt_br/event/details/131/2019view>. Acesso em: 20.jan.2025.

[Dask 2024] Dask (2024). Dask: Scalable parallel computing in python. Disponível em: <https://docs.dask.org/en/stable/>. Acesso em: 24 nov. 2024.

[MPI 2024] MPI (2024). Mpi 4.1: The complete mpi standard. Disponível em: <https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf>. Acesso em: 24 nov. 2024.

[Python 2024] Python (2024). multiprocessing — process-based parallelism. Disponível em: <https://docs.python.org/3/library/multiprocessing.html/>.Acesso em: 24 nov. 2024.