Relatório de Projeto: Comparação de Modelos de Paralelismo em Processamento de Dados Meteorológicos

Luís Felipe de Abreu Marciano

7 de Julho de 2025

Sumário

1 Introdução **1**

2 Descrição das Métricas Computadas **1**
2.1 Percentual de Anomalias por Sensor 1
2.2 Média Móvel Regional 1
2.3 Períodos de Coocorrência de Anomalias 2

3 Modelagem da Arquitetura das Soluções **2**
3.1 Abordagem A: Multiprocessamento Local 2
3.2 Abordagem B: Message Broker com Workers 2
3.3 Abordagem C: Processamento Distribuído com Apache Spark 2

4 Experimentos e Resultados **3**
4.1 Experimento 1: Carga de Dados Leve 3
4.2 Experimento 2: Carga de Dados Elevada 4

5 Discussão dos Resultados **6**
5.1 Análise de Desempenho 6
5.2 Vantagens e Desvantagens de Cada Abordagem 6
5.2.1 Abordagem A: Multiprocessamento Local 6
5.2.2 Abordagem B: Message Broker 7
5.2.3 Abordagem C: Apache Spark 7
5.3 Conclusão Final 7

1 Introdução

Este relatório detalha um experimento comparativo entre diferentes modelos de paralelismo aplicados a um problema de processamento de dados meteorológicos. O contexto envolve uma rede fictícia, a ClimaData, que coleta grandes volumes de dados de temperatura, umidade e pressão de múltiplas estações.

O objetivo principal é avaliar três abordagens de processamento paralelo para computar métricas diárias a partir desses dados: (1) multiprocessamento local, (2) um sistema baseado em message broker e (3) processamento distribuído com Apache Spark. O desempenho de cada solução foi medido em diferentes cenários de carga e graus de paralelismo, com os resultados e análises apresentados nas seções seguintes. As medições foram feitas em minha máquina pessoal, por isso sugiro fortemente que faça seus próprios experimentos acessando o repositório (https://github.com/lfamarciano/ModelosDeParalelismo) e seguindo as instruções.

2 Descrição das Métricas Computadas

O sistema foi encarregado de processar os dados brutos para extrair três métricas de alto valor, conforme descrito abaixo.

2.1 Percentual de Anomalias por Sensor

Esta métrica visa identificar a taxa de falha ou leituras atípicas de cada sensor em cada estação meteorológica.

- **Cálculo:** Para cada sensor (temperatura, umidade e pressão), foram definidos limiares (thresholds) de valores considerados normais. Uma leitura é classificada como anômala se estiver fora desse intervalo. O percentual é então calculado pela fórmula:

$$\%Anomalias = \left(\frac{NdeLeiturasAnômalas}{NTotaldeLeituras} \right) \times 100$$

Como pode ser observado, o método de detecção de anomalias é simples, visto que não é um dos objetivos do trabalho atingir um alto nível de eficiência na detecção de anomalias.

2.2 Média Móvel Regional

O objetivo desta métrica é suavizar os dados dos sensores ao longo do tempo para entender tendências regionais, desconsiderando os ruídos causados por anomalias.

- **Cálculo:** Primeiramente, todos os registros que continham qualquer valor de sensor classificado como anômalo foram excluídos do conjunto de dados. Em seguida, para cada região geográfica, os dados foram agrupados em janelas de tempo deslizantes de 10 minutos, e a média de cada sensor foi calculada para cada janela.

2.3 Períodos de Coocorrência de Anomalias

Esta métrica é projetada para detectar momentos em que uma estação meteorológica apresenta falhas sistêmicas, indicadas por anomalias em múltiplos sensores simultaneamente.

- **Cálculo:** Para cada estação, o sistema conta o número de janelas de tempo distintas de 10 minutos nas quais anomalias foram detectadas em pelo menos dois sensores diferentes (temperatura, umidade ou pressão).

3 Modelagem da Arquitetura das Soluções

Para atender aos requisitos, foram projetadas e implementadas três arquiteturas distintas, cada uma otimizada para um modelo de paralelismo específico. Todas as soluções foram containerizadas usando Docker para garantir isolamento e reprodutibilidade.

3.1 Abordagem A: Multiprocessamento Local

Esta solução utiliza o paralelismo de memória compartilhada, confinado a uma única máquina.

- **Tecnologia:** Módulo `multiprocessing` do Python.
- **Arquitetura:** Um script principal orquestra o processo. Ele lê o conjunto de dados completo para a memória e, em seguida, divide o trabalho (por estação ou por região) entre um pool de processos workers. Cada worker executa uma subtarefa (ex: calcular a média móvel para uma região). Os resultados são agregados de forma segura em uma estrutura de dados compartilhada, gerenciada pelo `multiprocessing.Manager`.

3.2 Abordagem B: Message Broker com Workers

Esta arquitetura implementa o padrão produtor-consumidor, ideal para sistemas distribuídos e assíncronos.

- **Tecnologias:** Celery como framework de tarefas, RabbitMQ como message broker.
- **Arquitetura:** Um script "Produtor"lê o conjunto de dados, divide-o em pequenas tarefas (ex: "calcular anomalias para a estação X"ou "calcular média móvel da região Y) e as envia como mensagens para uma fila no RabbitMQ. Múltiplos "Workers"(containers Celery independentes) consomem as tarefas dessa fila em paralelo. Cada worker executa sua tarefa e envia o resultado de volta através de um backend de resultados. O produtor aguarda a conclusão de todas as tarefas para finalizar o processo.

3.3 Abordagem C: Processamento Distribuído com Apache Spark

Esta solução utiliza um framework, padrão da indústria, que é conhecido por ser super eficiente para processamento de grandes cargas de dados, projetado para processamento em cluster.

- **Tecnologia:** Apache Spark, através da sua API Python (PySpark).
- **Arquitetura:** Uma única aplicação Spark é submetida. O Spark lê o dataset e o distribui em partições na memória de um cluster (neste experimento, simulado em uma única máquina, variando o número de cores disponíveis para o spark utilizar). As transformações (filtros, agregações) são executadas em paralelo nessas partições pelos "executors". Um nó "driver"coordena todo o fluxo de trabalho.

4 Experimentos e Resultados

Como eu já disse, eu encorajo para que você faça seus próprios experimentos em sua própria máquina para observar os resultados. As cargas foram escolhidas de forma que minha máquina pessoal conseguisse rodar sem encher a memória. Foram feitos dois experimentos principais para avaliar o desempenho das arquiteturas sob diferentes cargas de dados. Em ambos, o grau de paralelismo (unidades de processamento) variou entre 1, 2, 4 e 8 (8 é a quantidade de cores da minha máquina pessoal). A corretude de cada execução foi validada para garantir que todas as abordagens produzissem o mesmo número de anomalias detectadas.

4.1 Experimento 1: Carga de Dados Leve

- **Parâmetros:** 12 estações, 10.000 eventos por estação (Total: 120.000 eventos).

Tabela 1: Resultados do Experimento 1 (120.000 eventos).

Abordagem	**Paralelismo**	**Tempo (segundos)**	**Anomalias OK**
local-processing	1	6.19	1209
message-broker	1	5.17	1209
spark-processing	1	8.17	1209
local-processing	2	3.16	1209
message-broker	2	3.27	1209
spark-processing	2	6.26	1209
local-processing	4	1.92	1209
message-broker	4	2.79	1209
spark-processing	4	6.39	1209
local-processing	8	0.86	1209
message-broker	8	3.19	1209
spark-processing	8	6.13	1209

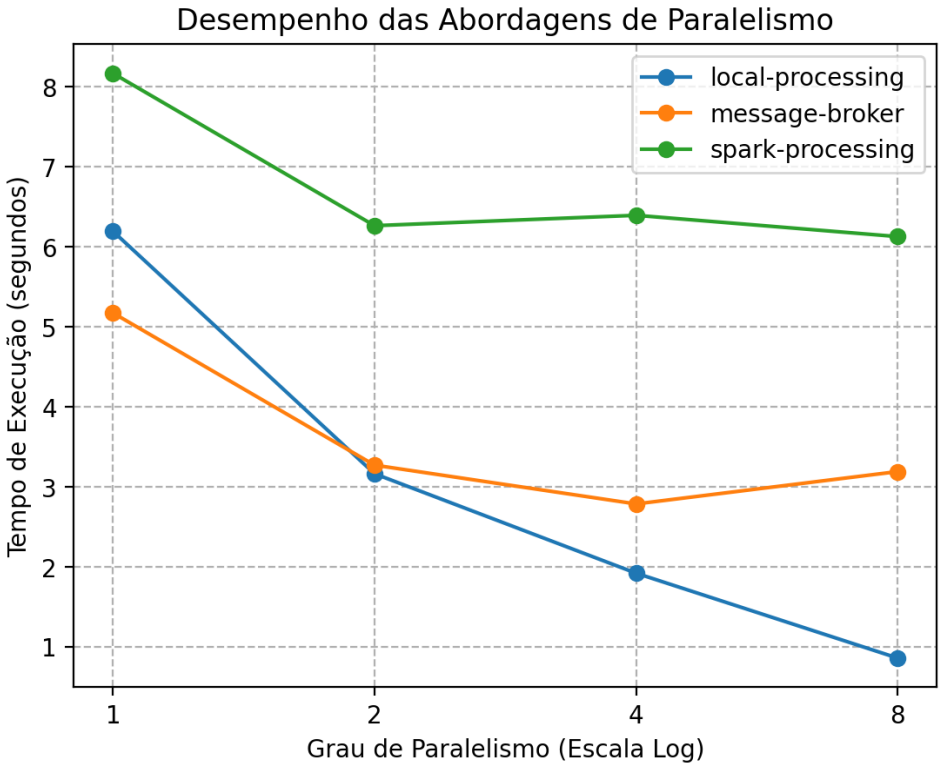

Figura 1: Gráfico de desempenho para o Experimento 1.

4.2 Experimento 2: Carga de Dados Elevada

- **Parâmetros:** 20 estações, 30.000 eventos por estação (Total: 600.000 eventos).

Tabela 2: Resultados do Experimento 2 (600.000 eventos).

Abordagem	Paralelismo	Tempo (segundos)	Anomalias OK
local-processing	1	9.57	5976
message-broker	1	30.23	5976
spark-processing	1	10.81	5976
local-processing	2	5.22	5976
message-broker	2	16.08	5976
spark-processing	2	8.70	5976
local-processing	4	3.33	5976
message-broker	4	16.04	5976
spark-processing	4	8.02	5976
local-processing	8	1.84	5976
message-broker	8	17.91	5976
spark-processing	8	8.65	5976

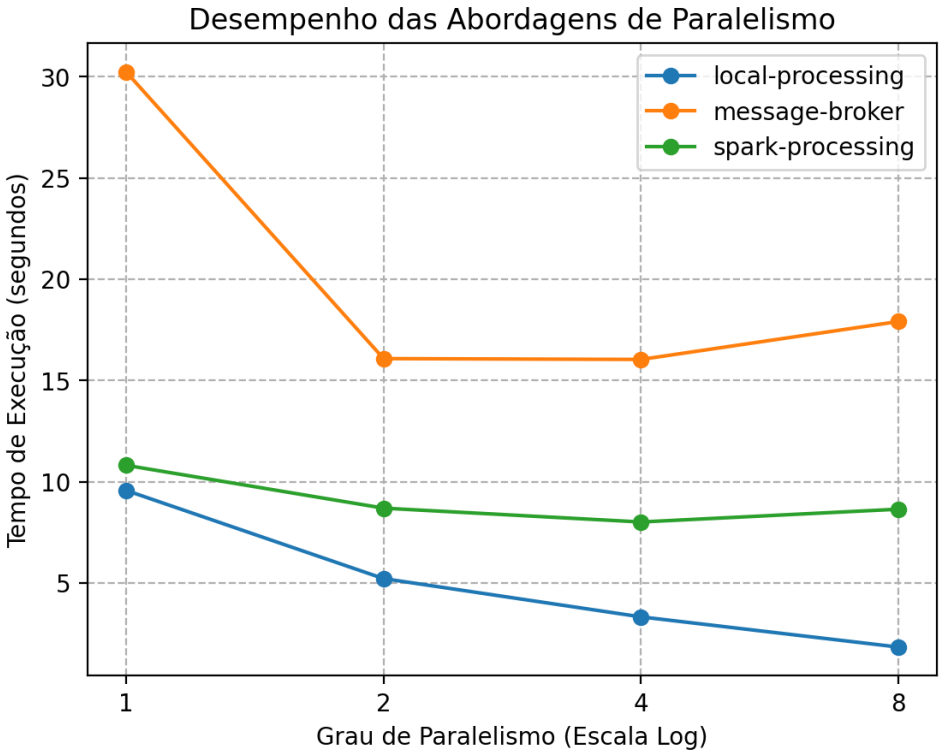

Figura 2: Gráfico de desempenho para o Experimento 2.

5 Discussão dos Resultados

A análise dos resultados dos experimentos revela nuances importantes sobre a aplicabilidade e eficiência de cada modelo de paralelismo.

5.1 Análise de Desempenho

- **Multiprocessamento Local:** Esta abordagem demonstrou o melhor desempenho e escalabilidade nos cenários testados. O tempo de execução diminuiu consistentemente com o aumento do grau de paralelismo, chegando a ser o mais rápido em todos os casos com 8 unidades de processamento. Isso se deve ao baixo custo de comunicação (a memória é compartilhada) e à simplicidade na criação de novos processos.

- **Message Broker:** O comportamento desta abordagem foi o mais complexo. No Experimento 1 (com carga leve), ela escalou bem até 4 workers, mas o desempenho piorou com 8. No Experimento 2 (carga elevada), o desempenho foi significativamente pior que as outras abordagens e não melhorou com o aumento do paralelismo. Acredito que isso tenha ocorrido porque, em uma única máquina, o "custo"do paralelismo (serialização de dados para JSON, tráfego de rede no broker, contenção de CPU e memória entre os workers) superou o benefício da divisão do trabalho. Talvez em uma máquina melhor ele tenha resultados melhores também conforme o grau de paralelismo aumente.

- **Apache Spark:** O Spark apresentou o tempo de execução mais consistente, porém o mais lento na maioria dos cenários de baixo paralelismo. Seu desempenho quase não variou com o aumento das unidades de processamento. Isso se deve ao alto custo fixo de inicialização da *Spark Session* e do seu complexo mecanismo de agendamento de tarefas. O Spark é projetado para "Big Data", e os volumes de dados dos experimentos não foram grandes o suficiente para que os benefícios de seu motor de processamento distribuído superassem seu overhead inicial.

5.2 Vantagens e Desvantagens de Cada Abordagem

5.2.1 Abordagem A: Multiprocessamento Local

- **Vantagens:**
  - **Desempenho Superior em Máquina Única:** Baixíssimo overhead de comunicação.
  - **Simplicidade:** Fácil de implementar e depurar usando bibliotecas padrão do Python.
- **Desvantagens:**
  - **Escalabilidade Vertical:** Limitado aos recursos (CPU, RAM) de uma única máquina. Não escala para clusters.
  - **Gerenciamento de Estado:** Compartilhar estado entre processos pode ser complexo e propenso a erros (race conditions).

5.2.2 Abordagem B: Message Broker

- **Vantagens:**
  - **Escalabilidade Horizontal:** Arquitetura ideal para distribuir trabalho entre múltiplas máquinas, bastando adicionar mais containers workers em outros nós.
  - **Desacoplamento e Resiliência:** O produtor e os consumidores são independentes. Se um worker falhar, o broker pode redistribuir a tarefa para outro.
- **Desvantagens:**
  - **Complexidade de Infraestrutura:** Requer a configuração e manutenção de mais componentes (broker, workers, rede).
  - **Overhead de Comunicação:** A serialização e o tráfego de mensagens pela rede introduzem uma latência significativa, que pode ser o gargalo em cenários de máquina única.

5.2.3 Abordagem C: Apache Spark

- **Vantagens:**
  - **Padrão para Big Data:** A melhor solução para processar datasets que não cabem na memória de uma única máquina.
  - **Tolerância a Falhas:** O Spark gerencia nativamente a falha de nós no cluster, recomputando partições de dados perdidas.
  - **API de Alto Nível:** Permite expressar transformações complexas de forma concisa (estilo SQL).
- **Desvantagens:**
  - **Alto Overhead de Inicialização:** Lento para tarefas pequenas devido ao tempo de setup da sessão e do plano de execução.
  - **Consumo de Recursos:** Exige uma quantidade considerável de memória RAM para funcionar de forma eficiente.

5.3 Conclusão Final

O experimento demonstrou que não existe uma única solução em computação paralela. A escolha da arquitetura ideal depende diretamente da escala do problema.

Para o volume de dados apresentado, o multiprocessamento local foi a abordagem vencedora, oferecendo a melhor relação entre simplicidade e desempenho.

No entanto, se o sistema ClimaData precisasse escalar para processar terabytes de dados distribuídos em um cluster de máquinas, a arquitetura de message broker seria ideal para tarefas em tempo real ou independentes, enquanto o Apache Spark seria a escolha correta para análises em lote (batch) de todo o conjunto de dados. O projeto, portanto, validou empiricamente os trade-offs teóricos de cada modelo de paralelismo.