Projeto de Micro-Framework

A1 Computação Escalável

FGV EMAp

Ana Júlia Amaro Pereira Rocha
Maria Eduarda Mesquita Magalhães
Mariana Fernandes Rocha
Paula Eduarda de Lima

Ciência de Dados e Inteligência Artificial
5º Período

Rio de Janeiro, 2025

Conteúdo

1 Introdução

Este relatório apresenta o desenvolvimento de um Micro-Framework voltado à construção de pipelines para processamento de dados. O principal objetivo é fornecer uma solução que promova maior eficiência e balanceamento de carga, por meio da execução concorrente e paralela das etapas de processamento. Embora a aplicação tenha sido especialmente construida para o monitoramento de uma doença infecciosa, sua estrutura foi projetada de forma genérica, com pontos de extensão (hot-spots) que permitem a personalização conforme as necessidades específicas de diferentes cenários. O desenvolvimento do projeto pode ser encontrado nesse **repositório GitHub**.

2 Modelagem

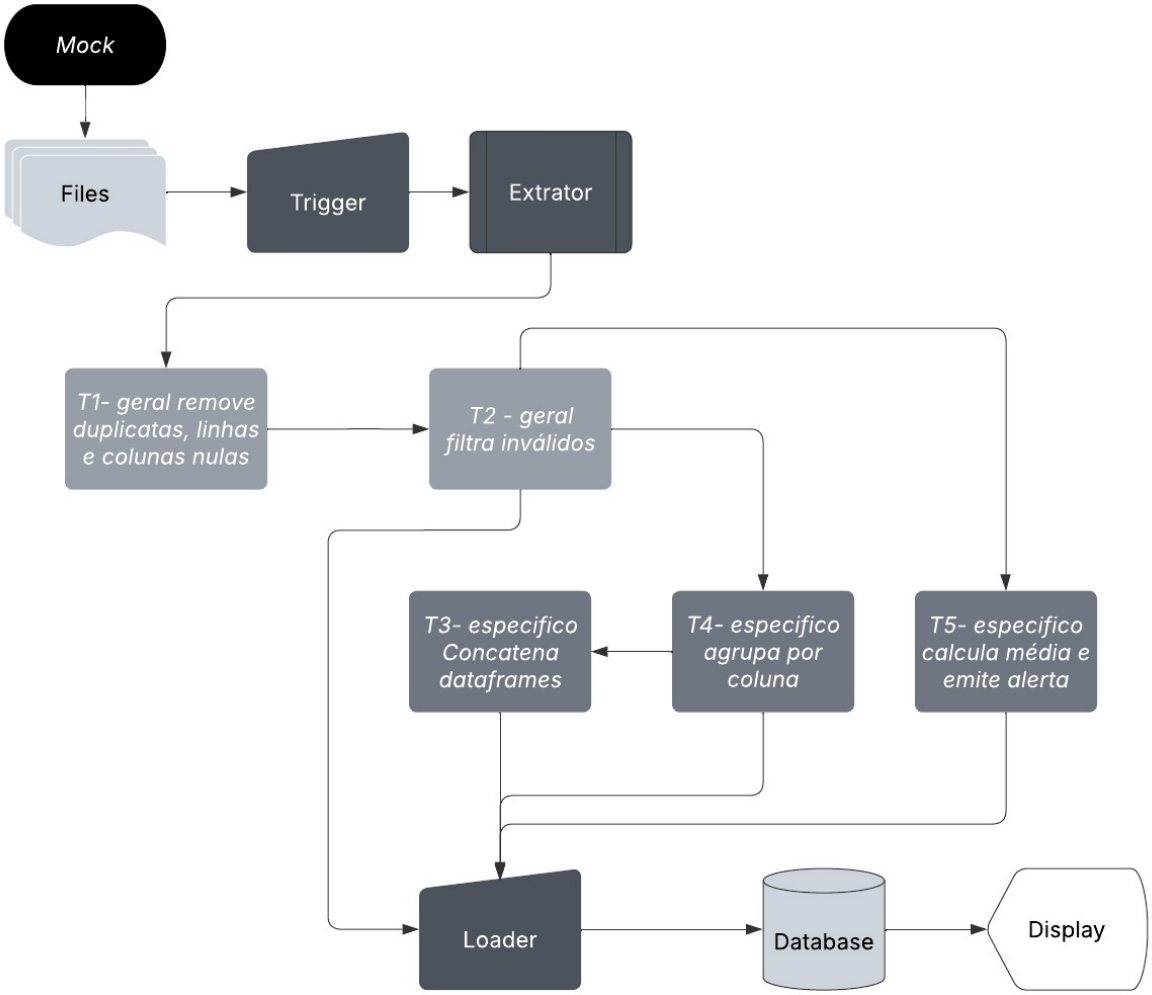

Figura 1: Fluxograma do framework

A modelagem, representada no fluxograma, é feita de um pipeline de processamento de dados baseado em um fluxo ETL (Extract, Transform, Load), acionado por diferentes triggers, organizado em etapas independentes, que se comunicam por meio de filas. O processo se inicia por um módulo mock que simula a chegada de dados reais de fontes distintas. Esses dados são armazenados em arquivos e

enviados para a primeira etapa do pipeline.

Quando o pipeline é ativado por um dos triggers, os arquivos são encaminhados para o extrator, que é responsável por identificar o tipo de arquivo e padronizar os dados extraídos em uma estrutura de dataframe. Esse processo garante que, independentemente do formato de origem, todos os dados estejam organizados de maneira uniforme para as próximas etapas do pipeline.

Após a extração, os dados passam por uma cadeia de tratadores gerais, que realizam limpezas e filtragens padronizadas, T1, T2 - geral.

Na sequência, os dados seguem para tratadores específicos, que aplicam transformações mais contextualizadas. O T3 - específico concatena diferentes DataFrames, unificando tabelas com a mesma estrutura. O T4 - específico realiza agrupamentos por colunas de interesse (como CEP ou hospital), preparando os dados para análises agregadas. Por fim, o T5 - específico calcula médias e emite um alerta, oferecendo uma visão resumida dos dados processados.

Após o tratamento, os dados são encaminhados para o Loader, responsável por carregá-los em arquivos csv's. Isso garante que os dados tratados fiquem disponíveis para consultas futuras e possam ser utilizados por módulos de visualização. Por fim, o componente Display consome essas informações armazenadas e as exibe relatórios semanais, fechando o ciclo do pipeline.

3 Mock

As fontes de dados serão simulações de hospitais, Secretaria de Saúde (SS) e Organização Mundial da Saúde (OMS). Receberemos conjuntos de dados advindos dessas fontes semanalmente. Além disso, temos dois tipos de segmentação regional: Ilhas, mais geral, e regiões, que são segmentações das ilhas, mais especifíco. A simulação apresenta 20 ilhas, cada uma identificada por um CEP de 2 dígitos, cada ilha tem 5 regiões, com CEPs de 5 dígitos, sendo os dois primeiros dígitos o CEP da ilha (ex: ilha 12 $\rightarrow$ regiões 12001 a 12005).

O objeto Mock é feito em python e implementa a criação de três tipos de tabelas diferes: SQLite, csv e txt.

Fontes e dados gerados

- **OMS (`oms_mock.txt`)**
  Dados agregados por ilha (CEP de 2 dígitos).
  Inclui número de óbitos, população, recuperados, vacinados e data.
  Gerado em formato `.txt` com tabulações.

- **Hospitais (`hospital_mock_*.csv`)**
  Dados individuais por paciente, com CEPs de 5 dígitos (regiões).

Contém informações como internação, idade, sexo, sintomas e data.
Gera múltiplos arquivos `.csv`, simulando diferentes hospitais.

- **Secretaria de Saúde (`secretary_data.db`)**
  Banco de dados SQLite com tabela `pacientes`.
  Dados por região (CEP de 5 dígitos): diagnóstico, vacinação, escolaridade, população e data.
  Registros inseridos diretamente em um banco relacional.

4 Triggers

O framework apresenta dois tipos de Triggers responsáveis por iniciar a execução de um pipeline:

- **TimerTrigger**: A cada X segundos dispara uma nova execução.
- **RequestTrigger**: A cada chamada de função (simulando uma requisição de rede) dispara uma nova execução.

5 Extrator

O extrator, implementado em C++, usa a classe `DataFrame` que organiza os dados gerados pelo mock em uma estrutura tabular com colunas bem definidas e suporta os diferentes tipos de dados (inteiros, reais e strings). Ele permite:

- **Validar os dados inseridos:** garante que cada valor esteja no formato correto (ex: inteiros em colunas inteiras).
- **Adicionar ou remover linhas e colunas:** facilita manipulações e filtragens nos dados carregados.
- **Visualizar os dados em forma de tabela:** com o método `display`, os dados são impressos com cabeçalho e valores formatados.
- **Acessar partes específicas:** como uma linha (`getRow`) ou coluna (`colIdx` e `typeCol`).
- **Verificar informações básicas:** número de colunas (`numCols`), número de linhas (`size`), se está vazio (`empty`), etc.

Aplicação com os dados gerados

O extrator identifica o tipo de arquivo da tabela em questão por meio da extensão pós ponto (.csv,.txt ou .db) e, com isso, chama a respectiva função de extração para esse tipo de dado, tornando os diferentes tipos de tabelas padronizadas para o uso dos tratadores.

Quando os dados da OMS (arquivo `.txt`) ou dos hospitais (arquivos `.csv`) são carregados, o `DataFrame` pode ser inicializado com os nomes das colunas e seus tipos, e cada linha é validada antes de ser inserida.

O extrator também pode ser usado para representar tabelas extraídas do banco de dados SQLite da Secretaria de Saúde (por exemplo, a tabela `pacientes`).

Ele garante integridade dos dados e fornece uma base comum para as análises posteriores.

6 Tratadores

Os tratadores implementados em C++ nesse projeto são responsáveis por preparar os dados para análises. Foram implementados 5 tratadores, sendo 2 gerais (todos os arquivos passam por eles) e 3 específicos:

- **T1 - geral:** O tratador 1 é responsável por eliminar duplicatas, além de colunas e linhas majoritariamente nulas. Ele recebe um dataframe vindo do extrator e retorna um dataframe para o tratador 2.
- **T2 - geral:** O tratador 2 é responsável por validar os dados do dataframe, isto é, eliminar linhas com idade negativa, números de CEP com menos ou mais dígitos do que deveria, número total de óbitos negativo, entre outros. Este tratador também tem grau de entrada e saída 1.
- **T3 - específico:** O tratador 3 tem grau de entrada 3 e grau de saída 4. Ele recebe 3 dataframes com duas colunas (vindos do T4 que é detalhado a seguir) e faz o merge dos três dataframes e de suas permutações também, isto é, considerando A, B e C os DFs que entram, temos que serão retornados os DFs AB, AC, BC e ABC. Tudo isso acontece por meio de uma coluna em comum a qual no escopo deste projeto é aquela referente ao CEP (única coluna que se repete em todos os DFs, ainda que em alguns seja CEP da ilha e em outros o CEP de uma região de uma ilha, mas essas diferenças foram tratadas observando apenas os 2 primeiros dígitos os quais se referem à ilha).

- **T4 - específico:** O tratador 4 recebe um Dataframe limpo do T2, uma coluna de agregação e uma coluna de agrupamento. Basicamente, é retornado um Dataframe com as duas colunas especificadas, sendo que a coluna de agregação terá seus dados agrupados de acordo com a outra coluna mencionada. Por exemplo, se a ideia é descobrir o número de casos por região, o T4 retorna um DF com a coluna referente ao número de casos e a coluna referente ao CEP da região que possuirá apenas valores únicos com o agrupamento. O DF de saída pode ir para o T3 ou direto para o Loader.
- **T5 - específico:** O tratador 5 recebe um DF limpo do T2 e é responsável por calcular a média dos dados em uma determinada coluna, criando uma coluna de alerta para indicar quais linhas estão acima da média. Por exemplo, se a ideia é identificar regiões com surtos de epidemia pode-se calcular a média do número de internados e gerar um alerta vermelho para os hospitais e/ou regiões que estejam com mais internações que o "normal"pela doença infecciosa a ser monitorada.

7 Loader

O loader tem uma função bastante simples que é enviar os dataframes tratados para o banco de dados fazendo com que os dados estejam disponíveis de forma mais confiável do que quando chegaram ao pipeline para futuros acessos e análises. Portanto, neste projeto, o loader recebe os dataframes tratados, transforma-os em CSV's e salva no banco de dados que no caso é apenas uma pasta chamada de database loader. A partir disso, pode-se fazer as análises a serem exibidas no dashboard para monitoramento da doença infecciosa em questão no projeto.

8 Dashboard

O dashboard é composto por cinco análises principais, cada uma com foco em aspectos distintos da saúde pública nas ilhas monitoradas. Temos 5 análises:

- **Análise 1: Alerta semanal por CEP**
  Esta análise processa os arquivos já tratados da OMS contendo alertas por CEP. Um alerta vermelho é emitido para os CEPs que apresentaram cinco ou mais ocorrências de alerta `True` ao longo da semana, enquanto os demais são classificados com alerta verde. O objetivo é sinalizar áreas críticas de forma direta e sumarizada.
- **Análise 2: Estatísticas globais de internações**
  Aqui, agregamos os dados de internação de todos os hospitais e calculamos

estatísticas gerais como média e desvio padrão do número total de internados. O processamento é realizado de forma paralela utilizando múltiplas threads para ganho de desempenho.

- **Análise 3: Estatísticas por hospital**
  Similar à Análise 2, mas os cálculos são feitos individualmente por hospital. Para cada hospital, são reportadas a média e o desvio padrão dos internados. Essa análise permite identificar hospitais com padrões anormais em relação ao restante.

- **Análise 4: Correlação entre vacinação e óbitos**
  Aqui, calculamos a correlação de Pearson entre o total de vacinados e o número de óbitos registrados, utilizando dados de um arquivo resultando do merge. O objetivo é avaliar se há relação linear entre o avanço da vacinação e a mortalidade.

- **Análise 5: Regressão linear entre vacinação e internações**
  Nesta análise fizemos uma regressão linear simples para estimar o número de internados com base na quantidade de vacinados. A intenção é investigar se o aumento na vacinação influencia a redução ou aumento nas internações.

Exemplos na prática

```
Modo timer: executando a cada 20 segundos...

=== Análise de Tempo por Estágio ===
1. Extração:    1.53467 segundos
2. Tratamento: 2.40603 segundos
3. Loader:      0.0044931 segundos
4. Merge:      3.88628 segundos
---------------------------------
Tempo Total:   8.34087 segundos

========== DASHBOARD ILHAS ==========

>> Análise 1: Alertas semanais por CEP
ALERTA DA SEMANA
   CEP de ilhas de alerta vermelho: 13, 16, 17, 21, 25, 26, 27
   CEP de ilhas de alerta verde: 11, 14, 15, 18, 19, 20, 22, 23, 24, 28, 29, 30

>> Análise 2: Estatísticas gerais dos hospitais

=== Estatísticas Hospitalares ===
Total de registros: 55
Média de internados: 6407.51
Desvio padrão: 494.827
================================
```

```
>> Análise 3: Estatísticas individuais por hospital

=== Estatísticas por Hospital ===
Hospital ID: 1
   Média de internados: 6456.18
   Desvio padrão: 517.533
Hospital ID: 2
   Média de internados: 6445.91
   Desvio padrão: 559.876
Hospital ID: 3
   Média de internados: 6450.82
   Desvio padrão: 572.393
Hospital ID: 4
   Média de internados: 6448.91
   Desvio padrão: 517.539
Hospital ID: 5
   Média de internados: 6477.64
   Desvio padrão: 450.181
================================

>> Análise 4: Correlação entre vacinação e internação
Correlação de Pearson (Total_Nº óbitos vs Total_Vacinado_C): -0.418377

   Desvio padrão: 517.539
Hospital ID: 5
   Média de internados: 6477.64
   Desvio padrão: 450.181
================================

>> Análise 4: Correlação entre vacinação e internação
Correlação de Pearson (Total_Nº óbitos vs Total_Vacinado_C): -0.418377

>> Análise 5: Regressão Linear para estimar o número de internados com base na quantidade de vacinados.
Regressão Linear (Total_Internado ~ Total_Vacinado_C):
  β0 (intercepto): 761.702
  β1 (coeficiente): 0.428674

=========== FIM DO DASHBOARD ===========
```

9 Pipeline

O Pipeline é feito de modo a organizar o fluxo de trabalho de maneira eficiente e paralela. O objetivo principal é o passo a passo desejado do ETL e a escolha da ordem e dos diferentes tratadores desejados. Além disso,essa arquitetura pode ser adaptada para qualquer outro contexto de processamento de dados.

Lógica Produtor-Consumidor

De maneira geral, o paralelismo no pipeline se dá pela lógica de Produtor-Consumidor, da seguinte forma:

Produtor: gera dados e os insere em uma fila compartilhada.

Consumidor: retira dados dessa fila para processá-los.

Fila compartilhada: estrutura de dados para comunicação entre threads.

Mutex (trava): garante que só uma thread acesse a fila por vez.

Condition Variable: notifica consumidores quando há dados disponíveis (ou produtores quando há espaço, se necessário).

Extração

O paralelismo na extração dos arquivos gerados pelo mock no pipeline é implementado com base no modelo produtor-consumidor utilizando threads, filas protegidas por mutex, e variáveis de condição para sincronização.

- Produtor coloca arquivos na fila (filaArquivos).
- Múltiplos consumidores-extratores (threads) consomem essa fila paralelamente (o número de consumidores é um argumento da função pipeline, definido como preferível).
- Cada consumidor-extrator:
  - Lê o arquivo (pode ser .csv, .txt, ou .db).
  - Constrói um DataFrame.
  - Envia esse DataFrame para a próxima etapa (extratorTratadorFila), que será tratada por outra thread.

Múltiplas threads `consumidorExtrator` são criadas (na função `executarPipeline`).

Cada thread opera de forma independente, competindo por arquivos na fila `filaArquivos`, gerando uma condição de corrida, corrigida por uma proteção da fila com mutex.

Isso permite que vários arquivos sejam processados **simultaneamente** por diferentes núcleos da CPU e haja um balanceamento de carga.

Pontos fortes da abordagem

- **Escalabilidade com threads:** permite processar múltiplos arquivos simultaneamente, aproveitando melhor os múltiplos núcleos da CPU, o que acelera a etapa de extração.
- **Balanceamento automático de carga:** como os consumidores competem por arquivos na fila, a distribuição é feita de forma natural e eficiente, sem divisão manual de tarefas.

- **Desacoplamento entre etapas:** o uso de filas entre as fases (extração, tratamento e carregamento) permite que cada parte do pipeline opere de forma independente, evitando gargalos.
- **Modelo confiável:** o padrão produtor-consumidor com mutex e variáveis de condição é amplamente utilizado e considerado seguro para aplicações concorrentes.
- **Paralelismo configurável:** o número de consumidores pode ser definido por parâmetro na execução, facilitando testes e adaptações ao ambiente de hardware.
- **Sincronização segura:** uso correto de `mutex` e `condition_variable` impede condições de corrida no acesso às filas compartilhadas.

Pontos fracos da abordagem

- **Overhead de gerenciamento de threads:** quando há muitos arquivos pequenos, o custo de criação e sincronização de threads pode superar os ganhos do paralelismo.
- **Ausência de priorização de arquivos:** a fila segue uma lógica FIFO simples, não considerando arquivos urgentes ou mais pesados que poderiam ser tratados primeiro.
- **Fila sem limite de tamanho:** caso a leitura seja muito mais rápida que o tratamento, a fila intermediária pode crescer indefinidamente e consumir muita memória.
- **Threads ociosas em cenários com poucos arquivos:** se a carga for baixa, as threads de extração encerram rapidamente e não são reaproveitadas.
- **Bloqueio com variáveis de condição:** consumidores ficam bloqueados esperando arquivos, o que pode causar ineficiência em cenários com picos esparsos de chegada de dados.

Tratamento

Na etapa de tratamento, a lógica produtor-consumidor se organiza da seguinte forma:

- **Produtor:** as threads consumidoras da etapa de extração enviam os DataFrames processados para a fila extratorTratadorFila, que serve como ponte para o tratamento.
- **Consumidores:** múltiplas threads tratadoras (criadas na função executarPipeline) consomem os DataFrames dessa fila. Além de ter vários agentes consumidores efetuando o tratamento dos arquivos, assim sendo paralelos entre os arquivos, as funções meanAlert, groupedDf e dataCleaner são paralelas nas linhas também, logo um agente consumidor de tratador se divide dentro de si mesmo para manipular várias partes do arquivos ao mesmo tempo, essa divisão é feita dividindo o número de threads para linhas e a quantidade de linhas para garantir o balanceamento.
- Cada consumidor-tratador:
  - Executa o tratamento dos dados conforme a lógica definida (e.g., limpeza, transformação).
  - Insere os dados tratados na fila tratadorLoaderFila, que será consumida pela próxima etapa.

Pontos fortes específicos da abordagem

- Além da escalabilidade, balanceamento e desacoplamento detalhados entre os pontos fortes do paralelismo na extração.
- Escalabilidade tanto horizontal quanto vertical, horizontalmente quando mais arquivos forem processados simultaneamente. Verticalmente, se os arquivos ficarem maiores (mais linhas), pois o processamento interno também se paraleliza.

Pontos fracos específicos da abordagem

- Em máquinas com poucos núcleos, o desempenho pode piorar com mais threads, devido à saturação do CPU e do barramento de memória.
- A lógica aninhada de paralelismo complica o controle de sincronização, concorrência e debug.

Loader

Na etapa de carregamento (loader), o paralelismo também segue a lógica produtor-consumidor:

- **Produtor:** as threads de tratamento inserem os dados tratados na fila tratadorLoaderFila.
- **Consumidores:** múltiplas threads loader consomem essa fila para enviar os dados para o destino final (e.g., banco de dados).
- Cada thread loader:
  - Lê o item da fila.
  - Realiza a operação de carga dos dados.

Pontos fortes específicos da abordagem

- **Aproveitamento de I/O paralelamente:** carregar dados em paralelo pode reduzir o tempo total quando o destino (como um banco) suporta múltiplas conexões simultâneas.

Além da escalabilidade, balanceamento e desacoplamento detalhados entre os pontos fortes do paralelismo na extração.

Pontos fracos específicos da abordagem

- **Concorrência com I/O externo:** se o destino de carga for um recurso compartilhado (como um banco com limite de conexões), muitas threads podem gerar contenção.
- **Dificuldade de controle de erro e rollback:** paralelismo em carregamento pode dificultar o rastreamento e recuperação de falhas, especialmente se os dados forem carregados em lotes ou transações distintas.

10 Análise de tempo de execução

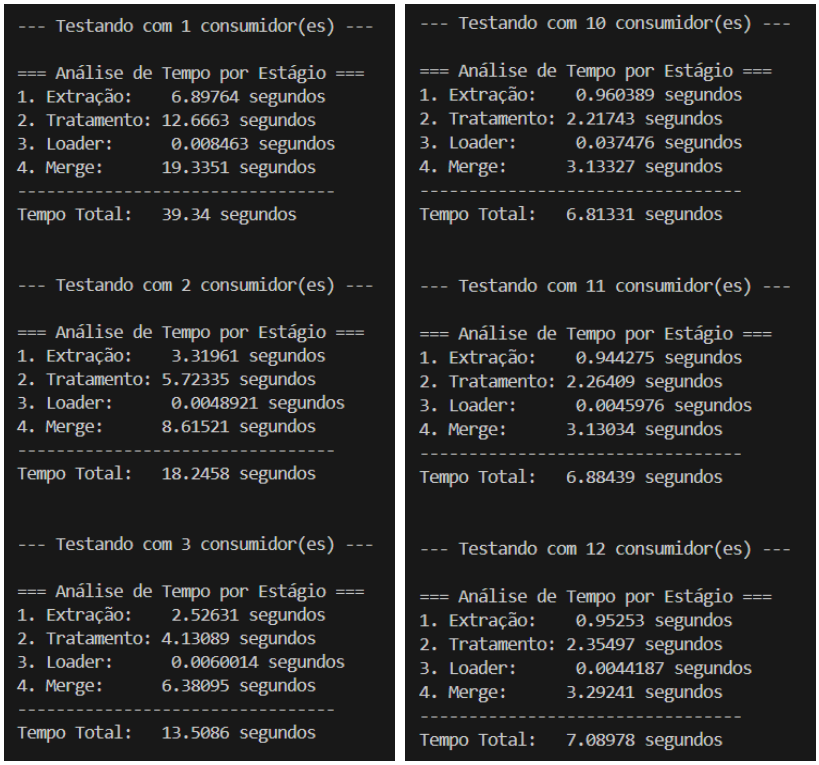

Figura 2: Testes com diferentes configurações de threads

Para demosntrar a corretude e eficiência do processo de paralelização temos acima um exemplo de uma sequência de testes com diferentes configurações de threads, assim como esperado no começo temos uma drástica diminuição no tempo, sendo praticamente reduzida metade do tempo, e no final pela saturação dos núcleos o ganho marginal de cada thread já se torna mais sútil. Uma observação muito importante, quando utilizamos 1 consumidor estamos na verdade utilizando no mínimo 3 threads, uma para cada etapa de extração, tratamento e carregamento que podem começar a trabalhar antes mesmo do processo anteriror se encerrar totalmente, e analogamente quando testamos para 2 consumidores são no mínimo 6 threads, 2 para cada etapa.