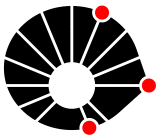

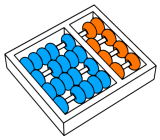

# Análise comparativa de algoritmos de balanceamento de carga em sistemas heterogêneos

*Júlia Alves de Arruda* *Lucas Hideki Carvalho Dinnouti*

Relatório Técnico - IC-PFG-24-01
Projeto Final de Graduação
2024 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

# Análise comparativa de algoritmos de balanceamento de carga em sistemas heterogêneos

Júlia Alves de Arruda Lucas Hideki Carvalho Dinnouti

**Resumo**

Este trabalho trata-se de uma análise comparativa entre diferentes técnicas de balanceamento de carga: Round Robin, Round Robin Ponderado, baseado em Metadados e Aprendizado de Máquina. A arquitetura foi baseada em uma plataforma de processamento de mensagens, que trafega conteúdos de diferentes tipos, utilizando dados reais. O objetivo foi encontrar a melhor estratégia para processar um volume grande de mensagens com diferentes tipos e tamanhos de instâncias, buscando entender se os algoritmos customizados para o domínio da aplicação apresentam melhor desempenho. Para o problema proposto, concluiu-se que tais algoritmos podem ser mais eficientes, a exemplo do baseado em Metadados. Por outro lado, os algoritmos baseados em Aprendizado de Máquina não apresentaram bom desempenho quando comparados às técnicas mais simples devido ao seu custo computacional.

***Palavras-Chave:*** Balanceamento de Carga; Sistemas Heterogêneos; Aprendizado de Máquina.

## 1 Introdução

Com o uso altamente disseminado da internet, aplicações que precisam lidar com um volume intenso de requisições têm se tornado cada vez mais comuns e, com elas, são necessárias técnicas para lidar com os limites impostos pelo hardware.

Uma técnica comum é replicar uma mesma aplicação em diversas máquinas, para que cada uma delas possa lidar com uma quantidade de requisições e assim evitar problemas de desempenho [1]. Para que isso seja possível, é necessário que haja alguma forma de dividir o fluxo entre essas máquinas, e a solução amplamente adotada é o balanceamento de carga.

Balanceamento de carga é uma técnica que busca otimizar o uso de recursos, ao mesmo tempo que fornece o máximo rendimento e tempo de resposta mínimo [2]. Diferentes formas de balanceamento de carga permeiam o desenvolvimento de sistemas distribuídos, variando em complexidade e objetivos. Porém, aplicar técnicas otimizadas a um domínio específico demanda conhecimento sobre os padrões de carga, recursos disponíveis, entre outros parâmetros que podem ser dinâmicos.

Isso pode não ser um problema a depender do caso de uso, e mesmo o Round Robin, argumentavelmente a forma mais simples de se distribuir carga entre sistemas, pode ser também a forma mais eficiente. No entanto, no caso de sistemas heterogêneos, pode haver

discrepância significativa na disponibilidade de recursos entre as máquinas, o que levaria esse algoritmo a distribuir a carga de forma ineficiente.

O intuito deste trabalho é explorar algoritmos que aproveitem melhor os recursos disponíveis. As seções a seguir descrevem: a implementação de um sistema heterogêneo e algoritmos de balanceamento de carga customizados; os resultados dos testes de desempenho aplicados aos diferentes algoritmos; e as conclusões obtidas da comparação dos resultados.

# 2 Objetivos e Relevância

## 2.1 Problema

Qual é o melhor algoritmo para balanceamento de carga em um sistema distribuído heterogêneo responsável pelo processamento de mensagens com diferentes requisitos computacionais?

## 2.2 Hipótese

É possível a implementação de algoritmos customizados para o domínio que levam em consideração características adicionais, como as mensagens trafegadas, e não só uma lista das máquinas disponíveis, sendo mais eficientes que algoritmos genéricos já conhecidos.

## 2.3 Justificativa

Sistemas heterogêneos são compostos por partes que atuam juntas, mas podem ser diferentes em diversos aspectos, como hardware, contendo diferentes tipos de processador, ou software, envolvendo sistemas operacionais diferentes. Tais sistemas podem ter vários objetivos, entre eles oferecer redundância ou melhorar o desempenho.

Uma forma comum dessa heterogeneidade é o sistema GPU-CPU, usado principalmente para redução do consumo de energia [3]. A Figura 1 ilustra tal forma de sistema heterogêneo, onde os servidores além de diferentes tipos de processadores, também possuem diferentes disponibilidades de recursos computacionais.

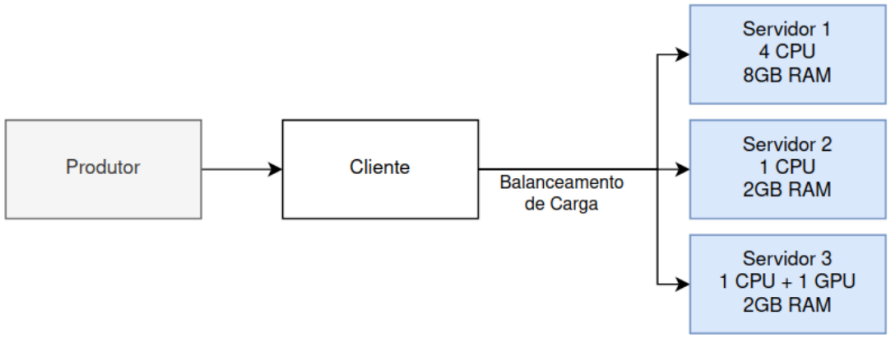


Figura 1: Diagrama ilustrativo de um sistema heterogêneo

Nesse cenário, caso fosse utilizado Round Robin como técnica de balanceamento, o Servidor 2 seria sobrecarregado enquanto o Servidor 1 seria subutilizado, já que um possui menos recursos computacionais que o outro.

Técnicas como o Round Robin Ponderado tratam o problema da má distribuição para as primeiras duas instâncias. Contudo, se for considerada ainda a terceira instância que possui um processador gráfico dedicado, é possível otimizar ainda mais a distribuição da carga ao considerar o conteúdo da requisição sendo feita.

Dada a grande variação entre as máquinas que um sistema heterogêneo pode oferecer, é importante encontrar um algoritmo de balanceamento que consiga distribuir a carga de forma justa. Caso contrário, o sistema pode acabar tendo problemas de desempenho ao sub-utilizar ou super-utilizar máquinas, se tornando ineficiente.

# 3 Metodologia

## 3.1 Modelagem

Para a análise comparativa das diferentes técnicas de balanceamento de carga, foi necessária a implementação de um sistema heterogêneo. Baseando-se em um sistema que fosse comum ao cotidiano, implementou-se a arquitetura ilustrada na Figura 2, inspirada em chat de mensagens. O sistema é formado por componentes de três tipos: Cliente, Proxy e Processador.

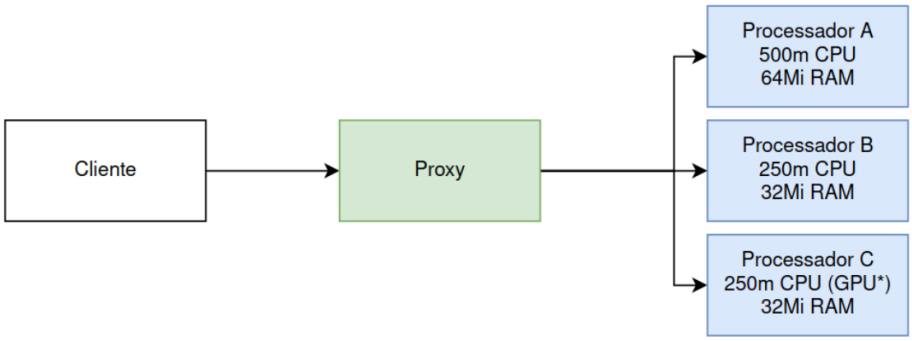


Figura 2: Diagrama do sistema heterogêneo implementado para testes

A primeira aplicação "Cliente" é o controlador dos testes, uma aplicação com instâncias homogêneas, que envia mensagens. As mensagens são de texto, imagem ou áudio, e foram obtidas através de arquivos de *backup* de conversas do *WhatsApp*, para que a proporção entre os diversos tipos de mensagem seja realista. Nessa proposta, o balanceamento de carga ocorre na sua comunicação com as instâncias de Processador de conteúdo.

Por sua vez, o Processador de conteúdo é um sistema heterogêneo, onde cada uma das instâncias, A, B e C, tem características computacionais específicas. Como por exemplo, o Processador C tem um marcador de processamento gráfico dedicado.

## 3.2 Algoritmos

A arquitetura descrita acima foi utilizada para comparar quatro algoritmos de balanceamento de carga. Dois deles são conhecidos e foram utilizados como referência: Round Robin e Round Robin Ponderado; e os outros dois são implementações customizadas que levam à validação da hipótese: baseado em Metadados e Aprendizado de Máquina.

### 3.2.1 Round Robin

Round Robin é um algoritmo estático amplamente utilizado que divide igualmente o número de requisições entre as máquinas, intercalando o envio destas.

### 3.2.2 Round Robin Ponderado

Existem vários problemas associados ao Round Robin, principalmente em sistemas heterogêneos onde as máquinas dispõem de recursos computacionais diferentes, e onde as requisições têm requisitos computacionais distintos. Consequentemente, o Round Robin Ponderado foi criado para resolver esses problemas. Nele, é atribuído um peso a cada servidor, que então recebe um número de requisições proporcional a esse valor [3], fazendo com que mais mensagens sejam enviadas às instâncias com maior capacidade.

### 3.2.3 Baseado em Metadados

Projetou-se esse algoritmo neste trabalho, especificamente para o domínio da aplicação que é inspirada em um chat de mensagens. Ele distribui as mensagens baseado em seu conteúdo, no caso, enviando mensagens que contém imagens para máquinas marcadas com processamento gráfico dedicado (processador C), focando em melhorar esse desempenho.

De forma geral esse algoritmo continua funcionando como um Round Robin, alternando o envio entre as réplicas. Quando o tipo da mensagem está marcado para ser executado em uma máquina específica, como mensagens de imagem, então a requisição é direcionada de acordo.

### 3.2.4 Aprendizado de Máquina

Esse algoritmo busca aplicar técnicas de aprendizado de máquina para, dadas as condições atuais das máquinas e a requisição em questão, prever qual será o tempo de resposta (latência) caso essa seja encaminhada para cada um das máquinas.

Assim, tendo uma estimativa da latência do envio para todas as máquinas, é possível decidir pelo roteamento para a máquina que oferecerá a menor latência.

Para que isso fosse possível, a técnica escolhida foi a Regressão Linear, uma técnica estatística usada para modelar relações entre variáveis. Ela pretende predizer o valor de variáveis dependentes (ou de resposta) a partir de um conjunto de variáveis independentes (ou preditoras) [4].

O modelo de Regressão Linear tem como variáveis independentes: instância, tipo da mensagem enviada, uso de CPU das instâncias disponíveis, uso de memória das instâncias disponíveis; e como variável dependente o tempo de resposta.

# 4 Implementação

Todas as informações disponíveis nessa seção são encontradas nesse repositório no GitHub: `https://github.com/lucasdinnouti/custom-reverse-proxy`.

O arquivo de *backup* do *WhatsApp* não está disponível no repositório, mas pode ser gerado pelo aplicativo de celular. As mensagens que são mencionadas ao longo do texto são mensagens de conversas reais. Seu conteúdo não é importante para os resultados, apenas a informação de seu tipo, isto é, se são mensagens de texto, áudio, imagem ou outra mídia.

## 4.1 Componentes

Os componentes descritos nesta seção foram implementados em Golang.

### 4.1.1 Cliente

A aplicação Cliente é responsável por controlar os testes e tem capacidade de ler os arquivos de backup de conversa e enviar as mensagens uma por vez para o Proxy.

Uma característica importante desse componente é a capacidade de controlar as Transações Por Segundo (TPS) das mensagens enviadas. Através de funcionalidades nativas, foi possível controlar o volume e tamanho dos testes dinamicamente.

### 4.1.2 Proxy

Existem diversas ferramentas que atuam como balanceadores de carga disponíveis no mercado. O NGNIX é um exemplo de ferramenta compatível com a arquitetura proposta [5].

Porém, para a análise comparativa sugerida é preciso um nível de customização maior do que o oferecido por essas ferramentas, principalmente por conta da integração com modelos de aprendizado de máquina. Assim, optou-se pela implementação de um *proxy* próprio.

Seria possível utilizar soluções prontas para os algoritmos conhecidos, como o Round Robin, e um Proxy customizado para os outros algoritmos. Porém, julgou-se ser mais justo para a comparação que todos os algoritmos fossem executados pelo mesmo *proxy*.

Utilizando *httputil.ReverseProxy*, uma funcionalidade nativa do Golang, foi possível construir uma aplicação que suporta todos os algoritmos e, através de uma variável de ambiente, é capaz de trocar entre eles. Assim, ao receber uma requisição, o Proxy seleciona uma instância de processador para qual a encaminha.

### 4.1.3 Modelo

O modelo de Aprendizado de Máquina foi treinado em Python, preferido por ter um melhor ferramental para treinamento de modelos quando comparado ao Golang. Mas especificamente, utilizou-se SciKit Learn, uma biblioteca conhecida e com suporte a vários algoritmos, entre eles o de Regressão Linear [6].

Como os componentes arquiteturais foram implementados em Golang, o que inclui o Proxy, o modelo já treinado foi portado em formato Open Neural Network Exchange

(ONNX) para a utilização em tempo de execução. Tal formato foi criado justamente para facilitar a interoperabilidade de modelos. Foi utilizada a biblioteca *onnxruntime_go* [7] para leitura do modelo em tempo de execução.

#### 4.1.3.1 Datasets

Para gerar um *dataset* de treinamento, foi necessário um mecanismo capaz de salvar dados de roteamento e estado das máquinas e transformar tais informações em arquivos CSV. Os componentes implementados para os testes foram executados várias vezes usando algoritmos diferentes, assim se obtendo dados suficientes para que o modelo conseguisse inferir a relação entre latência da requisição e os outros parâmetros.

Durante o treinamento notou-se que algumas características não estavam sendo abstraídas pelo modelo. O caminho escolhido foi trabalhar o conjunto de dados de treinamento, utilizando técnicas como remoção de *outliers*, *data augmentation*, e até mesmo explorar o uso de *datasets* sintéticos ou invés de dados reais.

Os dados sintéticos foram gerados da seguinte forma: os valores de utilização de CPU e memória foram gerados aleatoriamente de 0 a 1. A requisição então toma um tempo proporcional à CPU e à memória do processador de destino. Caso a mensagem seja de imagem e o processador de destino tenha marcador de GPU, a requisição leva menos tempo. Por exemplo, se uma requisição de imagem fosse roteada para o Processador C, que está com 50% de CPU e 10% de memória, a função seria: *tempo de resposta = (0,5 * 100 + 0,1 * 100) * 0,5 = 300*.

Como os valores são normalizados, a maior preocupação não é no valor absoluto, mas em modelar as especificidades que o sistema heterogêneo apresenta. Nesse caso, foi modelado que o tempo de resposta é proporcional ao uso de recursos e que existe uma relação especial entre imagens e processadores com GPU.

#### 4.1.3.2 Algoritmo de balanceamento

Se tratando da estratégia de balanceamento de carga em si, a primeira estratégia foi: para cada requisição, o Proxy usaria o modelo para prever o tempo de resposta de uma requisição para cada uma das instâncias, usando informações atuais das máquinas e o tipo da mensagem (informações de CPU e memória foram armazenadas em *background* em um *cache*, que será explicado na seção 4.2.1) e enviaria a mensagem para a instância com menos latência.

Ao longo da implementação, notou-se que essa estratégia não é eficiente, principalmente porque prever os valores é computacionalmente custoso e impacta o tempo de resposta se feito a cada requisição. Por isso, a solução adotada foi baseada no Round Robin Ponderado, e funciona da seguinte maneira: Um processo é disparado em um intervalo constante de tempo em *background* e utiliza o modelo para prever o tempo de resposta de cada instância, dadas as métricas de uso de recursos de cada uma. Vale ressaltar que uma das variáveis preditoras é o tipo da mensagem, e para tal, utilizou-se um contador dos tipos que passaram pelo Proxy recentemente. Com as previsões feitas, os pesos são determinados de forma a

encaminhar mais mensagens a instâncias de resposta mais rápida.

Por ser feito em *background*, o tempo de rotear uma requisição não é impactado pela execução do modelo, sendo análogo ao Round Robin Ponderado.

#### 4.1.4 Processor

Esse é o componente que recebe as requisições encaminhadas pelo Proxy, e é responsável por processá-las. Em um cenário real, esse processador seria capaz de interpretar as mensagens de forma realista, por exemplo, traduzí-las para fornecer acessibilidade.

Porém, para as análises, não se julgou necessária a implementação de tal processamento, se escolhendo simulá-lo. Para isso, utilizou-se o cálculo de números primos para simulação de uso de CPU e vetores de texto para uso de memória.

Tais cálculos e estruturas têm tamanhos dinamicamente definidos de acordo com o tipo da mensagem recebida, e com o tipo de processador da instância. Por exemplo, o custo de processamento de uma imagem ou áudio é maior do que de um texto; e caso uma imagem seja recebida por uma instância que possua GPU, o custo é reduzido. Note que, mesmo existindo várias instâncias de processadores, isto é, a aplicação roda em diversas máquinas, todos possuem o mesmo código base.

### 4.2 Infraestrutura

Preferiu-se utilizar um cluster Kubernetes [8] para administrar o sistema, a fim de facilitar a configuração de rede e gerenciamento de réplicas. Os componentes são recursos individuais, inclusive os processadores, que utilizam a mesma imagem docker mas possuem configurações diferentes dentro do cluster.

#### 4.2.1 Métricas

Para realizar análises e até mesmo obter informações de uso de recurso para tomada de decisão utilizando aprendizado de máquina, foi necessária a utilização de uma ferramenta capaz de capturar métricas do Kubernetes mas também métricas customizadas de cada aplicação (como latência das requisições).

Para tal, escolheu-se aplicar o Prometheus, uma plataforma *open-source* capaz de coletar e armazenar métricas em uma base de dados de séries temporais, utilizando um modelo de coleta via pull [9]. Além de oferecer uma interface de consulta e uma linguagem de *queries*, também oferece uma API que foi utilizada pelo processo em *background* do Proxy (citado na seção 4.1.3).

Também utilizou-se o Grafana, uma plataforma capaz de se conectar ao Prometheus para a construção de *dashboards*, facilitando e padronizando a consulta dos resultados dos testes [10].

### 4.3 Simulação

Houve 2 conjuntos de dados para teste, sendo ambos conversas de *WhatsApp*. As conversas têm suas mensagens categorizadas em: “texto”, “imagem”, “áudio” e “outros”, sendo

incluídas nessa última categoria documentos, contatos e vídeos. As estatísticas para cada conjunto de testes são apresentadas na Tabela 1.

| **Conjunto** | **Mensagens** | **Texto** | **Imagens** | **Áudio** | **Outros** |
|---|---|---|---|---|---|
| A | 3291 | 3137 | 62 | 59 | 33 |
| B | 1531 | 1169 | 279 | 2 | 81 |

Tabela 1: Tabela da proporção de tipos de mensagem por conjunto de teste

Foram escolhidas essas duas conversas pelas suas proporções distintas de texto e imagens. Chamaremos o conjunto de mensagens com maior proporção de textos de Conjunto A, e o com maior proporção de imagens, de Conjunto B. Assim, podemos comparar as estratégias de balanceamento de carga em cenários reais e diversos. A taxa de processamento em TPS foi controlada artificialmente. As taxas utilizadas nos testes foram de 30 a 140 TPS no Conjunto A, e de 10 a 60 TPS no Conjunto B.

## 5 Resultados

Como mencionado anteriormente, a taxa de TPS variou de acordo com o conjunto de dados utilizado. Foram executados testes para todos os algoritmos com ambos os conjuntos.

As Figuras 3 e 4 representam a evolução da taxa de TPS nos conjuntos de teste A e B, com volume máximo de 140 e 60 TPS respectivamente.

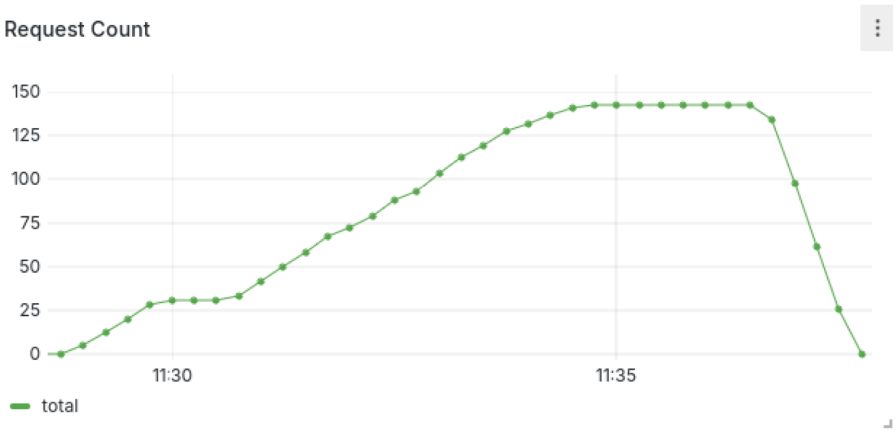


Figura 3: Evolução de TPS dos testes com Conjunto A, máximo de 140 TPS

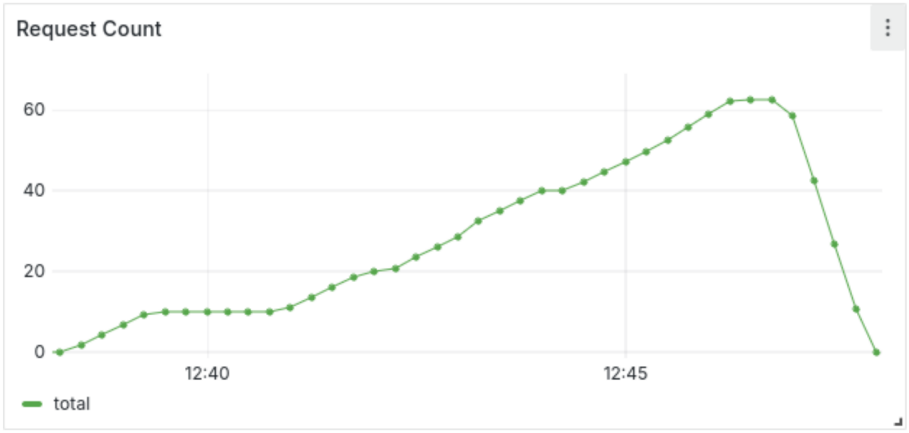


Figura 4: Evolução de TPS dos testes com Conjunto B, máximo de 60 TPS

## 5.1 Round Robin

Como mencionado anteriormente, esse algoritmo distribuiu as requisições igualmente entre as instâncias, sem considerar características da mensagem ou das máquinas.

A Figura 5 mostra que, no Conjunto A, o Processador B teve maior tempo de resposta, seguido do Processador C e então do Processador A. Assim, é possível observar que o tempo de resposta foi inversamente proporcional à capacidade do processador. O Processador A teve vantagem por ter mais CPU e memória, e o Processador C também manteve melhores tempos de resposta já que é mais eficiente em processar imagens. O Processador B teve degradação considerável, tendo tempos de resposta elevados e queda do serviço.

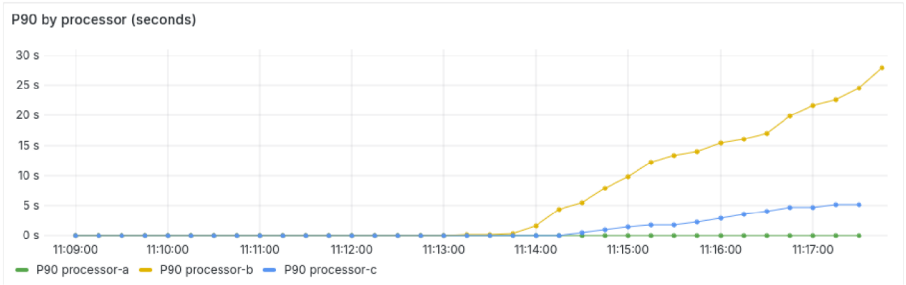


Figura 5: P90 do tempo de resposta por processador durante Round Robin, teste com Conjunto A

Já no conjunto B, pelo percentil de tempo de resposta é notável que a maioria das requisições se mantiveram com um tempo baixo, com algumas exceções ao final do teste que passaram de 2s. Olhando a Figura 6, que mostra um p90 de mais de 1 segundo para o Processador B, é evidente que ele foi o que mais impactou a média geral sendo responsável pelas exceções.

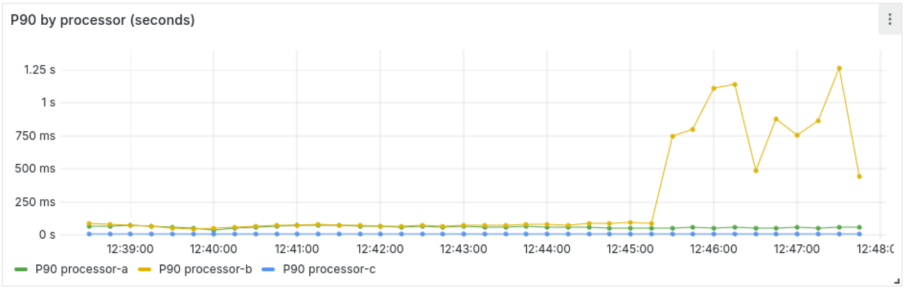


Figura 6: P90 do tempo de resposta por processador durante Round Robin, teste com Conjunto B

## 5.2 Round Robin Ponderado

Como previsto, as requisições nesse caso foram distribuídas proporcionalmente aos pesos pré-definidos, que refletiam as capacidades de cada instância. A Tabela 2 mostra os pesos de cada uma delas. A Figura 7, mostra efetivamente a distribuição de requisições.

| **Processador A** | **Processador B** | **Processador C** |
|---|---|---|
| 2 | 1 | 1 |

Tabela 2: Tabela da distribuição de pesos por processador no algoritmo de Round Robin Ponderado

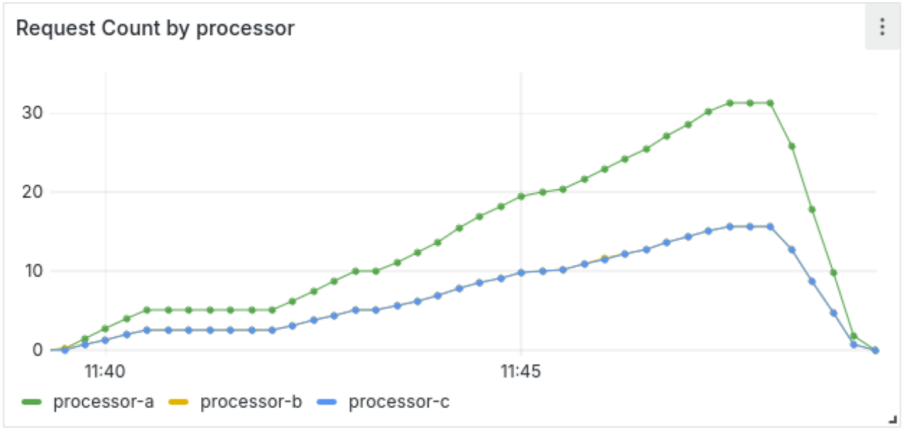


Figura 7: Distribuição das requisições entre Processadores, teste com Conjunto B

Ao longo do teste com o Conjunto A, houve utilização eficiente dos recursos dos três processadores, apesar de suas diferentes configurações. A Figura 8 mostra a utilização de CPU dos três processadores, e seus valores semelhantes ao longo do tempo.

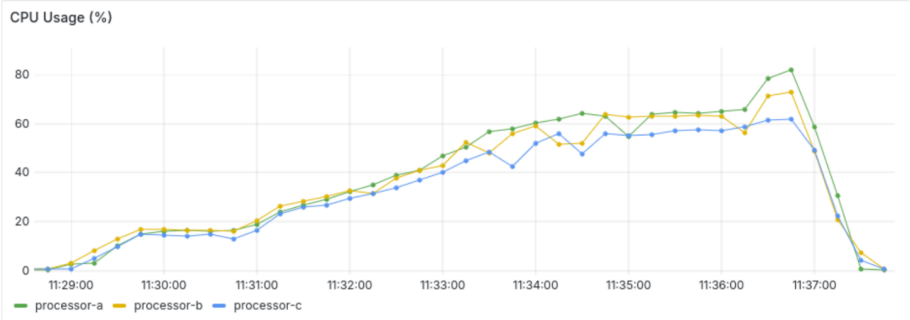


Figura 8: Uso de CPU durante Round Robin Ponderado, teste com Conjunto A

Apesar disso, foi possível perceber um aumento na latência em volumes mais altos de processamento, principalmente no Processador A, como visto na Figura 9, mostrando que a proporção entre recursos e eficiência em processamento de requisições não é completamente linear.

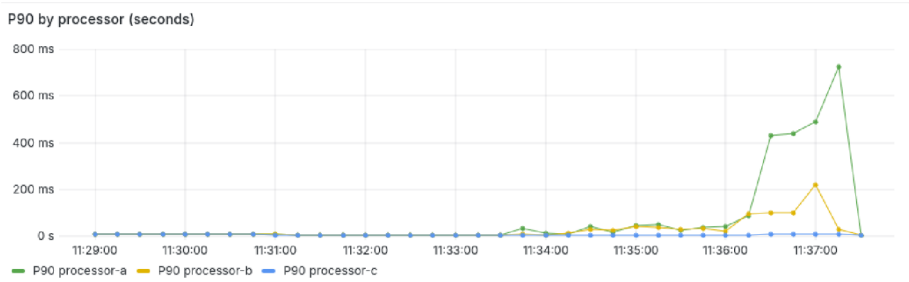


Figura 9: P90 do tempo de resposta durante Round Robin Ponderado, teste com Conjunto A

Já no teste com o Conjunto B, o processador C foi subutilizado, como visto na Figura 10, por esse teste contar com maior proporção de imagens, processadas eficientemente pelo Processador C.

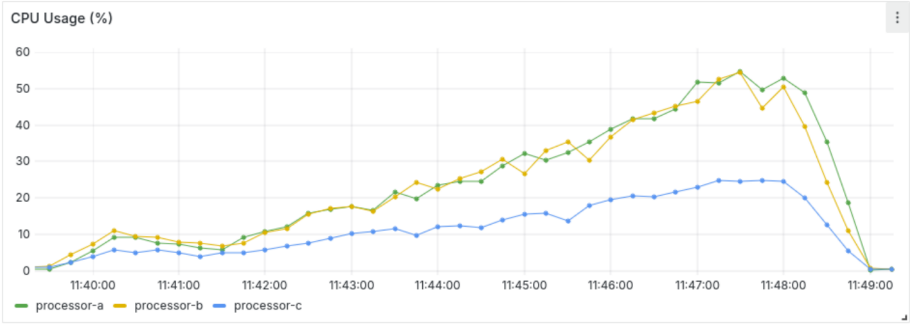


Figura 10: Uso de CPU durante Round Robin Ponderado, teste com Conjunto B

### 5.3 Baseado em Metadados

Para esse algoritmo, houve bastante diferença entre os conjuntos de testes. Para o conjunto A, onde houve mais mensagens de texto, foi possível ver que o Processador C sofreu uma sobrecarga, o que impactou no seu tempo de resposta. A Figura 11 mostra que ao final do teste o Processador C teve 100% de uso da CPU. Isso se deu pois, além de receber as mensagens de texto, também recebeu todas as imagens.

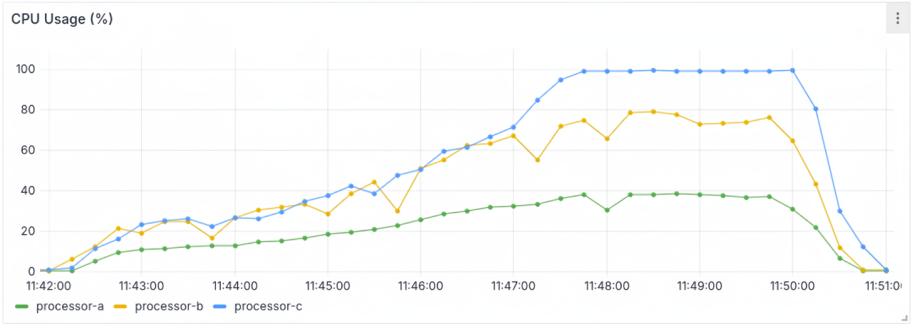


Figura 11: Uso de CPU por processador durante algoritmo Metadados, teste com Conjunto A.

Já para o conjunto B, o Processador C acabou recebendo um número significativamente maior de requisições quando comparado com as outras instâncias, justamente pelo fato de existirem mais imagens. Isso pode ser visto na Figura 12.

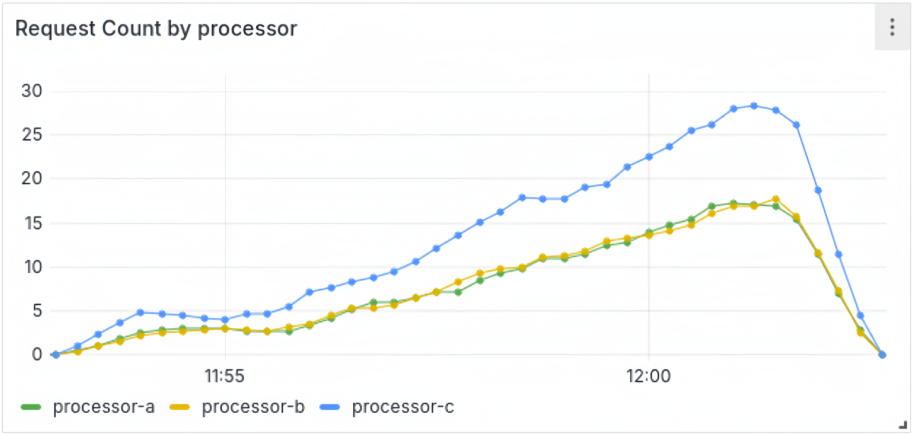


Figura 12: Distribuição de requisições durante algoritmo Metadados, teste com Conjunto B

Porém, isso não implicou em sobrecarga, mas sim em maior eficiência. Apesar do Processador C ter uso de recursos um pouco maior, isso não representou impacto em seu tempo de resposta, que permaneceu igual às outras instâncias. Se comparado ao Round Robin Ponderado, que teve o segundo melhor desempenho no Conjunto de dados B, o algoritmo de metadados conseguiu manter um P90 de 10 ms em todas as instâncias ao fim do teste, contra mais de 50ms no caso do Round Robin Ponderado.

A Figura 13 mostra o tempo de resposta com o algoritmo de Metadados, e a Figura 14

com o algoritmo de Round Robin Ponderado.

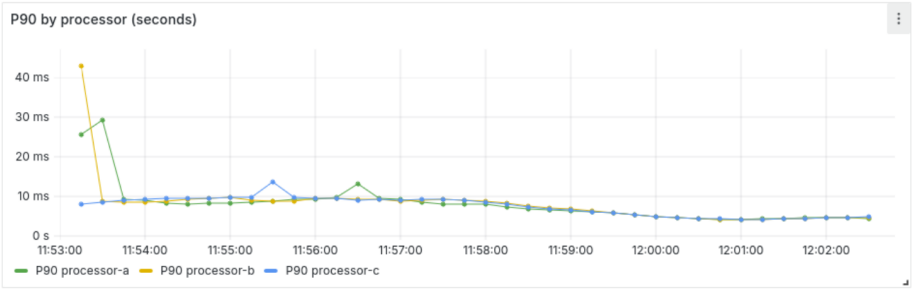


Figura 13: P90 do tempo de resposta durante algoritmo Metadados, teste com Conjunto B

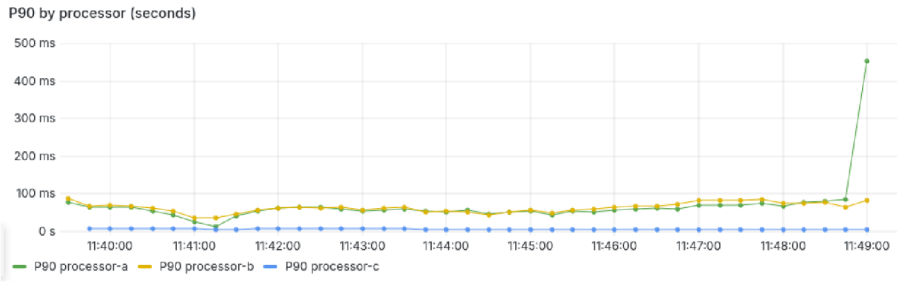


Figura 14: P90 do tempo de resposta durante algoritmo de Round Robin Ponderado, teste com Conjunto B

## 5.4 Aprendizado de máquina

Como mencionado na seção sobre a implementação do modelo (4.1.3), ele passou por mudanças até ser fixada uma versão final. A primeira mudança se deu na forma em que o modelo é usado: deixou de prever valores mensagem a mensagem e passou a fazer apenas uma previsão a cada 10 segundos em *background*. Como as Figuras 15 e 16 mostram, houve melhora significativa apenas com essa mudança.

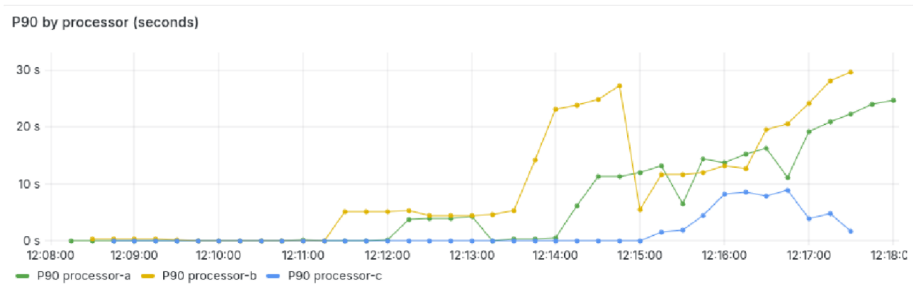


Figura 15: P90 do tempo de resposta usando Machine Learning 1 a 1, teste com Conjunto B

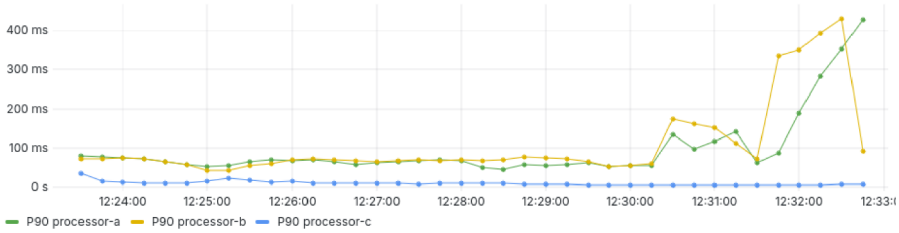


Figura 16: P90 do tempo de resposta usando durante Machine Learning otimizado, teste com Conjunto B

Outra variação importante foi nos conjuntos de dados para treinamento, sendo a diferença mais relevante a diferença entre dados reais e dados sintéticos. Os modelos foram avaliados pelo valor de "*score*", provido pela ferramenta SciKit Learn. O melhor valor possível para essa métrica é 1,0 e 0,0 representa um modelo que sempre prevê o mesmo valor. É possível obter valores negativos, pois os modelos podem ser arbitrariamente piores [6].

### 5.4.1 Dataset real

Nesse caso, foram utilizados valores extraídos durante testes com dados reais. O Cliente registrou os tempos de resposta de cada requisição, e posteriormente foram agregados os valores de métricas como a CPU e memória dos processadores. O modelo treinado com esses dados teve um *score* de apenas 0,24. Isso prejudicou as escolhas do modelo, o que sobrecarregou o Processador B como visto na Figura 17, que mostra também um período de indisponibilidade do Processador.

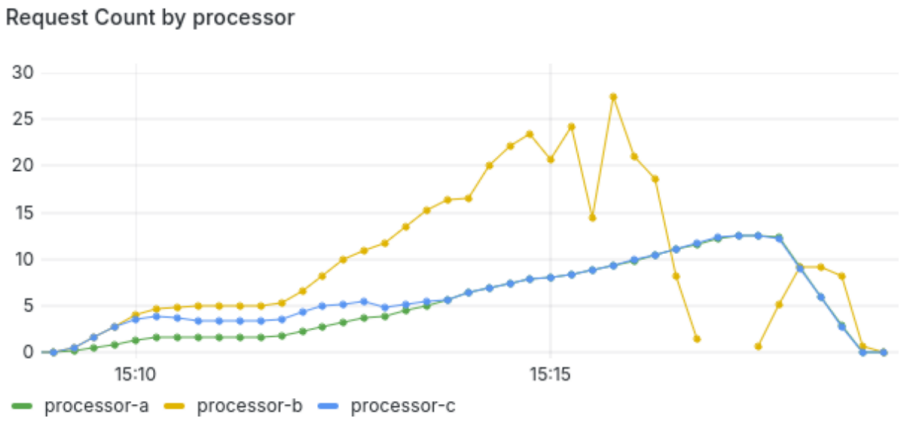


Figura 17: Distribuição de requisições com algoritmo de aprendizado de máquina treinado em dados reais, teste com Conjunto B

### 5.4.2 Dataset sintético

Com o *dataset* sintético, descrito na seção 4.1.3, obtivemos melhor comportamento do Proxy, e é possível observar a variação no roteamento de mensagens de acordo com o

conjunto de testes. As Figuras 18 e 19 mostram a distribuição de requisições em cada conjunto. No conjunto A, o Proxy optou por direcionar mais mensagens para o Processador A, e uma quantidade semelhante para os processadores B e C. No caso do conjunto B, com maior proporção de imagens, pode-se verificar que o Proxy optou por aumentar o envio ao processador C, o que aproveita sua eficiência em processar esse tipo de mensagens.

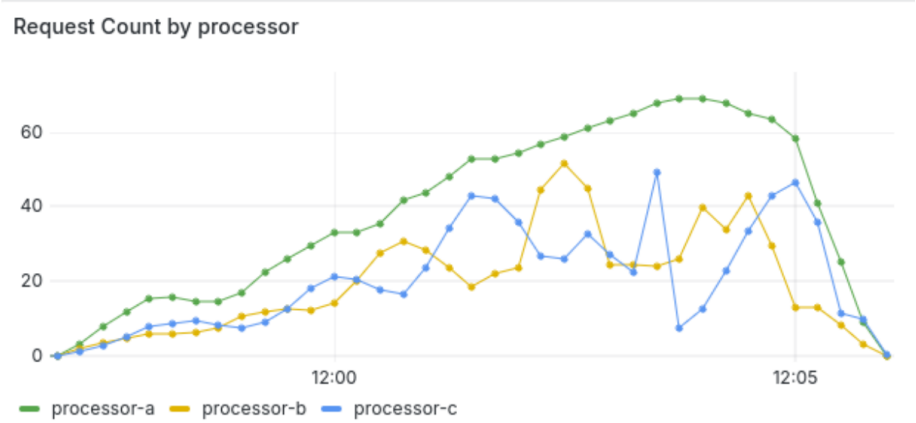


Figura 18: Distribuição de requisições com algoritmo de aprendizado de máquina treinado em dados sintéticos, conjunto de testes A

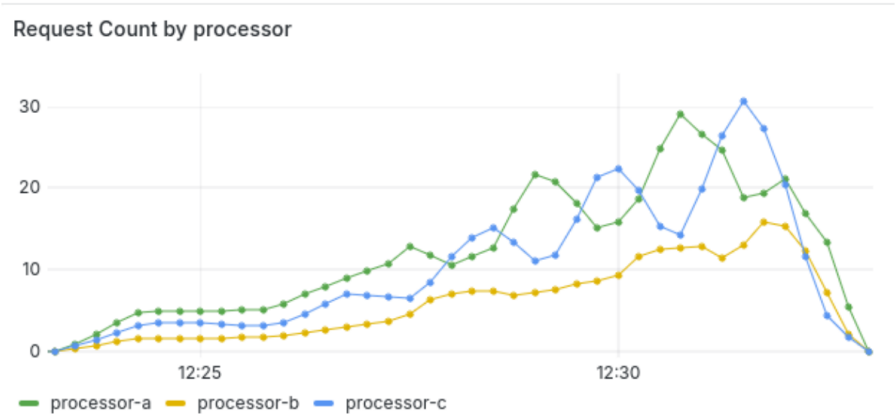


Figura 19: Distribuição de requisições com algoritmo de aprendizado de máquina treinado em dados sintéticos, conjunto de testes B

## 5.5 Síntese

O algoritmo de Round Robin se comportou de forma estável, tendo problemas apenas nos casos mais extremos de volume e carga. Porém, ele foi ineficiente se comparado aos outros algoritmos em ambos os conjuntos de dados.

O Round Robin Ponderado se mostrou uma opção eficiente para o balanceamento de carga em sistemas heterogêneos onde o conteúdo da mensagem não se relaciona aos recursos

das máquinas disponíveis, sendo a opção mais eficiente para o Conjunto de dados A, onde há poucas imagens.

O algoritmo baseado em Metadados se mostrou uma opção eficiente quando o conteúdo da mensagem está relacionado a recursos dedicados, que é o caso do Conjunto de testes B, onde há maior proporção de imagens que são melhor processadas por uma GPU dedicada.

## 6 Conclusão

É possível delimitar conclusões que relacionam a eficiência dos algoritmos aos conjuntos de dados A e B utilizando as métricas de uso de recursos e tempo de resposta. Uma conclusão parcial é que direcionar maior volume aos processadores com maior capacidade aumenta a eficiência do sistema; seja essa capacidade puramente disponibilidade de CPU e memória ou de recursos dedicados como processamento gráfico para imagens. Outra conclusão parcial é de que o conjunto de dados afeta diretamente a eficiência do sistema, sendo que diferentes algoritmos podem desempenhar melhor em diferentes casos de teste. Assim, é possível chegar a três afirmações:

1. O algoritmo de Round Robin Ponderado é significativamente mais eficiente que o Round Robin em sistemas distribuídos heterogêneos caso os pesos estejam calibrados.

2. O algoritmo de Metadados é significativamente mais eficiente que o Round Robin Ponderado caso haja relação entre o conteúdo trafegado e processadores dedicados.

3. Aprendizado de Máquina não se mostrou eficiente quando comparado a algoritmos mais comuns e computacionalmente mais simples.

Portanto, a hipótese inicialmente apresentada foi parcialmente comprovada. Algoritmos customizados podem sim ser mais eficientes do que aqueles agnósticos ao domínio da aplicação, como foi o caso do algoritmo baseado em metadados. Porém, em alguns cenários dentro de um sistema heterogêneo, algoritmos como o Round Robin Ponderado se mostram mais eficientes que técnicas rebuscadas como Aprendizado de Máquina.

Para continuidade deste trabalho, seria interessante explorar outras técnicas de Aprendizado de Máquina, para que se chegue em um modelo com maior precisão. A hipótese é que modelos mais precisos usados para previsão de pesos tenham capacidade de competir com os algoritmos com melhor desempenho, como o baseado em Metadados.

Nesse sentido, também é possível explorar técnicas de aprendizado por reforço, onde o modelo é capaz de se autorregular ao longo do tempo. Tal técnica pode também ajudar com o problema de precisão.

Outra forma de ampliar o estudo é mudando a natureza do domínio. Apesar deste trabalho usar arquivos de *backup* para simular requisições, todos os algoritmos avaliados não levam essa informação em consideração, propositalmente eles buscam tomar decisões em tempo real, sem contar com informações futuras. Porém, é possível o desenvolvimento de algoritmos que levem informações sobre o *backup* em suas decisões, por exemplo, se ajustando a depender de quais tipos de mensagem estão por vir.

## Referências

[1] VAN STEEN, Maarten; TANENBAUM, A. Distributed systems principles and paradigms. Network, v. 2, n. 28, p. 1, 2002.

[2] SAMAL, Pooja; MISHRA, Pranati. Analysis of variants in Round Robin algorithms for load balancing in cloud computing. International Journal of computer science and Information Technologies, v. 4, n. 3, p. 416-419, 2013.

[3] WANG, Guibin; REN, Xiaoguang. Power-efficient work distribution method for cpu-gpu heterogeneous system. In: International symposium on parallel and distributed processing with applications. IEEE, 2010. p. 122-129.

[4] MAROCO, J.: Análise Estatística, com utilização do SPSS.(2003) Edições Sílabo. Lisboa, Portugal.

[5] NGINX. Acesso em 24 junho de 2024. Disponível em: https://www.nginx.com/

[6] SCIKIT-LEARN. Acesso em 26 junho de 2024. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[7] YALUE. Cross-Platform onnxruntime Wrapper for Go. Acesso em 24 junho de 2024. Disponível em: https://github.com/yalue/onnxruntime_go

[8] KUBERNETES. Acesso em 26 junho de 2024. Disponível em: https://kubernetes.io/pt-br

[9] PROMETHEUS. Acesso em 24 junho de 2024. Disponível em: https://prometheus.io/

[10] GRAFANA. Acesso em 26 junho de 2024. Disponível em: https://grafana.com/