INSTITUTO FEDERAL DE SANTA CATARINA

MARINA SOUZA

# Comparativo de desempenho de um sistema de cache para aplicações web utilizando bancos de dados chave-valor

São José - SC

agosto/2022

# COMPARATIVO DE DESEMPENHO DE UM SISTEMA DE CACHE PARA APLICAÇÕES WEB UTILIZANDO BANCOS DE DADOS CHAVE-VALOR

Trabalho de conclusão de curso apresentado à Coordenadoria do Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Professor Ederson Torressini

Coorientador: Professor Jorge Henrique Busatto Casagrande

São José - SC

agosto/2022

Marina Souza
Comparativo de desempenho de um sistema de cache para aplicações web utilizando bancos de dados chave-valor/ Marina Souza. – São José - SC, agosto/2022-
105 p. : il. (algumas color.) ; 30 cm.

Orientador: Professor Ederson Torressini

Monografia (Graduação) – Instituto Federal de Santa Catarina – IFSC
Campus São José
Engenharia de Telecomunicações, agosto/2022.

1. Invalidação de cache. 2. Aplicação web. 2. Banco de dados chave-valor. I. Ederson Torressini. II. Instituto Federal de Santa Catarina. III. Campus São José. IV. Comparativo de desempenho de um sistema de cache para aplicações web utilizando bancos de dados chave-valor

MARINA SOUZA

# COMPARATIVO DE DESEMPENHO DE UM SISTEMA DE CACHE PARA APLICAÇÕES WEB UTILIZANDO BANCOS DE DADOS CHAVE-VALOR

Este trabalho foi julgado adequado para obtenção do título de Engenheiro de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 19 de agosto de 2022:

**Professor Ederson Torressini, Me.**
Orientador
Instituto Federal de Santa Catarina

**Professor Jorge Henrique Busatto Casagrande, Dr.**
Co-orientador
Instituto Federal de Santa Catarina

**Professor Marcos Moecke, Dr**
Instituto Federal de Santa Catarina

**Professor Odilson Tadeu Valle, Dr**
Instituto Federal de Santa Catarina

# AGRADECIMENTOS

À minha família pelo suporte durante a elaboração desse projeto.

Ao Instituto Federal de Santa Catarina (IFSC) e todo corpo docente, pela dedicação, qualidade de ensino, ambiente e apoio oferecido durante todos esses anos.

Aos meus orientadores Ederson Torresini, Jorge Casagrande por aceitarem tratar desse tema e me guiarem na elaboração desse projeto. Aos professores(as) Saul, Maria Lúcia, Marcelo, Robertinho, Marcos Moeke, Odilson, Volney, Volneizinho, Rubem e Ramon por marcaram a minha trajetória no curso com seus ensinamentos e personalidades.

Aos meus amigos(as) Jeneffer, Paulo, Maria Fernanda e Vitor pela parceria e risadas até nas fases difíceis da graduação.

À minha fiel escudeira Pally pelas palavras de apoio, incentivo e paciência de sempre.

Ao Chico por estar sempre presente.

*Keep a little fire burning*
*(Cormac McCarthy)*

# RESUMO

O cache vem sendo amplamente utilizado em diversas etapas de um sistema web como técnica que possibilita a redução de consultas redundantes a servidores e banco de dados, tráfego na rede e latência. Contudo, dada a diversidade de aplicações disponíveis e suas particularidades, necessidades distintas entre as aplicações devem ser consideradas durante o desenvolvimento para melhor utilização dos recursos disponíveis. O objetivo deste trabalho é analisar diferentes abordagens de uso de cache em aplicações web, comparando o desempenho através da métrica de tempo de resposta da requisição, representando a percepção do usuário como utilizador do sistema e também métricas de eficiência de uso do recurso de cache como *hit rate* e uso de memória. A aplicação web em avaliação é um sistema de consulta por termos, no qual os dados serão armazenadas no banco de dados não relacional Cassandra e serviço de cache implementado através do banco de dados não relacional Redis. O sistema foi submetido à um teste de carga para análise do comportamento. Ao fim do teste, em média, mais de 78% das requisições estavam sendo atendidas por cache nos cenários que implementavam a técnica. O cenário que implementa o cache passivo e cenário que implementa cache pró-ativo com *Time to Live* (TTL) adaptativo apresentaram os menores valores de tempo de resposta sendo que o cenário com cache pró-ativo com TTL adaptativo apresentou também, a maior taxa de *hit rate*. A memória livre e o uso de *Central Process Unit* (CPU), embora impactados durante a execução dos testes, não apresentaram resultados conclusivos quando a um cenário mais eficiente comparado as demais.

**Palavras-chave**: Invalidação de cache. Aplicação web. Banco de dados chave-valor.

# ABSTRACT

The cache has been widely used in several stages of a web system such as technique that makes it possible to reduce redundant queries to servers and databases, network traffic and latency. However, given the diversity of available applications and their particularities, different needs between applications must be considered during development to make better use of available resources. The purpose of this work is to analyze different approaches to the use of cache in web applications, comparing performance through the request response time metric, representing the user perception as a system user and also usage efficiency metrics cache resource like hit rate and memory usage. The web application being evaluated is a query by terms system, in which the data will be stored in the database Cassandra non-relational data and caching service implemented through the database Redis non-relational data. The system was subjected to a load test to analyze the behavior. At the end of the test, on average, more than 78% of requests were being processed served by cache in scenarios that implemented the technique. The scenario that implements passive caching and scenario that implements proactive caching with TTL adaptive presented the lowest response time values and the scenario with proactive caching with adaptive TTL also presented the highest hit rate. Free memory and CPU usage, although impacted during execution of the tests, did not present conclusive results when to a more efficient compared to others.

**Keywords**: Cache invalidation. Web application. Key-value database.

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

# SUMÁRIO

# 1 INTRODUÇÃO

Com o desenvolvimento de tecnologias de conectividade, cada vez mais pessoas e dispositivos estão tendo acesso à Internet. Além do crescimento na quantidade de usuários, a forma de utilização do sistema também evoluiu, principalmente na última década, com os usuários finais deixando de serem apenas consumidores de conteúdo e tornando-se fonte ativa de informação (TRISTAO, 2008). Uma consequência do alto volume de informações e transferência de dados pela web é o custo que os provedores de conteúdos e serviços precisam arcar com a infraestrutura necessária para suprir a nova demanda, de tal modo que se mantenha uma boa performance, principalmente do ponto de vista do usuário.

O aumento na quantidade de requisições simultâneas e a solicitação de recursos não estáticos impactam diretamente na infraestrutura de rede e, principalmente, na capacidade de processamento no servidor web (PEñA-ORTIZ et al., 2013). O esgotamento de recursos do servidor, principalmente processamento e da memória RAM, podem ocasionar ações que refletem na usabilidade da aplicação do lado do cliente, como por exemplo, aumento no tempo de resposta de uma solicitação, recusa de conexões, indisponibilidade de serviços entre outros (Hwang et al., 2016).

Segundo Nakata, Arakawa e Murata (2015), o tráfego por redes móveis dobra a cada ano e esse fato se deve principalmente ao aumento do número de pessoas com acesso a Internet e ao aumento de dispositivos portáteis como *smartphones*, *tablets* e devices para *Internet of things* (IoT). O modelo tradicional de infraestrutura de servidores, no qual cada servidor possui a sua infraestrutura individual, tende a ter uma limitação na velocidade em que consegue escalar seus recursos e atender um pico de demanda repentino, como uma oferta em *e-commerces* ou uma notícia relevante em um portal de notícias. Neste cenário, mesmo que haja um planejamento para atender uma grande demanda, sendo ela esporádica, surge o desafio de evitar que os recursos fiquem subutilizados fora dos momentos de pico. Além da ociosidade de recursos, o excesso de capacidade de *hardware* pode levar a um desperdício significativo de energia (YASIN, 2007).

Além da otimização de servidores web, desafios surgem no projeto de cada componente que integra um sistema, a fim de melhorar a experiência do usuário e a utilização dos recursos. No lado cliente, as versões mais recentes dos navegadores web estão operando cada vez melhor, focando em velocidade e segurança NOVAK (2017). Já o projeto da arquitetura dos servidores planeja o uso de *cache*, replicação e escalabilidade, levando em consideração diversas premissas, dentre as quais podemos citar: regionalidade do conteúdo, tipo de recurso que será ofertado, seja *stream* de vídeos, conteúdos estáticos, sazonalidade de acesso, visando atender a solicitações em um tempo de resposta rápido,

taxa de transferência satisfatória, baixa taxa de falhas e alta disponibilidade. O tempo de carregamento de uma página é especialmente crítico para sites de comércio eletrônico, pois a espera por conteúdo estimula evasão do usuário e, consequentemente, impacta na taxa de conversão e na satisfação do cliente (STADNIK; NOWAK, 2018).

Neste contexto, soluções em nuvem são alternativas para redução de custos na sustentação de servidores web de alta demanda, por permitirem a expansão de recursos sob demanda de forma veloz. Ao alocar a infraestrutura de um servidor em serviço de nuvem, os recursos são provisionados pelos padrões de carga de trabalho e taxas de utilização, diminuindo a subutilização de *hardware* (Hwang et al., 2016). Outra estratégia complementar e amplamente adotada para preservar o processamento do servidor web é a utilização de *cache*. O *cache* é um armazenamento temporário de informação em memória, que permite guardar a resposta de uma solicitação para retorná-la posteriormente caso a mesma solicitação seja feita. Essa técnica possibilita reduzir latência e o re-processamento da solicitação no servidor.

A avaliação da qualidade ou desempenho de um sistema envolve principalmente o conhecimento do sistema como um todo para definição da metodologia mais adequada, escolha das ferramentas de análise, delimitação da carga de trabalho e métricas. As etapas para definição de uma análise de desempenho, se dividem principalmente em técnicas de modelagem, técnicas de simulação e técnicas de medição (SILVA, 2015). A avaliação da aplicação dessas técnicas pode trazer respostas sobre qual sistema de *cache* melhor se encaixa em uma dada aplicação web.

## 1.1 Justificativa do tema

A motivação para escolha do tema proposto neste trabalho, se deu principalmente ao meu interesse pessoal e profissional ao deparar com aplicações web que recebem um grande volume de requisições. Essas aplicações necessitam utilizar variadas técnicas de *cache* em nível de aplicação, visando reduzir a quantidade de consultas no servidor e o tempo de resposta. O estudo apresentado por Zulfa, Fadli e Wardhana (2020), reforça que criar um sistema de *cache* a partir nível de banco de dados pode reduzir gargalos e a latência na rede.

Um outro fator estimulante para seguir neste estudo, é a contemporaneidade do tema. Conforme abordado por Silva-Muñoz, Franzin e Bersini (2021), bancos de dados não relacionais (como Cassandra e Redis) tem ganhando evidência nos últimos anos, devido à sua alta disponibilidade e aplicabilidade como *cache* de aplicação. Ademais, explorar e experimentar uma proposta nesta área de conhecimento, seria uma oportunidade de consolidar a relação efetiva de cenários do mundo real com unidades curriculares da graduação, tais como banco de dados, sistemas distribuídos, entre outras.

## 1.2 Objetivo geral

O propósito deste trabalho é comparar diferentes abordagens de implementação de técnicas de *cache* em uma aplicação web, a fim de analisar o desempenho dos cenários e entender qual é a mais adequada para a aplicação web proposta.

### 1.2.1 Objetivos específicos

- Definir, dentre as opções de técnicas de *cache* disponíveis, quais se adéquam ao cenário proposto.
- Implementar uma aplicação web simples, utilizando tecnologias empregadas em cenários reais e comparando cada uma das abordagens de *cache* propostas.
- Aplicar a técnica de aferição por medição, utilizando um software de geração de requisições para fornecimento de carga de trabalho.
- Avaliar o desempenho das abordagens de *cache*, com base nos indicadores de tempo de resposta (pelo ponto de vista do usuário final) e a taxa de uso de objetos armazenados em *cache* (*Hit Rate*).

## 1.3 Estrutura do trabalho

O capítulo 2 possui a fundamentação teórica, apresentando os principais conceitos citados nesse trabalho. Já o capítulo 3 detalha o sistema implementado e a configuração do cenários de *cache*. O capítulo 4 descreve os testes realizados e a análise dos resultados. Por fim, o capítulo 5 apresenta a conclusão e considerações finais

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os conceitos básicos e as tecnologias citadas por este trabalho. Inicialmente, uma abordagem sobre os elementos que compõem um sistema web. Na sequência os serviços que fornecem uma melhoria ao sistema em cenários de alta demanda e por fim as técnicas de avaliação de desempenho e métricas que serão utilizadas na análise.

## 2.1 A web

Idealizada com o objetivo de compartilhar informações entre cientistas e pesquisadores, a web teve seu primeiro protótipo apresentado em 1991, e após o lançamento do primeiro navegador web gráfico, o Mosaic, em 1993, cresceu rapidamente, contando com cerca de de 1,7 bilhões de páginas web disponíveis em poucos anos de existência (CâNDIDO; BUENO; SANTOS, 2020). Conceitualmente, a web é um padrão de arquitetura para implementação de um sistema capaz de acessar conteúdo vinculado como uma teia (cuja tradução para o inglês é *web*), por meio de links, no qual conteúdo é armazenado em servidores (S.TANENBAUM; J.WETHERALL, 2002). Sendo assim, é um sistema complementar da Internet em conjunto com serviços como *File Transfer Protocol* (FTP), aplicações *mobile* e *e-mail*.

A estrutura tradicional da web segue o modelo cliente-servidor, no qual computadores clientes acessam uma aplicação *front-end* por meio de um navegador web, sendo este responsável por realizar requisições via protocolo *Hypertext Transfer Protocol* (HTTP) a um servidor web e renderizar graficamente a resposta. Cada elemento que compõe uma página web (seja imagem, página HTML, *scripts* entre outros) é denominado `objeto web`. As aplicações de *back-end* são responsáveis por prover determinado recurso ou operação, tratando as requisições simultâneas e o acesso ao banco de dados. Geralmente, servidores *Domain Name System* (DNS) são incluídos no sistema web a fim de facilitar a localização das aplicações, visto que são responsáveis por traduzir uma *Uniform Resource Identifier* (URI) amigável à um endereço IP. Conforme definição apresentada por Dissanayake e Dias (2017), diferentemente de aplicações desktop, onde todos os componentes do aplicativo estão localizados dentro do dispositivo e são executados em um único espaço de endereço da memória do sistema, uma aplicação web pode ser definida como:

> ... um sistema, com componente(s) do aplicativo no lado do cliente, que se comunica com o(s) componente(s) do aplicativo em um servidor web, para processamento de dados. Eles utilizam o serviço da web, com base

na arquitetura cliente-servidor, modelo de solicitação-resposta, HTTP padrão e outras técnicas e tecnologias relacionadas.

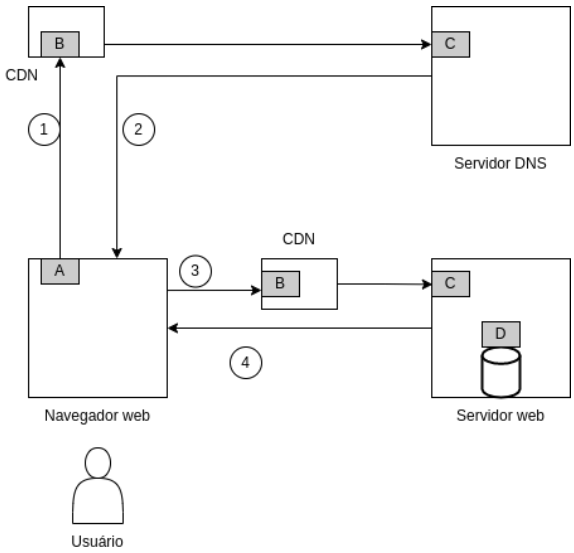
As aplicações web clientes também podem ser categorizadas como baseada em navegador (ou *browser-based*) ou não baseadas em navegadores (ou *non-browser-based*) (DISSANAYAKE; DIAS, 2017). As aplicações baseadas em navegadores utilizam o navegador (que é uma aplicação *desktop*) para comunicação com o servidor. Aplicações *desktop* podem nativamente se comunicar com o servidor, sem o intermédio de um navegador, como por exemplo as aplicações implementadas na plataforma Java EE. Já as aplicações híbridas, são aplicações implementadas utilizando javascript, *Hypertext Markup Language* (HTML) e *Cascading Style Sheets* (CSS), mas que também utilizam recursos da máquina local ou dispositivos móveis.

Em aplicações web predominantemente *Server Side Rendering* (SSR) o processamento das regras de negócio da aplicação, consulta a banco de dados e criação da página HTML são realizados no servidor web. Linguagens comuns para implementações *server-side* são o *Java Server Pages* (JSP) e o *Hypertext Preprocessor* (PHP). O PHP é amplamente usado para implementar aplicativos da web, em parte devido ao seu suporte de biblioteca para interação de rede, processamento HTTP e acesso a banco de dados, gerando páginas HTML prontas que são renderizadas pelo navegador web (ARTZI et al., 2010). Esse tipo de arquitetura evita a transmissão de dados sensíveis, como senhas de acesso a banco de dados e *tokens* pela rede. No entanto, esta prática de renderização pode não ser adequada em cenário de elevada interatividade, uma vez que cada interação do usuário com a aplicação exigirá a realização de um novo pedido ao servidor para obter o HTML atualizado (BASTOS, 2020).

Como a tendência da web são páginas cada vez mais interativas, e dada a necessidade de diminuir o tempo de renderização de objetos HTML, abordagens *Client Side Rendering* (CSR) foram ganhando relevância nos últimos anos. Segundo Maniezzo et al. (2019), as plataformas móveis amadureceram a um ponto em que podem fornecer a infraestrutura necessária para suportar códigos de otimização complexos, desta forma, novos padrões de desenvolvimento transferem para o navegador a responsabilidade de processar regras de negócio, consultar *Application Programming Interface* (API) externas e solicitar objetos ao servidor quando necessário. O padrão CSR tem a vantagem de reagir mais rapidamente às interações do usuário, devido a não estar dependente de sucessivos acessos ao servidor para renderização completa da página (BASTOS, 2020).

Na Figura 1 é apresentado o fluxo de básico de uma comunicação via web, no qual um usuário especifica uma *Universal Resource Locator* (URL) (por exemplo: <https://www.worldometers.info/coronavirus/>) em seu navegador. Essa URL identifica o servidor web e o recurso que o usuário deseja acessar.

Figura 1 – Fluxo de requisições em um sistema web.




Fonte: Elaborado pela autora.

Primeiramente o navegador realiza uma consulta a um servidor DNS (1), que retorna o endereço *Internet Protocol* (IP) correspondente ao endereço da URL especificada. Nesta etapa ocorre uma das mais importantes causas de atraso nas solicitações web, pois para traduzir o nome do *site* ao seu respectivo endereço IP é necessário uma consulta a um banco de dados, processo conhecido como resolução DNS. Após obter endereço IP (2) do servidor que contém o recurso, o navegador estabelece uma conexão com o servidor (3) e consulta a informação no caminho especificado na URL. No servidor web há um custo temporal de processar a informação, o qual pode ser ainda maior se a requisição solicitar um conteúdo dinâmico, ou seja, conteúdo que requer acesso em um banco de dados, ou exigir o processamento de uma aplicação no servidor. A resposta é entregue ao usuário (4) por meio do protocolo HTTP, e seu conteúdo pode ser uma página HTML, imagens, áudio, vídeo entre outros tipos de conteúdo. Durante todo o percurso da mensagem pela rede, a solicitação é suscetível aos gargalos e latência de rede que podem elevar o tempo total de resposta da solicitação.

Também com objetivo de aprimorar usabilidade da aplicação web do ponto de vista do usuário, o *cache* é um componente de desempenho crucial de um sistema web de grande escala, pois ajuda a reduzir os tempos médios de resposta de consulta e as cargas de trabalho de processamento de consulta em *clusters* de pesquisa de *back-end* (LEE; KIM, 2012). Na Figura 1 também são apresentadas as etapas de *cache* que uma requisição pode ser submetida até obter uma resposta do servidor web. A primeira etapa é o *cache* do navegador (A) que armazena na memória do computador cliente, arquivos estáticos que são consultados com frequência. Durante a trajetória até o servidor de DNS ou até o servidor web, a requisição pode ser atendida por uma *Content Delivery Network* (CDN) (B), que armazena replicas das informações contidas nos servidores a fim de reduzir o trajeto da

requisição pela rede. Na borda dos servidores (C), a técnica de *cache* armazena a resposta de requisições que já foram solicitadas anteriormente e envia a resposta armazenada, evitando um reprocessamento da solicitação pela aplicação de *back-end*. Por fim, o *cache* de dados (D) tem o objetivo de evitar leitura no banco dados pela aplicação de *back-end*.

### 2.1.1 Protocolo HTTP

O protocolo HTTP é um protocolo de camada de aplicação, responsável por definir os padrões de sintaxe e a semântica da comunicação entre servidores e navegadores. Baseado em modelo solicitação e resposta, isto é, *stateless*, ele não armazena nenhuma informação sobre conexões anteriores (OVIEDO et al., 2019). Através de *cookies* HTTP é possível que as sessões compartilhem informações, permitindo que o servidor reconheça uma navegação de um mesmo usuário, por exemplo. A estrutura de uma solicitação HTTP consiste em solicitação do cliente (método, URL e versão do protocolo), cabeçalho e corpo da solicitação, enquanto a estrutura da resposta é composta pela linha de status, o cabeçalho e conteúdo resposta (MIN; FU, 2016). Desde a sua implementação inicial o protocolo está na sua terceira versão e atualmente a web se comunica principalmente com as versões HTTP1.1 e HTTP2, definidas a seguir:

- HTTP/1.1: Vigente desde 1997, essa versão é caracterizada por manter uma conexão persistente (com cabeçalho *keep-alive*), permitindo múltiplas requisições em uma mesma conexão *Transmission Control Protocol* (TCP). O HTTP/1.1 *pipelining*, permite ainda envio de mais de uma requisição de forma não sequencial, ou seja, uma nova solicitação pode ser enviada, antes que a resposta da solicitação anterior tenha sido retornada (MATTSON; GHOSH, 2009). Para prover paralelismo nas requisições, o HTTP/1.1 permite a abertura de várias conexões TCP simultâneas entre cliente e servidor, no entanto, um grande número de conexões redundantes gera em uma grande quantidade de dados duplicados, podendo afetar o desempenho da rede. (AL-SHARIF, 2015)

- HTTP/2: Essa versão do protocolo permite a compactação de cabeçalhos e encapsulamento binário (ao invés de textual), reduzindo o tamanho da mensagem que trafega pela rede. Implementa também a multiplexação para permitir interações bilaterais simultâneas em uma mesma conexão TCP, segmentando o fluxo de mensagens HTTP em diferentes *streams* através do uso de *frames* (OLIVIERA, 2016). Como o paralelismo ocorre em uma mesma conexão TCP, essa versão do protocolo reduz o tráfego e a sobrecarga de informações redundantes na rede. (AL-SHARIF, 2015).

Em uma conexão via protocolo HTTP, os recursos são localizados através de uma request-URI. A request-URI é o URI do recurso ao qual a solicitação se aplica e pode

possuir ou não um parâmetro de autoridade (aplicável apenas método CONNECT). O protocolo HTTP possui um conjunto de métodos de requisição conhecidos também como verbos HTTP, que segundo ás RFC2616, RFC7231 e a RFC5789 são definidos da seguinte forma:

- GET: Utilizado para obter um recurso do servidor. Com o objetivo de reduzir o uso desnecessário da rede, um método GET pode possuir um cabeçalho condicional, onde solicita a transferência da entidade apenas nas circunstâncias descritas pelo(s) campo(s) de cabeçalho condicional (Exemplo: *If-Modified-Since*, *If-Unmodified-Since*, *If-Match*, *If-None-Match* ou *If-Range*). Já o método GET parcial também destina-se a reduzir o uso desnecessário da rede, permitindo que entidades parcialmente recuperadas sejam concluídas sem a transferência de dados já mantidos pelo cliente.
- POST: A função real executada pelo método POST é determinada pelo servidor e geralmente depende da URI de solicitação.
- PUT: Solicita que a entidade incluída seja armazenada no URI de solicitação fornecido. Se o request-URI se referir a um recurso já existente, a entidade deve ser considerada como uma versão modificada daquela que reside no servidor de origem.
- DELETE: Solicita que o servidor de origem exclua o recurso identificado pela Request-URI.
- HEAD: Idêntico ao GET, exceto que o servidor retorna um conteúdo na resposta. Este método pode ser usado para obter metainformações sobre a entidade implícita na solicitação, sem transferir o próprio corpo da entidade. Este método é usado para testar links de hipertexto quanto à validade, acessibilidade e modificações recentes.
- OPTIONS: Representa um pedido de informação sobre as opções de comunicação disponíveis na cadeia de pedido / resposta identificada pela request-URI. As respostas a este método não podem ser armazenadas em cache.
- TRACE: Usada como mecanismo de depuração. Retorna a mensagem solicitada a fim de identificar se houve alterações por servidores intermediários.
- PATCH: Usado para aplicar modificações parciais a um recurso. Ele solicita que um conjunto de mudanças descritas na entidade de solicitação seja aplicada ao recurso identificado pela request-URI.
- CONNECT: solicita que o destinatário estabeleça um túnel para o servidor de origem de destino identificado pelo destino da solicitação. Túneis são usados para criar conexões virtuais de ponta a ponta, por meio de um ou mais *proxies*, que podem ser protegidos usando *Transport Layer Security* (TLS)

O protocolo HTTP implementa campos no cabeçalho de requisições e respostas a fim de auxiliar na identificação de conteúdo que pode ser armazenado em *cache*. O campo `cache-control` é uma diretiva que define várias configurações como quanto o tempo uma resposta pode ser utilizada pelo navegador (campo *max-age*), se a resposta não pode ser reutilizada sem verificar se houve alterações no servidor (campo *no-cache*), se a resposta não deve ser armazenada em *cache* nem reutilizada (campo *no-store*). O campo `age` indica quanto tempo a informação permaneceu armazenada em *cache* no proxy e o `expires` define uma data e hora na qual a informação é considerada obsoleta.

O *cache-control* também fornece orientações para cabeçalhos de resposta indicando se o conteúdo pode ser compartilhado (campo *public*) ou usado apenas pelo *browser* que fez a solicitação (campo *private*). O cabeçalho de resposta `Vary` determina que os destinatários não devem usar esta resposta para satisfazer um pedido futuro, a menos que o pedido posterior tenha os valores para os campos listados como a solicitação original. Também é possível a inclusão de uma *tag* de entidade (*e-tag*) no cabeçalho da resposta HTTP, para identificar as várias representações do mesmo recurso, permitindo que servidor web não precise reenviar uma resposta completa se o conteúdo, validado pelo código da *e-tag*, não mudou. Ao receber o pedido, o servidor envia de volta um código de status e, opcionalmente, o conteúdo no restante do corpo da resposta. Cada classe de status tem como objetivo indicar à a aplicação requisitante se a solicitação pode ou não ser atendida, a fim de possibilitar uma tratativa para uma possível falha. As classes de status possíveis são:

- 1XX: Resposta de natureza informativa, indica uma resposta temporária para comunicar o status da conexão ou o andamento da solicitação antes de completar a ação solicitada.
- 2XX: Indica que a solicitação foi bem-sucedida e o conteúdo dependerá do tipo de método HTTP utilizado.
- 3XX: Especifica que a solicitação possui mais de uma resposta possível. O servidor permite que o cliente opte pela representação apropriada.
- 4XX: Informa que o servidor não pode ou não processará a solicitação devido a algo que é percebido como um erro de cliente
- 5XX: Indica que o servidor falhou ou é incapaz de realizar o método solicitado.

O status de resposta também é um indicativo de que o conteúdo deve ou não ser armazenado em cache. As requisições comumente armazenadas, segundo Mozilla (2021), são requisições GET bem sucedidas (status 200), redirecionamentos permanentes (status 301), requisições mal sucedidas (status 404) e requisições com respostas incompletas (status 206).

### 2.1.2 Servidores web

Segundo a Nginx (2021) servidores web são computadores capazes de armazenar e servir conteúdos estáticos e dinâmicos. Logo, computadores de diversas capacidades e sistemas operacionais podem operar como servidores web, desde computadores pessoais, Raspberrys Pi até *clusters* de grande porte alocados em nuvem. A capacidade de hardware requerida dependerá da quantidade de conteúdo, disponibilidade, requisições simultâneas entre outras características que o servidor deverá atender. O tipo de conteúdo também é um fator importante pois poderá implicar em uma maior necessidade de processamento e memória. Conteúdos estáticos podem ser páginas HTML, imagens, arquivos de fontes, texto, folhas de estilo (CSS) e entre outros arquivos que não sofrem alteração a cada requisição e estão armazenados no sistema de arquivos do servidor. Já os conteúdos dinâmicos podem ser páginas HTML geradas dinamicamente, consulta de dados baseada nas informações de formulários recebidos pela requisição do cliente ou qualquer recurso que necessite de processamento por uma aplicação em execução no servidor.

A integração da comunicação entre a aplicação cliente e servidor ocorre através de serviços web (ou *web services*) que são aplicações modulares que podem ser invocadas através da rede. Segundo a IBM (2022), um serviço web possui uma interface que oculta os detalhes de implementação para que possa ser usado independentemente da plataforma de hardware ou software em que é implementado e independentemente da linguagem de programação em que está escrito. Desta forma, para que tanto a aplicação cliente quanto a aplicação do servidor possam ser fracamente acopladas em relação a linguagem de desenvolvimento, a mensagens precisam seguir um padrão para tornar a comunicação possível.

Os padrões de troca de mensagens amplamente utilizados na web são protocolo *Simple Object Access Protocol* (SOAP) ou padrão *Representational State Transfer* (REST). O SOAP é fundamentalmente um paradigma de troca de mensagens unilateral sem estado, que atua sobre HTTP, utilizando o padrão *Extensible Markup Language* (XML). A comunicação através do padrão é REST é mais simples e dispensa a necessidade de um *Universal Description Discovery and Integration* (UDDI), pois cada serviço corresponde a um recurso, e cada recurso corresponde a um URI que deve ser única e deve estar armazenada em um único local (RICHARDSON; RUBY, 2007). O REST também permite a troca de mensagens em diversos formatos, como HTML, XML, textual e *JavaScript Object Notation* (JSON).

Servidores web também se baseiam em alguns paradigmas importantes da computação para definir sua estrutura de tratamento de conexões:

- Servidores baseados em processos ou `Process-Based`: iniciam um novo processo a cada nova conexão. A vantagem desse tipo de arquitetura é a simplicidade no

tratamento das solicitações, visto que as tarefas ocorrem de forma sequencial, de forma isolada (em relação à outros processos) e não compartilham memória.

- Servidores baseado em eventos ou `Event-Driven`: também conhecidos como *Single-Process Event-Driven* (SPED), uma única *thread* é responsável por gerenciar um *pool* de encadeamentos de acordo com as notificações recebidas. Essa arquitetura, é do tipo não bloqueante, ou seja, o encadeamento retorna ao *pool* de encadeamentos (conjunto de processos encadeados) antes que a resposta seja concluída (método assíncrono), podendo atender outra solicitação, tornando esse modelo mais eficiente para atender alta demanda de requisições.

A quantidade de solicitações simultâneas que um computador consegue atender acompanhou a evolução de capacidade disponível em hardware dos computadores, principalmente em relação à memória e processamento. No ano de 2000 Dan Kegel relatou em seu blog pessoal a limitação técnica dos servidores web da época em atender mais de 10 mil conexões simultâneas em uma única máquina, o desafio ficou conhecido como *C10k problem* (ou problema das 10 mil conexões) (KEGEL, 2000). O problema foi resolvido em 2004, após o lançamento do servidor NGINX (cuja arquitetura é *event-based*) capaz de atender essa demanda (SONI, 2016). No ano de 2013, Robert Graham publicou um manifesto apresentando a dificuldade naquele ano em atender 10 milhões de conexões simultâneas em uma única máquina, tal manifesto relatou o desafio de *C10M problem*(ou problema das 10 milhões de conexões simultâneas) em referência ao clássico problema apontado por Dan Kegel (GRAHAM, 2015), e segundo o artigo de Rotaru, Olariu e Rivière (2017) o servidor MigratoryData lançado em 2015 é capaz de atender a demanda.

### 2.1.3 Proxy

A existência de *proxy* em um sistema web indica que há uma entidade intermediária que atua entre o servidor e o cliente capaz de analisar conteúdo das solicitações. Segundo, Oliviera (2016), um servidor *proxy* deve interpretar e, se necessário, modificar mensagens HTTP antes de encaminhá-las.

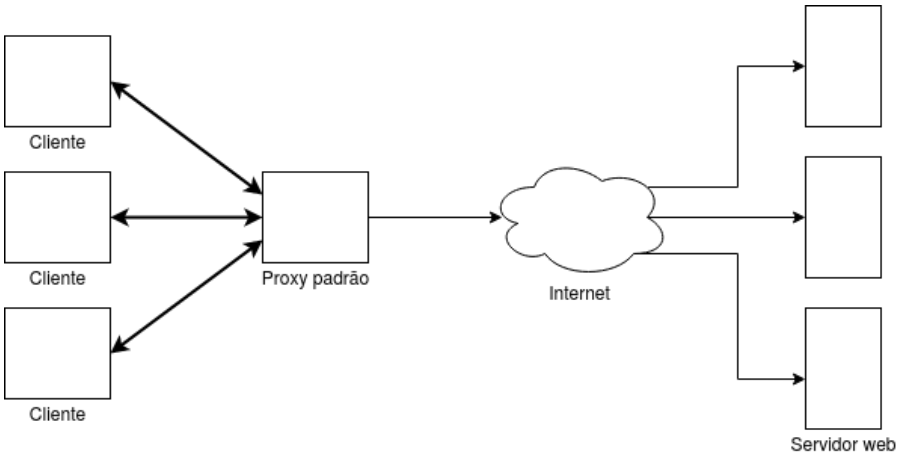
#### 2.1.3.1 Proxy padrão ou proxy de encaminhamento

Um *proxy* de encaminhamento tem como objetivo centralizar as solicitações da web de clientes, podendo acelerar o tempo de resposta de requisições (provendo informação armazenda em *cache*) ou filtrar a conectividade de clientes.

Se um objeto solicitado for encontrado no *cache*, o *proxy* servirá o objeto do *cache*, caso contrário, ele recuperará o objeto solicitado do servidor web de origem e o armazenará no *cache*. Este cenário é exemplificado na Figura 2, onde o servidor *proxy* pode ficar na

borda da rede local (como de uma empresa, por exemplo) podendo aplicar políticas de segurança, bloqueios e análise de tráfego.

Figura 2 – Proxy padrão



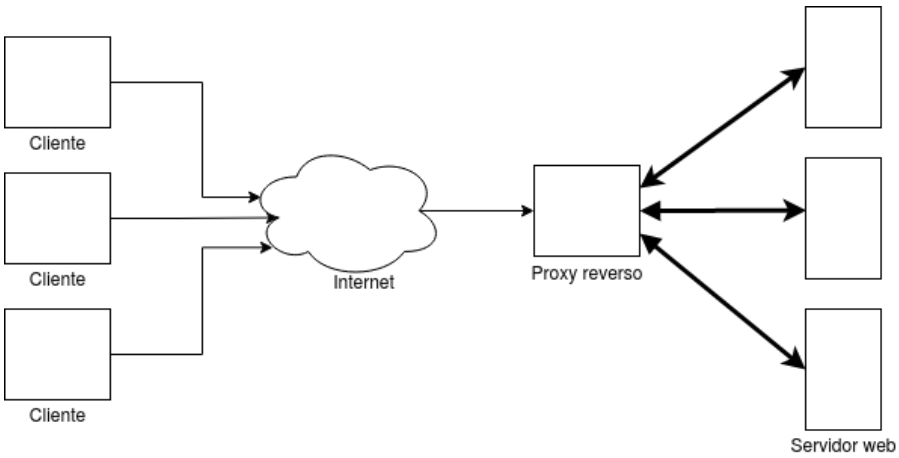

Fonte: Elaborado pela autora.

#### 2.1.3.2 Proxy reverso

Um *proxy* reverso é geralmente colocado diretamente na frente de um grupo de servidores para reduzir o número de solicitações que os servidores devem manipular (DATTA et al., 2003).

A Figura 3 representa um servidor de proxy reverso atuando como distribuidor de requisições entre um servidores web. Nesta configuração, os diversos servidores web são percebidos pela aplicação de forma transparente, ou seja, como se fossem um único serviço respondendo as solicitações.

Figura 3 – Proxy reverso




Fonte: Elaborado pela autora.

### 2.1.4 *Content Delivery Network* (CDN)

Uma rede de entrega de conteúdo, também conhecida como CDN, objetiva reduzir a distância física entre o cliente e o conteúdo do servidor, replicando e copiando os objetos web em locais estratégicos, por diversas cidades do mundo. O serviço de CDN é contratado ou implementado pelo servidor da aplicação e realizado através de um redirecionamento do DNS do servidor de aplicação para o domínio de CDN da empresa contratada. Desta forma, a camada da CDN, também permanece transparente ao cliente, visto que a aplicação cliente solicita o recurso através da URL e resposta poderá ser fornecida por uma das réplicas da CDN ou até pelo servidor original.

Empresas como Amazon, Akamai e CloudFlare fornecem a infraestrutura de CDN como serviço, alugando a estrutura física sob demanda. Prover *stream* de video costuma ser um desafio para os servidores, visto a necessidade de manter uma alta taxa de transferência de dados (medida por *bitrate* ou bits por segundo) e alta disponibilidade. A Netflix, empresa provedora de *stream* de vídeo, possui seu próprios servidores responsáveis distribuir o conteúdo de forma regionalizada, alocando, próximo aos clientes, conteúdos frequentemente acessados na região ou produzidos no país.

## 2.2 Computação em nuvem

A computação em nuvem pode ser definida como um conjunto de recursos computacionais disponibilizados na rede que possibilitam abstração da infraestrutura de hardware e também a virtualização de software (PEREIRA et al., 2016). Esse modelo facilita escalabilidade de infraestrutura que mantém a aplicação, pois permite a expansão de recursos conforme a necessidade, tornando-o indicado para cenários onde a carga de trabalho pode variar com frequência. Servidores web, por exemplo, podem receber picos de solicitação a um recurso e permanecer ociosos (ou com baixa demanda) em outros momentos. A ociosidade da infraestrutura indica subutilização ou desperdício de recursos, tais como energia elétrica, memória e capacidade de processamento.

Segundo, Mell e Grance (2010) a computação em nuvem é um paradigma que pode ser aplicado a diferentes tipos de serviços, como redes, servidores, armazenamento, aplicativos e serviços e deve garantir as seguintes características:

- Serviço sob-demanda auto servido (*On-Demand Self-Service*): Um cliente pode provisionar recursos computacionais como servidores ou armazenamento, à medida que o recurso for necessário, e de maneira automática, sem a necessidade de interação humana com o provedor de cada serviço.
- Amplo acesso à rede (*Ubiquitous Network Access*): Os recursos estão disponíveis na rede e são acessados por meio de plataformas heterogêneas (acessíveis via celular ou

notebook, por exemplo).

- Compartilhamento de recursos (*Resource Pooling*): Recursos físicos e virtuais atribuídos e reatribuídos dinamicamente de acordo com a demanda do consumidor e este não tem controle ou conhecimento sobre a localização exata dos recursos fornecidos.
- Rápida elasticidade (*Rapid Elasticy*): Os recursos podem ser provisionados de forma rápida e elástica, em alguns casos automaticamente.
- Serviço contabilizado (*Measured Service*): O uso de recursos pode ser monitorado, controlado e relatado, proporcionando transparência tanto para o provedor quanto para o consumidor do serviço utilizado.

A categorização dos serviços de um sistema em nuvem pode ser definida pelo nível de abstração em que utilizador consome demanda computacional, segundo Mell e Grance (2010), as principais categorizações são:

- FaaS (Function as a Service): Possibilita que os desenvolvedores executem trechos de código em resposta a eventos utilizando o processamento em nuvem, dispensando necessidade de um servidor tratando as solicitações. Os serviços em nuvem populares que possibilitam esse modelo de serviço são: AWS Lambda, Google Cloud Functions, Microsoft AzureFunctions.
- SaaS (*Software as a Service*): O recurso provido ao consumidor é uma aplicação sendo executada sobre uma infraestrutura de nuvem, ou seja, ao invés de instalar a aplicação na própria máquina o usuário pode acessá-la e utilizá-la remotamente. Algumas aplicações populares desenvolvidas nesse modelo são Netflix e o Dropbox.
- PaaS (*Platform as a Service*): O desenvolvimento da aplicação é feito usando as linguagens de programação, bibliotecas, serviços e ferramentas suportadas e disponibilizadas pelo provedor. Os consumidores desse tipo de modelo de serviço são normalmente desenvolvedores ou empresas proprietárias de aplicações.
- IaaS (*Infrastructure as a Service*): Nesse modelo o consumidor tem controle sobre os sistemas operacionais, armazenamento, aplicativos implantados e, possivelmente, controle limitado de componentes de rede *firewalls*. Empresas como *Amazon Web Services* (AWS), Red Hat OpenShift, Google Cloud, Windows Azure Cloud oferecem os serviços de infraestrutura e também proveem soluções para as camadas de abstração PaaS e SaaS.

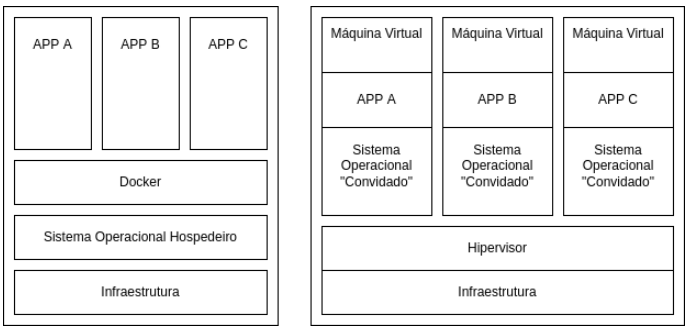
Segundo a RedHat (2018), um sistema em nuvem pode ser classificado como:

- Nuvem privada: ambientes de nuvem dedicados a um usuário final. Todas as nuvens se tornam privadas quando a infraestrutura é dedicada e o cliente tem acesso totalmente isolado a ela.
- Nuvem pública: são ambientes de nuvem geralmente criados em uma infraestrutura de TI que não é de propriedade do usuário final. Eles podem ser gratuitos ou vendidos sob demanda, permitindo que os clientes paguem apenas pelo seu consumo de ciclos de CPU, armazenamento ou largura de banda.
- Nuvem híbrida: ambiente de TI aparentemente único criado a partir de vários outros ambientes conectados por redes locais (ou *Local Area Network* (LAN)), redes de área ampla (ou *Wide Area Network* (WAN)), redes privadas virtuais (ou *Virtual Private Network* (VPN)) e/ou API.

### 2.2.1 Virtualização

A virtualização é a técnica que permite particionar um único sistema computacional em vários outros, denominados de máquinas virtuais (CARISSIMI, 2008). Dentre os principais benefícios da técnica, destacam-se o isolamento entre os sistemas em execução dentro de cada máquina virtual e o compartilhamento de recursos. Além da virtualização, existe a técnica de conteinerização que ao invés de isolar todo o sistema operacional, isola apenas a aplicação e suas dependências.

Figura 4 – Contêiner x Máquina Virtual




Fonte: Docker (2022), traduzido pela autora.

A Figura 4 apresenta um comparativo entre os dois modelos. No modelo de conteinerização várias aplicações contêineres executam em uma mesma máquina, compartilhando o kernel do sistema operacional, enquanto a máquina virtual inclui uma cópia completa de um sistema operacional, o aplicativo, binários e bibliotecas necessários

#### 2.2.1.1 Docker

O Docker é uma tecnologia de conteinerização, onde contêiner são ambientes isolados para a execução de aplicações e serviços (REDHAT, 2021). A implementação de aplicações

utilizando Docker evita problemas como versões divergentes em bibliotecas e dependências, além de facilitar a portabilidade da aplicação entre várias máquinas. Para replicar a aplicação, com todas as suas dependências é gerada uma imagem Docker, que segundo a documentação do software Docker, é um pacote de software leve, autônomo e executável que inclui tudo o que é necessário para executar um aplicativo: código, ferramentas do sistema, bibliotecas do sistema e configurações(DOCKER, 2021).

O Docker possui um repositório público chamado Docker Hub onde empresas e pessoas podem compartilhar suas imagens. É possível, por exemplo, encontrar imagens prontas com a pilha Linux, Apache, MySQL e PHP (LAMP) que são comumente usadas em conjunto para servir aplicações web. Após criada, uma imagem é imutável, ou seja, qualquer alteração na configuração da imagem não será mantida caso o contêiner seja encerrado. Para manter uma alteração realizada em uma imagem, deve-se gerar uma nova imagem.

Para administrar vários contêiners o Docker possui uma ferramenta chamada Docker Compose, conhecido também como coordenador de contêineres. Ele é um arquivo de configuração onde são definidas as parametrizações de cada serviço, como portas e diretório de arquivos. Utilizando o Docker Compose é possível especificar as imagens que irão compor uma aplicação e quais arquivos serão executados ao iniciar cada serviço.

## 2.3 Banco de dados

Segundo a Oracle (2022), um banco de dados é uma coleção organizada de informações ou dados estruturadas, normalmente armazenadas eletronicamente em um sistema de computador. O gerenciamento e controle de acesso desses dados armazenados, é realizado através de um Sistema Gerenciador de Banco de Dados (SGBD). O armazenamento dos dados em *hardware* pode ser realizado em diferentes níveis de hierarquias. Segundo Sudarshan, Silberschatz e Korth (2006), o tipo primário inclui as mídias que operam diretamente sobre o controle da CPU como a memória *Random-Access Memory* (RAM), as quais são mais rápidas, porém limitadas e mais caras em comparação com as memórias do tipo secundário que incluem disco rígido (*Hard Disk* (HD)). Por fim, o nível terciário engloba dispositivos removíveis e ópticos.

As atribuições de um SGBD envolvem, dentre outras coisas, a movimentação dos dados da memória para o processador. Esse processo de leitura e gravação em dispositivos como disco rígido é um dos pontos de latência da operação, devido à limitações de *hardware*. Kepe (2019) aponta que a transferência de dados para o processador dentro da CPU, afeta não só o tempo de execução da consulta como também consumo de energia do sistema. Devido à necessidade de operar quantidades cada vez maiores de dados, diversas técnicas são aplicadas a fim de reduzir o tempo de execução das consultas. Segundo, Silva (2010)

algumas delas são:

- A criação de algoritmos sofisticados de classificação que exploram a quantidade de memória disponível.
- Utilização de estruturas auxiliares de acesso aos dados conhecidas como índices. Um índice é uma estrutura em disco ou em memória, associada a uma tabela ou exibição que acelera a recuperação de linhas de uma tabela ou exibição (MICROSOFT, 2022).
- Particionamento de segmentos de dados, visando diminuir o conjunto de pesquisa com base em um critério conhecido (normalmente o tempo).
- Armazenamento dos dados lidos em áreas intermediárias compartilhadas de memória cache.

O processamento de grandes volumes, fluxos rápidos e dados complexos em um tempo limitado exige elasticidade também dos sistemas de banco de dados (WU; NI; XIAO, 2021). Apesar das diversas vantagens sobre o uso de softwares em ambientes virtualizados, a virtualização de SGBD apresenta algumas ressalvas. Segundo BINI (2016), as técnicas de otimização e até o mesmo o desempenho do SGBD pode ser afetado, devido à camada adicional de virtualização em dispositivos como disco rígido, o autor também aponta que:

> Uma vez que os SGBDs não foram inicialmente projetados para serem executados em ambientes virtualizados, seu modelo de custos, base para a tomada de decisões e otimizações, não levam em consideração sua execução em ambientes elásticos, que implicam em um provisionamento dinâmico de recursos. A existência de cargas de trabalho concorrentes, oriundas de outras Máquinas Virtuais (MVs) sobre um mesmo hardware, também é um exemplo que não se pode ignorar. E necessário que o mecanismo de auto-configuração do SGBD reconheça essas variações que ficam de posse do Monitor de Máquinas Virtuais (MMV), responsável pelo escalonamento de recursos. Para isso é preciso conceber uma nova arquitetura de custos para os SGBD.

Segundo Wu, Ni e Xiao (2021), a dificuldade de alocar banco de dados relacionais em *clusters* elásticos foi um dos fatores que motivou o movimento denominado *Not Only SQL* (NoSQL). As seções a seguir vão descrever as principais diferenças e particularidades entre os dois modelos de dados.

### 2.3.1 Banco de dados relacionais

Banco de dados relacionais utilizam o formato de tabelas para representar os dados, onde cada tabela representa um tipo de registro e as colunas correspondem aos atributos do tipo de registro, por exemplo, em uma tabela do tipo de registro "Cliente"as colunas podem definir um "id_cliente", "Nome", "Endereço"entre outras informações Sudarshan,

Silberschatz e Korth (2006). A consulta de informações nesse tipo de banco de dados é realizada através de *Structured Query Language* (SQL) e visam garantir as premissas Atomicidade, Consistência, Isolamento, Durabilidade (ACID).

- **A**tomicidade: indica que a transação será feita integralmente ou então não será feita;
- **C**onsistência: uma transação deve manter o estado consistente. O estado inconsistente ocorre durante uma transação enquanto a informação está sendo modificada, mas toda a transação é revertida se houver um erro durante qualquer estágio do processo;
- **I**solamento: é a execução sequencial de transações de modo que uma transação não interfira ou acesse concorrentemente a mesma informação que outra transação;
- **D**urabilidade: os dados armazenados devem persistir em caso de falta de luz, falha ou reinicialização da máquina;

### 2.3.2 Banco de dados não relacionais

Um banco de dados do tipo não relacional pode ou não utilizar SQL para acesso das informações e armazenam dados estruturados e não estruturados (LEAVITT, 2010). Normalmente são classificados em quatro categorias: *Key-value Stores*, *Graph Stores*, *Column Stores* e *Document Stores* sendo a mais adequada definida pela natureza da aplicação. Segundo Lourenço et al. (2015), diferentemente dos bancos de dados relacionais, os bancos de dados NoSQL abrem mão do suporte para a transações ACID afim de prover maior disponibilidade, escalabilidade e acessos simultâneos às informações.

Um armazenamento de tipo chave-valor (ou *Key-value*) está representado na Figura 5a. Trata-se de um sistema que define uma chave única para associar à um valor, também chamadas de "tabelas de *hash*", onde recuperação de dados geralmente é realizada usando as chaves associadas (BARON, 2016). O software Redis é um exemplo de software que utiliza esse tipo de armazenamento.

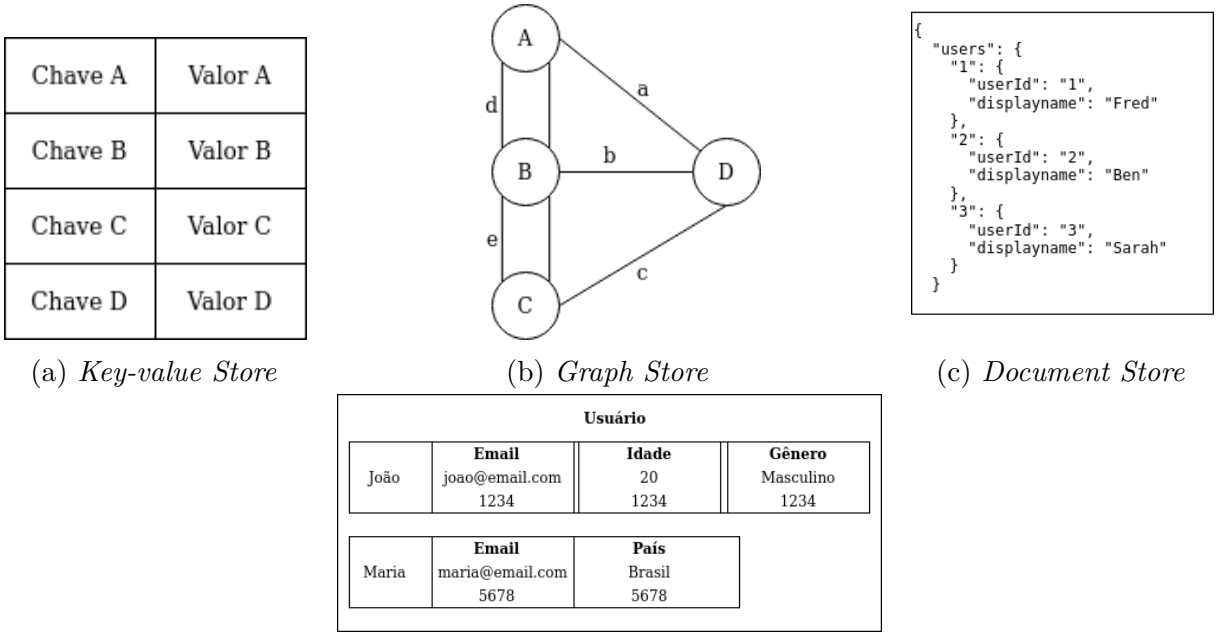
As estruturas baseadas em grafos (ou *Graph stores*) são focados em manipular informações que apresentam relacionamentos complexos com agilidade, definindo nós e arestas conforme a teoria de grafos. A Figura 5b representa uma estrutura de grafos onde os vértices representam as entidades, suas propriedades representam os atributos e as arestas representam as relações (CATARINO, 2017), o software Neo4j é um exemplo de software que utiliza esse tipo de armazenamento.

O armazenamento de colunas (ou *Column stores*), representado na Figura 5d, é indicado para armazenamento distribuído de dados, em grande escala, e é implementada por softwares como Cassandra ou BigTable. Sua organização é semelhante ao banco de dados de relacional mas as suas linhas não precisam possuir as mesmas colunas. Nesse

modelo pode-se formar super colunas que agregam outro conjunto de colunas, tornando-o ideal para relacionamento hierárquico entre entidades, no entanto, costuma exigir *queries* complexas para acesso das informações (SILVA, 2016).

Por fim, o armazenamento orientado à documentos (ou *Document stores*) é semelhante à estrutura chave-valor, no entanto, o valor associado à uma chave pode armazenar dados semiestruturados em formatos como JSON, *Binary JSON* (BSON) e XML (BARON, 2016). Os populares software de banco de dados MongoDB e CouchDB utilizam esse modelo.

Figura 5 – Categorias de bancos não relacionais.




(a) *Key-value Store* (b) *Graph Store* (c) *Document Store*

(d) *Column Store*

Fonte: (CATARINO, 2017)

As diferentes metodologias de implementação de banco de dados (relacionais e não relacionais) podem ser utilizadas de forma complementar, dada a importância de ser preservar as premissas ACID no tratamentos de dados e também da necessidade de acesso rápido à informação exigido por cenários de grande volume de acesso. Essa combinação é aplicada especialmente como estratégia *caching* para acelerar acesso à banco de dados (ZULFA; FADLI; WARDHANA, 2020).

O uso serviços de banco de dados em aplicativos da web também pode exigir que o servidor de banco de dados seja replicado em diferentes locais ou implante uma arquitetura paralela e/ou distribuída. A principal vantagem da replicação de dados é que ela fornece *backup* das informações e disponibilidade em caso de falhas na rede ou em um dos servidores.

#### 2.3.2.1 Banco de dados de séries temporais

Segundo Stonebraker e Cetintemel (2018), embora os banco de dados relacionais, tenham sido historicamente capazes de gerenciar uma ampla gama de cenários, eles foram considerados ineficientes, ou mesmo inadequados, no manuseio da velocidade e do volume das grandes infraestruturas da atualidade. A velocidade de gravação e precisão temporal da métrica é especialmente importante para sistemas de monitoramento e sensores. Os *Time Series Database* (TSBD) são especialmente adaptados à natureza das leituras dos sensores, onde cada entrada está associada a uma marcação de datahora (*timestamp*), podendo representá-los de forma eficiente como uma sequência de valores ao longo do tempo (CALATRAVA et al., 2021). Os softwares InfluxDB e TimescaleDB são exemplos de aplicações que oferecem esse formato de armazenamento de informação.

Conceitualmente, séries temporais são constituídas por dados que possuem uma observação sequencial sobre um período de tempo, sendo que elas possuem padrões como tendência, sazonalidade e ciclos (FILHO et al., 2021). No contexto de armazenamento de dados, as séries temporais consistem em fluxos de valores com seu valor de *timestamp* pertencentes à mesma métrica e ao mesmo conjunto de dimensões rotuladas. Cada série temporal é identificada exclusivamente por seu nome de métrica e pares de valores-chave opcionais chamados rótulos.

#### 2.3.2.2 Cassandra

O banco de dados não relacional Cassandra é um sistema distribuído *open source*, mantido pela Apache Software Foundation que não utiliza a linguagem SQL para a manipulação dos dados e sim *Cassandra Query Language* (CQL). Sua arquitetura possibilita alta escalabilidade, alta disponibilidade e durabilidade e segundo Milošević et al. (2016) é capaz lidar com alto rendimento de gravação sem sacrificar a eficiência de leitura. Os dados no Cassandra são estruturados em linhas e colunas, e organizadas em tabelas agrupadas em *keyspaces*, que são objetos que definem a estratégia de replicação dos nós e definições que se aplicam a todas as tabelas que compõem o *keyspace*. As estratégias de replicação configuráveis são:

- SimpleStrategy: Estratégia simples que define um fator de replicação para que os dados sejam distribuídos por todo o *cluster*.
- NetworkTopologyStrategy: Define o fator de replicação de forma independente para cada data center.

A Figura 6 apresenta a estrutura de uma coluna no Cassandra. O campo `nome` caracteriza a informação armazenada, por exemplo, ao registrar dados sobre um usuário, o `nome` pode ser nome, e-mail, endereço, entre outros (BAAKIND, 2013). O atributo `valor`

contém o valor armazenado. O *timestamp* de data/hora é o real momento em que a coluna foi inicialmente armazenada.

Figura 6 – Estrutura da coluna dentro do Cassandra

| Nome | Valor | *Timestamp* |
|---|---|---|

Fonte: (BAAKIND, 2013)

Segundo Brewer (2012), o teorema *Consistence, Availability and Partition Tolerance* (CAP) define que os sistemas de armazenamento distribuídos conseguem atender plenamente no máximo duas das suas três propriedades (*Consistence, Availability and Partition Tolerance*), implementando um *"trade-off"* entre disponibilidade e consistência caso ocorra uma partição. A partição é a reorganização dos nodos que compõem o sistema distribuído quando ocorre uma falha. Ainda segundo o teorema, a consistência (C) determina que todos os nós vejam a mesma informação ao mesmo tempo, a disponibilidade (A) indica que toda solicitação de cliente que chega a uma instância de serviço deve ser respondida e por fim a tolerância de partição (P) se refere ao sistema se manter em funcionamento em caso de falha ou particionamento da rede de nós. Sendo um sistema de armazenamento distribuído e com foco em aplicações web, o Cassandra prioriza disponibilidade e tolerância de partição (CASSANDRA, 2021).

#### 2.3.2.3 Redis

O Redis é um banco de dados não relacional que implementa a arquitetura do tipo chave-valor. Segundo Li et al. (2017) em arquiteturas de servidores web atualmente, técnicas de *cache* em memória de forma distribuída, como Redis e Memcached, são componentes vitais para garantir serviço de baixa latência para solicitações de usuários. A baixa latência citada se deve principalmente ao armazenamento de dados ser realizado em memória ao invés de armazenamento em disco ou *Solid-State Drive* (SSD), que possibilita alta velocidade de leitura, tornando-o popular não apenas para armazenamento em *cache*, mas também para jogos e aplicações IoT (AWS, 2021).

Diferente do Memcached, outro banco de dados chave-valor não relacional que armazena dados em memória, o Redis suporta *string*, listas, *sets*, valores estruturados como *hashs*, entre outros. Conforme abordado por Zhang e Jia (2020), uma tabela de *hash* é uma estrutura de dados projetada para acessar rapidamente os dados correspondentes através de uma chave, mapeando códigos-chave para uma posição na matriz por meio de uma função *hash* e faz uso das características de pesquisa rápida na matriz, de modo a ter uma função de pesquisa eficiente.

As informações no banco podem ser manipuladas pela aplicação através de métodos simples como *get*, *set*, *hset*(para armazenamento *hash*) e caso necessário também há possibilidade de armazenar as informações em disco. O Redis disponibiliza o número de

*bytes* que uma chave e seu valor requerem para serem armazenados na RAM e também implementam o paradigma de messageria *Publish/Subscribe*.

Um estudo apresentado por Zulfa, Fadli e Wardhana (2020), demonstrou que a utilização do Redis como *cache* de dados foi capaz de aumentar significativamente o desempenho do sistema de banco de dados MySQL em até 3,3 vezes mais rápido para exibir os dados consultados.

### 2.3.3 Prometheus

O software Prometheus é uma ferramenta para monitoramento e geração de alertas. As métricas são coletadas como dados de séries temporais, onde cada métrica possui um valor e uma informação de *timestamp* com precisão de milissegundo associada. O software possui quatro tipos de métricas:

- Contador (ou *counter*): Métrica cumulativa que pode ser incrementada ou atribuída para 0.
- Medidor (ou *gauge*): Valor numérico que pode ser incrementado ou decrementado.
- Histograma (ou *histogram*): Um histograma mostra uma série de observações relacionadas a uma métrica, úteis por exemplo, para mostrar a frequência de um acontecimento ou ver a distribuição temporal da coleta de um métrica. Ele também fornece uma soma de todos os valores observados.
- Resumo (ou *summary*): Semelhante ao histograma, um resumo também é capaz mostrar observações como durações de solicitação e tamanhos de resposta, no entanto, também calcula quantis configuráveis em uma janela de tempo deslizante.

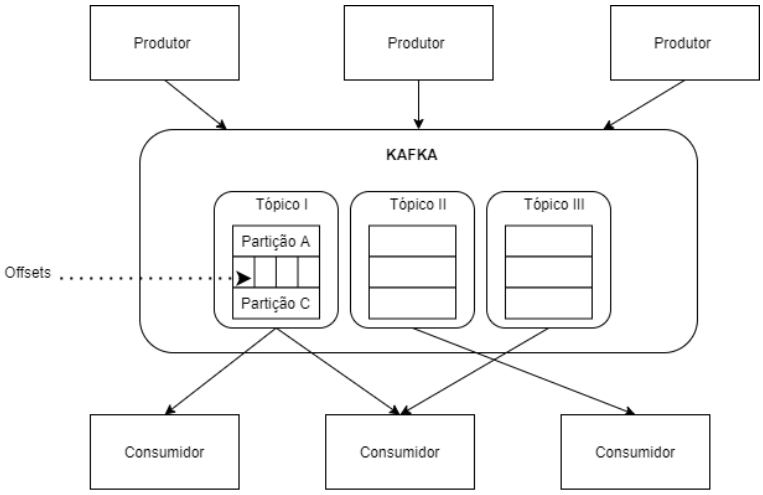
Por recomendação da documentação do Prometheus (2022), a notação das métricas seguem o seguinte padrão: `<nome_métrica>{<nome_label>=<valor_label>}`, por exemplo, `interval_length_sec{quantile=0.2}`. As métricas podem ser visualizadas em gráficos gerados pelo próprio software ou acessadas por outros softwares de análise e monitoramento de métricas como Grafana.

## 2.4 Kafka

O software Kafka é uma plataforma capaz de processar *streaming* e *batch* de eventos, originados por diversas fontes (por exemplo, APIs, sensores, banco de dados) e entregá-los de forma ordenada à várias aplicações clientes. Os*streamings* são fluxos contínuos de dados sem um momento definido para início ou final da transmissão, e *batchs* representam lotes de eventos com uma quantidade definida. O Kafka aplica o paradigma orientado à eventos

(ou *event-based*) para tratamentos de notificações, ou seja, cada mudança de estado em uma fonte de informação gera um evento. Um evento possui uma chave, valor, definição de data e hora (ou *timestamp*) e cabeçalhos de metadados opcionais. A Figura 7 apresenta a estrutura dos componentes do software.

Figura 7 – Fluxo *publisher/subscriber* do Kafka




Fonte: Elaborado pela autora.

Para garantir tolerância a falhas o Kafka pode atuar de forma distribuída, adicionando mais uma complexidade em manipular eventos de forma ordenada. Segundo a documentação do Kafka, o sistema distribuído do software consiste em servidores e clientes que se comunicam por meio de um protocolo de rede TCP (KAFKA, 2021). As principais aplicações do software incluem sistema de mensageria (ou *publisher/subscriber*) com possibilidade de confirmação de recebimento pelo consumidor, reenvio de eventos se necessário, sistema de armazenamento temporário e processamento em *streams* (como conversão de formato de dados gerados em tempo real). Os principais conceitos que compõe o software são:

- *Logs*: são registros de eventos passados e imutáveis associados a uma marcação temporal (*ou timestamp*).
- *offset*: É um identificador de cada registro em uma partição. Seu objetivo é rastrear a ordem sequencial em que as mensagens recebidas;
- Partições: são coleções ordenadas de eventos e replicadas entre os *brokers*. Eventos com a mesma chave são gravados na mesma partição.
- Tópicos: é onde os produtores assinam para publicar e consumir mensagens. Um tópico é constituído de uma ou várias partições, são armazenados de forma durável, ou seja, escritos em disco e replicáveis. Os tópicos no Kafka são sempre multi-produtor e multi-consumidor.

- Servidores: podem formar a camada de armazenamento (atuando como *brokers*) ou executar o *Kafka Connect* para importar e exportar dados continuamente como fluxos de eventos.
- Produtores: são as aplicações que geram eventos e os enviam para os tópicos.
- Consumidores: são as aplicações que assinam um tópico e consomem os eventos gerado pelos produtores.
- Zookeeper: é um software de gerenciamento do cluster Kafka, responsável por controle do status e comunicação entre os tópicos e partições de cada nó.

## 2.5 Cache

Em computação, o conceito de *cache* pode estar presente em várias etapas e componentes do sistema. A técnica consiste em utilizar um armazenamento de rápido acesso para guardar informações que serão utilizadas com mais frequência e de forma temporária. Processadores utilizam memorias RAM, que são memórias de acesso rápido, fisicamente próximas, porém mais caras, para evitar acessos à dispositivos de leitura lenta como disco rígido ou SSD. A infraestrutura de um sistema web utiliza o conceito de *cache* para armazenar informações frequentemente acessadas em local que evite o acesso a um banco de dados ou o reprocessamento da informação pelo servidor web. Em todas as aplicações, o desafio de tratar a informação envolve `o que` armazenar, `onde` armazenar e `quando` armazenar em *cache*.

Terminologias comuns do paradigma *cache* também são aplicáveis aos caches da web, como `cache miss` que é ação de buscar e não encontrar uma informação no *cache* e `cache hit` que é a ação de encontrar um objeto no *cache*. *cache miss* e *cache hit* são métricas usadas para tomada de decisões, como em algoritmos de invalidação de *cache*. As principais métricas segundo Bahn et al. (1999), usadas para medir a eficiência da regra de *cache* implementada e uso dos recursos destinados ao cache, além do uso de memória são:

- Taxa de acerto (*Hit Ratio* (HR)): É a taxa que contabiliza a quantidade de requisições que foram respondidas pelo *cache* em relação à quantidade de requisições recebidas no total. Um alto valor de hits é desejável, visto que representa um bom uso do recurso alocado e também economia no tempo de resposta do conteúdo solicitado.

$$HR = \frac{\sum h_i}{\sum r_i} \tag{2.1}$$

  O $\boldsymbol{h}$ é a quantidade de *cache hit* e $\boldsymbol{r}$ é a quantidade de requisições que o servidor recebe.

- Taxa de acerto de bytes (*Byte Hit Ratio* (BHR)): Percentual de bytes requisitados pelo cliente que foram enviados pelo cache, sem solicitação ao servidor de origem.

$$BHR = \frac{\sum s_i h_i}{\sum s_i r_i} \tag{2.2}$$

  O $\boldsymbol{s}$ é valor em bytes do objeto consultado.

- Taxa de economia de atraso (*Delay Saving Ratios* (DSR)): determina a fração de atrasos (de comunicação e de consultas ao servidor) que é economizada ao satisfazer as referências do *cache* em vez de um servidor remoto.

$$DSR = \frac{\sum d_i h_i}{\sum d_i r_i} \tag{2.3}$$

  O $\boldsymbol{d}$ é valor da latência ao consultar o objeto no servidor.

Para implementação de algoritmos de invalidação e substituição de *cache* algumas métricas padronizadas são mensuradas sobre os objetos armazenados em cache. A recência indica o tempo desde a última referência ao objeto, a frequência contabiliza o número de solicitações anteriores ao objeto, a latência mede tempo entre as solicitações de um mesmo objeto, o tamanho indica o tamanho da informação armazenada e o tempo de expiração ou TTL indica o tempo de validade da informação armazenada.

### 2.5.1 Substituição de *cache*

As políticas de substituição de *cache* (ou *Eviction Policy*) determinam qual elemento em *cache* será substituído quando os elementos em *cache* excederem um determinado limite de memória, visando preservar o recurso disponível. Conforme Sarma e Govindarajan (2003), diferentes políticas de substituição têm melhor desempenho em termos de número de objetos encontrados no cache, tráfego de rede evitado ao buscar o objeto referenciado do *cache* ou a economia no tempo de resposta. Logo, a politica mais eficiente para uma aplicação web dependerá tanto do tipo de aplicação quanto da métrica que deve ser priorizada. Segundo Pan et al. (2019) existem politicas básicas, comuns a diversas etapas de *cache* e aprimoramentos dessas. Dentre as politicas básicas destacam-se:

- *Lowest Latency First* (LLF): O primeiro objeto a ser removido é o que apresenta menor latência.
- *First In First Out* (FIFO): O primeiro objeto adicionado no *cache* é o primeiro a ser removido.
- *Least Frequently Used* (LFU): O objeto menos solicitado é o primeiro a ser removido.

- *Least Recently Used* (LRU): O objeto que ficou a mais tempo sem ser requisitado é o primeiro a ser removido.
- Tamanho (ou *Size*): O maior objeto é removido primeiro.
- Randômica: Remove um objeto aleatoriamente.

Devido ao enorme volume de dados armazenados em *proxies* web e a limitação de recurso de memória, as técnicas de substituição são especialmente importantes nessa etapa e diversos algoritmos são descritos e comparados na literatura. Algoritmos básicos de substituição de *cache* ganharam aprimoramentos e passaram a considerar mais de uma métrica (como o algoritmo *Least Recently Frequently Used* (LRFU)) ou utilizar características mais específicas, como tamanho. Bancos de dados comumente usados para armazenamento de cache, como o Redis permitem nativamente políticas de substituição comuns como LRU, LFU (a partir da versão 4.0) e também aprimoradas como *volatile-ttl*, que remove as chaves com o tempo de vida mais curto, *volatile-random* que remove chaves aleatoriamente mas desde que contenham um TTL definido.

Algoritmos baseados em funções como o *Greedy Dual-Size* (GDS) utilizam o tamanho do objeto armazenado para decidir pela substituição. Segundo Lam (2015), se os tempos de busca para itens armazenados em um *cache* variam significativamente, o GDS oferece melhor desempenho em termos de tempo total de busca, quando comparado ao LRU, no entanto agrega maior complexidade de implementação. Outras políticas baseadas em funções como *Greedy Dual Size Frequency* (GDSF) e *Weighted Greedy Dual Size Frequency* (WGDSF), consideram múltiplos parâmetros como custo de obtenção do objeto, tamanho, frequência, frequência anterior, entre outras (JULIAN; PAHUJA; SIDHU, 2020).

A abordagem preditiva também vem sendo incorporada em algorítimos de substituição de cache. O modelo preditivo busca prever a probabilidade de acesso do objeto para definir a estratégia de substituição. Segundo o estudo apresentado por Venketesh e Venkatesan (2009), algoritmos tradicionais têm dificuldades em prever o comportamento de cargas na web devido a variação no tamanho dos objetos solicitados e também aos picos de carga que podem ocorrer. O autor apresenta um comparativo entre algoritmos que aplicam inteligência artificial como *Neural Network Proxy Cache Replacement* (NNPCR) e lógica *fuzzy* para caracterizar a carga e apresentaram uma métrica de *hit rate* até 90% maior que algoritmos LFU e LRU.

### 2.5.2 Invalidação de *cache*

Estratégias de invalidação de *cache* são necessárias para manter a consistência dos dados. Uma informação armazenada em *cache* pode ser invalidada pelo valor do TTL ou também por ação de um algoritmo externo indicando que a informação original foi alterada.

Assim como as políticas de substituição de *cache*, existem abordagens tradicionais para invalidação de dados que podem ser aplicadas nas diferentes etapas de *cache* e adaptações que utilizam tecnologias específicas como ferramentas de mensageria ou regras definidas via código. A terminologia *aside* indica que o sistema de *cache* opera isoladamente do sistema de gravação/leitura de dados da origem enquanto nos padrões *System Of Record* (SoR) (como *read-through*, *write-through* e *write-behind*) a manipulação do *cache* ocorre no fluxo de escrita e leitura de dados na origem (EHCACHE, 2021). Dentre as abordagens tradicionais destacam-se:

- TTL: Essa abordagem define um valor estimado sobre o tempo que uma informação irá permanecer válida em *cache* e ao expirar o tempo o objeto é removido do armazenamento.
- *cache-aside*: as solicitações consultam primeiramente o armazenamento de *cache* e caso ocorra um *cache miss* o aplicativo busca as informações na origem, atualizam o valor em *cache* e responde a solicitação.
- *read-through*: nessa abordagem a manipulação do *cache* ocorre como consequência de uma leitura, ou seja, quando ocorre um *cache miss*, o algoritmo de *cache* solicita a informação na origem, armazena o valor em *cache* e retorna para o responsável pela chamada.
- *write-through*: nessa abordagem a manipulação do *cache* ocorre como consequência de uma escrita, ou seja, ao atualizar um informação na origem ela também é inserida no *cache*. Neste cenário dedica-se mais esforço na gravação do dado e também favorece à consistência das informações pois cada alteração na base irá atualizar também o *cache*. No entanto, uma desvantagem dessa abordagem é a alocação de informações que podem não ser solicitadas com frequência, ocasionando um desperdício de recursos.

Em documentações de software e literatura é comum encontrar as terminologias (como *write-through* e *read-through*) em algoritmos que consideram sistemas de banco de dados e sistema de *cache* independentes, bem como adaptações e união de mais de uma abordagem.

### 2.5.3 Etapas de *cache* web

O *cache* de conteúdo da web pode ser implantado em vários locais em toda a arquitetura da web típica MERTZU (2017). As principais etapas são: navegador (ou *browser*) do usuário, servidor *proxy*, servidor de DNS, servidor web e ainda *cache* de dados nas aplicações de *back-end*.

- Navegador: o software do navegador web armazena localmente algumas informações frequentemente acessadas, visto que é um comportamento comum do usuário re-acessar uma página já navegada ou solicitar recarregamento da página. Embora ativa por padrão, a configuração pode ser desabilitada pelo usuário. Os principais recursos armazenados são imagens, fontes, arquivos estáticos e segundo o estudo apresentado por Horsman (2018), podem armazenar inclusive, imagens ou recursos não visualizados pelo usuário mas que pela estratégia do navegador teriam chance de serem solicitados.

- Proxy: O servidor *proxy* pode estar alocado na borda de uma rede local ou na borda de um *cluster* de servidores. Em ambas as configurações ele atua armazenando informações estáticas e dinâmicas e é capaz de compartilhar as informações armazenadas em *cache* entre os usuários que solicitam recursos através desse serviço. Segundo Wessels (2001), armazenar objetos populares em locais próximos aos clientes, é considerada uma das soluções eficazes para evitar gargalos de serviços web, reduzir o tráfego na Internet e melhorar a escalabilidade do sistema.

- DNS: Os servidores DNS são responsáveis por traduzir endereços IP ao nome de domínio amigável. Como são consultas que terão a mesma resposta para um grande número de consumidores, essa informação é geralmente cacheada a fim de prover agilidade na tradução dos nomes de domínio.

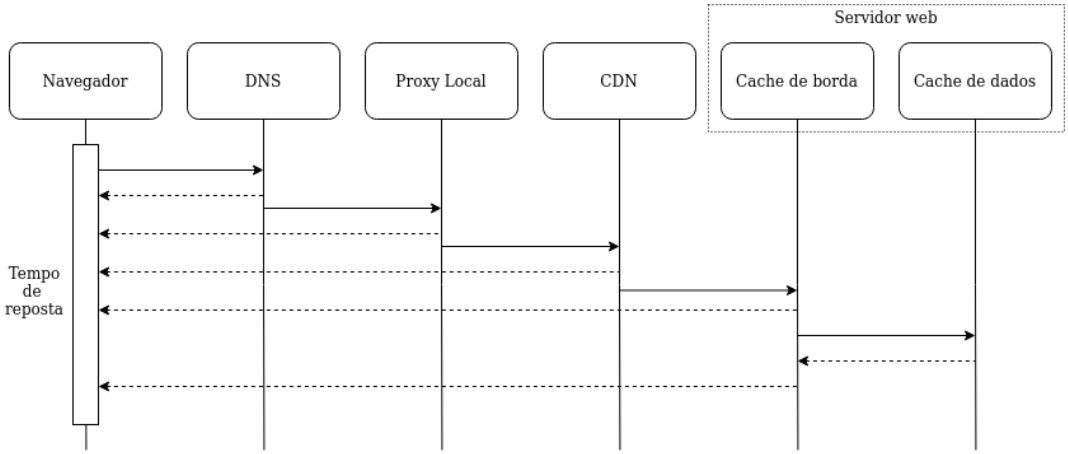
- CDN: A CDN provê informação através de replicações do conteúdo em pontos de presença, especialmente importantes na redução de latência de mídia, como *streams* de vídeo e jogos.

- Servidor web: As técnicas de *cache* em servidores envolvem tanto a resposta inteira de um requisição quanto o cacheamento de dados de tratativas internas dentro de aplicativos, como retorno de *queries* à banco de dados ou respostas APIs e manipulações internas.

A Figura 8 exemplifica as camadas de *cache* que compõe o tempo de resposta de uma solicitação. Cada etapa, com exceção do *cache* de DNS, caso possua um objeto web armazenado, poderá retornar ao cliente evitando que a requisição precise ser reprocessada totalmente. Ao mesmo tempo, cada etapa a mais adiciona complexidade em manter a consistência da informação, exigindo políticas de invalidação através de cabeçalhos HTTP e principalmente, tempo de vida da informação.

Segundo Mertz e Nunes (2018), devido à variedade de tipos de aplicativos, características de carga de trabalho e padrões de acesso, nenhuma solução de *cache* da web é universal e supera outras opções de *cache* em todos os cenários possíveis.

Figura 8 – Etapas de *cache* na composição do tempo de resposta




Fonte: Elaborado pela autora.

### 2.5.4 Estratégias de *cache* para aplicações web

O estudo apresentado por Zulfa, Hartanto e Adhistya (2020) classifica as técnicas atuais de *cache* web em três categorias: *web caching*, *web prefetching* e *cache* em nível de aplicativo.

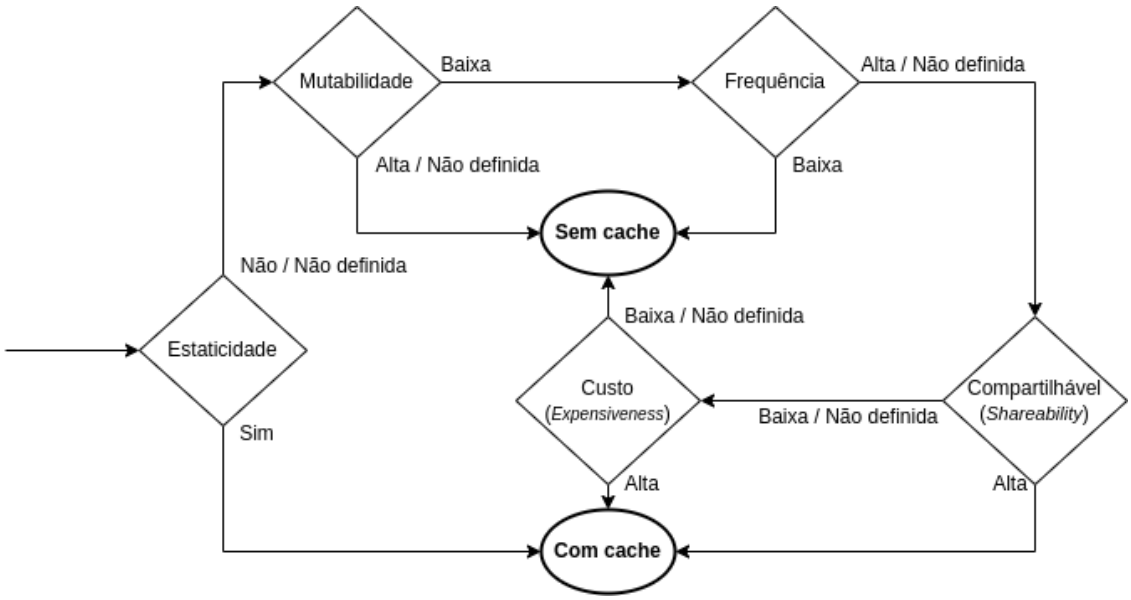
Na técnica *web caching* o algoritmo de *cache* atua de forma passiva, ou seja, as informações são obtidas e armazenadas apenas como consequência de uma consulta do cliente. Na técnica *web prefetching* o algoritmo de *cache* atua de forma ativa, buscando prever quais objetos o cliente pode consultar no futuro. O *cache* no aplicativo pode armazenar em *cache* qualquer informação utilizada pela aplicação para acelerar o processamento da resposta, como uma resposta de API, consulta a banco de dados ou algum valor pré-calculado.

Segundo Mertz e Nunes (2018), o *cache* no aplicativo é normalmente abordado de forma *ad hoc*[1], visto que depende de detalhes específicos do aplicativo. Uma boa implementação de *cache* de aplicativo, também requer conhecimento dos cenários de uso típicos, frequência de solicitação do conteúdo que será armazenado em cache, consumo de memória e frequência de atualização. Um ponto de atenção abordado pelo autor nas implementações neste nível é a junção de regras de negócio com regras de cache, aumentando a complexidade do código e tornando-a uma possível fonte *bugs*.

O fluxograma apresentado por Mertz e Nunes (2018) e exposto na Figura 9 exibe alguns critérios relevantes para definir o uso ou não de abordagem de *cache* na aplicação. Onde a estaticidade indica quantas vezes um método retorna o mesmo valor quando recebe os mesmos parâmetros. A mutabilidade busca entender se o dado altera mais do que é utilizado. A frequência caracteriza se o dado é requisitado repetidamente. A propriedade compartilhável (ou *shareability*) indica se a informação é especifica de um usuário ou

[1] Definição: Para determinado fim; locução que enfatiza que algo tem determinado propósito; destinado a esse fim. Neste contexto, indica que a implementação é destinada à um cenário especifico.

Figura 9 – Critérios associados ao padrão de uso ou não de *cache* (*Cacheability*).




Fonte: Mertz e Nunes (2018), traduzido pela autora.

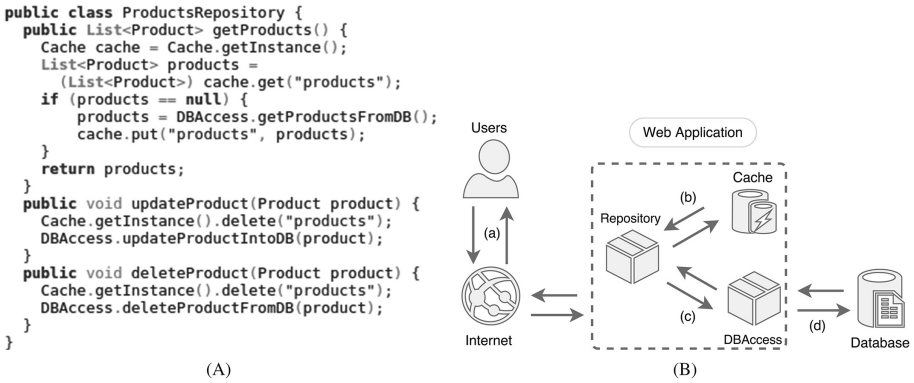
pode ser compartilhada, como por exemplo, tratativa de dados pessoais (específico) ou buscas em *e-commerce* (compartilhável). E por fim, o custo de se obter o dado novamente, relevante em consultas em banco de dados relacionais que necessitam associar tabelas, por exemplo.

Embora cada aplicação possua suas especificidades e regras de negócio, algumas questões relevantes em comum podem ser analisadas a fim de avaliar a necessidade do uso de *cache* padrão ou de forma mais elaborada com técnicas *prefching*. Conforme abordado por Zulfa, Hartanto e Adhistya (2020), o tipo de conteúdo e se é ou não compartilhável por mais de uma requisição podem ser fatores de decisão para a implementação de *cache* pró-ativo.

A Figura 10A apresenta uma abordagem para *cache* de aplicação, com o propósito de reduzir a carga de trabalho do banco de dados, segundo Mertz e Nunes (2018). Na Figura 10B, o usuário realiza uma consulta a aplicação (etapa `a`), as informações são consultadas no armazenamento de *cache* (etapa `b`), caso ocorra um `cache miss` as informações são consultadas no banco de dados (etapa `c` e `d`).

A fim de reduzir a quantidade de premissas a serem consideradas pelos desenvolvedores, Mertz e Nunes (2018) apresenta o *framework* APLCACHE, implementado na linguagem de programação Java. O APLCACHE é uma abordagem de *cache* proativo que calcula a alterabilidade, frequência de uso, compartilhamento, complexidade de recuperação e propriedades de tamanho para definir se um conteúdo deve ser ou não armazenado em cache. O estudo concluiu, que essa abordagem identificou um número maior de objetos em *cache* (46,66% -100%), quando comparado ao número total de objetos em *cache* identificados pela abordagem anterior.

Figura 10 – Fluxo da abordagem apresentada por Mertz e Nunes (2018)




Fonte: Mertz e Nunes (2018) .

## 2.6 Métricas e análise de desempenho

Uma importante etapa de uma analise de desempenho é a definição da métrica a ser utilizada. Conforme apresentado por Ejiogu (1991) uma métrica de software deve ser consistente no seu valor, ou seja, produz o mesmo resultado se utilizada sob as mesmas condições, consistente no seu uso das unidades e dimensões, simples de obtenção e entendimento, entre outras características. Conceitualmente, uma métrica pode ser definida como:

> Métricas são escalas de unidades padronizadas que servem para apurar a quantidade que essas unidades participam na composição de uma grandeza. O estabelecimento de uma unidade de medida (grandeza cujo valor é definido como exatamente um) possibilita a comparação entre duas medições da mesma grandeza, facilitando a análise econômica e qualitativa dos aspectos produtivos que as envolvem. (FILHO, 2003)

Uma outra definição importante é apresentada por Pressman (2009), diferenciando medida, métrica e indicadores:

> Um engenheiro de software coleta medidas e desenvolve métricas para obter indicadores. Um indicador é uma métrica ou combinação de métricas que proporcionam informações sobre o processo de software, em um projeto de software ou no próprio produto. Um indicador proporciona informações que permitem ao gerente de projeto ou aos engenheiros de software ajustar o processo, o projeto ou o produto para incluir melhorias.

Características computacionais comumente mensuradas para parametrizar a qualidade de um sistema web são:

- Vazão: Indica a quantidade de solicitações processadas por unidade de tempo, normalmente, medido em bytes por segundo. Essa métrica é afetada pela latência da rede, tamanho do conteúdo solicitado, existência do conteúdo em cache, por exemplo.

- Tempo de resposta: É a métrica que contabiliza o tempo em que uma requisição demora para chegar ao servidor, ser processada e retornar o conteúdo ao cliente. A medida é calculada considerando *Time To First Byte* (TTFB) ser recebido pelo cliente. Excluindo a latência da rede que é uma característica externa ao servidor web mas que impacta diretamente nessa métrica, fatores internos como consultas síncronas à banco de dados, baixo recurso de hardware (memória e CPU) e código da aplicação oneroso ou pouco otimizado irão contribuir com o aumento desse valor.

- Utilização de CPU: A utilização da CPU em um servidor web está diretamente relacionada ao número de solicitações processadas simultaneamente. As principais especificações de um processador incluem capacidade de memória cache, velocidade de *clock* e quantidade de núcleos. A memória *cache* é responsável por permitir que o processador armazene e acesse os dados frequentemente necessários. Já a o *clock* indica a velocidade na qual o processador executa as instruções, quanto mais rápido o *clock*, mais instruções a CPU pode executar por segundo. Por fim, a quantidade de núcleos permite a execução simultânea de instruções, melhorando o desempenho do processador.

- Utilização de memória: Em um cenário com alta quantidade de requisições HTTP, os processos ficarão na memória em uma fila aguardando processamento e se a memória não for grande o suficiente, eles terão que fazer um *swap* com o HD, aumentando o tempo de resposta da requisição (SILVA, 2015). Além da quantidade de requisições, a complexidade do código da aplicação (como a quantidade de laços de repetição e excesso de alocação de variáveis) e a quantidade de acesso ao banco de dados para leitura e escrita de informação podem impactar o uso da memória disponível.

- Requisições por Segundo (RPS): Mede a quantidade de solicitações que o servidor é capaz de processar por segundo. O tamanho do arquivo que é solicitado e a necessidade de criptografar e descriptografar a requisição (em caso de requisições *Hypertext Transfer Protocol Secure* (HTTPS)) tendem a aumentar o tempo de processamento e por consequência diminuir a quantidade de requisições por segundo que o servidor é capaz de atender.

- Disponibilidade: Geralmente medida em porcentagem, indica o tempo em que o sistema se mantém disponível em relação ao seu tempo total de operação. As falhas do servidor podem ser originadas por problemas no software, na comunicação entre as aplicações ou no hardware.

- Conexões simultâneas: Indica a quantidade de requisições que o servidor pode receber ao mesmo tempo, são impactadas principalmente pela capacidade da CPU, memória e software que opera o servidor web.

### 2.6.1 Estatística descritiva

A estatística descritiva tem como objetivo resumir as principais características de um conjunto de dados por meio de tabelas, gráficos e resumos numéricos (GUIMARãES, 2008). Medidas estatísticas comumente aplicadas à um conjunto de dados são média aritmética, mediana, moda, variância, percentis e desvio padrão.

Segundo Rios (2014), os percentis são os 99 valores que separam uma série em 100 pares iguais de uma métrica. A mensuração de métricas em percentil, diferente do cálculo da média, costuma representar melhor a informação. Conforme abordado por Lima et al. (2018), isso se deve ao impacto dos *outliers* no cálculo da média. *Outliers* são valores que apresentam um padrão distinto dos demais dados coletados e resultam em um valor médio pouco representativo às demais medidas. O cálculo é feito ordenando as amostras e obtendo o valor da posição $(L_p)$, onde **n** é a quantidade de amostras e P o valor do percentil.

$$L_p = ((n + 1) * P)/100$$

logo:

$$L_p = ((7 + 1) * 50)/100 = 4$$

Por exemplo, na amostra de valores 20, 60, 57, 45, 89, 75 e 250, a média será 85,14 e o $P_{50}$ será 60. Em testes de performance é comum o uso de percentis para avaliar métricas como tempo de resposta. Dentre os percentis mais populares em testes, destacam-se o $P_{50}$ e $P_{99}$.

- Percentil: Medidas que dividem a amostra (por ordem crescente dos dados) em 100 partes, cada uma com uma percentagem de dados aproximadamente igual. Por exemplo, a análise do percentil 99 ($P_{99}$) indica que 1% das amostras estarão acima desse valor e todo o restante estará abaixo.
- Média aritmética ou média simples: Obtida dividindo-se a soma das observações pelo número delas.
- Mediana: É o valor central de um conjunto de dados. Por exemplo, no conjunto de dados 1, 2, 3, 6, 7, 8, 9, por exemplo, a mediana é 6.
- Moda: É o valor mais frequente na amostra.
- Variância: Indica a dispersão em geral os seus valores se encontram do valor esperado (normalmente a média aritmética).
- Desvio padrão: Também indica uma medida de dispersão dos dados em torno de média amostral, mas para reduzir a influência de valores críticos que ocorrem no cálculo da variância, o desvio padrão é o resultado positivo da raiz quadrada da variância.

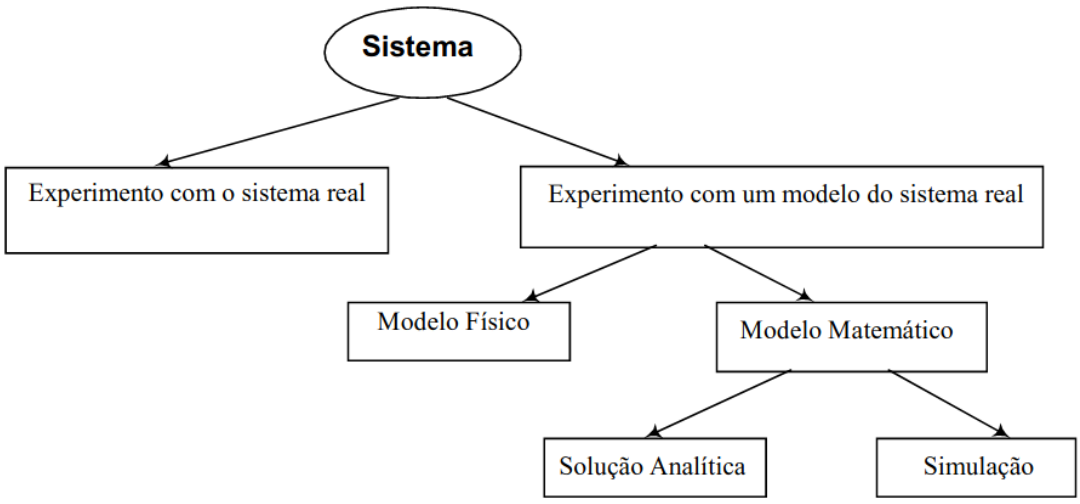
### 2.6.2 Medidas qualitativas

Algumas características são comumente avaliadas a fim de mensurar a qualidade de um sistema computacional, seja de forma quantitativa e/ou qualitativa. A norma ISO/IEC 9126 define como atributos de qualidade de software manutenibilidade, que indica o quanto a estrutura software é fácil de analisar e realizar alterações; portabilidade, indicando a capacidade de um software de ser instalado ou substituído; eficiência, como análise o comportamento temporal do software (lentidão e velocidade) e também bom uso dos recursos; usabilidade é a avaliação do sistema em relação a facilidade de opera-lo e atratividade da interface; confiabilidade indica a tolerância à falhas do software e a sua capacidade de recuperação; funcionalidade indica a capacidade do software de cumprir ao que se propõe, realizar tarefas de forma segura entre outras ações.

Devido as especificidades apresentadas por aplicações web (como necessidade de manutenção, alta disponibilidade e escalabilidade), Nabil, Mosad e Hefny (2011) apresentam uma extensão das características desejáveis definidas na norma ISO/IEC 9126, incluindo fatores de qualidade do ponto de vista do desenvolvedor como reusabilidade e modularidade e sub-fatores como velocidade de download e compatibilidade (entre versões de navegadores, por exemplo).

A Figura 11 apresenta as sub categorizações das metodologias clássicas para aplicar uma análise de desempenho de sistema computacional. Segundo Raeder (2007), realizar a análise de desempenho de um determinado sistema, primeiramente deve-se construir um modelo condizente com ele, visto que a técnica de modelagem pode inclusive prever o desempenho de um sistema antes de sua implementação. Logo, a técnica a ser utilizada depende da característica do sistema a ser avaliada e das ferramentas disponíveis para execução da análise.

Figura 11 – Metodologias para análise de desempenho de sistemas computacionais




Fonte: (GAVIRA, 2003).

As abordagens de análise de desempenho envolvem a definição de uma metodologia, da carga de trabalho e das ferramentas para análise dos dados. A metodologia pode envolver técnicas de aferição de sistemas reais (como protótipos e *benchmarks*) ou uma modelagem baseada no sistema real, seja de forma analítica, simulada entre outras. A etapa de metodologia também pode ser categorizada conforme seguinte definição:

> Existem três técnicas fundamentais que podem ser usadas para encontrar a solução desejada, medidas de sistemas existentes, simulação e modelagem analítica. As medições reais geralmente fornecem os melhores resultados, uma vez que, dadas as ferramentas de medição necessárias, nenhuma suposição simplificada precisa ser feita. Essa característica também torna os resultados baseados em medições de um sistema real os mais verossímeis quando apresentados a outras pessoas. (LILJA, 2005)

A definição de carga de trabalho (ou *workload*) é fortemente relacionada com a característica que está sendo avaliada. Por exemplo, podem ser requisições HTTP à servidores, *queries* em um sistema de banco de dados, opiniões de usuários, cliques, vazão, ou seja, os dados de entrada que causam a ação do sistema em análise. Softwares como JMeter, HP LoadRunner ou Apache-Bench, são capazes de simular carga de requisições HTTP para validação de sistemas web.

Por fim as ferramentas de análise devem ser capazes de coletar ou monitorar de forma padronizada as métricas que irão parametrizar a qualidade do sistema. O software Prometheus por exemplo, permite a coleta de métricas em intervalos definidos e possibilitando acompanhar a saúde do sistema e gerar alertas caso alguma medida ultrapasse um limiar definido. Um outro exemplo é o banco de dados Redis que possui ferramentas de *benchmark* que permite a simulação de testes de carga pontuais, indicando o tempo de resposta para algumas operações básicas, visto que podem variar de acordo com os parâmetros configurados e quantidade de dados armazenados.

### 2.6.3 Testes de performance

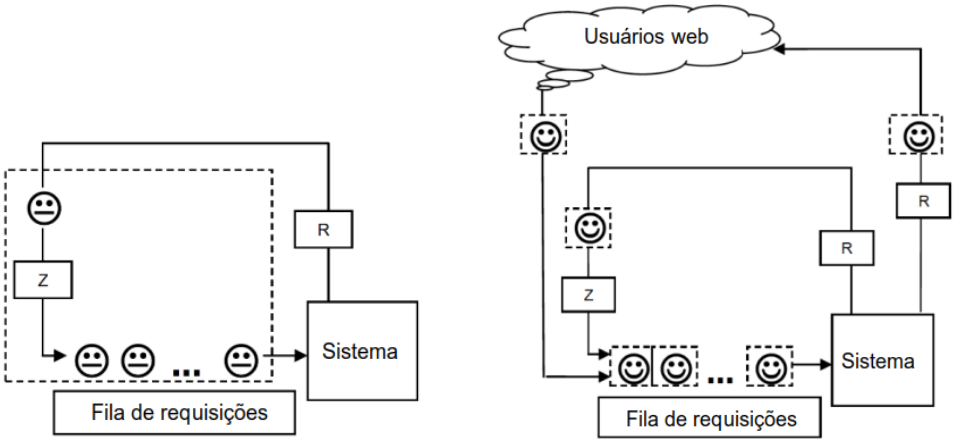
O estudo teórico de filas, busca caracterizar sistemas geradores de espera, que são fluxos de usuários requisitando algum serviço limitado (FONTANELLA; MORABITO, 1997). Esta limitação pode se dar com relação ao número máximo de usuários que o sistema pode atender simultaneamente, dentro de um tempo de resposta limitado dentre outros tipos de restrições.

Durante a avaliação de um sistema computacional, o estudo das características das requisições auxilia na identificação de comportamentos divergentes entre simulações e cenários reais. Segundo Brady e Gunther (2016), usuários virtuais em um teste de carga convencional, formam um padrão de chegada síncrona, visto que a lógica em um *script* de geração de carga permite apenas uma solicitação pendente por vez e cada usuário

virtual não pode emitir sua próxima solicitação até que a solicitação atual seja concluída, enquanto os usuários baseados na web geram um padrão assíncrono de chegada.

A Figura 12 apresenta a divergência entre o comportamento de requisições de usuários virtuais e web. Na Figura 12a cada usuário virtual faz uma solicitação, calcula o tempo de resposta (R) e aguarda um tempo de pausa (Z). A linha pontilhada representa os usuários virtuais compartilhando o mesmo sistema operacional. Na Figura 12b está representado o cenário real com cada usuário web fazendo requisições a partir do seu sistema operacional. Cada usuário da web, também pode emitir mais de um tipo de solicitação HTTP para recuperar vários objetos da web que podem incluir uma página da web completa.

Figura 12 – Comparação dos modelos de filas síncronas e assíncronas



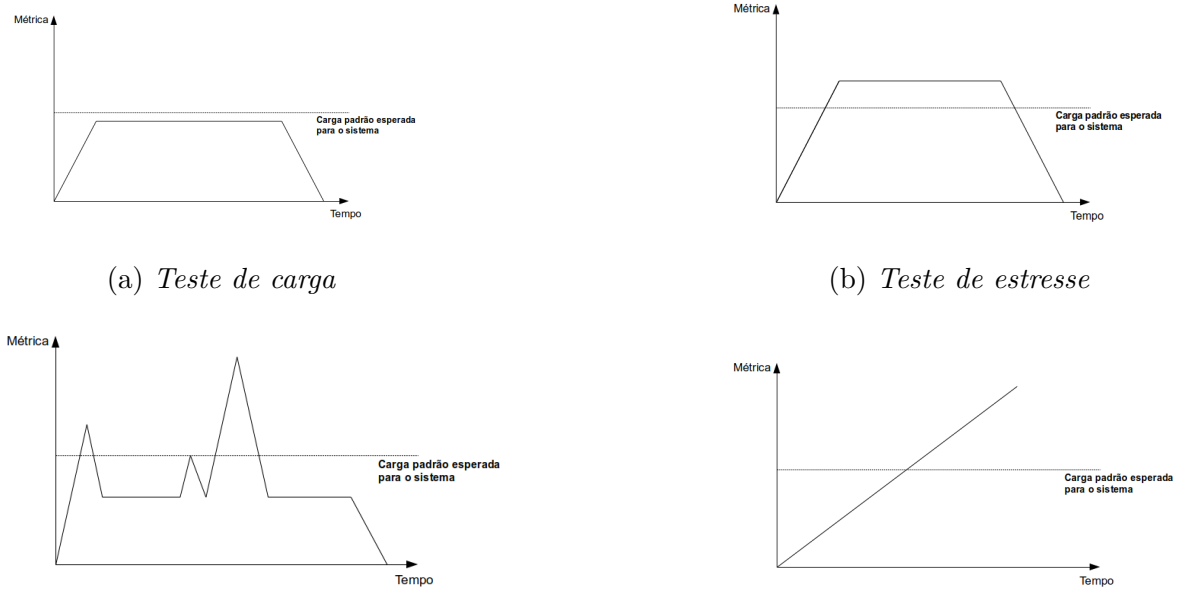

(a) *Fila de um software de simulação* (b) *Fila de usuários reais*

Fonte: (BRADY; GUNTHER, 2016).

Ainda segundo Brady e Gunther (2016), essas distinções nos dizem que os usuários reais da web agem como um conjunto de *threads* de tamanho variável que se movem dentro e fora do *pool* de encadeamentos ativos para iniciar solicitações HTTP. Além da origem das requisições, a taxa com a qual elas são recebidas pelo servidor também poderá levar a diferentes análises. Essa taxa é definida de acordo com a métrica que está sendo mensurada no teste de carga.

A Figura 13a representa o comportamento das requisições em um teste de carga. Segundo Jiang e Hassan (2015), o teste de carga tem como objetivo identificar problemas funcionais que aparecem apenas sob carga, mas dentro das especificações de software e hardware, por exemplo, *deadlocks*, corrida, *buffer overflows* e *memory leaks*. Já o teste de estresse apresentado na Figura 13b ultrapassa os valores de carga para qual o sistema foi projetado a fim de testar a robustez dos recursos de computação, como por exemplo, consumo da CPU, memória, acesso à banco de dados e afins. Testes de pico como apresentado na Figura 13c avaliam o comportamento do sistema em casos de oscilações

Figura 13 – Tipos de carga de trabalho




(a) *Teste de carga*

(b) *Teste de estresse*

(c) *Teste de pico*

(d) *Teste de ponto de ruptura*

Fonte: Elaborado pelo autor.

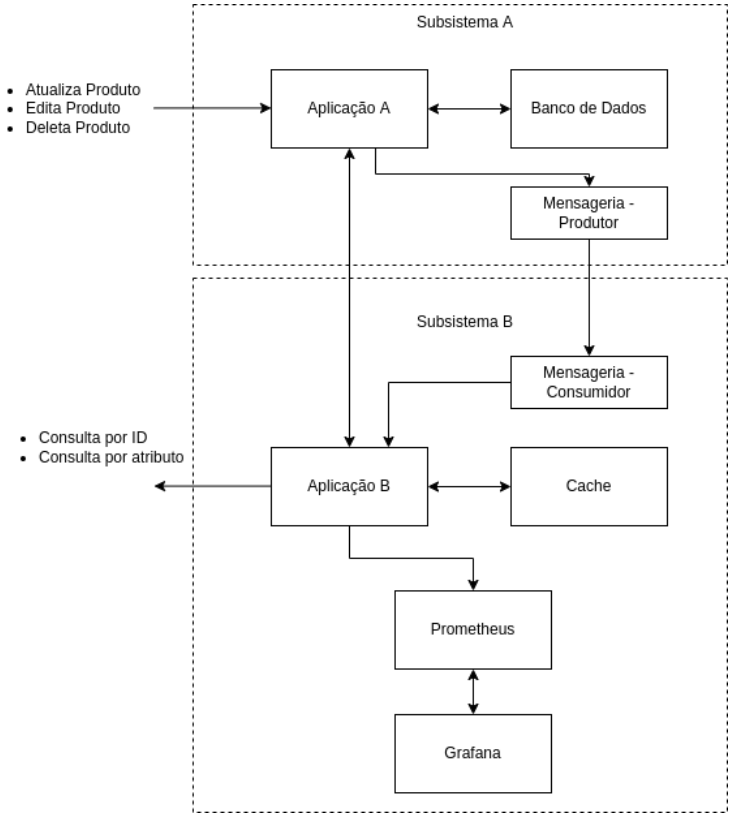
abruptas de carga, situação comuns principalmente em aplicações web. O teste de ruptura apresentado na Figura 13d busca encontrar os limites dos recursos e/ou qual recurso será o primeiro a falhar de modo a interromper o pleno funcionamento do software.

# 3 IMPLEMENTAÇÃO

A implementação visa comparar diferentes estratégias de armazenamento de objetos em *cache* de uma aplicação responsável por consultar informações dinâmicas em um banco de dados. O projeto envolve dois subsistemas que se comunicam apenas por intermédio de uma API e sistema de mensageria, conforme apresentado no diagrama de blocos da Figura 14. O `subsistema A` é composto por um banco de dados responsável por armazenar os dados de forma persistente e pela `aplicação B`, capaz de prover uma camada de abstração para banco de dados e notificar um sistema de mensageria. Já o `subsistema B` é responsável pelo comportamento e regras de negócio do sistema de consulta.

A `aplicação B` será submetido a 3 cenários de tratamento de cache distintos, a fim de analisar qual é o mais adequado em termos de menor tempo de resposta, uso de memória e uso de CPU. A comunicação entre as aplicações se dará através das APIs e também via sistema de mensageria, conforme demonstrado na Figura 14.

Figura 14 – Sistema proposto




Fonte: Elaborado pela autora

As seções a seguir irão detalhar cada componente do sistema, bem como a justificativa para escolha de cada software. Na sequência serão descritos os cenários onde

serão aplicadas as regras de *cache* e a metodologia utilizada para aplicação do teste de desempenho.

## 3.1 Subsistema A

O `subsistema A` é responsável por armazenar a base dados. A `aplicação A` foi implementada na linguagem de programação Node.js e tem como principal objetivo prover uma API para manipulação dos itens (*Create, Read, Update, Delete* (CRUD)) e notificar um tópico Kafka a cada atualização de produto. O sistema em questão não faz uso de dados inter-relacionados, prioriza velocidade no tempo de resposta e é tolerante a algum nível de inconsistência das informações, logo, não é essencial o uso de um SGBD relacional. A escolha do Cassandra como banco de dados se deve ao propósito do software em atender com alta disponibilidade e escalabiliade um grande volume de dados, situação especialmente relevante em aplicações web. O arquivo docker-compose.yaml com as configurações das aplicações que compõe o `subsistema A` está disponível como apêndice na subseção A.1.1.

### 3.1.1 Banco de dados

A primeira ação para criação do banco de dados foi a definição do *keyspace*. Conforme abordado na seção subseção 2.3.2.2 diferentes estratégias de replicação dos dados podem ser adotadas caso o banco de dados esteja em um *cluster*, a fim de organizar a informação e reduzir latências de consulta. Como banco de dados da aplicação não será distribuído em um *cluster*, a estratégia escolhida foi a *SimpleStrategy* com fator de replicação 1.

Após criação do *keyspace*, é definido o *schema* da tabela, indicando o tipo de cada coluna. As colunas dos itens armazenados no banco de dados foram baseadas em documentações plataformas de comércio online como Shopify[1] e Magento[2] para criação de catálogo de produto. A estrutura de cada item armazenado no banco de dados possui a estrutura apresentada na Figura 15:

Segundo o relatório apresentado pela MagazineLuiza (2019), a quantidade de itens em um e-commerce com *marketplace* envolve cerca 5 milhões de itens. Já em lojas virtuais convencionais, a quantidade de produtos é na casa de milhares, como a loja Marisa com cerca 6 mil itens (Souza (2021) e Decathlon com aproximadamente 7 mil (MercadoeConsumo (2021)). A fim de manter uma quantidade de itens próxima ao utilizado em cenários reais e ainda provocar um esforço na consulta, foi definido que o sistema irá manter 10 mil itens armazenados no banco de dados.

[1] <https://shopify.dev/api/storefront/2022-04/objects/Product#fields>
[2] <https://devdocs.magento.com/guides/v2.4/graphql/interfaces/product-interface.html>

Figura 15 – Schema de dados de produto

| Propriedade | Tipo |
|---|---|
| id | *text* PRIMARY KEY |
| availability | *text* |
| brand | *text* |
| categories | *text* |
| created | *timestamp* |
| list_price | *float* |
| product_url | *text* |
| sale_price | *float* |
| title | *text* |

Fonte: Elaborado pela autora

A base de dados foi gerada a partir de uma base de dados aleatória disponível no *site* Data World[3]. Foram realizadas algumas modificações nos valores a fim ocultar links para sites de terceiros, conforme Figura 16:

Figura 16 – Exemplo de um produto

```
{
  "id": "3093",
  "availability": "TRUE",
  "brand": "Gomadic",
  "categories": "Sports & Outdoors | Outdoor Sports |
    Paddling | Kayaks | Dry Boxes",
  "created": "2020-03-10 09:53:07",
  "list_price": 26.95,
  "product_url": "https://www.randomstore.com/ip/Gomadic
    -Clean-and-Dry-Waterproof-Protective-Case-Suitablefor
    -the-LG-Cosmos-2-to-use-Underwater/194195835",
  "sale_price": 26.95,
  "title": "Gomadic Clean and Dry Waterproof Protective Case
    Suitablefor the LG Cosmos 2 to use Underwater"
}
```

Fonte: Elaborado pela autora

Conforme mencionado na subseção 2.3.2.2, o banco de dados Cassandra utiliza a linguagem de consulta CQL, que embora muito semelhante ao SQL, possui algumas limitações a fim de priorizar a eficiência nas consultas. Por exemplo, para que seja possível realizar consultas por todos os valores que possuam determinado valor (exemplo `SELECT * FROM <tabela> WHERE <condição>`) é necessário definir o parâmetro *ALLOW FILTERING* nas consultas.

Além desta, por padrão o Cassandra também não permite a consulta de termos parciais contidos no valor de um registro, consulta que em SQL pode ser realizada com o operador de consulta "LIKE". Esse tipo de funcionalidade se faz necessária no projeto

[3] <https://data.world/promptcloud/product-data-from-walmart-usa>

para que seja possível o retorno de itens que atendam a pelo menos um termo do titulo do item, caso contrário, o usuário teria que pesquisar exatamente pelo título completo item.

Para que esse tipo de consulta fosse possível, foi necessária a criação de uma estrutura auxiliar (índices) com a coluna "Title". Segundo a documentação do Cassandra, o índice indexa os valores da coluna em uma tabela oculta separada daquela que contém os valores que estão sendo indexados, a fim de otimizar a pesquisa de dados que correspondem a uma determinada condição. Para a criação de uma tabela de índices também foi necessário habilitar a configuração *enable_sasi_indexes* no arquivo de configuração `cassandra.yaml`.

### 3.1.2 Aplicação A

A `aplicação A` é, essencialmente, uma API para iteração com o banco de dados. Ela disponibiliza um servidor web, que aceita requisições REST para manipulação dos itens do banco de dados Cassandra. A requisições permitidas, bem como os parâmetros requeridos estão definidos na Figura 17

Figura 17 – URIs Aplicação A

| Método | URI | Parâmetros |
|---|---|---|
| POST | /add | id, created, product_url, title, list_price, sale_price, brand, categories, availability |
| PUT | /update | id, collumn, value |
| DELETE | /delete:id | |
| GET | /get:id | |
| GET | /getAll:collumn:property | |
| GET | /search:term | |

Fonte: Elaborado pela autora

A conexão com o Cassandra foi estabelecida através da biblioteca cassandra-drive[4] implementada em Node.js. Para criação da instância foi necessário definir os IPs do cluster no parâmetro *contactPoints* e a *keyspace* criada no banco.

A conexão com o Kafka foi estabelecida utilizando a biblioteca KafkaJs[5]. O cliente é instanciado definindo o *clientId*, *brokers* e política de *retry*. Segundo Kafka (2018), o *clienteId* é utilizado como identificador da aplicação, útil por exemplo, para rastrear a origem de solicitações. No parâmetro *brokers* foi definido o endereço IP da máquina na qual foi criado o tópico do Kafka e que receberá as mensagens da aplicação. Como política de re-envio (*retry*), foi definida 5 tentativas de re-envio para caso ocorra instabilidades entre a aplicação e o tópico. Também foi definido um valor de 300 ms (default) para parâmetro *initialRetryTime*. Esse parâmetro define o atraso para a primeira requisição de

[4] <https://www.npmjs.com/package/cassandra-driver>
[5] <https://www.npmjs.com/package/kafkajs>

*retry*, as requisições seguintes terão valores aleatórios de acordo com o mecanismo de retry implementado pelo Kafka.

## 3.2 Tópico Kafka

O *broker* Kafka é criado de forma independente do `subsistema A` e `subsistema B`. Ele iniciado através de um *docker-compose* composto por contêiner do Zookeeper baseado em uma imagem *confluentinc/cp-zookeeper* e um contêiner (*broker*) baseado na imagem *confluentinc/cp-kafka*. No contêiner *broker* está definido quais IPs podem se cadastrar como consumidores do tópico e quais portas podem ser acessadas. O arquivo docker-compose.yaml dessa configuração está disponível como apêndice na subseção A.1.3.

Após execução dos contêineres, foi criado o tópico **updatedProduct** por onde as mensagens serão publicadas pela `aplicação A` e consumidas pela `aplicação B`. A imagem Figura 18 apresenta algumas mensagens produzidas pela **aplicação A** listadas o tópico *updateProduct*.

Figura 18 – Lista de mensagens publicadas no tópico Kafka

```
Last login: Sun Jul 10 18:04:14 2022 from 45.237.109.168
root@tele-boi-tcc-marina:~# docker exec --interactive --tty broker kafka-console-consumer --bootstrap-server
 tccmarina.sj.ifsc.edu.br:9092 --topic updatedProduct --from-beginning
{"type":"update","product":"11259","route":"update","timestamp":1657395094339}
{"type":"update","product":"11259","route":"update","timestamp":1657395304597}
{"type":"update","product":"11245","route":"update","timestamp":1657395423694}
{"type":"update","product":"11200","route":"update","timestamp":1657395423714}
{"type":"update","product":"11200","route":"update","timestamp":1657395425409}
{"type":"update","product":"11200","route":"update","timestamp":1657395426209}
{"type":"update","product":"11200","route":"update","timestamp":1657395426778}
{"type":"update","product":"11200","route":"update","timestamp":1657395427306}
{"type":"update","product":"11200","route":"update","timestamp":1657395565706}
{"type":"update","product":"1542","route":"delete","timestamp":1657474199794}
{"type":"update","product":"5723","route":"delete","timestamp":1657474225892}
{"type":"update","product":"2674","route":"delete","timestamp":1657475659794}
{"type":"update","product":"2677","route":"delete","timestamp":1657476301303}
{"type":"update","product":"2674","route":"delete","timestamp":1657480024512}
{"type":"update","product":"25","route":"delete","timestamp":1657480042416}
```

Fonte: Elaborado pela autora

Cada objeto *json* listado é uma ação de manipulação de produto solicitada pelo usuário à `aplicação A`. A propriedade **type**, indica que se trata de mensagens de atualização de itens no banco de dados. O **product** informa o id do item que sofreu a alteração. A propriedade **route** indica se foi uma ação de remoção ou atualização (terão ações diferentes na `aplicação B` para cada caso) e por fim o **timestamp** indicando o momento em que a mensagem foi enviada pela `aplicação A`.

## 3.3 Subsistema B

O subsistema B é composto por uma API que busca itens que contenham determinado termo no título e serviços auxiliares de cache, mensageria, armazenamento e exibição de métricas. O arquivo *docker-compose.yaml* com as configurações das aplicações que compõe o `subsistema A` está disponível como apêndice na subseção A.1.2

### 3.3.1 Aplicação B

A `aplicação B` foi desenvolvida em Node.js e implementa as regras de negócio para realização das consultas. Conforme apresentado na Figura 19, o termo buscado deve ser passado como caminho na URL.

Figura 19 – Requisição e resposta da API

| Método | URL |
|---|---|
| GET | /search/:termo |

Fonte: Elaborado pela autora

A resposta da requisição é composta por metainformações sobre os dados consultados. O `requestId` identificador único de cada requisição, `source` indicando se o conteúdo veio do banco do dados ou cache, `products` com a lista de itens que atendem a requisição e por fim `responseTime` contendo a duração da consulta em segundos. A Figura 20 apresenta um exemplo de consulta na `aplicação B`.

Figura 20 – Resposta de uma requisição pelo termo "*bike*"para aplicação B


```
Not secure | 191.36.8.53:4040/search/bike
Raw
{
  "requestId": "l564wxz8",
  "source": "storage",
  "products": [
    {
      "id": "1542",
      "availability": "TRUE",
      "brand": "lowrider",
      "categories": "Sports & Outdoors | Bikes | Bike Components | Bike Brakes",
      "created": "2020-01-24 17:59:34",
      "list_price": 20.989999771118164,
      "product_url": "https://www.randomstore.com/ip/730-Front-Alloy-Brake-Set-Black-for-bicycle-brake-bike-brake/151769955",
      "sale_price": 20.989999771118164,
      "title": "730 Front Alloy Brake Set Black. for bicycle brake  bike brake"
    },
    {
      "id": "5723",
      "availability": "TRUE",
      "brand": "Lowrider",
      "categories": "Sports & Outdoors | Bikes | Bike Components | Bike Headsets",
      "created": "2020-03-10 09:13:51",
      "list_price": 4.989999771118164,
      "product_url": "https://www.randomstore.com/ip/Headset-Spacer-1-1-8-15mm-Gold-bicycle-headset-bike-headset-chopper-headset-headset-parts-Bike-part/125837486",
      "sale_price": 4.989999771118164,
      "title": "Headset Spacer 1-1/8 15mm Gold. bicycle headset  bike headset  chopper headset  headset parts Bike part"
    },
    {
      "id": "2674",
      "availability": "TRUE",
      "brand": "lowrider",
      "categories": "Sports & Outdoors | Bikes | Bike Components | All Bike Components",
      "created": "2020-01-24 18:13:05",
      "list_price": 21.989999771118164,
      "product_url": "https://www.randomstore.com/ip/Tire-Duro-16-x-2-125-Black-Black-Side-Wall-HF-143G-Bicycle-tire-bike-tire-kids-bike-tire-lowrider-bike-tire-bmx-
      tire/105544114",
      "sale_price": 21.989999771118164,
      "title": "Tire Duro 16\" x 2.125\" Black/Black Side Wall HF-143G.Bicycle tire  bike tire  kids bike tire  lowrider bike tire  bmx bike tire"
    },
```


Fonte: Elaborado pela autora

#### 3.3.1.1 Cache

Assim como afirmado por Mertz e Nunes (2018), o *cache* a nível de aplicação é tipicamente endereçado de maneira *ad-hoc*, uma vez que depende de detalhes específicos da aplicação para ser desenvolvida. Considerando o fluxo de decisão apresentado na Figura 9, o sistema em questão apresenta as seguintes características:

- **Estaticidade**: Não retorna conteúdo estático, visto que com as atualizações e remoções no banco de dados, o retorno será diferente ao longo do tempo.

- **Compartilhável (*Shareability*)**: É destinado à usuários, visto que a informação armazenada em cache é uma consulta por um termo definido pelo usuário. É compartilhável entre usuários, a resposta da consulta por um mesmo termo será a mesma para vários usuários.
- **Frequência**: Um mesmo termo pode ser consultado várias vezes.
- **Mutabilidade**: De acordo com a característica da aplicação, informação será atualizada, mas não de forma extremamente frequente.
- **Custo**: Conforme abordado na subseção 3.1.1, para a execução de filtros por termos no Cassandra, algumas propriedades que impactam no desempenho do banco foram alteradas a fim de tornar possível a consulta. Logo, há um custo a ser considerado nas consultas executadas.

Logo, para essa aplicação, se justifica um tratamento de cache, e também o uso de técnicas pró ativas (*prefetching*). Dado que o fluxo da aplicação está definido, busca-se também determinar **o que**, **como**, **quando** e **onde** armazenar dados em cache:

- **o que:** Visto que a informação consultada não é específica a único usuário e pode compartilhada entre novas consultas, cada item retornado poderá ser armazenado em cache e reaproveitado em consultas seguintes. A informação de quais itens atendem determinada consulta também é relevante para manter em cache, pois também poderá ser compartilhada por mais de uma requisição.
- **como:** A manipulação de objetos no cache será implementada diretamente no código fonte da `aplicação B`.
- **quando:** O momento de inserção ou leitura de itens em cache será diferente em cada cenário implementado. A invalidação se dará por item, visto que o que define a obsolescência da informação é a atualização/remoção do item.
- **onde:** Por implementar o armazenamentos as informações em memória e apresentar baixo tempo de resposta na consulta dos objetos, o software escolhido para armazenamento do cache é o REDIS, banco de dados não relacional, orientado a chave-valor.

O software Redis mantém as configurações padrão da imagem oficial. A Figura 21 apresenta um exemplo de armazenamento de informações no software.

Figura 21 – Armazenamento das informações no Redis (*cache*)

```
127.0.0.1:6379> get bike
"{\"idList\":[\"10396\",\"4382\",\"117\",\"5563\",\"9863\",\"1499\",\"11085\",\"7721\",\"414\",\"13347\",\"7599\",\
"1127\",\"7370\",\"1207\",\"6739\",\"1165\",\"6127\",\"13915\",\"7766\",\"9528\",\"428\",\"106\",\"4872\",\"355\",\
"442\",\"2135\",\"13961\",\"2601\",\"5836\",\"8753\",\"1585\",\"2213\",\"13599\",\"678\",\"2022\",\"1177\",\"8228\"
,\"1599\",\"232\",\"14275\",\"8289\",\"11088\",\"2906\",\"2627\",\"5369\",\"2144\",\"4330\",\"10709\",\"2946\",\"48
07\",\"995\",\"7022\",\"13163\",\"9693\",\"7557\",\"4368\",\"5416\",\"11302\",\"11834\"],\"size\":59}"
127.0.0.1:6379> get 117
"{\"id\":\"117\",\"availability\":\"TRUE\",\"brand\":\"Origami\",\"categories\":\"Sports & Outdoors | Bikes | Adult
 Bikes | Folding Bikes\",\"created\":\"2020-01-24 17:30:30\",\"list_price\":299,\"product_url\":\"https://www.rando
mstore.com/ip/Origami-Gazelle-7-speed-folding-bike/112521103\",\"sale_price\":299,\"title\":\"Origami Gazelle 7-spe
ed folding bike\"}"
127.0.0.1:6379> keys *
 1) "4368"
 2) "5563"
 3) "9863"
 4) "428"
 5) "4872"
 6) "13599"
 7) "442"
 8) "2144"
 9) "5416"
10) "5836"
11) "11302"
12) "11834"
```

Fonte: Elaborado pela autora

## 3.4 Cenários

Baseada na documentação sobre estratégias de cache da AWS[6] e também nos artigos de Mertz e Nunes (2018) e Zulfa, Hartanto e Adhistya (2020), foram definidos 3 cenários possíveis de configuração de cache neste sistema, com diferentes níveis de manipulação da informação. O TTL padrão de 5 minutos foi baseado publicação de Gray e Shenay (1999) e também já foi observado o uso desse valor em aplicações reais. Os detalhes de cada cenário que serão detalhado a seguir:

### 3.4.1 Cenário 1 - Cache passivo

Esse fluxo armazena os objetos em cache somente como decorrência de um requisição, ou seja, sem nenhum tipo de pró-atividade da `aplicação B` em ir buscar dados que possivelmente serão acessados.

- Substituição: Política definida no Redis, sem substituição (propriedade *maxmemory_policy* definida como *noeviction*), usando a memória disponível.
- Invalidação: A invalidação ocorre quando o tempo de vida do objeto (termo ou ID de produto) atingir 5 minutos ou quando o consumidor Kafka receber uma mensagem de atualização/remoção.
- Prós: Pouca manipulação na aplicação, evita o armazenamento de informações que nunca foram (e podem nunca ser) consultadas.
- Contras: Itens novos ou atualizados só retornarão na consulta após a expiração do TTL do termo. Caso vários itens de uma mesma consulta sofra atualizações a

[6] <https://docs.aws.amazon.com/AmazonElastiCache/latest/mem-ug/Strategies.html>

quantidade de itens disponível em cache fica reduzida, logo, foi incluída também um regra de quantidade miníma de retorno por termo, para caso o cache não possua itens o suficiente, a consulta é feita para a `aplicação A`.

**Algorithm 1** Cenário 1 - Cache passivo

```
1: itens ←consultaCache(termo)
2: if itens = 0 then
3:     itens ←consultaBancodeDados(termo)
4:     armazenaCache(itens)
5: end if
6: while true do
7:     if kafkaMessage ←update then
8:         invalidaCache(id)
9:     end if
10: end while
```

O Algoritmo 1 apresenta um pseudo-código com a tratativa feita na `aplicação B`. Ao receber uma requisição, o termo é consultado em cache e caso não possua resultado é consultado na `aplicação A`. Paralelamente há um cliente da biblioteca do kafka (representada pelo *loop while true*) em execução que executa funções a cada mensagem recebida. No cenário atual, a ação executada é apenas invalidar o ID do item cache.

### 3.4.2 Cenário 2 - Cache pró-ativo

No `cenário 2`, como *cache* possui a informação de quais IDs devem ser retornados por cada consulta (Figura 21), ao receber uma mensagem indicado que o ID foi atualizado a aplicação além de remover o item do cache, a `aplicação B` realiza a consulta da informação atualizada no banco de dados, mantendo-a atualizada em cache.

- Substituição: Política definida no Redis, sem substituição (propriedade *maxmemory_policy* definida como *noeviction*), usando a memória disponível.
- Invalidação: A invalidação ocorre quando o tempo de vida do objeto atingir 5 minutos ou quando o consumidor Kafka receber uma mensagem de atualização/remoção.
- Prós: Enquanto o cache do termo não expirar, os itens consultados estarão atualizados. Garante a quantidade de produtos associados à um termo contida em cache em caso de atualizações.
- Contras: Pode manter em cache itens que nunca serão consultados.

No Algoritmo 2, as ações adicionais são executadas a cada nova mensagem recebida pelo consumidor. Conforme descrito inicio da seção, além de invalidar o item em cache

**Algorithm 2** Cenário 2 - Cache pró-ativo

```
    itens ←consultaCache(termo)
 2: if itens = 0 then
        itens ←consultaBancodeDados(termo)
 4:     armazenaCache(itens)
    end if
 6: while true do
        if kafkaMessage ←update then
 8:         invalidaCache(id)
            itens ←consultaBancodeDados(id)
10:         armazenaCache(id)
        end if
12: end while
```

(linha 8 do algoritmo), a aplicação irá consultar a informação atualizada na `aplicação A` e armazenar em cache.

### 3.4.3 Cenário 3 - Cache pró-ativo com TTL adaptativo

O `cenário 3` engloba as regras do `cenário 2`, no entanto, busca favorecer consultas frequentes, mantendo-as em cache com um tempo de expiração (TTL) maior. Nesse cenário, quando um termo é consultado, é criado um registro no Redis que é incrementado a cada nova consulta. Quando o contador do termo ultrapassa 100 consultas o TTL considerado passa a ser 10 minutos. Caso ultrapasse 1000 consultas o TTL é alterado para 30 minutos.

- Substituição: Politica definida no Redis, sem substituição (propriedade *maxmemory_policy* definida como *noeviction*), usando a memória disponível.
- Invalidação: A invalidação ocorre quando o tempo de vida do objeto atingir o TTL definido ou quando o consumidor Kafka receber uma mensagem de atualização/remoção.
- Prós: Enquanto o cache do termo não expirar, os itens consultados estarão atualizados. Garante a quantidade de produtos associados à um termo contida em cache em caso de atualizações. Os termos consultados com maior frequência permanecem por mais tempo em cache.
- Contras: Maior manipulação no código; Mais uma chave armazenada no Redis para computar a quantidade de consultas por termo; Pode manter em cache itens que nunca serão consultados.

O Algoritmo 3 demonstra a adição da etapa de definição do TTL ao inserir objetos no cache, tanto no momento em que a requisição é tratada pela `aplicação B` (linha 4), quando ao receber mensagens do consumidor (linha 11).

**Algorithm 3** Cenário 3 - Cache pró-ativo com TTL adaptativo

```
    itens ←consultaCache(termo)
    if itens = 0 then
3:      itens ←consultaBancodeDados(termo)
        TTL ←calculaTTL(id)
        armazenaCache(itens, TTL)
6:  end if
    while true do
        if kafkaMessage ←update  then
9:          invalidaCache(id)
            itens ←consultaBancodeDados(id)
            TTL ←calculaTTL(id)
12:         armazenaCache(id, TTL)
        end if
    end while
```

### 3.4.4 Cenário 4 - Sem cache

O `cenário 4` não implementa nenhuma tratativa de *cache*. Conforme apresentado no algoritmo 4, a requisição consulta os dados diretamente no banco de dados.

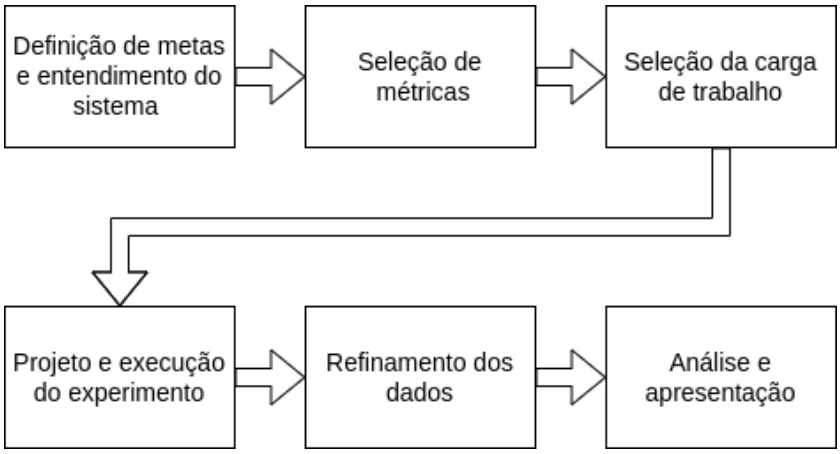
**Algorithm 4** Cenário 4 - Sem cache

```
    itens ←consultaBancodeDados(termo)
```

# 4 TESTE E RESULTADOS

Para realizar a análise comparativa entre os cenários propostos, foi aplicada a metodologia de medição conforme abordado na subseção 2.6.2. A dissertação de SILVA (2015) detalha as etapas de medição de uma abordagem de para a avaliação de desempenho de serviços web, tais etapas foram usadas como base durantes os testes desse projeto.

Figura 22 – Metodologia adotada para execução dos testes




Fonte: SILVA (2015)

A Figura 22 apresenta um fluxograma com cada etapa descrita e as subseções a seguir irão descrever a aplicação de cada etapa no contexto do projeto.

## 4.1 Definição de metas e entendimento do sistema

Segundo SILVA (2015), a etapa de entendimento do sistema envolve identificar estruturas de *hardware* e *software*, dado que escolha do sistema afeta as métricas de desempenho. Como o objetivo do projeto é analisar métricas de tempo de resposta, *hit rate* e uso de memória em 4 cenários, foram usadas quatro máquinas disponibilizadas pelo IFSC para aturem como servidor dos subsistemas, onde cada uma serve um dos cenários implementados. As especificações do *software* foram detalhadas na Capítulo 3.

A Tabela 1 apresenta a especificação técnica de cada máquina utilizadas para servir as aplicações. Além dessas, foi utilizada uma quinta máquina para execução do teste de carga e *script* de simulação do comportamento.

Tabela 1 – Especificação dos computadores usados nos testes

| | **Máquina 1** | **Máquina 2** | **Máquina 3** | **Máquina 4** |
|---|---|---|---|---|
| Processador | GenuineIntel Common KVM processor 2599.996MHz 8 Cores | GenuineIntel Intel(R) Xeon(R) CPU E3-1225 V2 @ 3.20GHz - 2782.401 4 cores | GenuineIntel Common KVM processor 2000.070 MHz 2 cores | GenuineIntel Common KVM processor 2000.070MHz 2 cores |
| Sistema Operacional | Ubuntu 20.04.4 LTS | Ubuntu 20.04.4 LTS | Ubuntu 18.04.6 LTS | Ubuntu 18.04.6 LTS |
| RAM | 16384 MB | 4096 MB | 4096 MB | 4096 MB |

## 4.2 Seleção de métricas e indicadores

Todas as medidas geradas pela `aplicação B` são armazenadas no formato de séries temporais, que conforme mencionado na subseção 2.3.3, são fluxos de valores associados à um *timestamp* pertencentes à mesma métrica. Todas as métricas são disponibilizadas para consulta no caminho /metrics da aplicação (Exemplo http://191.36.8.54:4040/metrics), rota que será utilizada pelo software Grafana para importar os dados e gerar os gráficos. Também foram analisadas as métricas geradas pelo relatório do JMeter que contabiliza dados de cada requisição.

- Total de requisições: Métrica do tipo contador, incrementada a cada requisição recebida pela rota /search da `aplicação B`.
- Total de requisições *cache*: Métrica do tipo contador, incrementada sempre o *cache* possui a informação requerida.
- Total de requisições ao banco de dados: Métrica do tipo contador, incrementada sempre que o *cache* não possui a informação e a busca precisa ser feita na `aplicação A`.
- Memória: Métrica do tipo medidor, mensurando o percentual de memória livre, obtido a cada 1 segundo, pelo método *freeMemPercentage*() da biblioteca node-os-utils[1].
- Uso de CPU: Métrica do tipo medidor, mensurando o percentual de uso de CPU, obtido a cada 1 segundo, pelo método *cpuPercentage*() da biblioteca node-os-utils.
- Tempo de resposta total: Métrica do tipo histograma, contabiliza o tempo total, agregando a consulta ao *cache* e ao banco de dados quando esse fluxo ocorre, por exemplo.

[1] <https://www.npmjs.com/package/node-os-utils>

- Tempo de resposta *cache*: Métrica do tipo histograma, contabiliza o tempo da consulta ao *cache*, quando a informação existe no *cache*. Como essa medida é composta de valores na casa de ms, foi necessária a configuração mais *buckets* em relação aos histogramas do tempo de resposta total e do banco de dados.

- Tempo de resposta banco de dados: Métrica do tipo histograma, contabiliza o tempo da consulta ao banco de dados.

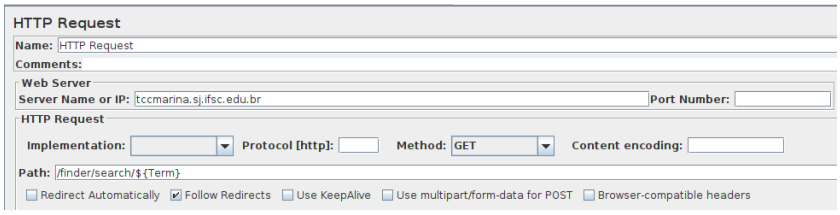
- Hit rate: Conforme Equação 2.1, foi gerada diretamente no software Grafana, apresentando a razão do `Total de requisições` ***`cache`*** em relação ao `Total de requisições`

- RT cliente: Gerado pelo software JMeter, contabiliza a diferença temporal entre a requisição ser enviada e recebimento do primeiro byte (TTFB).

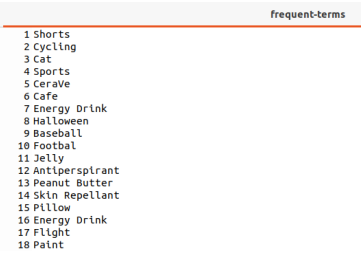
## 4.3 Seleção da carga de trabalho

A carga de trabalho é essencialmente requisições HTTP para a API da `aplicação B`. Primeiramente foi gerada uma lista de termos baseada no título de itens aleatórios da base dados. Na sequência foi definido uma quantidade de repetições para cada termo e gerada uma nova lista com os termos repetidos, porém desordenados. Essa lista final é importada no software JMeter como variável das requisições HTTP.

Figura 23 – Configuração do software JMeter




(a) Configuração do JMeter para iteração na lista de termos




(b) Trecho da lista de termos

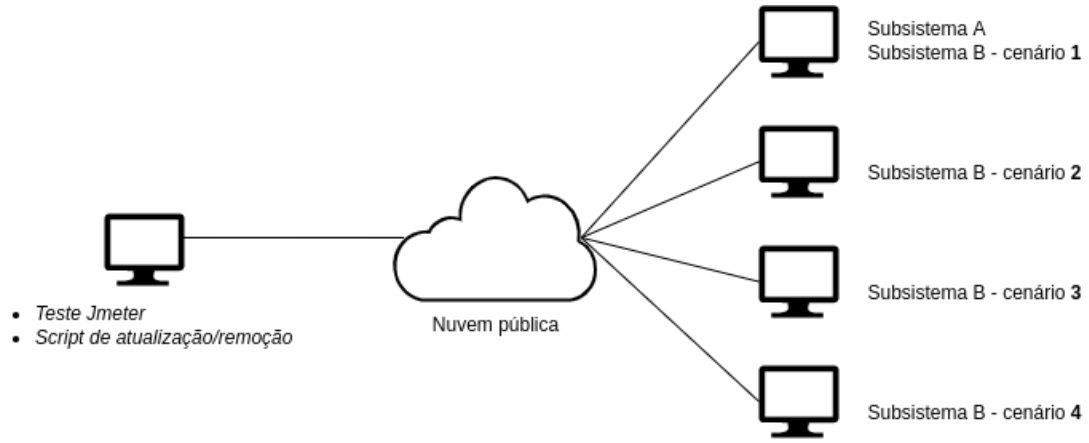
A Figura 23a apresenta a configuração de uma requisição HTTP no software JMeter para iterar uma lista de termos (Figura 23b), que já apresenta repetição de item na posição 7 e 16. No plano de testes é a realizada a mesma configuração para cada um dos servidores (Máquinas 1, 2, 3 e 4), onde as requisições são feitas paralelamente e seguindo a mesma ordem de termos consultados.

Também foram selecionados itens que seriam atualizados e removidos durante a execução do teste para provocar invalidação de informações no *cache*. O *script* responsável por atualizar e remover IDs envia requisições para a API da `aplicação A` e é iniciado junto ao início da execução do teste de carga pelo JMeter.

## 4.4 Projeto e execução do experimento

Foram realizadas 4 execuções de testes a fim de rotacionar os cenários em cada máquina. Cada rodada de teste tem duração de 1 hora e ao fim é capturada as imagens dos gráficos gerados pelo Grafana e o relatório gerado pelo JMeter.

Figura 24 – Distribuição dos sistemas nas máquinas de teste




Fonte: Elaborado pela autora

A Figura 24 apresenta um exemplo de configuração de uma das rodadas de testes. O `subsistema A` permanece na maquina 1 em todas as rodadas enquanto a configuração do `subsistema B` é alternada entre as máquinas a cada execução de teste. Como cada subsistema é instanciado via Docker Compose, entre cada rodada de testes todos os contêineres e imagens são removidos e construídos novamente. Ao final das quatro rodadas de testes foram gerados um total de 16 dashboards no Grafana, que estão apresentadas pela Figura 29 até Figura 44 de forma ampliada como apêndice no Apêndice A.

O primeiro painel apresenta o acumulativo do total de requisições recebidas pela `aplicação B`. Nele, é apresentada de forma isolada as requisições ao *cache*, as requisições ao banco de dados e somatória de ambas. O segundo painel apresenta o percentual de hit rate, ou seja, representa a porcentagem de requisições que chegaram e foram respondidas pelo *cache*. O terceiro painel exibe o tempo de resposta em percentil 95, calculado dentro da *aplicação B*. O quarto painel exibe em porcentagem, a degradação da memória livre ao decorrer do teste. E por fim, o ultimo painel exibe o uso da CPU.

A Figura 25 apresenta o dashboard resultante do cenário 1 - Cache Passivo na primeira execução de testes. O primeiro painel exibe um total de 8683 requisições à `aplicação B`, sendo que 6788 foram atendidas pelo *cache* e 1895 foram requisitadas à `aplicação A`. O segundo painel apresenta o hit rate em torno de 78,2%. Como no cenário de teste adotado, as requisições atendidas pelo *cache* evoluem proporcionalmente ao aumento de requisições, esse valor possui pouca variação decorrer do teste.

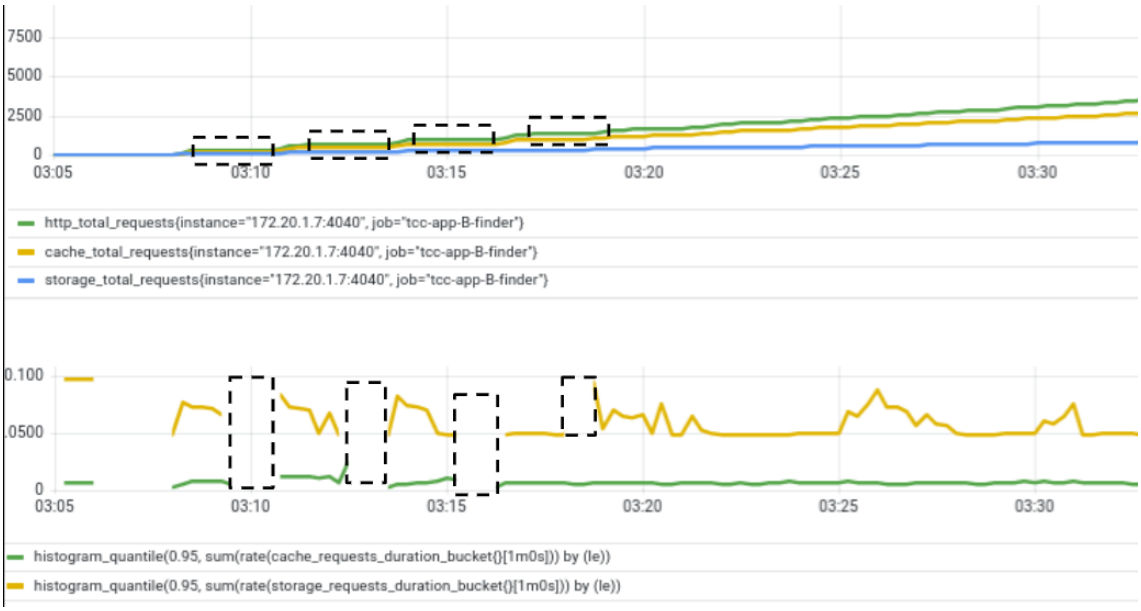
Analisando s gráficos, foi observada a ausência de valores de tempo de resposta em

Figura 25 – Gráfico resultante do cenário 1 - Cache Passivo após a primeira execução de testes


Fonte: Elaborado pela autora

determinados períodos de execução. Ao comparar com o comportamento das requisições recebidas, nota-se que a `aplicação B` não estava conseguindo responder requisições nesses momentos.

Figura 26 – Comparação entre o comportamento das requisições com os intervalos do gráfico de tempo de resposta




Fonte: Elaborado pela autora

A Figura 26 apresenta um recorte ampliado da Figura 25, destacando um compa-

rativo entre o gráfico de requisições e o de tempo de resposta, salientando em retângulos pontilhados os trechos críticos, onde a `aplicação B` estava apresentando instabilidade no inicio dos testes. O comportamento se repetiu em todos os cenários, e pode ter sido provocado pela lentidão no processamento de alguma requisição pelo `subsistema A`.

Através dos gráficos é possível observar uma diminuição na quantidade de memória livre e aumento no uso da CPU a partir do inicio dos testes, mas sem um comportamento padrão que possibilite caracterizar a responsabilidade de um cenário específico.

Figura 27 – Comparação entre o tempo de reposta com a porcetagem de memória utilizada e uso de CPU


Fonte: Elaborado pela autora

A Figura 27 também apresenta uma ampliação da Figura 25, demostrando um comportamento instável na `aplicação B` e que ocorreu de forma isolada nessa rodada de testes (primeira execução) na máquina M3. Na figura, tanto o tempo de resposta das requisições solicitadas ao *cache* quanto as requisitadas ao banco de dados sofreram um aumento no valor coincidindo com a degradação de memória de CPU pela máquina.

## 4.5 Refinamento dos dados

Na etapa de refinamento dos dados, foi coletado os valores apresentados nos dashboards do Grafana e agregado o tempo de resposta gerado pelo JMeter a fim de organizar a comparação dos valores, a Figura 28 exposta como apêndice no Apêndice A, apresenta um trecho de um dos relatórios gerados. Como no relatório gerado pelo JMeter, temos os valores de tempo de resposta por cada requisição, foi calculado também o percentil 50, média simples e o desvio padrão das amostras. A coleta de cada rodada de execução é apresentada a seguir e a análise das informações será feita na seção seguinte.

Tabela 2 – Primeira Execução

| | **Cenário 1 Cache Passivo** | **Cenário 2 Cache Pró-ativo** | **Cenário 3 Cache pró-ativo com TTL adaptativo** | **Cenário 4 Sem cache** |
|---|---|---|---|---|
| Máquina | M3 | M4 | M1 | M2 |
| Hit Rate [%] | 78,20% | 78,20% | **78,60%** | 0,00% |
| RT cache percentill 95[s] | 0,00394 | 0,00265 | **0,00108** | 0 |
| RT banco de dados percentil 95 [s] | 0,0488 | 0,0535 | **0,0217** | 0,046 |
| RT cliente percentil 95 [s] | 3,069 | 3,068 | **3,065** | 3,087 |
| RT cliente (média simples) [s] | 0,676 | **0,573** | 0,686 | 0,599 |
| RT cliente percentil 50 [s] | 0,040 | 0,040 | **0,033** | 0,058 |
| Desvio padrão | 3,599 | 2,875 | 3,434 | **2,686** |

Tabela 3 – Segunda execução

| | **Cenário 1 Cache Passivo** | **Cenário 2 Cache Pró-ativo** | **Cenário 3 Cache pró-ativo com TTL adaptativo** | **Cenário 4 Sem cache** |
|---|---|---|---|---|
| Máquina | M2 | M3 | M4 | M1 |
| Hit Rate [%] | 78,10% | 78,10% | **78,30%** | 0,00% |
| RT cache percentill 95[s] | 0,00261 | **0,00215** | 0,00241 | 0 |
| RT banco de dados percentil 95 [s] | **0,0379** | 0,0591 | 0,0539 | 0,0501 |
| RT cliente percentil 95 [s] | 3,079 | 3,083 | 3,081 | **3,078** |
| RT cliente (média simples) [s] | 0,718 | **0,714** | 0,72 | 0,795 |
| RT cliente percentil 50 [s] | 0,045 | 0,048 | 0,049 | **0,036** |
| Desvio padrão | 3,113 | **2,973** | 3,113 | 3,638 |

Tabela 4 – Terceira execução

| | **Cenário 1 Cache Passivo** | **Cenário 2 Cache Pró-ativo** | **Cenário 3 Cache pró-ativo com TTL adaptativo** | **Cenário 4 Sem cache** |
|---|---|---|---|---|
| Máquina | M1 | M2 | M3 | M4 |
| Hit Rate [%] | 78,20% | 78,20% | **78,40%** | 0,00% |
| RT cache percentill 95[s] | 0,00245 | **0,00128** | 0,00212 | 0 |
| RT banco de dados percentil 95 [s] | **0,0269** | 0,039 | 0,0523 | 0,0594 |
| RT cliente percentil 95 [s] | **3,07** | 3,081 | 3,08 | 3,104 |
| RT cliente (média simples) [s] | 0,734 | 0,837 | **0,708** | 0,723 |
| RT cliente percentil 50 [s] | **0,035** | 0,042 | 0,046 | 0,071 |
| Desvio padrão | 3,226 | 3,826 | **2,974** | 3,327 |

Tabela 5 – Quarta execução

| | **Cenário 1 Cache Passivo** | **Cenário 2 Cache Pró-ativo** | **Cenário 3 Cache pró-ativo com TTL adaptativo** | **Cenário 4 Sem cache** |
|---|---|---|---|---|
| Máquina | M4 | M1 | M2 | M3 |
| Hit Rate [%] | 78,60% | 78,70% | **78,90%** | 0,00% |
| RT cache percentill 95[s] | 0,00593 | **0,00108** | 0,00132 | 0 |
| RT banco de dados percentil 95 [s] | 0,0507 | **0,0216** | 0,0371 | 0,0587 |
| RT cliente percentil 95 [s] | 3,082 | **3,067** | 3,079 | 3,104 |
| RT cliente (média simples) [s] | 0,758 | **0,708** | 0,75 | 0,771 |
| RT cliente percentil 50 [s] | 0,045 | **0,033** | 0,039 | 0,07 |
| Desvio padrão | **3,238** | 3,312 | 3,249 | 3,518 |

## 4.6 Análise

- Hit Rate: Em todos os cenários que utilizam *cache*, o percentual de requisições atendidas pelo *cache* é em média 78% (média simples de todas as execuções) do total de requisições recebidas na `aplicação B`. O valor de hit rate do cenário C foi maior em todos as rodadas de teste.

- Tempo de resposta (RT): No tratamento da requisição pela `aplicação B`, a consulta de objetos em *cache* é em média 94% mais rápida que a consulta na API da `aplicação A`. No entanto, como o indicador relevante é o tempo de resposta percebido pelo cliente, a análise foi feita utilizando os valores de percentil 50 obtidos no relatório JMeter. Outro ponto considerado na comparação foi a execução de cada cenário na mesma máquina, visto que a capacidade de recursos da máquina afeta o desempenho da aplicação.

  Conforme apresentado na Tabela 6, a execução do cenários na máquina M1, que possui maior recurso computacional, resultou em um tempo de resposta menor em comparação com a execução em outras máquinas. O cenário 4, que não implementa estratégia de*cache* resultou em um valor de percentil 50 maior em todas as execuções de teste. No entanto, observando os valores de percentil 95 e média simples expostos na seção 4.5 o cenário 4 chegou a ter valores muito próximo ou até melhor que outros cenários, demostrando que fatores externos à etapa de coleta de dados (banco de

Tabela 6 – RT cliente percentil 50 [s]

| | **Cenário 1 Cache passivo** | **Cenário 2 Cache pró-ativo** | **Cenário 3 Cache pró-ativo com TTL adaptativo** | **Cenário 4 Sem cache** |
|---|---|---|---|---|
| M1 | 0,035 | **0,033** | **0,033** | 0,036 |
| M2 | 0,045 | 0,042 | **0,039** | 0,058 |
| M3 | **0,040** | 0,048 | 0,046 | 0,070 |
| M4 | **0,045** | 0,048 | 0,049 | 0,071 |

dados ou *cache*) podem sobrepor a vantagem temporal oferecida pelo *cache*.

- Memória: A análise do uso de memória foi realizada através dos gráficos. Quando executado na Máquina 3, o Cenário 1 obteve uma maior oscilação entre os valores máximo e mínimo, mas pelo indicador utilizado nos testes, não fica clara uma vantagem relevante em relação ao uso percentual de memória pelos cenários que utilizam *cache*.

- Uso de CPU: Assim como o indicador de memória, é possível observar alguns comportamentos críticos no uso da CPU, como o atingimento de 100% do uso de CPU na execução do cenário 1 na Máquina 3 e também do cenário 3 na Máquina 4, mas o comportamento não se repete com consistência em todas as execuções de teste, logo, não é possível afirmar a influência do algoritmo de *cache* nos valores do indicador.

# 5 CONCLUSÕES

Este trabalho procurou apresentar conceitos relacionados à *cache* e questões relevantes a serem consideradas durante a implementação, através do desenvolvimento de uma aplicação que utiliza diferentes tipos de abordagens para implementação de cache.

Durante a fase de estudo teórico, foi constatado que diversas métricas, além da popular *hit rate*, podem ser utilizadas para analisar a eficiência do cache, como a taxa de acerto em *bytes* e economia de atraso, bem como diferentes políticas de substituição para economia de recurso de memória. Dadas as opções disponíveis, cabe ao projetista do *software* entender como e quando implementar a estratégia de cache a fim de obter maior otimização dos recursos.

Na etapa de implementação, foram selecionados *softwares* mais robustos que o necessário para execução do projeto (Cassandra e o Kafka) mas que proporcionaram entendimento das configurações disponíveis e limitações para utilização. O Cassandra, por exemplo, é destinado a sistemas distribuídos e possui meios de organizar a informação que impedem a pesquisa por termo e filtros, funcionalidades necessárias para implementação do projeto. Para evitar a utilização de banco de dados ou serviços auxiliares, foram alteradas configurações que impactam no desempenho do Cassandra para busca de informações.

Conhecer a carga de trabalho no qual o *software* será exposto e como a informação será apresentada ao usuário final é relevante para propor uma abordagem antes de qualquer implementação ser iniciada, visto que a indicação de que o resultado das requisições pode ser compartilhado ente os usuários e que alguns termos se repetem com mais frequência que outros, foram as premissas para propor o Cenário 3 - Cache pró-ativo com TTL adaptativo, que resultou em uma maior taxa de *hit rate*.

Embora seja possível presumir qual dos cenários utilizaria maior quantidade de memória pela quantidade de objetos que são criados no Redis e pelo tempo que os objetos ficam armazenados, a métrica de porcentagem memória livre não refletiu um comportamento consistente entre os cenários, logo, não foi possível concluir qual cenário foi mais vantajoso em relação ao uso de memória. De forma semelhante, a métrica de uso de CPU não apresentou um comportamento consistente entre os cenários nas execuções e não é possível afirmar um impacto maior de um dos cenários no uso de CPU.

Além dos indicadores utilizados no projeto para comparar os cenários, existem outras métricas quantitativas relevantes que não foram consideradas, como o tempo até a informação retornar atualizada nas requisições (após ocorrer uma invalidação no cache), economia de *bytes* trafegando entre os subsistemas e as métricas qualitativas, como a quantidade de regras incluídas na aplicação que podem impactar na manutenibilidade e

complexidade do código.

Conforme proposto no objetivo geral do trabalho, a implementação propiciou observar cenários mais vantajosos quanto à hit rate e tempo de resposta, no entanto, as métricas escolhidas para avaliação de uso de memória e uso de CPU (definidas na seção 4.2) não foram as mais adequadas, visto que não foram capazes de caracterizar consistentemente o comportamento dos cenários. Além disso, o projeto proporcionou o entendimento de conceitos e premissas relevantes a serem consideradas durante implementações de regras de cache em aplicações.

Como sugestão de trabalhos futuros, pode-se avaliar os cenários a partir de outra fonte de informação para métrica de uso de memória, como o uso de memória isoladamente do software Redis ou avaliar o desempenho do uso de cache em uma aplicação que utilize um banco de dados relacional (que faça o uso de tabelas relacionadas e uso da cláusula JOIN, por exemplo). Também é possível reavaliar os cenários a partir de outros indicadores, como taxa de acerto de bytes (BHR) ou taxa de economia de atraso (DSR), citados na seção 2.5 deste trabalho ou analisar diferentes politicas de substituição de cache, abordadas na subseção 2.5.1.

# REFERÊNCIAS


AL-SHARIF, I. *HTTP 1.1 ,SPDY3.1 And HTTP2 Protocols Performance Comparison*. 2015. Citado na página 21.

ARTZI, S. et al. Finding bugs in web applications using dynamic test generation and explicit-state model checking. *IEEE Transactions on Software Engineering*, IEEE Computer Society, New York, v. 36, n. 4, p. 474–494, 2010. ISSN 00985589. Disponível em: <http://search.proquest.com/docview/732184800/>. Acesso em: 10 jun. 2022. Citado na página 19.

AWS. *O que é o Redis?* 2021. Disponível em: <https://aws.amazon.com/pt/elasticache/what-is-redis/>. Citado na página 35.

BAAKIND, T. A. *Automatic scaling of Cassandra clusters*. Dissertação (Mestrado) — UNIVERSITY OF OSLO Department of Informatics, Oslo, 2013. Citado 2 vezes nas páginas 34 e 35.

BAHN, H. et al. Using full reference history for efficient document replacement in web caches. In: *Second USENIX Symposium on Internet Technologies & Systems (USITS 99)*. Boulder, CO: USENIX Association, 1999. Disponível em: <https://www.usenix.org/conference/usits-99/using-full-reference-history-efficient-document-replacement-web-caches>. Acesso em: 08 jun. 2022. Citado na página 38.

BARON, C. A. Nosql key-value dbs riak and redis. *Database systems journal*, Bucharest Academy of Economic Studies Publishing House, VI, n. 4, p. 3–10, 2016. ISSN 2069-3230. Disponível em: <https://doaj.org/article/833ae7a4fcd3411e97003486e95242aa>. Acesso em: 08 jun. 2022. Citado 2 vezes nas páginas 32 e 33.

BASTOS, J. P. da S. *Desenvolvimento WebOrientado a MicroFrontends*. Dissertação (Mestrado) — Instituto Politécnico do Porto, Portugal, julho 2020. Citado na página 19.

BINI, T. A. *ANALISE DA APLICABILIDADE DAS REGRAS DE OURO AO TUNING DE SISTEMAS GERENCIADORES DE BANCOS DE DADOS RELACIONAIS EM AMBIENTES DE COMPUTACÃO EM NUVEM*. Dissertação (Mestrado) — Universidade Federal do Paraná, Curitiba - PR, 2016. Citado na página 31.

BRADY, J. F.; GUNTHER, N. J. How to emulate web traffic using standard load testing tools. 2016. Citado 2 vezes nas páginas 49 e 50.

BREWER, E. Cap twelve years later: How the "rules"have changed. *Computer*, IEEE, v. 45, n. 2, p. 23–29, 2012. ISSN 0018-9162. Citado na página 35.

CALATRAVA, C. G. et al. Nagaredb: A resource-efficient document-oriented time-series database. *Data*, v. 6, n. 8, 2021. ISSN 2306-5729. Disponível em: <https://www.mdpi.com/2306-5729/6/8/91>. Acesso em: 08 jun. 2022. Citado na página 34.

CARISSIMI, A. Virtualização: da teoria a soluções. In: . [s.n.], 2008. Disponível em: <https://docplayer.com.br/533812-Virtualizacao-da-teoria-a-solucoes.html>. Acesso em: 24 julho 2022. Citado na página 29.

CASSANDRA. *Guarantees*. 2021. Disponível em: <https://cassandra.apache.org/doc/latest/cassandra/architecture/guarantees.html>. Citado na página 35.

CATARINO, M. H. *Integrando banco de dados relacional e orientado a grafos para otimizar consultas com alto grau de indireção*. Dissertação (Mestrado) — Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo - SP, novembro 2017. Citado 2 vezes nas páginas 32 e 33.

CâNDIDO, P. H. V.; BUENO, D. C.; SANTOS, C. H. da S. Rev. bras. deiniciação científica (rbic). In: *ANÁLISE DA UTILIZAÇÃO DE FRAMEWORK FRONT-END EM SISTEMA WEB ADAPTATIVO: UM ESTUDO DA PERSPECTIVA DO DESENVOLVEDOR*. [s.n.], 2020. p. 31–54. Disponível em: <https://periodicos.itp.ifsp.edu.br/index.php/IC/article/download/1592/1191>. Acesso em: 4 abril 2021. Citado na página 18.

DATTA, A. et al. World wide wait: A study of internet scalability and cache-based approaches to alleviate it. *Management Science*, v. 49, n. 10, p. 1425–1444, 2003. ISSN 0025-1909. Citado na página 26.

DISSANAYAKE, N.; DIAS, K. Web-based applications: Extending the general perspective of the service of web. In: . [S.l.: s.n.], 2017. Citado 2 vezes nas páginas 18 e 19.

DOCKER. *Docker overview*. 2021. Disponível em: <https://docs.docker.com/get-started/overview/#the-docker-platform>. Acesso em: 08 jun. 2022. Citado na página 30.

DOCKER. *Comparing Containers and Virtual Machines*. 2022. Disponível em: <https://www.docker.com/resources/what-containe>. Citado na página 29.

EHCACHE. *Cache Usage Patterns*. 2021. Disponível em: <https://www.ehcache.org/documentation/3.3/caching-patterns.html>. Acesso em: 08 jun. 2022. Citado na página 41.

EJIOGU, L. Tm: a systematic methodology of software metrics. *ACM SIGPLAN Notices*, ACM, v. 26, n. 1, p. 124–132, 1991. ISSN 03621340. Citado na página 45.

FILHO, F. M. d. A. et al. Um estudo comparativo de ensembles híbridos para aplicações de previsão de séries temporais. v. 13, n. 2, p. 58–72, 2021. ISSN 2176-6649. Citado na página 34.

FILHO, J. R. D. S. F. *ESTIMAÇÃO DE MÉTRICAS DE DESENVOLVIMENTO AUXILIADA POR REDES NEURAIS ARTIFICIAIS*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DO MARANHÃO, São Luis - MA, 2003. Citado na página 45.

FONTANELLA, G. C.; MORABITO, R. Modelagem por meio de teoria de filas do tradeoff entre investir em canais de atendimento e satisfazer o nível de serviço em provedores internet a queueing model to analyse the tradeoff between investing in attending channels and satisfying service level in internet providers. *Gestão & produção*, Universidade Federal de São Carlos, v. 4, n. 3, p. 278–295, 1997. ISSN 0104-530X. Disponível em: <https://doaj.org/article/789592cbd28c494fb5e531537dd09166>. Acesso em: 08 jun. 2022. Citado na página 49.

GAVIRA, M. de O. *SIMULAÇÃO COMPUTACIONAL COMO UMA FERRAMENTA DE AQUISIÇÃO DE CONHECIMENTO*. Dissertação (Mestrado) — USP - São Carlos, São Carlos - SP, 2003. Citado na página 48.

GRAHAM, R. *The C10K problem*. 2015. Disponível em: <http://c10m.robertgraham.com/p/manifesto.html>. Citado na página 25.

GRAY, J.; SHENAY, P. Rules of thumb in data engineering. In: *ICDE '00 Proceedings of the 16th International Conference on Data Engineering*. Institute of Electrical and Electronics Engineers, Inc., 1999. p. 7. ISBN 0-7695-0506-6. Disponível em: <https://www.microsoft.com/en-us/research/publication/rules-of-thumb-in-data-engineering/>. Acesso em: 08 jun. 2022. Citado na página 59.

GUIMARãES, P. R. B. *Métodos Quantitativos Estatísticos*. [S.l.]: IESDE Brasil S.A, 2008. Citado na página 47.

HORSMAN, G. I didn't see that! an examination of internet browser cache behaviour following website visits. *Digital Investigation*, v. 25, p. 105–113, 2018. ISSN 1742-2876. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1742287617301524>. Acesso em: 08 jun. 2022. Citado na página 42.

Hwang, K. et al. Cloud performance modeling with benchmark evaluation of elastic scaling strategies. *IEEE Transactions on Parallel and Distributed Systems*, v. 27, n. 1, p. 130–143, 2016. Citado 2 vezes nas páginas 15 e 16.

IBM. *O que é um serviço da web?* 2022. Disponível em: <https://www.ibm.com/docs/pt-br/integration-bus/10.0?topic=services-what-is-web-service>. Acesso em: 14 mai 2022. Citado na página 24.

JIANG, Z.; HASSAN, A. E. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, v. 41, p. 1–1, 11 2015. Citado na página 50.

JULIAN, B. P.; PAHUJA, K.; SIDHU, M. S. Enhancements to content caching using weighted greedy caching algorithm in information centric networking. *Procedia Computer Science*, v. 171, p. 2435–2444, 2020. ISSN 1877-0509. Third International Conference on Computing and Network Communications (CoCoNet'19). Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050920312552>. Acesso em: 08 jun. 2022. Citado na página 40.

KAFKA. *Client Configuration*. 2018. <https://kafka.js.org/docs/1.12.0/configuration>. Acesso em: 08 jun. 2022. Citado na página 55.

KAFKA. *What is event streaming?* 2021. Disponível em: <https://kafka.apache.org/documentation/>. Acesso em: 08 jun. 2022. Citado na página 37.

KEGEL, D. *The C10K problem*. 2000. Disponível em: <http://www.kegel.com/c10k.html>. Citado na página 25.

KEPE, T. R. The design and implementation of query execution in modern processing-in-memory hardware. In: . [S.l.: s.n.], 2019. Citado na página 30.

LAM, J. Cache optimization for the modern web. *UC Irvine Electronic Theses and Dissertations*, 2015. Disponível em: <https://escholarship.org/uc/item/555925rp>. Citado na página 40.

LEAVITT, N. Will nosql databases live up to their promise? *Computer*, v. 43, n. 2, p. 12–14, 2010. Citado na página 32.

LEE, D.; KIM, K. Improving web cache server performance through arbitral thread and delayed caching. *Cluster Computing*, Springer US, Boston, v. 15, n. 1, p. 17–25, 2012. ISSN 1386-7857. Citado na página 20.

LI, M. et al. Memsc: A scan-resistant and compact cache replacement framework for memory-based key-value cache systems. Springer US, New York, v. 32, n. 1, p. 55–67, 2017. ISSN 1000-9000. Citado na página 35.

LILJA, D. J. *Measuring computer performance : a practitioner's guide*. 1st pbk.. ed. Cambridge ; New York: Cambridge University Press, 2005. ISBN 9780521646703. Citado na página 49.

LIMA, L. F. M. et al. A influência de outliers nos estudos métricos da informação: uma análise de dados univariados. *Em Questão*, Universidade Federal do Rio Grande do Sul, Faculdade de Biblioteconomia e Comunicação, Porto Alegre, v. 24, n. especial, p. 216–235, 2018. ISSN 1807-8893. Citado na página 47.

LOURENçO, J. et al. Choosing the right nosql database for the job: a quality attribute evaluation. *Journal of Big Data*, Springer Nature B.V., Heidelberg, v. 2, n. 1, p. 1–26, 2015. ISSN 21961115. Disponível em: <http://search.proquest.com/docview/1987960462/>. Citado na página 32.

MAGAZINELUIZA. *Vendas do Magazine Luiza crescem 28% no primeiro trimestre e clientes ativos chegam a 18,2 milhões*. 2019. Disponível em: <https://ri.magazineluiza.com.br/Download.aspx>. Acesso em: 14 mai 2022. Citado na página 53.

MANIEZZO, V. et al. Client-side computational optimization. *ACM Trans. Math. Softw.*, Association for Computing Machinery, New York, NY, USA, v. 45, n. 2, abr. 2019. ISSN 0098-3500. Disponível em: <https://doi-org.ez130.periodicos.capes.gov.br/10.1145/3309549>. Citado na página 19.

MATTSON, R. L.; GHOSH, S. Http-mplex: An enhanced hypertext transfer protocol and its performance evaluation. *Journal of Network and Computer Applications*, v. 32, n. 4, p. 925–939, 2009. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804509000046>. Citado na página 21.

MELL, P.; GRANCE, T. The nist definition of cloud computing. *Association for Computing Machinery. Communications of the ACM*, Association for Computing Machinery, New York, v. 53, n. 6, p. 50, 2010. ISSN 00010782. Disponível em: <http://search.proquest.com/docview/577585592/>. Citado 2 vezes nas páginas 27 e 28.

MERCADOECONSUMO. *Decathlon lança aplicativo próprio para compras online no Brasil*. 2021. Disponível em: <https://mercadoeconsumo.com.br/2021/07/06/decathlon-lanca-aplicativo-proprio-para-compras-online-no-brasil/>. Acesso em: 14 mai 2022. Citado na página 53.

MERTZ, J.; NUNES, I. Automation of application-level caching in a seamless way. *Software: Practice and Experience*, v. 48, n. 6, p. 1218–1237, 2018. ISSN 0038-0644. Citado 7 vezes nas páginas 8, 42, 43, 44, 45, 57 e 59.

MERTZU, J. *Understanding and AutomatingApplication-level Caching*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL INSTITUTO DE INFORMÁTICA PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO, Porto Alegre - RS, março 2017. Citado na página 41.

MICROSOFT. *Guia de arquitetura e design de índices do SQL Server e do SQL do Azure*. 2022. Disponível em: <https://docs.microsoft.com/pt-br/sql/relational-databases/sql-server-index-design-guide?view=sql-server-ver15>. Acesso em: 22 abr 2022. Citado na página 31.

MILOšEVIć, D. et al. Weighted moore-penrose generalized matrix inverse: Mysql vs. cassandra database storage system. *Sadhana*, v. 41, n. 8, p. 837 – 846, 2016. ISSN 02562499. Disponível em: <http://search-ebscohost-com.ez130.periodicos.capes.gov.br/login.aspx?direct=true&db=aph&AN=117955750&lang=pt-br&site=ehost-live>. Citado na página 34.

MIN, C.; FU, T. M. Server program analysis based on http protocol. *MATEC web of conferences*, EDP Sciences, v. 63, p. 05023, 2016. ISSN MATEC Web of Conferences. Disponível em: <https://doaj.org/article/f4ea63a492d943a4b6c292ab7c321d26>. Citado na página 21.

MOZILLA. *Cacheamento HTTP*. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Caching>. Citado na página 23.

NABIL, D.; MOSAD, A.; HEFNY, H. A. Web-based applications quality factors: A survey and a proposed conceptual model. *Egyptian Informatics Journal*, v. 12, n. 3, p. 211–217, 2011. ISSN 1110-8665. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1110866511000405>. Citado na página 48.

NAKATA, Y.; ARAKAWA, S.; MURATA, M. Analyzing the evolution and the future of the internet topology focusing on flow hierarchy. *Journal of Computer Networks & Communications*, v. 2015, p. 1 – 18, 2015. ISSN 20907141. Disponível em: <http://search-ebscohost-com.ez130.periodicos.capes.gov.br/login.aspx?direct=true&db=iih&AN=109063187&lang=pt-br&site=ehost-live>. Citado na página 15.

NGINX. *What Is a Web Server?* 2021. Disponível em: <https://www.nginx.com/resources/glossary/web-server/>. Citado na página 24.

NOVAK, C. *Why is my computer so slow? Your browser needs a tune-up*. 2017. Disponível em: <https://blog.mozilla.org/en/products/firefox/computer-slow-browser-needs-tune-up/>. Citado na página 15.

OLIVIERA, I. N. de. *ANÁLISE DE PERFORMANCE DO PUSH EM CONEXÕES HTTP/2 NO CARREGAMENTO DE PÁGINAS WEB*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, RECIFE -PE, março 2016. Citado 2 vezes nas páginas 21 e 25.

ORACLE. *O que É um Banco de Dados?* 2022. Disponível em: <https://www.oracle.com/br/database/what-is-database/>. Acesso em: 21 abr 2022. Citado na página 30.

OVIEDO, B. et al. Visualizador de trafico de red de comunicación basadas en la Arquitectura TCP/IP. *Revista Universidad y Sociedad*, scielocu, v. 11, p. 193 – 202, 06 2019. ISSN 2218-3620. Disponível em: <http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2218-36202019000200193&nrm=iso>. Citado na página 21.

PAN, C. et al. Lightweight and accurate memory allocation in key-value cache. *International Journal of Parallel Programming*, Springer US, New York, v. 47, n. 3, p. 451–466, 2019. ISSN 0885-7458. Citado na página 39.

PEREIRA, A. L. et al. ComputaÇÃo em nuvem:a seguranÇa da informaÇÃo em ambientes na nuvem eem redes fÍsicas. *Brazilian Journal of Production Engineering,*, v. 2, p. 12–27, 2016. ISSN 2447-5580. Disponível em: <https://www.periodicos.ufes.br/bjpe/article/download/EO02_2016/pdf/34008>. Citado na página 27.

PEñA-ORTIZ, R. et al. Analyzing web server performance under dynamic user workloads. *Computer Communications*, v. 36, n. 4, p. 386–395, 2013. ISSN 0140-3664. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0140366412003908>. Citado na página 15.

PRESSMAN, R. *Engenharia de Software - 7.ed.* McGraw Hill Brasil, 2009. ISBN 9788580550443. Disponível em: <https://books.google.com.br/books?id=y0rH9wuXe68C>. Citado na página 45.

PROMETHEUS. *METRIC AND LABEL NAMING*. 2022. Disponível em: <https://prometheus.io/docs/practices/naming/>. Citado na página 36.

RAEDER, M. *Um Estudo Sobre Técnicas de Avaliação de Desempenho e Modelos de Complexidade para Computação Paralela*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul Faculdadede Informática Programa de Pós-Graduação em Ciência da Computação, Porto Alegre - RS, 2007. Citado na página 48.

REDHAT. Tipos de cloud computing. 2018. Disponível em: <https://www.redhat.com/pt-br/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-cloud>. Citado na página 28.

REDHAT. *What Is an API Gateway?* 2021. Disponível em: <https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container>. Citado na página 29.

RICHARDSON, L.; RUBY, S. *RESTful Web Services*. [S.l.]: O'Reilly Media, Incorporated, 2007. Citado na página 24.

RIOS, E. de M. *Estatística Descritiva, Probabibilodade e Estimação: Noções para o Ensino Básico*. Dissertação (Dissertação de Mestrado) — Universidade Federal de Goias, 2014. Citado na página 47.

ROTARU, M.; OLARIU, F.; RIVIèRE, E. Reliable messaging to millions of users with migratorydata. *Proceedingsof Middleware Industry '17: Proceedings of the Industrial Track of the 18thInternational Middleware Conference (Middleware Industry '17)*, 2017. Citado na página 25.

SARMA, A. R.; GOVINDARAJAN, R. An efficient web cache replacement policy. In: PINKSTON, T. M.; PRASANNA, V. K. (Ed.). *High Performance Computing - HiPC 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 12–22. ISBN 978-3-540-24596-4. Citado na página 39.

SILVA, G. H. M. A. d. *Um modelo de visualização de dados utilizando banco de dados orientado a grafo suportado por big data*. Dissertação (Mestrado) — Universidade de Brasília, Brasília, julho 2016. Citado na página 33.

SILVA, M. J. S. D. *UMA ABORDAGEM PARA AVALIAÇÃO DE DESEMPENHO DE SERVIÇOS WEB*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO, Recife - PE, maio 2015. Citado na página 63.

SILVA, M. J. S. da. *UMA ABORDAGEM PARA AVALIAÇÃO DE DESEMPENHO DE SERVIÇOS WEB*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO, Recife - PE, 2015. Citado 2 vezes nas páginas 16 e 46.

SILVA-MUñOZ, M.; FRANZIN, A.; BERSINI, H. Automatic configuration of the cassandra database using irace. *PeerJ Computer Science*, PeerJ, Inc., San Diego, v. 7, 2021. ISSN 23765992. Disponível em: <http://search.proquest.com/docview/2558263304/>. Citado na página 16.

SILVA, P. P. de Souza Bento da. *Banco de Dados em Memória Principal, um Estudo de Caso Oracle TimesTen Solução de Alto Desempenho*. Dissertação (Mestrado) — Universidade de Brasília, São Paulo - SP, 2010. Citado na página 30.

SONI, R. *Nginx*. Apress, 2016. Disponível em: <https://doi.org/10.1007/978-1-4842-1656-9>. Citado na página 25.

SOUZA, K. *Marisa avança no online com marketplace e já mira carteira digital*. 2021. Disponível em: <https://exame.com/negocios/marisa-avanca-no-on-line-com-marketplace-e-ja-mira-carteira-digital/>. Acesso em: 14 mai 2022. Citado na página 53.

STADNIK, W.; NOWAK, Z. The impact of web pages' load time on the conversion rate of an e-commerce platform. In: . [S.l.: s.n.], 2018. p. 336–345. ISBN 978-3-319-67219-9. Citado na página 16.

S.TANENBAUM, A.; J.WETHERALL, D. *COMPUTER NETWORKS*. 5. ed. [S.l.: s.n.], 2002. Citado na página 18.

STONEBRAKER, M.; CETINTEMEL. "one size fits all": An idea whose time has come and gone. In: _____. *Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker*. Association for Computing Machinery and Morgan & Claypool, 2018. p. 441–462. ISBN 9781947487192. Disponível em: <https://doi.org/10.1145/3226595.3226636>. Citado na página 34.

SUDARSHAN, S.; SILBERSCHATZ, A.; KORTH, F. H. *Sistema De Banco De Dados*. [S.l.]: Elsevier, 2006. ISBN 8535211071. Citado 2 vezes nas páginas 30 e 32.

TRISTAO, M. L. R. *WEB 2.0: ESTRATÉGIA E USABILIDADE*. Dissertação (Mestrado) — PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO - PUC-RIO, Rio de Janeiro - RJ, agosto 2008. Disponível em: <https://www.maxwell.vrac.puc-rio.br/12035/12035_2.PDF>. Citado na página 15.

VENKETESH, P.; VENKATESAN, R. A survey on applications of neural networks and evolutionary techniques in web caching. *IETE Technical Review*, Taylor & Francis Ltd., New Delhi, v. 26, n. 3, p. 171–180, 2009. ISSN 02564602. Disponível em: <http://search.proquest.com/docview/857840579/>. Citado na página 40.

WESSELS, D. *Web Caching*. [S.l.]: O'Reilly Media, Incorporated, 2001. Citado na página 42.

WU, J.; NI, D.; XIAO, Z. N-tier soft set data model: An approach to combine the logicality of sql and the flexibility of nosql. *Mobile information systems*, Hindawi, v. 2021, p. 1–23, 2021. ISSN 1574-017X. Citado na página 31.

YASIN, R. The means to go green: how to eke out energy savings in servers, processors, networks and storage systems.(fcw gcn wt 360)(cover story). *Government Computer News*, 1105 Media, Inc., v. 26, n. 29, p. 18, 2007. ISSN 0738-4300. Citado na página 15.

ZHANG, J.; JIA, Y. Redis rehash optimization based on machine learning. *Journal of Physics: Conference Series*, IOP Publishing, v. 1453, n. 1, p. 012048, jan 2020. Disponível em: <https://doi.org/10.1088/1742-6596/1453/1/012048>. Citado na página 35.

ZULFA, M.; HARTANTO, R.; ADHISTYA, E. Caching strategy for web application – a systematic literature review. Emerald Group Publishing Limited, Bingley, v. 16, n. 5, p. 545–569, 2020. ISSN 17440084. Disponível em: <http://search.proquest.com/docview/2457789214/>. Citado 3 vezes nas páginas 43, 44 e 59.

ZULFA, M. I.; FADLI, A.; WARDHANA, A. W. Application caching strategy based on in-memory using redis server to accelerate relational data access. Diponegoro University, v. 8, n. 2, p. 157–163, 2020. ISSN 2338-0403. Disponível em: <https://doaj.org/article/8db9b85da50f4d60817efbda1532524b>. Citado 3 vezes nas páginas 16, 33 e 36.

# Apêndices

# APÊNDICE A – APÊNDICES

## A.1 Arquivos de configuração

### A.1.1 Arquivo docker-compose.yaml do subsistema A

```
version: '3.3'

services:
  cassandra:
    image: cassandra:latest
    container_name: cassandra
    hostname: cassandra
    networks:
      app-a-network:
        ipv4_address: 172.20.0.6
    ports:
      - '9042:9042'
    volumes:
      - ./src/cassandra/data/product.cql:/product.cql
      - ./src/cassandra/cassandra.yaml:/etc/cassandra/cassandra.yaml

  cassandra-load-keyspace:
      container_name: cassandra-load-keyspace
      image: cassandra:latest
      networks: ["app-a-network"]
      depends_on:
        - cassandra
      volumes:
        - ./src/cassandra/data/product.cql:/product.cql
      command: /bin/bash -c "sleep 120 && echo loading cassandra keyspace && cqlsh
    cassandra -f /product.cql"

  appa-api:
    image: app-a
    container_name: appA-api
    networks:
      app-a-network:
        ipv4_address: 172.20.0.5
    ports:
      - '6060:6060'
    depends_on:
      - cassandra

```

```
38 networks:
39   app-a-network:
40     driver: bridge
41     ipam:
42       config:
43         - subnet: 172.20.0.0/24
```

### A.1.2 Arquivo docker-compose.yaml do subsistema B

```
version: '3.3'
services:
  redis:
    image: redis
    container_name: redis
    ports:
      - "6379:6379"
    networks:
      app-b-network:
        ipv4_address: 172.20.1.5
    command: redis-server
    hostname: redis

  prometheus:
    build:
      dockerfile: src/metrics/prometheus/prom.dockerfile
      context: .
    image: marina/prom
    container_name: prometheus
    ports:
      - "9090:9090"
    networks:
      app-b-network:
        ipv4_address: 172.20.1.6
    extra_hosts:
      - host.docker.internal:host-gateway
  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    networks:
      - app-b-network
    environment:
      - GF_AUTH_BASIC_ENABLED=false
      - GF_AUTH_DISABLE_LOGIN_FORM=true
      - GF_AUTH_DISABLE_SIGNOUT_MENU=true
      - GF_AUTH_ANONYMOUS_ENABLED=true
      - GF_AUTH_ANONYMOUS_ORG_ROLE=Admin
      - GF_SERVER_ROOT_URL=http://localhost:3000/grafana/
      - GF_SERVER_SERVE_FROM_SUB_PATH=true
    extra_hosts:
      - host.docker.internal:host-gateway

  node1:
    build:
```

```
47        dockerfile: Dockerfile
48        context: .
49      image: app-b
50      container_name: tcc-app-b-1
51      ports:
52        - "4040:4040"
53      networks:
54        app-b-network:
55          ipv4_address: 172.20.1.7
56      depends_on:
57        - "redis"
58
59
60  networks:
61    app-b-network:
62      driver: bridge
63      ipam:
64        config:
65          - subnet: 172.20.1.0/24
```

### A.1.3 Arquivo docker-compose.yaml do tópico Kafka

```
1  ---
2  version: '3'
3  services:
4    zookeeper:
5      image: confluentinc/cp-zookeeper:7.0.1
6      container_name: zookeeper
7      environment:
8        ZOOKEEPER_CLIENT_PORT: 2181
9        ZOOKEEPER_TICK_TIME: 2000
10
11   broker:
12     image: confluentinc/cp-kafka:7.0.1
13     container_name: broker
14     ports:
15       - '9094:9094'
16       - '9092:9092'
17       - '9095:9095'
18       - '9096:9096'
19       - '9097:9097'
20     depends_on:
21       - zookeeper
22     environment:
23       KAFKA_BROKER_ID: 1
24       KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
```

```
25      KAFKA_LISTENERS: INTERNAL://0.0.0.0:9092,OUTSIDE://0.0.0.0:9094,M2
    ://0.0.0.0:9095,M3://0.0.0.0:9096,M4://0.0.0.0:9097
26      KAFKA_ADVERTISED_LISTENERS: INTERNAL://191.36.8.52:9092,OUTSIDE
    ://191.36.8.52:9094,M2://tccmarina.sj.ifsc.edu.br/:9095,M3://191.36.8.53:9096,
    M4://191.36.8.54:9097
27      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,OUTSIDE:PLAINTEXT,M2
    :PLAINTEXT,M3:PLAINTEXT,M4:PLAINTEXT
28      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
29      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
30      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
31      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
```

## A.2 Relatório JMeter

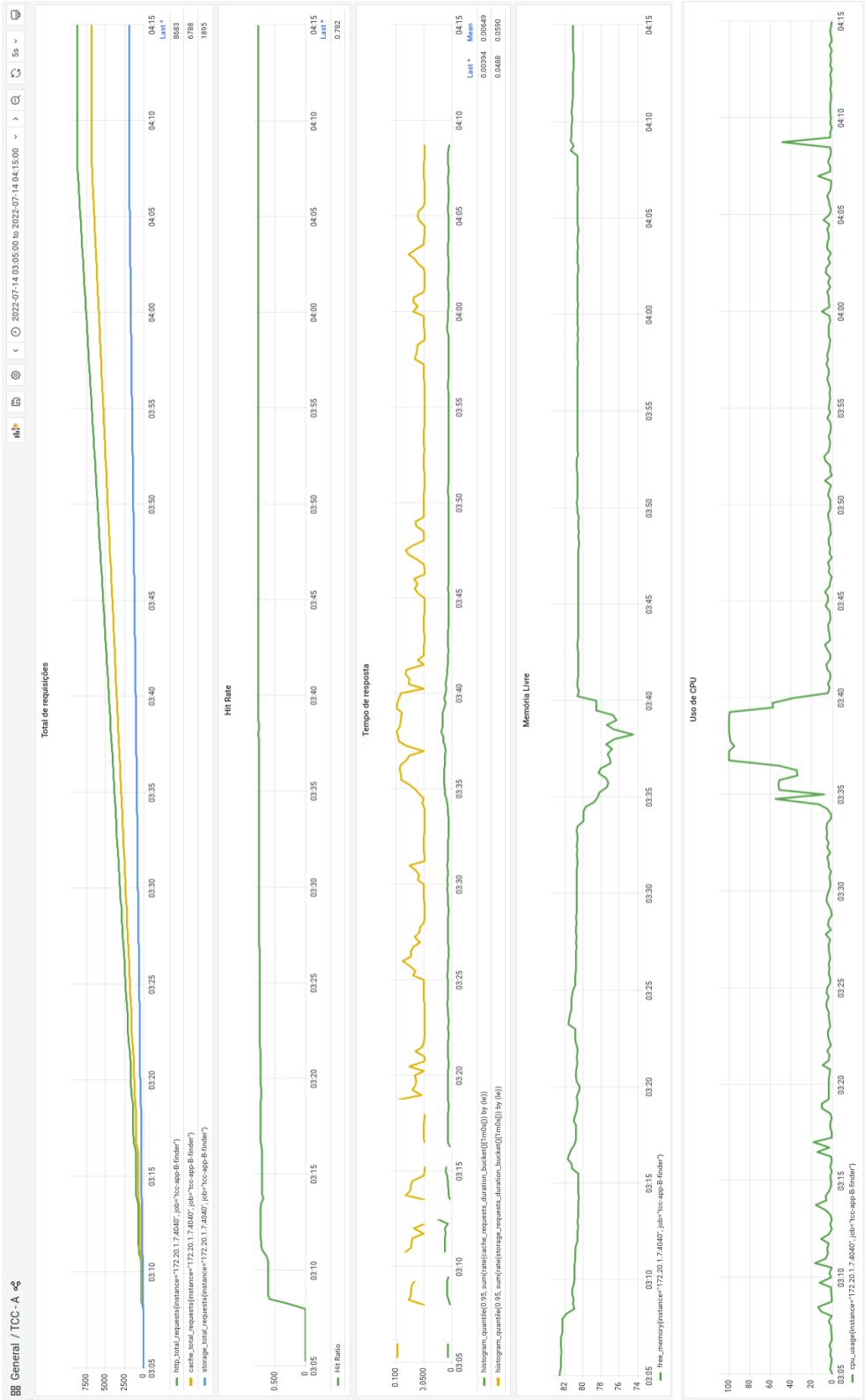
Figura 28 – Trecho do relatório gerado pelo JMeter

| Timestamp | Loadtime | Response Code | Size in Bytes | Threads Active | Threads Total | Request | Latency |
|---|---|---|---|---|---|---|---|
| 1657769953399 | 315 | 200 | 83213 | 1 | 1 | https://tccmarina.sj.ifsc.edu.br/finder/search/Cat | 32 |
| 1657769954422 | 111 | 200 | 6511 | 1 | 1 | https://tccmarina.sj.ifsc.edu.br/finder/search/Energy%20Drink | 34 |
| 1657769954768 | 115 | 200 | 11724 | 1 | 1 | https://tccmarina.sj.ifsc.edu.br/finder/search/Jelly | 35 |
| 1657769955109 | 159 | 200 | 39783 | 1 | 1 | https://tccmarina.sj.ifsc.edu.br/finder/search/Pillow | 33 |
| 1657769955609 | 106 | 200 | 3106 | 1 | 1 | https://tccmarina.sj.ifsc.edu.br/finder/search/Chicago%20Cubs | 35 |
| 1657769955936 | 108 | 200 | 7405 | 1 | 1 | https://tccmarina.sj.ifsc.edu.br/finder/search/Fisher | 32 |
| 1657769956258 | 245 | 200 | 92170 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Pool | 30 |
| 1657769956401 | 299 | 200 | 127433 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Sports | 35 |
| 1657769956985 | 219 | 200 | 75005 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Butter | 32 |
| 1657769957279 | 180 | 200 | 52688 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Mountain | 33 |
| 1657769957656 | 106 | 200 | 1915 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Scar%20Removal | 35 |
| 1657769957841 | 132 | 200 | 20714 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Cereal | 32 |
| 1657769957969 | 339 | 200 | 158569 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Women | 30 |
| 1657769958242 | 95 | 200 | 2738 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Clorox | 30 |
| 1657769958568 | 109 | 200 | 6238 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Neutrogena | 33 |
| 1657769958888 | 96 | 200 | 6509 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Energy%20Drink | 34 |
| 1657769958891 | 97 | 200 | 1968 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Jelly%20Beans | 31 |
| 1657769959195 | 142 | 200 | 20205 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Disney | 31 |
| 1657769959143 | 234 | 200 | 90725 | 2 | 2 | https://tccmarina.sj.ifsc.edu.br/finder/search/Baseball | 29 |
| 1657769959400 | 221 | 200 | 80664 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/Chocolate | 33 |
| 1657769959840 | 111 | 200 | 1849 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/Piano%20Keyboard | 35 |
| 1657769959610 | 381 | 200 | 229774 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/Water | 30 |
| 1657769960147 | 140 | 200 | 786 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/CollarCups | 35 |
| 1657769960031 | 354 | 200 | 168680 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/Bicycle | 31 |
| 1657769960612 | 130 | 200 | 8258 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/Dark%20Chocolate | 38 |
| 1657769960648 | 94 | 200 | 6509 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/Energy%20Drink | 34 |
| 1657769960958 | 94 | 200 | 6509 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/Energy%20Drink | 35 |
| 1657769960962 | 100 | 200 | 4161 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/OralB | 30 |
| 1657769960908 | 192 | 200 | 90723 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/Baseball | 33 |
| 1657769961194 | 102 | 200 | 2352 | 3 | 3 | https://tccmarina.sj.ifsc.edu.br/finder/search/Star%20Wars | 32 |

Fonte: Elaborado pela autora

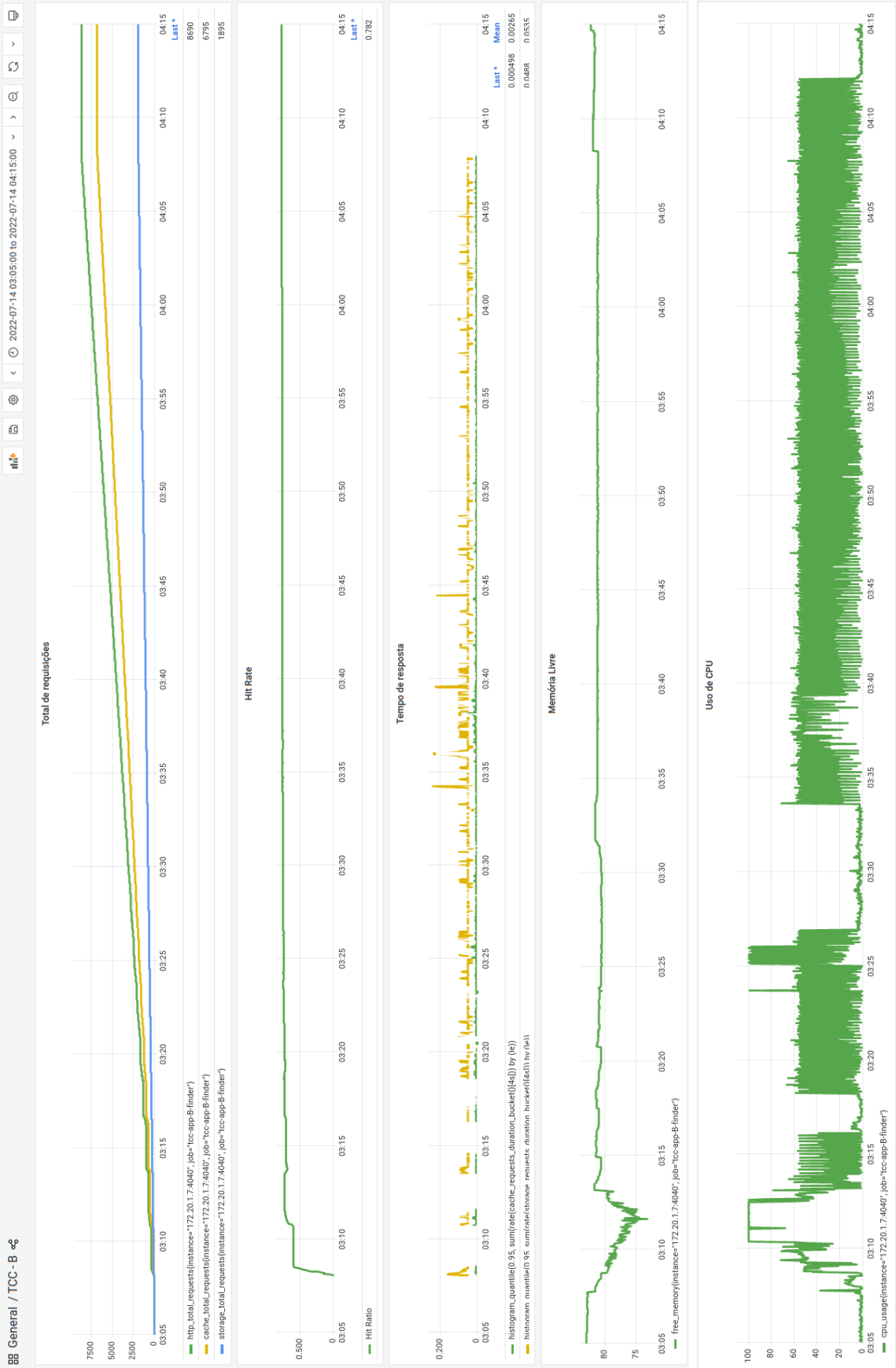
## A.3 Gráficos resultantes

Figura 29 – Gráfico resultante do cenário 1 - Cache Passivo após a primeira execução de testes



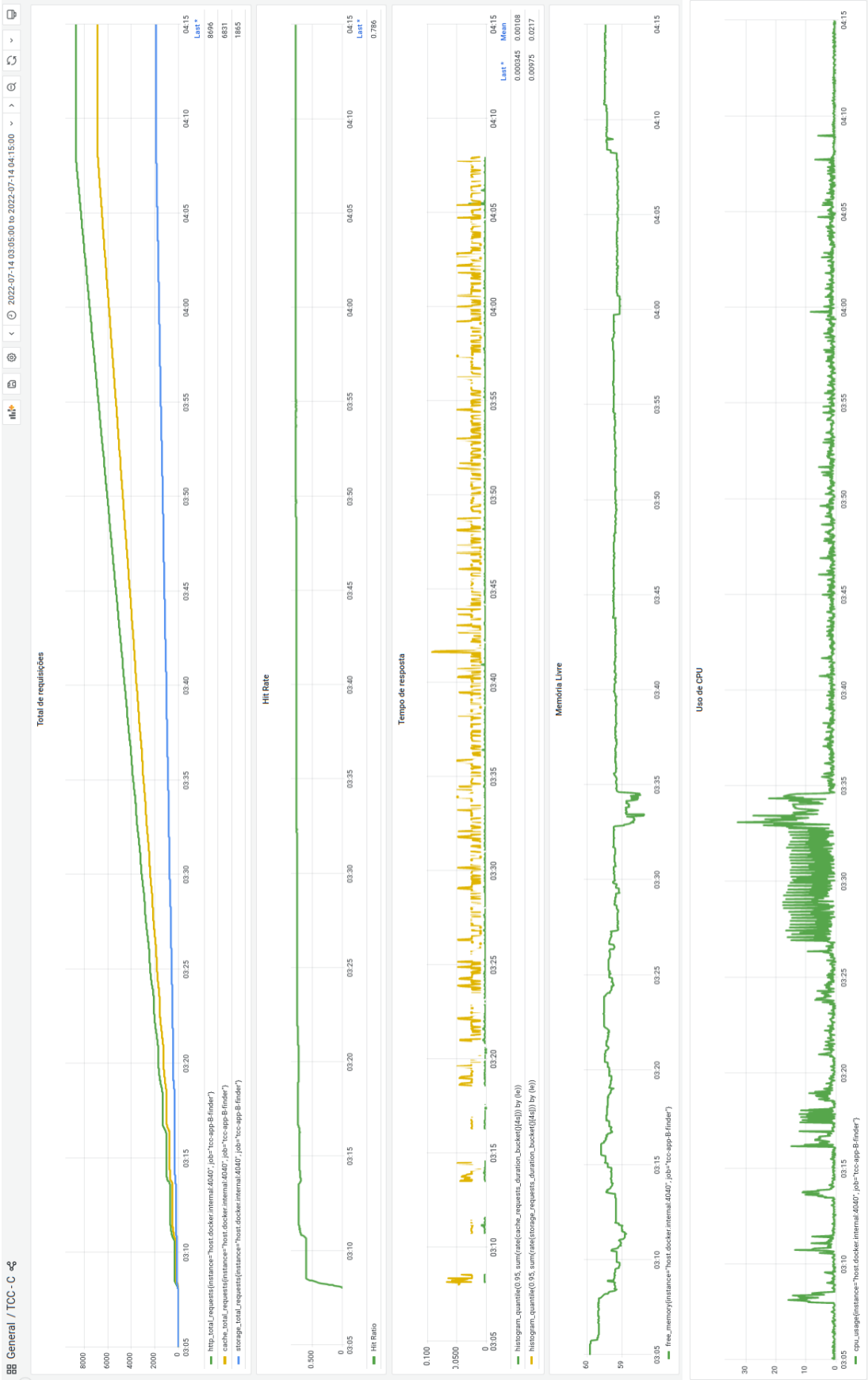

Fonte: Elaborado pela autora

Figura 30 – Gráfico resultante do Cenário 2 - Cache pró-ativo após a primeira execução de testes



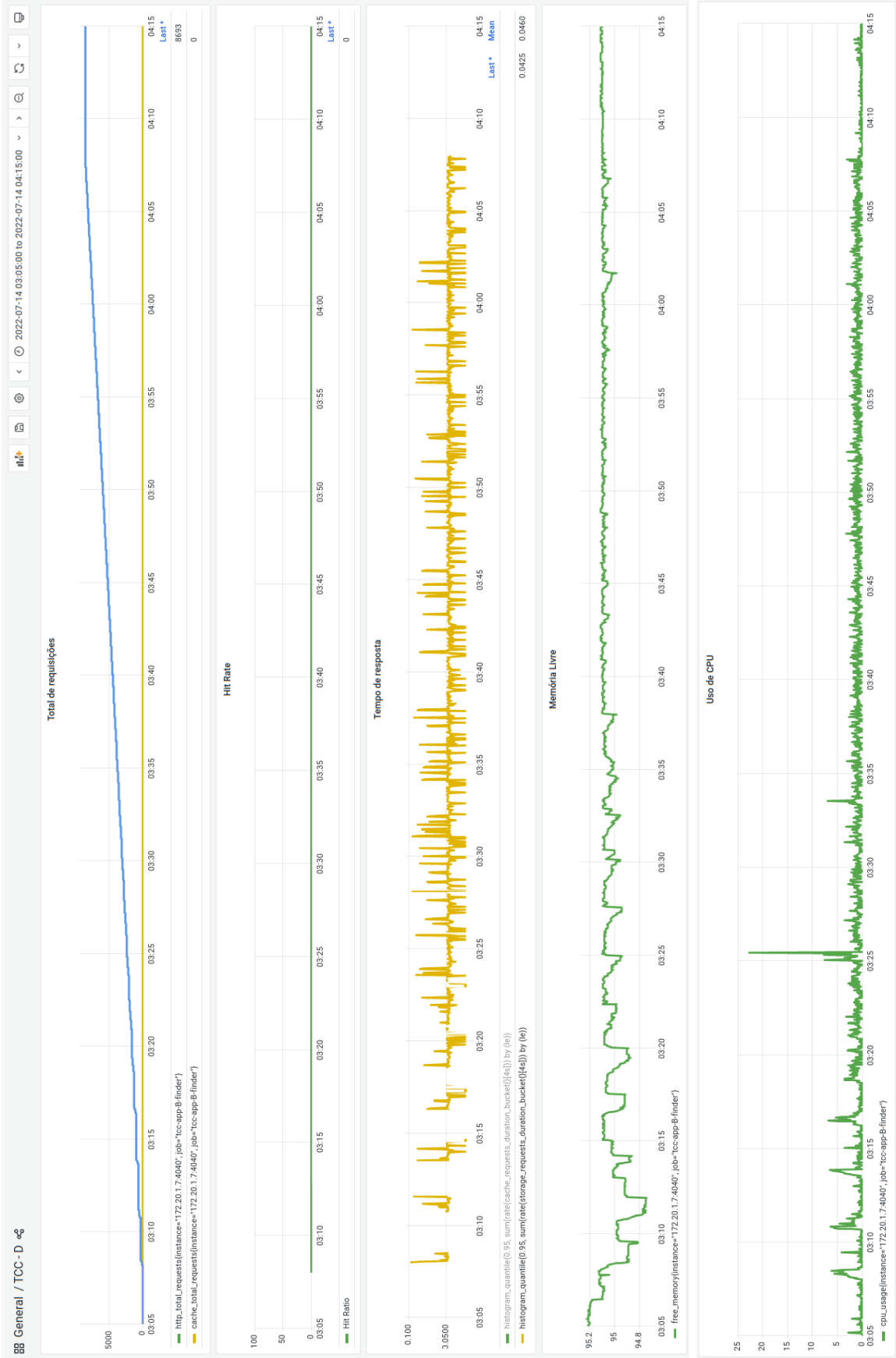

Fonte: Elaborado pela autora

Figura 31 – Gráfico resultante do Cenário 3 - Cache pró-ativo com TTL adaptativo após a primeira execução de testes



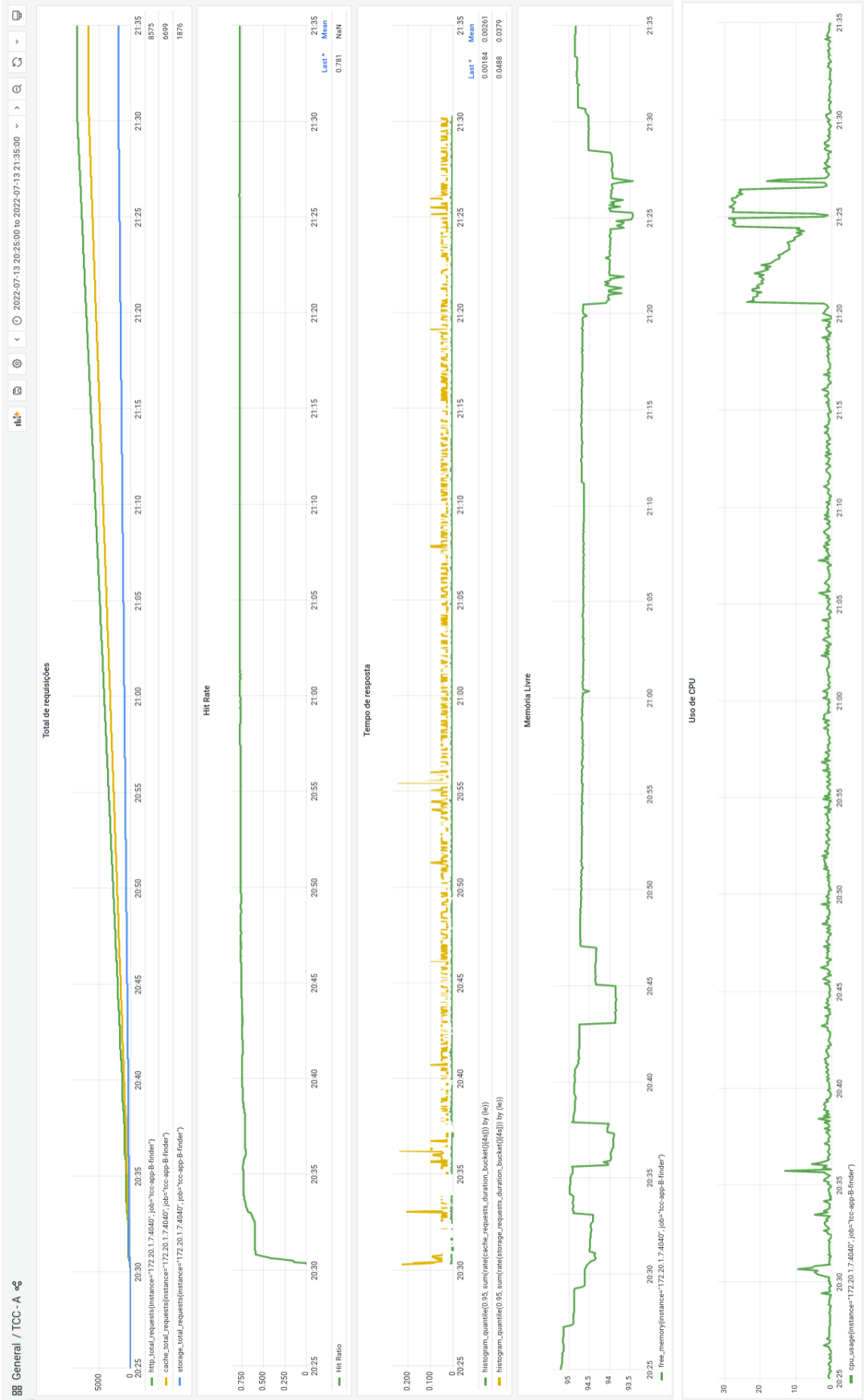

Fonte: Elaborado pela autora

Figura 32 – Gráfico resultante do Cenário 4 - Sem cache após a primeira execução de testes



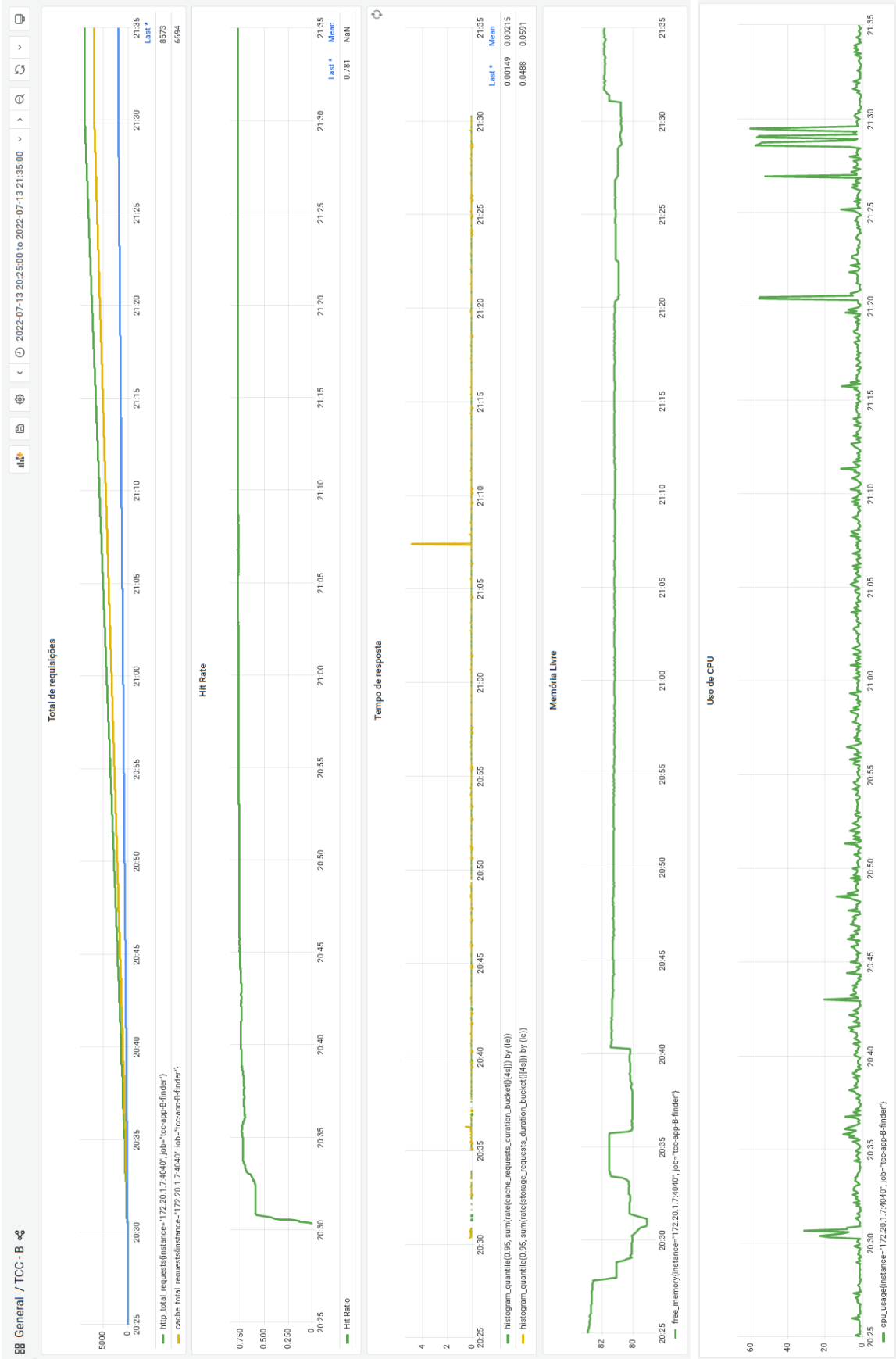

Fonte: Elaborado pela autora

Figura 33 – Gráfico resultante do cenário 1 - Cache Passivo após a segunda execução de testes




Fonte: Elaborado pela autora

Figura 34 – Gráfico resultante do Cenário 2 - Cache pró-ativo após a segunda execução de testes

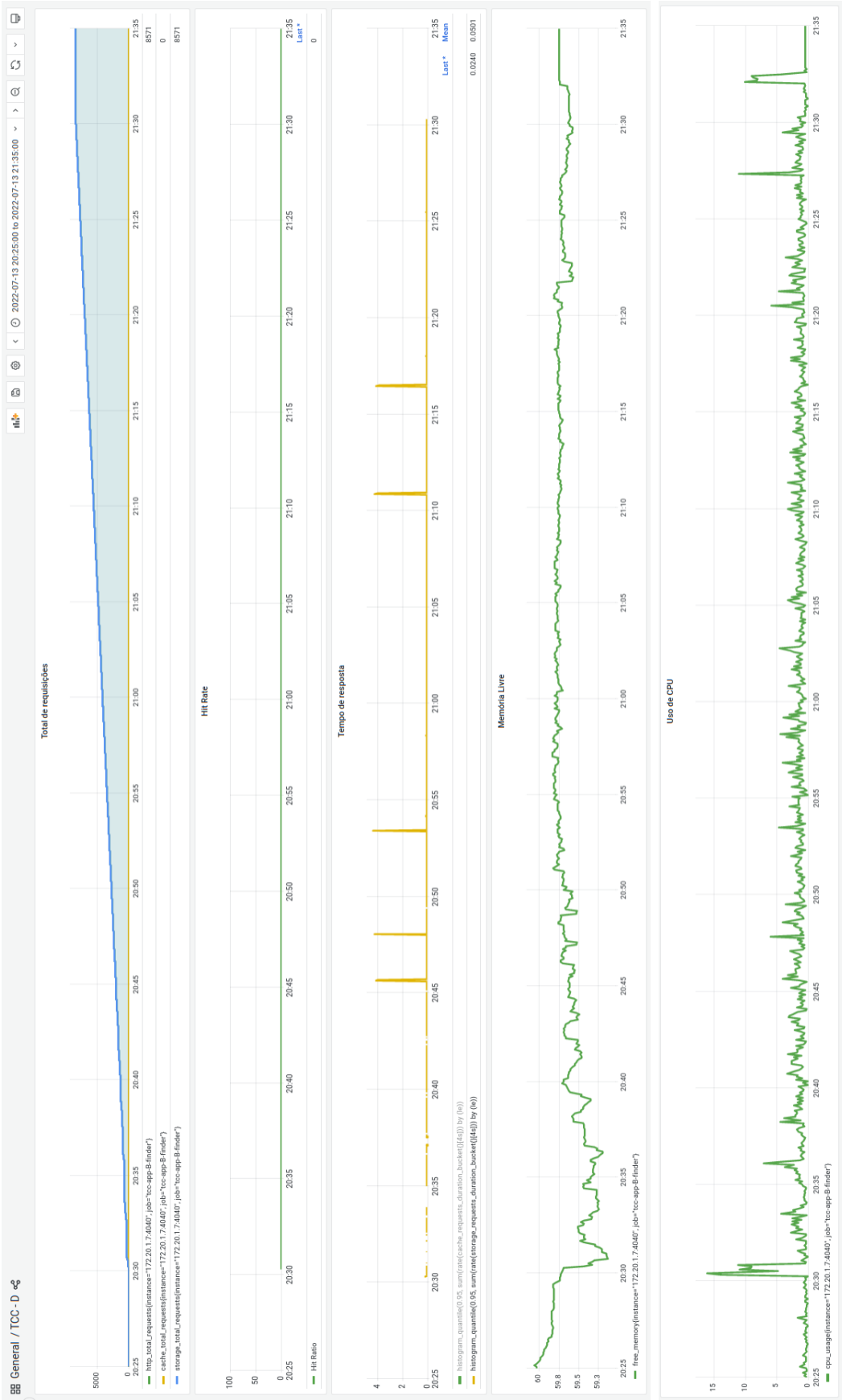



Fonte: Elaborado pela autora

Figura 35 – Gráfico resultante do Cenário 3 - Cache pró-ativo com TTL adaptativo após a segunda execução de testes

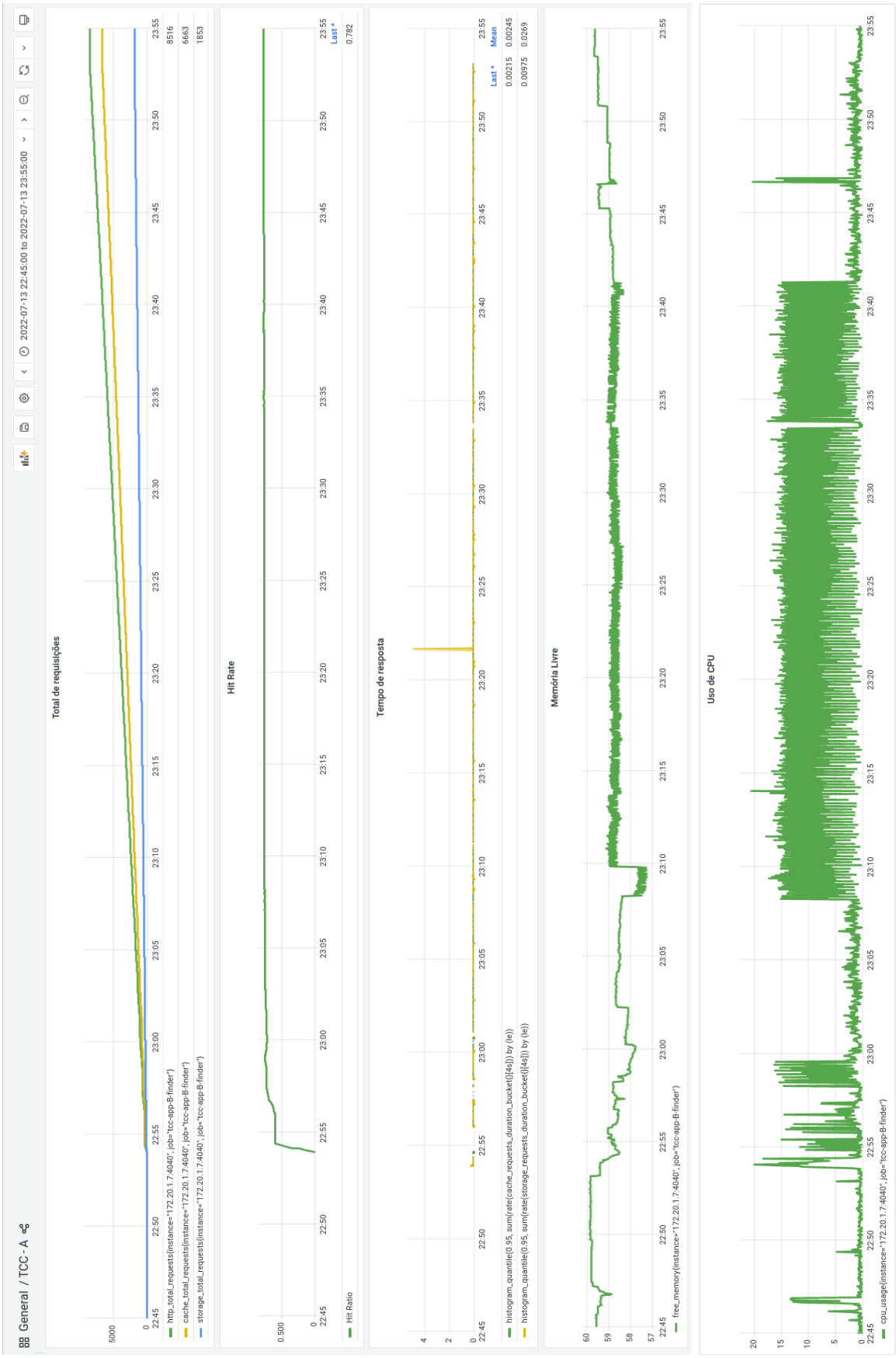

Fonte: Elaborado pela autora

Figura 36 – Gráfico resultante do Cenário 4 - Sem cache após a segunda execução de testes



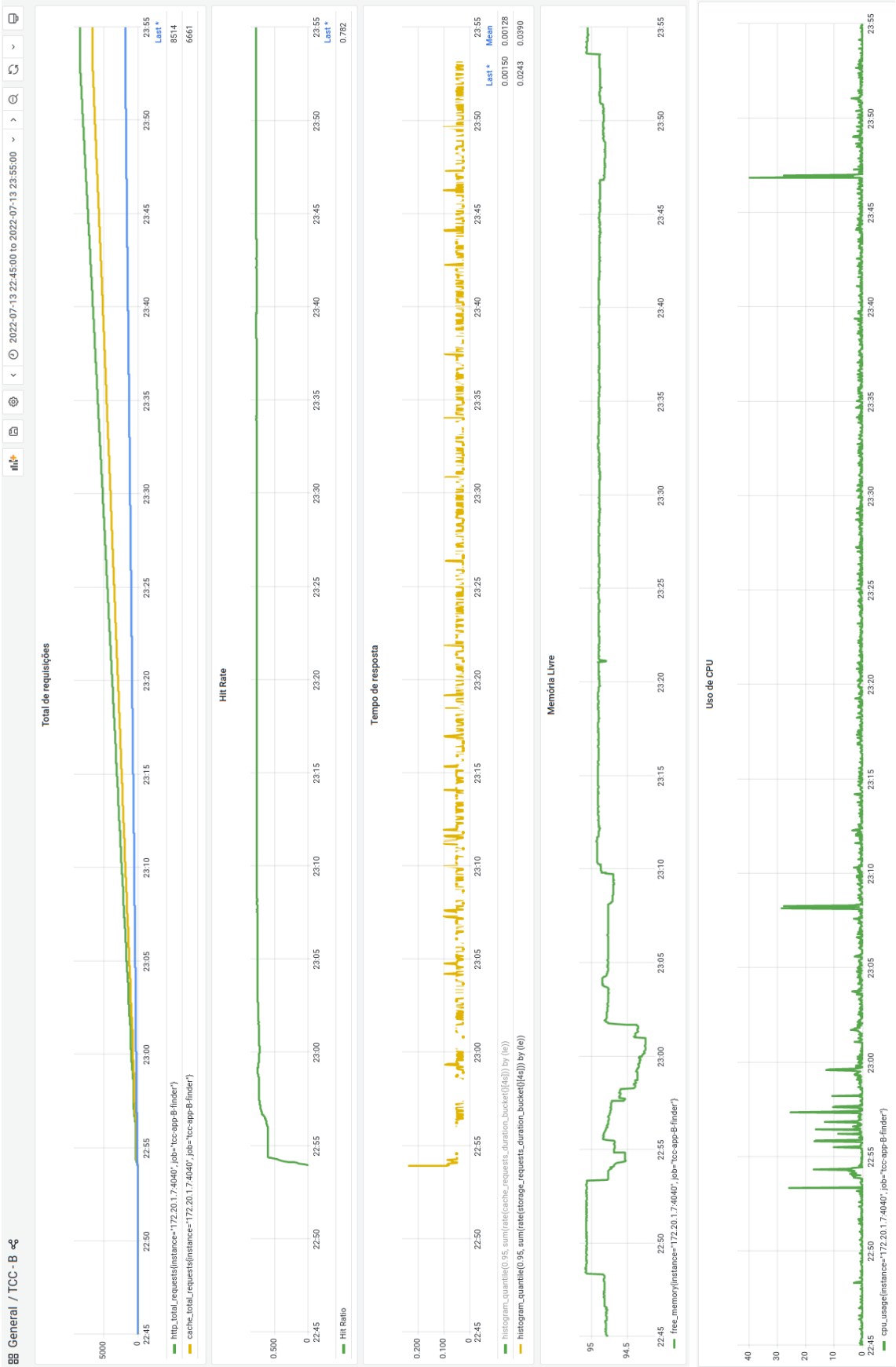

Fonte: Elaborado pela autora

Figura 37 – Gráfico resultante do cenário 1 - Cache Passivo após a terceira execução de testes




Fonte: Elaborado pela autora

Figura 38 – Gráfico resultante do Cenário 2 - Cache pró-ativo após a terceira execução de testes

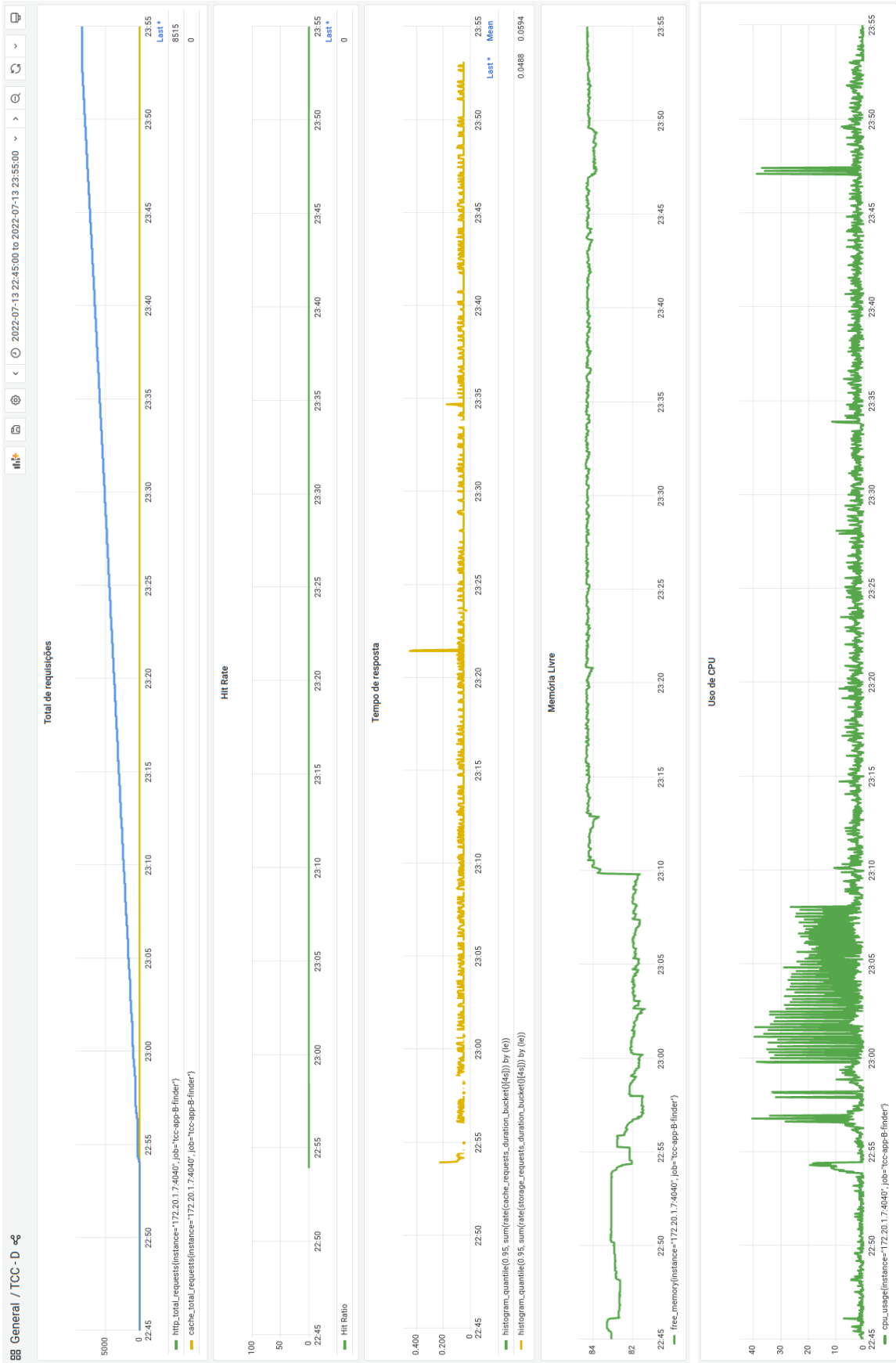



Fonte: Elaborado pela autora

Figura 39 – Gráfico resultante do Cenário 3 - Cache pró-ativo com TTL adaptativo após a terceira execução de testes

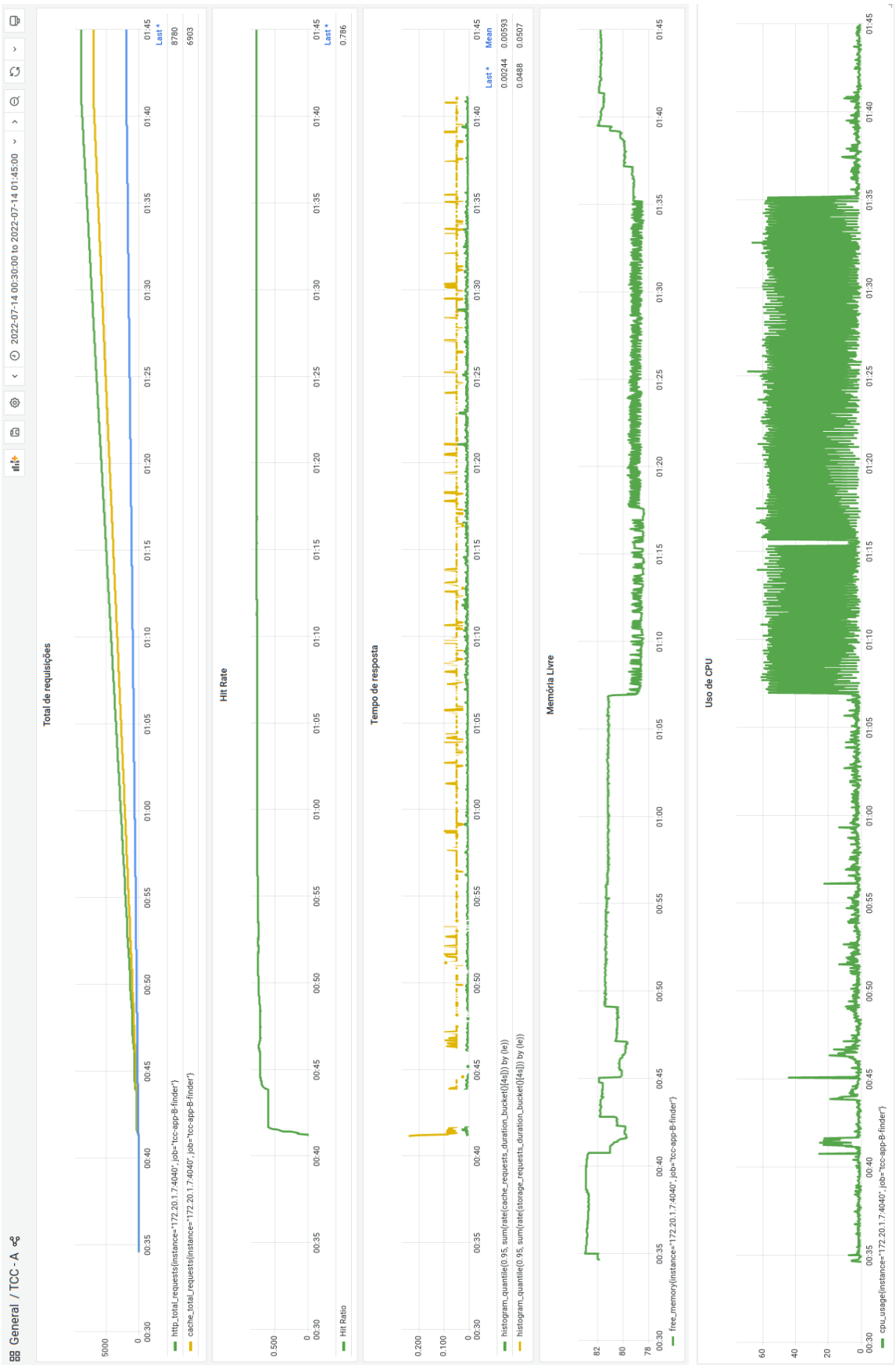

Fonte: Elaborado pela autora

Figura 40 – Gráfico resultante do Cenário 4 - Sem cache após a terceira execução de testes



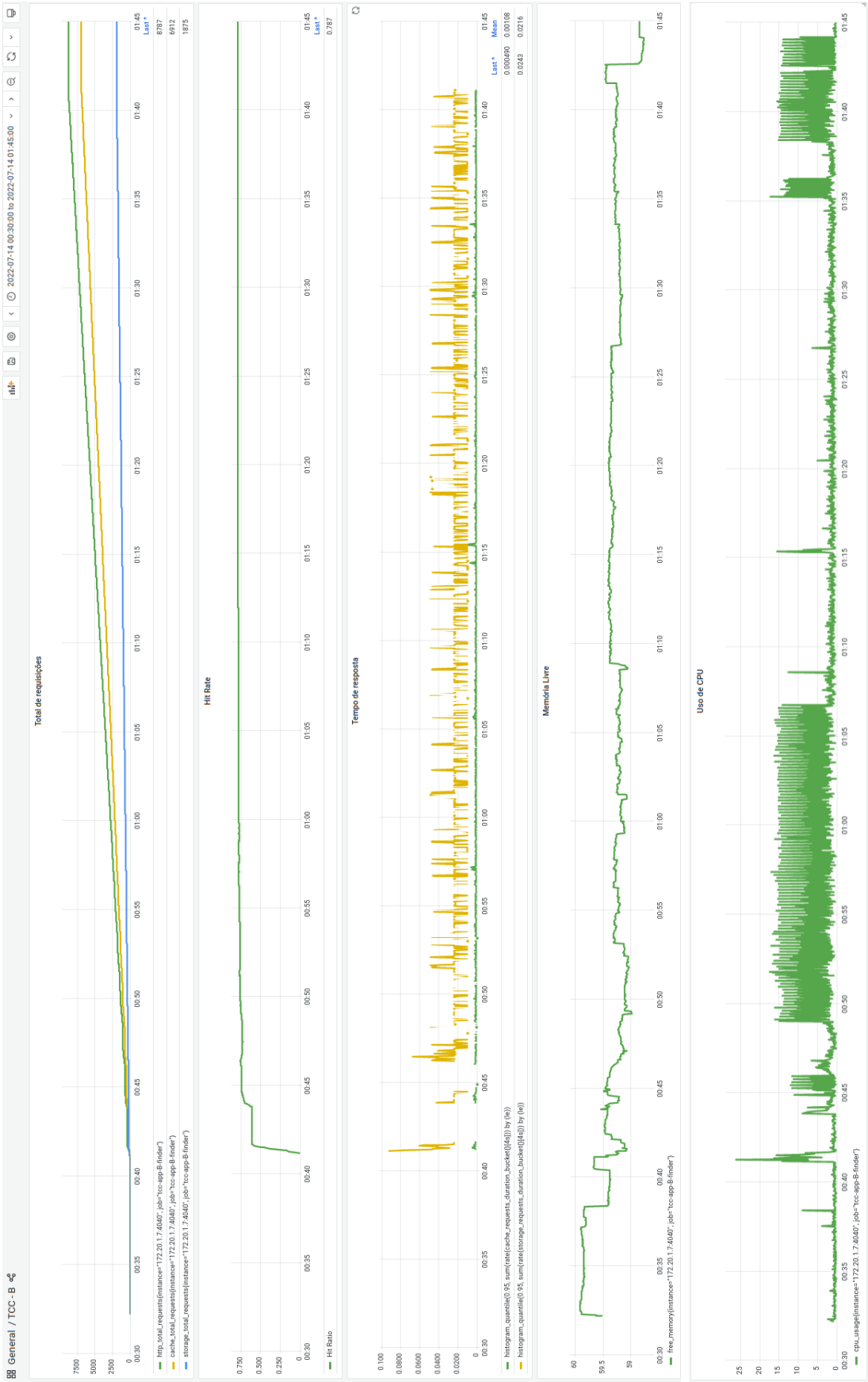

Fonte: Elaborado pela autora

Figura 41 – Gráfico resultante do cenário 1 - Cache Passivo após a quarta execução de testes



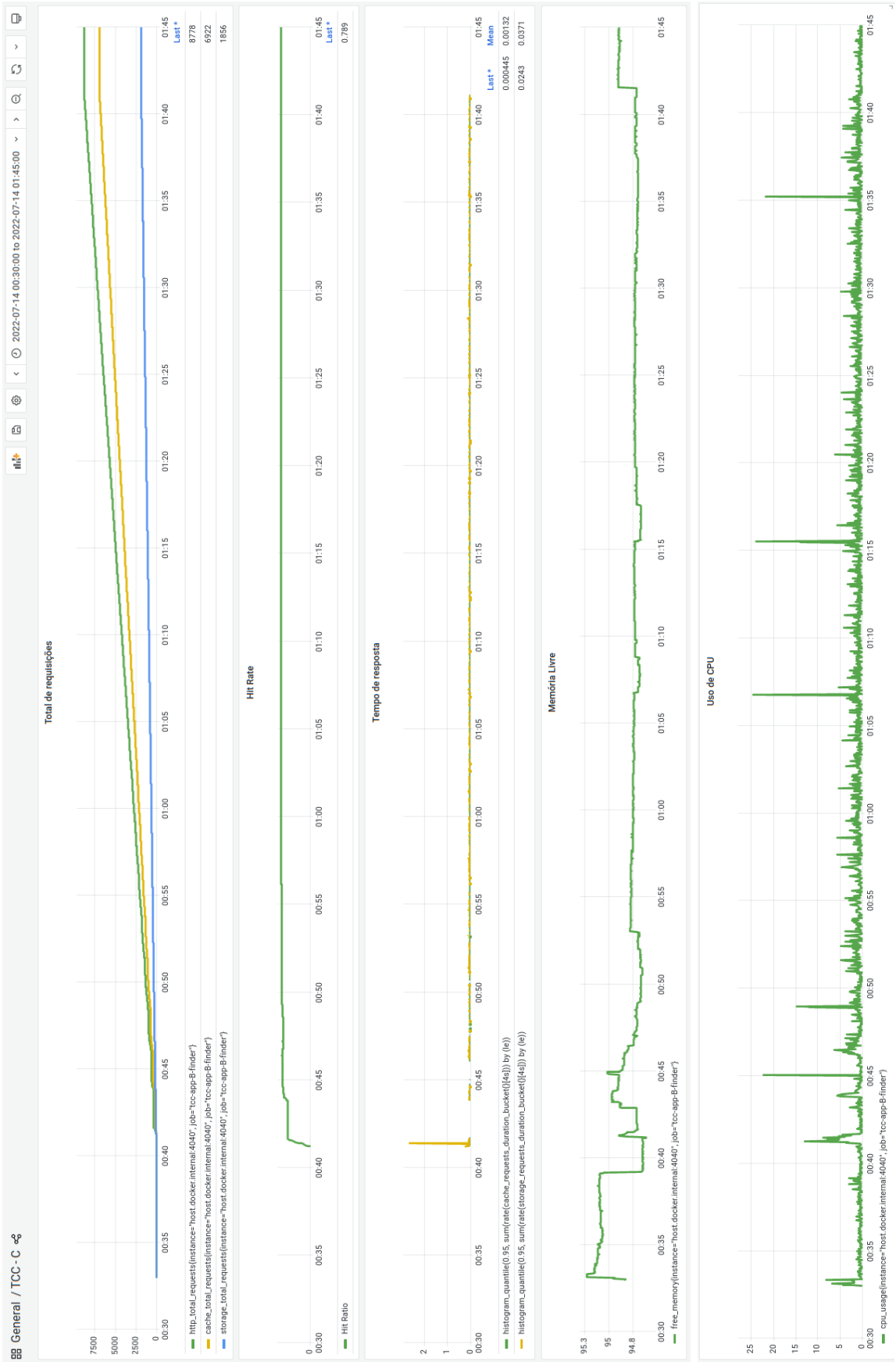

Fonte: Elaborado pela autora

Figura 42 – Gráfico resultante do Cenário 2 - Cache pró-ativo após a quarta execução de testes



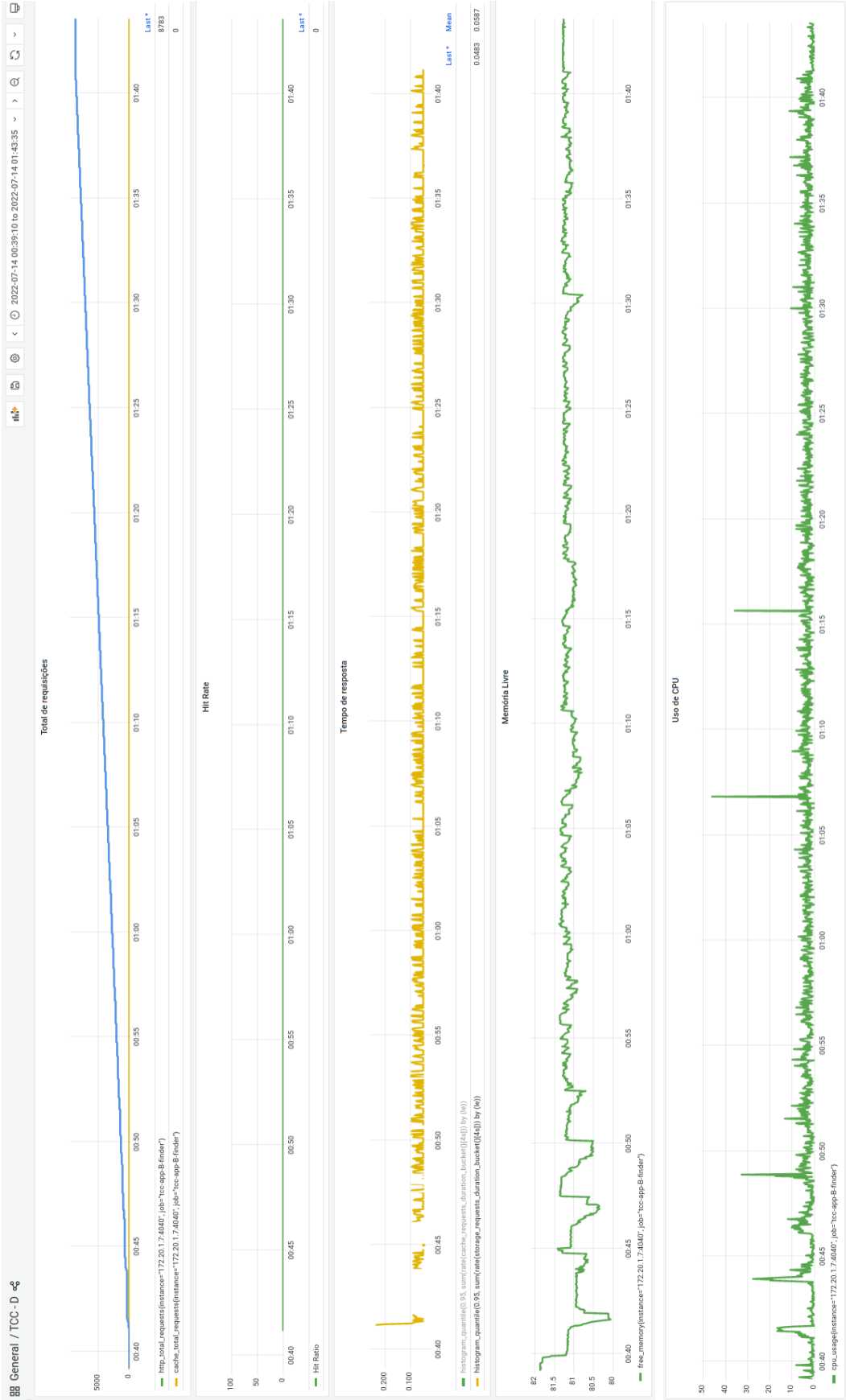

Fonte: Elaborado pela autora

Figura 43 – Gráfico resultante do Cenário 3 - Cache pró-ativo com TTL adaptativo após a quarta execução de testes




Fonte: Elaborado pela autora

Figura 44 – Gráfico resultante do Cenário 4 - Sem cache após a quarta execução de testes




Fonte: Elaborado pela autora