Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

REVISTA SOCIEDADE CIENTÍFICA, VOLUME 9, NÚMERO 1, ANO 2026

ARTIGO ORIGINAL

Análise da eficiência do uso de recursos de máquina na aplicação de diferentes práticas de observabilidade em ambientes com *hardware* limitado

Gabriel Augusto Souza Borges[1]; Danilo de Quadros Maia Filho[2]

Como Citar:
BORGES, Gabriel Augusto Souza; MAIA FILHO, Danilo de Quadros. Análise da eficiência do uso de recursos de máquina na aplicação de diferentes práticas de observabilidade em ambientes com hardware limitado. Revista Sociedade Científica, vol. 9, n. 1, p. 1451-1477, 2026. https://doi.org/10.61411/rsc2026134919

DOI: 10.61411/rsc2026134919

Área do conhecimento:
Ciências Exatas e da Terra
Sub-área:
Ciência da Computação

Palavras-chave: Observabilidade; Monitoramento de Sistemas; Otimização de Desempenho.

Publicado: 10 de junho de 2026.

Resumo

A observabilidade é uma prática essencial para monitorar sistemas modernos, mas sua implementação pode gerar custos elevados em recursos de *hardware*, como processamento, memória e armazenamento. O problema central abordado é a ineficiência no uso desses recursos, que ocorre devido à ausência de diretrizes para equilibrar a qualidade do monitoramento e o custo computacional. Para investigar soluções, este trabalho seguiu uma metodologia experimental que comparou a eficiência de diferentes práticas de observabilidade em um ambiente controlado com *hardware* limitado. Uma API foi submetida a testes de estresse, enquanto o consumo de recursos era monitorado. Os resultados demonstraram que a aplicação de diretrizes específicas de observabilidade — amostragem simples, amostragem adaptativa e redução da cardinalidade de métricas — permitiu reduzir o uso da CPU em mais de 40% e diminuir o tempo médio de resposta em aproximadamente 45%, em comparação com uma abordagem sem otimizações. Com base nesses achados, o estudo busca direcionar a coleta otimizada de dados, oferecendo um equilíbrio entre eficiência no uso de recursos e qualidade da observabilidade em ambientes de *hardware* restritos.

Analysis of Machine Resource Utilization Efficiency in the Implementation of Different Observability Practices in Hardware-Constrained Environments

Abstract

Observability is an essential practice for monitoring modern systems, but its implementation can generate high hardware resource costs in processing, memory, and

[1]Pontifícia Universidade Católica de Minas Gerais (PUC Minas), Belo Horizonte-MG, Brasil. Email: ⋈
[2]Pontifícia Universidade Católica de Minas Gerais (PUC Minas), Belo Horizonte-MG, Brasil. Email: ⋈

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

storage. The central problem addressed is the inefficient use of these resources, which stems from a lack of guidelines to balance monitoring quality and computational cost. To investigate solutions, this work followed an experimental methodology that compared the efficiency of different observability practices in a controlled, resource-constrained hardware environment. An API was subjected to stress tests while resource consumption was monitored. The results demonstrated that the application of specific observability guidelines — simple sampling, adaptive sampling, and metric cardinality reduction — enabled a reduction in CPU usage by over 40% and decreased the average response time by approximately 45% compared to a non-optimized approach. Based on these findings, The study aims to direct the data optimization collection, offering a balance between resource use efficiency and observability quality in constrained hardware environments.

Keywords: Observability; System Monitoring; Performance Optimization.

1. Introdução

Para garantir a confiabilidade e o desempenho de aplicações modernas, o conceito de Observabilidade surge como um conjunto de práticas que visa compreender o estado interno de um sistema por meio da análise de seus dados externos [12]. Em sistemas distribuídos modernos, onde a complexidade é alta e falhas podem ser difíceis de diagnosticar, a observabilidade torna-se essencial para detecção e resolução ágil de problemas [12]. Seus principais pilares são métricas (dados quantitativos sobre desempenho), *logs* (registros de eventos temporais) e traces (rastreamento de requisições entre serviços) [10].

A implantação incorreta de diretrizes de observabilidade, ou seja, regras práticas para coletar, processar e armazenar dados de monitoramento, pode ser custosa em infraestrutura e tempo humano quando feita sem critérios de otimização, especialmente se houver alta cardinalidade de métricas, registro completo de *logs* e rastreamento de

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

todas as requisições. Essas práticas exigem maior capacidade de armazenamento, enquanto o processamento desses dados em tempo real pode gerar custos computacionais adicionais [7].

Embora existam trabalhos que estudem o impacto negativo nos recursos de máquina durante o uso de ferramentas de observabilidade como o ElasticSearch [2], a diferenciação (ou seja, o que este trabalho se diferencia do que já existe) é a abordagem do problema da ineficiência no uso de recursos de *hardware* causada pela ausência de diretrizes para equilibrar qualidade do monitoramento e custo computacional em práticas de observabilidade.

Resolver este problema é relevante, pois a escolha inadequada de práticas de observabilidade como a captura excessiva de dados sem restrições pode causar problemas no desempenho do sistema [9] aumentando custos de infraestrutura e reduzindo a escalabilidade dos *softwares*. Com os gargalos gerados, os sistemas de computador conseguem lidar com menos tráfego novo. Outro aspecto importante é que em cenários de ambientes com restrições de *hardware*, a otimização desses recursos é crítica, sabendo que a performance de ferramentas de observabilidade pode ser diretamente impactada pela saturação dos recursos computacionais das máquinas nas quais a aplicação é executada [4].

Portanto, o objetivo geral desse estudo é avaliar como diferentes tratativas de observabilidade afetam a eficiência no uso de recursos de *hardware* em ambientes de produção. A definição e a caracterização dessas tratativas serão apresentadas na metodologia. Como objetivos específicos, têm-se: (i) Quantificar o impacto de diferentes tratativas de observabilidade no consumo de CPU (CPU, do inglês Central Processing Unit), memória e latência; (ii) Investigar a relação entre o custo computacional da coleta de dados e o valor em *insights* gerados para a Engenharia de *software*; e (iii) Melhor direcionar o uso de diretrizes presentes na literatura para otimizar a coleta de dados, como estratégias de amostragem adaptativa.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

As expectativas de desempenho foram baseadas em evidências preliminares e na literatura existente. Madupati [7] observou que a instrumentação de observabilidade pode gerar sobrecarga de aproximadamente um quarto na CPU, sugerindo que estratégias de otimização poderiam reverter significativamente esse impacto. Paralelamente, Costa e Araújo [2] alcançaram redução de 80% no armazenamento de dados de observabilidade sem comprometer a qualidade. Neste contexto, espera-se que as estratégias propostas resultem em redução de pelo menos 30% no uso de CPU, 20% no consumo de memória e 15% no tempo médio de resposta em comparação com a abordagem sem otimizações. A amostragem adaptativa, em particular, é esperada como a estratégia mais eficaz por acumular todas as otimizações propostas.

As próximas seções do artigo estão divididas da seguinte forma: na Seção 2, são apresentados o referencial teórico e os trabalhos relacionados. A Seção 3, descreve a metodologia adotada para a realização da pesquisa. Na Seção 4, são apresentados e discutidos os resultados obtidos, bem como as ameaças à validade do estudo. A Seção 5 apresenta as considerações finais. Por fim, a Seção 6 reúne as indicações de trabalhos futuros e o pacote de replicação.

2. Referencial teórico

A observabilidade, termo originado da teoria de controle, refere-se a capacidade de entender o comportamento de um sistema por meio dos dados que ele gera, facilitando o monitoramento e a análise em tempo de execução [5]. Na computação, seu principal objetivo é reduzir o tempo necessário para diagnosticar problemas. Essa capacidade é sustentada por três pilares fundamentais: *logs*, métricas e *traces* [10].

O armazenamento regular de métricas, prática conhecida na literatura técnica como monitoramento, é um dos mecanismos fundamentais para detectar anomalias em sistemas computacionais [10]. Esse processo gera séries temporais que, ao serem analisadas, revelam padrões e tendências no comportamento do sistema. Essas métricas

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

são organizadas em diferentes níveis ou categorias, cada um responsável por monitorar aspectos específicos, como desempenho, comportamento e integridade do sistema [3].

Logs são registros imutáveis de eventos, podendo ser estruturados ou não, coletados de diversas fontes, como aplicações e sistemas operacionais e possuem um nível de detalhamento maior que as métricas [3]. Os *logs* são coletados diretamente de processos em execução. No entanto, ao contrário das métricas, que são registradas de forma periódica, os *logs* são gerados de maneira não programada, pois dependem da ocorrência de eventos específicos [10]. A estrutura dos *logs* pode ser de forma organizada (em formatos como JSON, do inglês Javascript Object Notation) ou não estruturada (como linhas de texto livre).

Enquanto métricas e *logs* oferecem dados isolados sobre cada serviço, os traces mapeiam o caminho das requisições em um ambiente de *software* [9]. Sejam esses caminhos internos, passando por classes do sistema ou rastros de outros serviços externos. Assim, os traces mostram claramente as relações causais entre requisições, mapeando as complexas interações típicas dos sistemas. Essa capacidade é fundamental para rastrear em tempo de execução como um evento se propaga por toda a arquitetura [10].

Além do objetivo principal de reduzir tempo necessário para diagnosticar falhas no sistema sustentado pelo uso de métricas, *logs* e *traces*, outras vantagens podem ser providas pela observabilidade, como a ampliação da visibilidade do sistema, tornando mais transparente seu funcionamento interno. Ela vai além do monitoramento superficial, revelando desde métricas de desempenho até padrões operacionais otimizados, permitindo às equipes não apenas identificar problemas [4].

Além disso, a observabilidade fornece dados estratégicos sobre o comportamento do sistema em produção, permitindo aprimorar continuamente o ciclo de desenvolvimento. Esses *insights* operacionais em tempo real orientam decisões

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

técnicas mais embasadas, desde ajustes pontuais até evoluções arquiteturais mais profundas, garantindo que o sistema se adapte às necessidades reais de uso [4].

Por fim, a observabilidade proporciona ao cliente (ponta final do *software*) uma melhor experiência com o sistema em questão. Uma boa observabilidade tende a entregar performance superior e maior estabilidade, resultando em uma experiência mais fluida e satisfatória para os usuários finais. Essa excelência operacional não apenas eleva a satisfação do cliente, mas também impulsiona diretamente os resultados de negócio, fortalecendo a reputação da aplicação e aumentando sua taxa de adoção [4].

Na prática, a adoção da observabilidade não é binária (ter ou não ter), mas envolve escolhas de projeto sobre como coletar, processar e armazenar os dados gerados pelos três pilares. Entre essas escolhas, existem, por exemplo: a cardinalidade das métricas, o nível de retenção e detalhamento dessas métricas, e a taxa de amostragem aplicada a elas. Cada uma dessas diretrizes representa diferentes estratégias de observabilidade, cujas implicações em termos de custo de infraestrutura e eficiência diagnóstica serão discutidas neste trabalho (e detalhadas na Seção Metodologia).

2.1. Trabalhos relacionados

Os trabalhos relacionados discutidos nesta seção envolvem a análise da experimentação de diferentes práticas e ferramentas de observabilidade em diferentes tipos de infraestrutura de *hardware*.

Li *et al*. [6] propõem o TraceRCA, uma abordagem não supervisionada para localização de causas-raiz em sistemas de microsserviços por meio da análise de traces. O método utiliza três etapas principais: detecção de anomalias em traces, mineração de conjuntos suspeitos de microsserviços e ranqueamento baseado em escores de suspeição. Os autores demonstram que o TraceRCA supera abordagens anteriores em mais de quarenta e quatro porcento na detecção do serviço causador da falha, validando sua eficácia em sistemas reais. Embora o foco do artigo seja a acurácia na diagnose de falhas, ele destaca a importância da eficiência computacional ao utilizar técnicas leves

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

(como seleção adaptativa de métricas) para reduzir o espaço de busca. Essa otimização é relevante para o contexto deste trabalho, pois alinha-se ao objetivo de definir estratégias de extração de informação para a observabilidade de sistemas.

Madupati [7] explora o uso de observabilidade com microserviços utilizando uma abordagem empírica, aplicando as ferramentas Prometheus para coleta de métricas, o Grafana para visualização e o Jaeger para rastreamento distribuído em sistemas. O autor defende que essa cadeia de ferramentas melhora a detecção de falhas e o monitoramento, embora seu trabalho não apresente métricas quantitativas que dimensionem essa melhoria (como a redução no tempo médio para resolver incidentes). Por outro lado, o estudo é mais preciso ao quantificar um dos principais desafios: a instrumentação pode gerar uma sobrecarga de aproximadamente 25% no uso da CPU. Essa análise dos custos computacionais oferece uma referência valiosa para a presente investigação sobre o impacto da observabilidade no consumo de recursos.

Rodrigues *et al*. [10] fazem uma avaliação da efetividade do uso combinado de dois pilares da observabilidade, métricas e *logs*, para diagnosticar anomalias em um sistema 5G. A metodologia é experimental, baseada na implantação de um ambiente de testes que emula uma rede 5G completa em um cluster Kubernetes, além da condução de testes de conectividade e desempenho sob subdimensionamento de CPU, comparando a detecção de anomalias. Os autores demonstram como a análise combinada de métricas e *logs* pode melhorar a observabilidade de sistemas 5G e ajudar a antecipar a ocorrência de falhas. Já que, no teste de desempenho, os *logs* emitiram alertas anômalos repetidos antes que a métrica de throughput indicasse a queda, permitindo uma detecção antecipada em vários segundos. Diante disso, esse estudo se assemelha por também trabalhar com práticas semelhantes que focam em manipulação de métricas. Entretanto, Rodrigues *et al.* [10] abordam muito mais a facilidade na detecção de falhas com a aplicação de boas técnicas de observabilidade que na análise do uso de recursos de *hardware*.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

Gomes *et al.* [3] analisam o uso da observabilidade para avaliar o desempenho de arquiteturas monolíticas e baseadas em microsserviços, empregando a solução OpenTelemetry (OTel) por meio da migração de uma aplicação de referência baseada em microsserviços para um sistema monolítico. O resultado principal indica que o OTel pode impactar negativamente o desempenho da aplicação em ambientes com recursos computacionais limitados, em que houve uma redução de 0,44% no número de requisições atendidas e um aumento de 0,48% no tempo de resposta para cada solicitação. Este trabalho constrói um ambiente semelhante, uma vez que visa-se limitar propositalmente os recursos disponíveis para uma análise que simula ambientes restritos. Porém o foco não é em restringir o *hardware*, isso faz parte da metodologia. Este estudo foca na manipulação de diretrizes de observabilidade que visam melhorar a performance geral da aplicação.

Costa e Araújo [2] abordam a sobrecarga computacional causada por ferramentas de observabilidade em ambientes de Computação em Névoa, utilizando um cenário prático com o aplicativo Mobile IoT-RoadBot. O estudo avalia o impacto de ferramentas de código aberto, como Prometheus, ELK Stack e OpenTelemetry, no consumo de CPU, memória e armazenamento em dispositivos IoT e nós de Névoa. Os autores propõem estratégias para reduzir o volume de dados de observabilidade, alcançando uma diminuição de oitenta por cento no armazenamento sem comprometer a qualidade dos dados. Este trabalho relacionado é relevante para o presente estudo, pois quantifica o custo computacional de práticas de observabilidade em um ambiente com restrições de recursos, similar aos cenários abordados aqui. Contudo, os autores focam em otimizar o volume de dados em um contexto específico de Névoa, enquanto este estudo explora uma análise em ambiente local com recursos de *hardware* limitados.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

3. Metodologia

Este trabalho adota uma abordagem experimental quantitativa e aplicada, com o objetivo de investigar, de forma explicativa, o impacto de diferentes práticas de observabilidade no consumo de recursos de *hardware* através de métricas como o tempo de execução de um *endpoint* da API (API, do inglês, Application Programming Interface) construída para esse estudo e o consumo computacional das ferramentas. A metodologia foi planejada para traduzir informações em números que podem ser analisados e classificados [11] e garantir a obtenção de dados confiáveis, permitindo comparações entre configurações de monitoramento em sistemas de computador. Compreende-se como procedimentos, cinco etapas que serão detalhadas na subseção 3.2.

3.1. Instrumentos Utilizados

A execução da pesquisa utilizará um conjunto integrado de ferramentas de código aberto, selecionadas por sua adequação aos objetivos do estudo. Para a geração de carga de trabalho, a ferramenta Grafana k6 foi escolhida devido à sua arquitetura leve e capacidade de integração nativa com pipelines de métricas, permitindo não apenas simular tráfego realista mas também exportar dados de desempenho diretamente para o Prometheus. No k6 será possível analisar a quantidade de requisições feitas, a média de tempo de cada requisição e o p95 de tempo de cada uma.

Complementarmente, a interface do Prometheus será utilizada para analisar graficamente os usos de recursos de *hardware* (CPU e memória) no tempo de execução de cada estratégia de observabilidade. A combinação entre Prometheus e Grafana k6 forma o núcleo da solução de monitoramento, seguindo padrões consolidados do mercado para coleta, armazenamento e visualização de métricas quantitativas em tempo real.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

3.2. Procedimentos

A Figura 1 resume as principais etapas, desde a preparação da API que será estressada até a proposição de diretrizes de observabilidade para aproveitamento de recursos de máquina. A seguir, é detalhada cada uma delas.

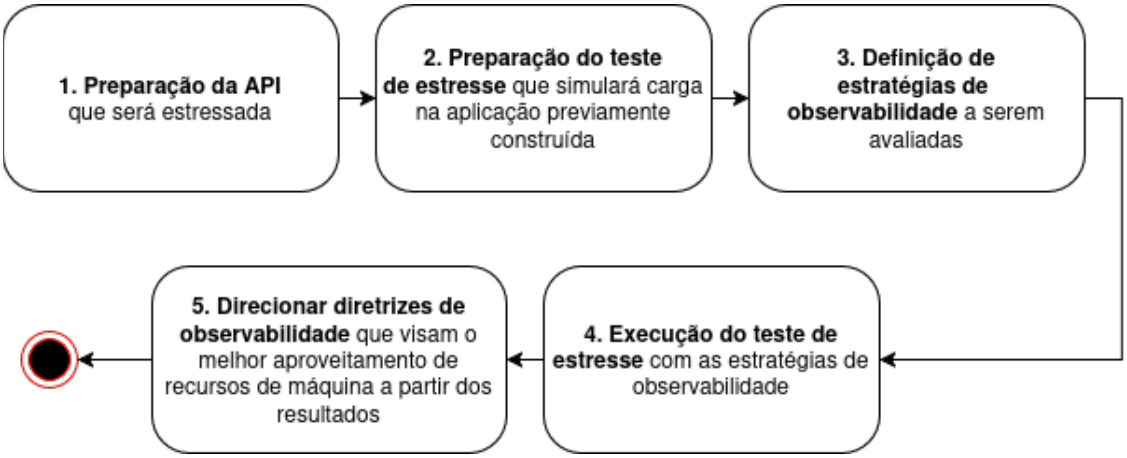

Figura 1: Metodologia de cinco etapas para o desenvolvimento deste projeto

Fonte: Autores (2026).

3.2.1. Preparação da API que será estressada

Inicialmente, será construída a API orquestrada por um ambiente com restrições de recurso pelo Docker. Ela será desenvolvida na linguagem de programação PHP utilizando o framework HyperF, com banco de dados local MySQL. Todos os serviços serão configurados através de um arquivo docker-compose.yml único, com limites de recursos por contêiner, redes bridge isoladas e volumes para persistência dos dados coletados. A API possuirá um *endpoint* POST que fará consultas ao banco de dados com duas tabelas relacionais pré montadas antes de fazer inserções em uma terceira tabela e obterá dados no formato JSON.

3.2.2. Preparação do teste de estresse

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

Nessa etapa, será implementado um teste de estresse para simular condições realistas de carga no sistema. Para isso, utilizaremos a ferramenta k6 para gerar requisições HTTP (HTTP, do inglês Hypertext Transfer Protocol) automatizadas, configuradas com parâmetros específicos. A configuração do teste de estresse é composta por:

- Usuários virtuais: variam entre 10 e 200 por execução.
- Execuções por estratégia: serão feitas 30 execuções do teste de estresse para cada estratégia de observabilidade.
- Duração por execução: cada execução terá o tempo de vida de 120 segundos.
- Perfil de carga: serão definidas cargas diferentes ao longo de cada período do teste de estresse com a finalidade de simular o volume de tráfego variável de um sistema real.

3.2.3. Definição das Estratégias de Observabilidade

Nesta etapa, serão avaliadas três abordagens conceituais de observabilidade, independentes de ferramentas específicas, na seguinte ordem:

- Redução da cardinalidade de métricas: Consiste na remoção seletiva de métricas que possuem alta cardinalidade, ou seja, aquelas cujos rótulos podem assumir muitos valores distintos (ex.: IDs de usuário, endereços IP).
- Amostragem com parâmetro fixo: Implementação de um mecanismo determinístico que filtra as métricas coletadas com base em uma taxa constante predefinida. Além disso, esta estratégia aumenta o intervalo de coleta do Prometheus, reduzindo a frequência de consulta de informações.
- Amostragem Adaptativa: Implementação de um sistema dinâmico que ajusta a granularidade do monitoramento conforme a carga do sistema

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

(ex.: uso de CPU, latência de requisições). Durante picos de demanda, a coleta de métricas é reduzida; em condições normais, é restaurada. Aqui também será mantido o aumento no intervalo de coleta das métricas pelo Prometheus.

Vale ressaltar que a cada execução do teste de estresse, o banco de dados estará previamente populado apenas com os dados de configuração inicial.

3.2.4. Execução do teste de estresse e coleta e análise de dados

Esta etapa consistirá na execução dos testes de carga para cada cenário de observabilidade. Serão registrados: tempo médio de resposta e p95 do tempo de resposta, utilização de CPU e memória do sistema. Os dados de CPU e memória serão armazenados e analisados no Prometheus enquanto as métricas relacionadas à quantidade e tempo das requisições serão analisados no k6, no output de finalização do teste de estresse.

3.2.5. Direcionar diretrizes de observabilidade

Com base na análise comparativa dos resultados, serão elaboradas recomendações práticas de observabilidade que visam aproveitar o consumo de recursos de máquina.

3.3. Ambiente Experimental

O estudo será conduzido em ambiente local isolado utilizando contêineres Docker em uma máquina hospedeira que executa qualquer distribuição Linux com Docker Compose para orquestração dos serviços. O ambiente conteinerizado usará uma imagem do Alpine e terá limitações de recursos pré-definidas pelo Docker. Será destinado à API 1 cpu e 128MB (MB, do inglês Mega Bytes) de memória. Enquanto o banco de dados da aplicação possuirá 1 cpu e 512MB de memória.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

O presente estudo utilizará recursos limitados para que seja possível aplicá-lo em vários tipos de máquinas que têm como sistema operacional Linux. Além de simular um ambiente com pouco recurso de *hardware* disponível com a finalidade de extrair o máximo possível de desempenho do ambiente isolado.

3.4. Métricas de avaliação

A seleção das métricas seguiu a abordagem GQM (Goal-Question-Metric) [1], que estabelece uma relação sistemática entre objetivos, questões de pesquisa e métricas de avaliação. Neste estudo, o objetivo era avaliar a eficiência de diferentes práticas de observabilidade no consumo de recursos de *hardware*. As questões de pesquisa investigaram: (i) Qual estratégia oferece melhor desempenho em tempo de resposta? (ii) Como cada abordagem impacta o consumo de CPU e memória? (iii) Quais são os *trade-offs* entre qualidade de monitoramento e custo computacional?

Serão coletadas e analisadas quatro métricas principais. Duas delas, o tempo de resposta médio e latência do percentil da API, medido em milissegundos, servirão como indicadores diretos do impacto na experiência do usuário final e performance percebida. Ademais, essas métricas foram escolhidas por serem entregues pela ferramenta de teste de estresse Grafana K6, usada neste trabalho. Já os dados de memória e CPU foram coletados por meio do pacote de observabilidade hyperf/metric do framework HyperF utilizado para a construção da API. O consumo de recursos de *hardware* será quantificado através do uso de CPU e memória, monitorado pelo Prometheus, que captura o *overhead* computacional introduzido por cada abordagem de coleta de dados.

4. Desenvolvimento e discussão

4.1. Resultados Quantitativos

Esta seção apresenta os resultados obtidos a partir da anáalise dos quatro cenários de configuração: Sem Diretrizes, Redução da Cardinalidade, Amostragem

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

Simples e Amostragem Adaptativa. Para visualizar e comparar a distribuição das métricas de desempenho entre os cenários, optou-se pela utilização de gráficos box plot. O box plot é uma ferramenta robusta para detecção de *outliers* baseada em quartis e na amplitude interquartil (IQR). Por utilizar medidas resistentes de tendência central (mediana) e dispersão (IQR), é menos sensível a valores extremos, identificando eficazmente observações atípicas [8].

A Figura 2 apresenta a distribuição das médias dos tempos de resposta das requisições HTTP entre os quatro cenários. Os cenários de Amostragem Adaptativa é Amostragem Simples apresentaram medianas de 8,645 ms (milissegundos) e 9,045 ms, respectivamente, enquanto Redução da Cardinalidade e Sem Diretrizes obtiveram médias nas de 14,55 ms e 14,65 ms. Em relação a dispersão, o cenário Sem Diretrizes apresentou menor variabilidade, com intervalo interquartil de 0,59 ms (Q1=14,37; Q3=14,96), seguido pela Redução da Cardinalidade com 0,8 ms (Q1 = 14,11; Q3 = 14,91). Os cenários de amostragem mostraram maior dispersão, com intervalos interquartis de 0,59 ms para Amostragem Simples (Q1 = 8,63; Q3 = 9,22) e 1,59 ms para Amostragem Adaptativa (Q1=7,86; Q3=9,45).

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

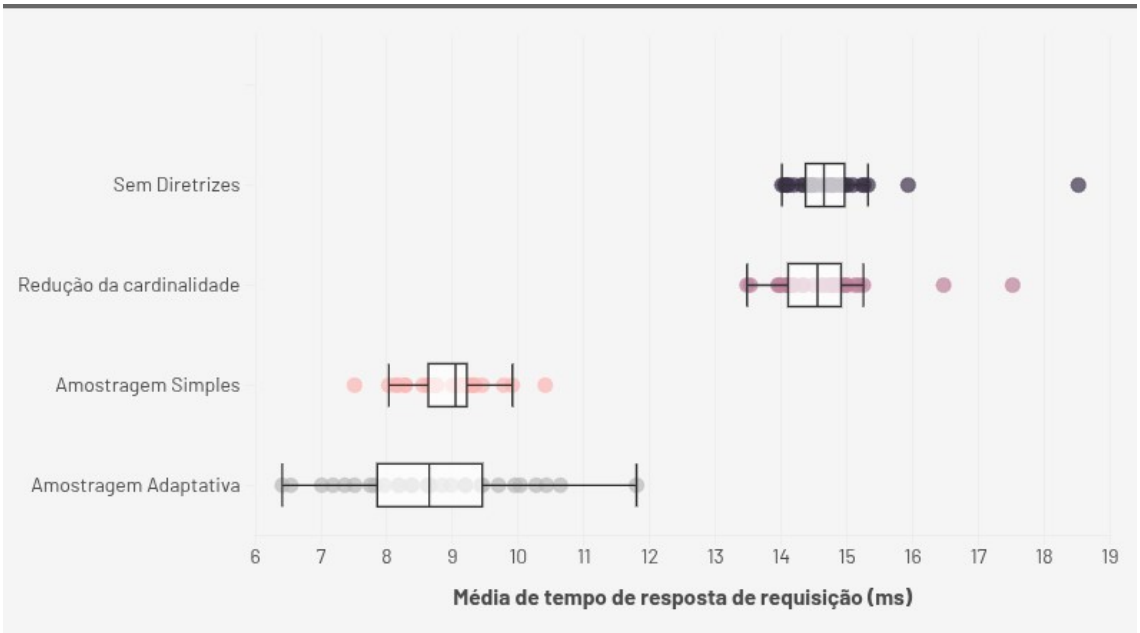

Figura 2: Médias nos tempos de resposta das requisições HTTP

Fonte: Autores (2026).

A Figura 3 mostra a distribuição do percentil 95 (P95) dos tempos de resposta. O cenário Redução de cardinalidade apresentou a menor mediana (17,29 ms), seguido pelo Sem Diretrizes (17,56 ms), Amostragem Simples (24,59 ms) e Amostragem Adaptativa (23,59 ms). A dispersão foi mais acentuada no cenário de Amostragem Simples, com intervalo interquartil de 10,78 ms (Q1 = 19,59; Q3 = 30,37). Observa-se a presença de *outliers* em todos os grupos com exceção da Amostragem Simples, com maior concentração no cenário de Redução de Cardinalidade.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

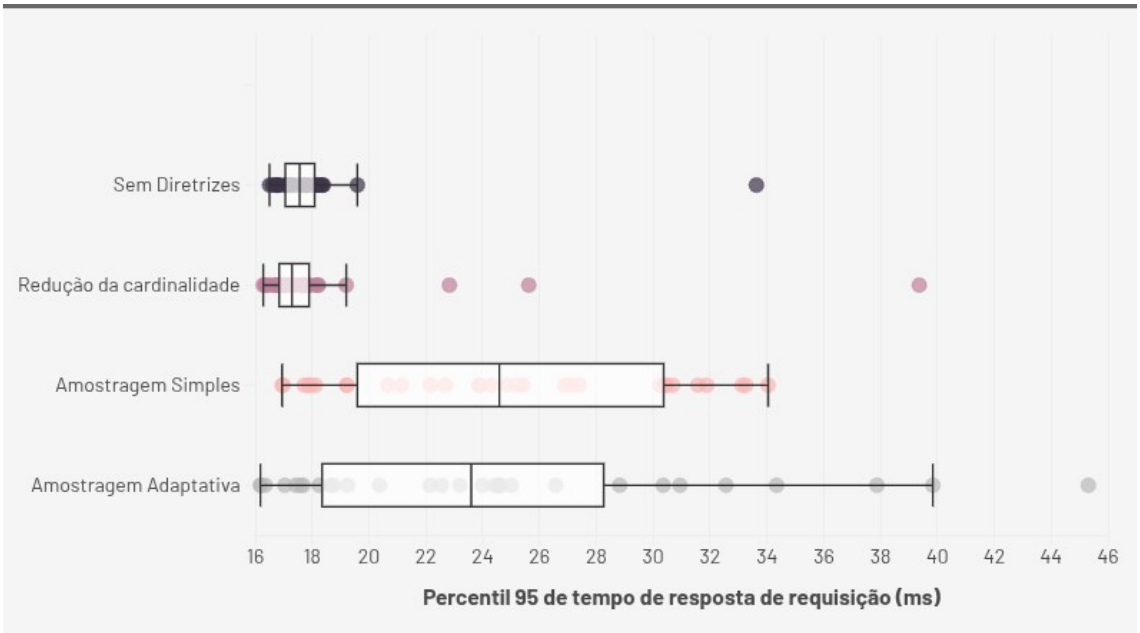

Figura 3: Percentil 95 dos tempos de resposta das requisições HTTP

Fonte: Autores (2026).

A Figura 4 apresenta a distribuição do uso de memória entre os cenários. As medianas variaram de 6,10 MB (Amostragem Adaptativa) a 6,14 MB (Sem Diretrizes). Os cenários de amostragem mostraram maior variabilidade, com intervalos interquartis de 0,04 MB para Amostragem Adaptativa (Q1=6,06; Q3=6,10), 0,04 MB para Amostragem Simples (Q1 = 6,06; Q3 = 6,10) e 0,04 MB para Redução de Cardinalidade (Q1 = 6,13; Q3 = 6,17). Já o cenário Sem Diretrizes apresentou menor dispersão 0,3 MB (Q1=6,13; Q3=6,16) mas, com mais *outiliers* (4).

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

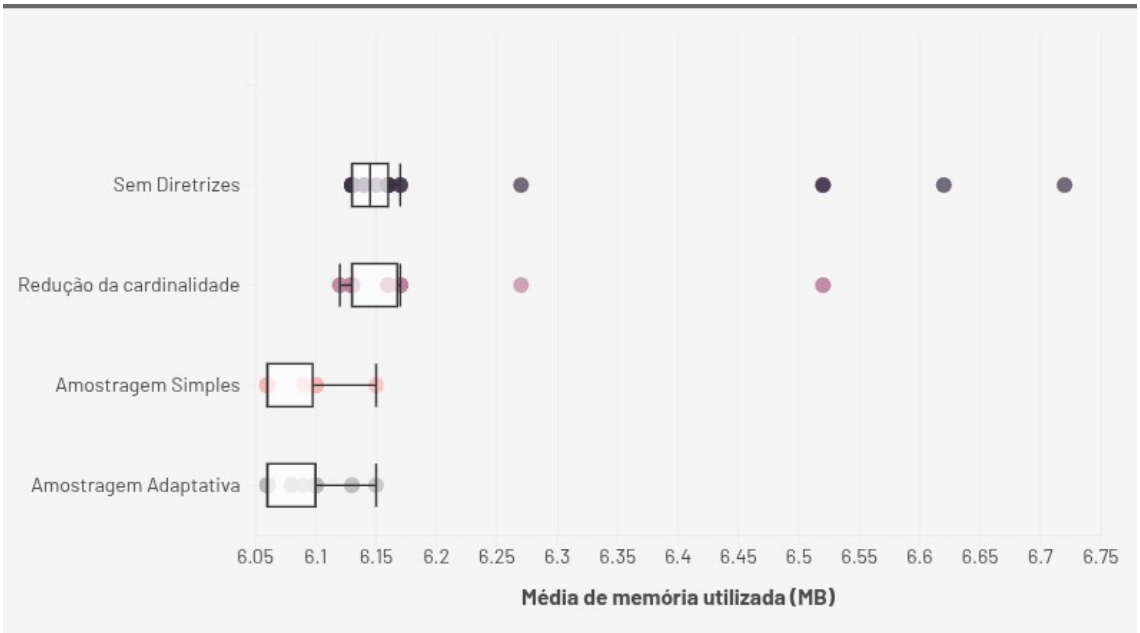

Figura 4: Médias de memória utilizada pelos diferentes cenários

Fonte: Autores (2026).

A Figura 5 exibe a distribuição do uso de CPU medido em microssegundos por segundo (µs/s). As medianas observadas foram de 44.744 µs/s (Sem Diretrizes), 47.234 µs/s (Redução da Cardinalidade), 25.638 µs/s (Amostragem Simples) e 27.805 µs/s (Amostragem Adaptativa). Quanto à dispersão, o cenário Amostragem Simples apresentou o maior intervalo interquartil (7.814,75 µs/s), enquanto a Amostragem Adaptativa mostrou a menor variabilidade (6.595 µs/s). Nota-se a presença de valores *outliers* apenas no cenário de Amostragem Simples.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

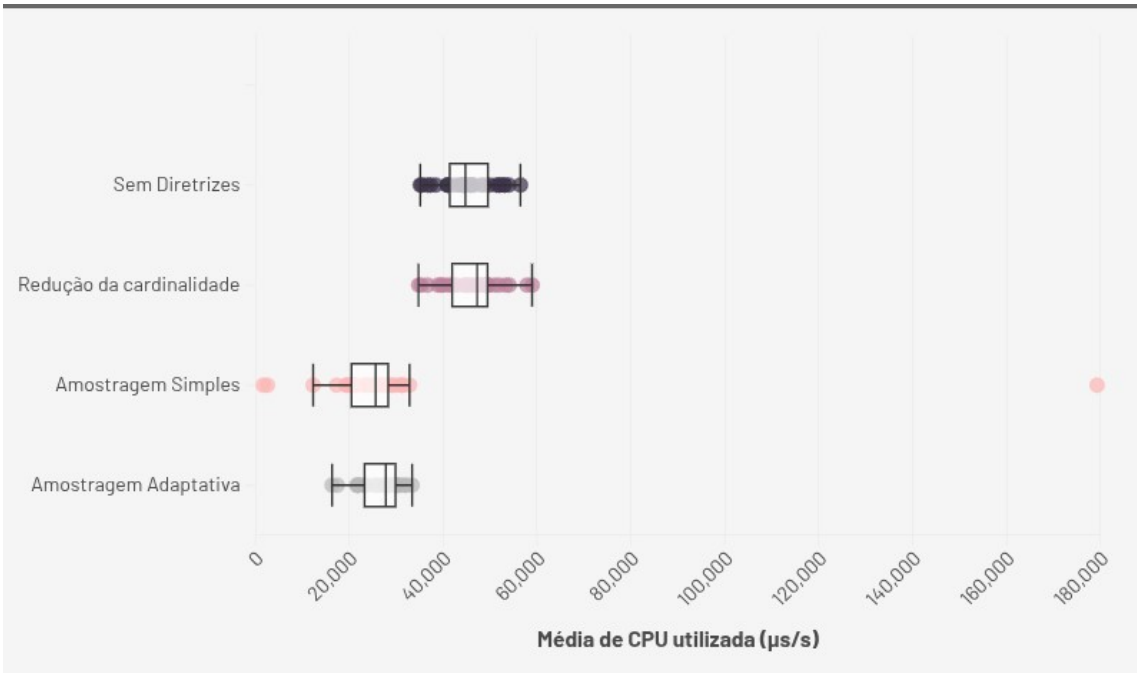

Figura 5: Médias de CPU utilizada pelos diferentes cenários

Fonte: Autores (2026).

4.2. Discussão dos resultados

Os resultados quantitativos apresentados na seção anterior revelam um padrão claro de melhoria progressiva a medida que estratégias de otimização de observabilidades são acumuladas. Como primeiro achado, identificou-se que a combinação de múltiplas técnicas produz ganhos superiores aos alcançados por abordagens isoladas, oferecendo *insights* valiosos para a implementação de observabilidade em ambientes com recursos limitados.

Para facilitar a comparação direta entre os cenários, a Tabela 1 consolida os valores medianos das principais métricas de desempenho. Esta visão sintetizada permite identificar rapidamente os *trade-offs* e benefícios de cada abordagem.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

Tabela 1: Resumo Comparativo das Métricas de Desempenho por Cenário

Métrica	Sem Diretrizes	Red. Card.	Amost. Simples	Amost. Adapt.
Tempo de Resposta Médio (ms)	14,65	14,55	9,045	8,645
Percentil 95 do Tempo de Resposta (ms)	17,56	17,29	24,59	23,59
Uso de Memória (MB)	6,14	6,14	6,10	6,10
Uso de CPU (μs/s)	44.744	47.234	25.638	27.805

Fonte: Autores (2026).

4.3. Impacto nas métricas de desempenho e latência

A evolução desde o cenário Sem Diretrizes até a Amostragem Adaptativa mostra uma melhoria consistente e substancial no desempenho do tempo de resposta, conforme ilustrado pela Figura 2. O tempo médio, que começou em 14,65 ms no cenário base, foi reduzido para aproximadamente 8-9 ms nas estratégias mais avançadas – representando uma melhoria de cerca de 45%. Esse ganho progressivo indica que cada camada de otimização contribui para a eficiência geral do sistema, com a combinação de redução de cardinalidade, amostragem básica e ajuste dinâmico demonstrando ser a abordagem mais eficaz para a métrica de tempo médio.

Entretanto, a análise da latência do percentil 95 (P95), apresentada na Figura 3, revela um trade-off importante: Enquanto a amostragem beneficia o usuário médio, ela pode introduzir uma variabilidade maior que impacta negativamente uma pequena fração das requisições mais lentas. A Redução de Cardinalidade isolada obteve a melhor mediana de P95 (17,29 ms), sugerindo que esta técnica é eficaz para aplicações que exigem baixa latência consistente. No entanto, a medida que as estratégias de amostragem são incorporadas, observa-se um aumento na variabilidade e nas medianas de P95 (atingindo aproximadamente 23-24 ms). A escolha da estratégia deve, portanto,

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

considerar se o objetivo é otimizar a experiência média ou garantir a consistência da latência para todos os usuários.

A partir dos resultados acima, a pergunta levantada pelo objetivo i) pode ser respondida da seguinte forma: A combinação de estratégias até a amostragem adaptativa é o cenário em que se oferece melhor desempenho em tempo de resposta.

4.4. Eficiência no uso de recursos de sistema

O consumo de CPU, mostrado na Figura 5, apresenta um padrão. Enquanto a aplicação isolada da Redução de Cardinalidade mostrou um consumo ligeiramente superior ao cenário sem otimizações (47.234 µs/s vs 44.744 µs/s), a introdução da Amostragem Simples sobre esta base reduziu drasticamente o uso para 25.638 µs/s – uma economia de aproximadamente 43%. A evolução para a Amostragem Adaptativa manteve ganhos significativos (27.805 µs/s), demonstrando que a combinação estratégica de técnicas é essencial para otimizar recursos computacionais.

Esse resultado é crucial, pois sugere que a Redução de Cardinalidade, quando aplicada isoladamente, pode introduzir *overheads* de processamento que superam seus benefícios. No entanto, quando combinada com estratégias de amostragem, ela contribui para um sistema mais eficiente no uso do processador. Esta descoberta corrobora observações de Madupati [7], em que o autor evidencia o aumento de um quarto na sobrecarga da CPU em cenários de coleta detalhada de informações.

Em contraste, o consumo de memória, ilustrado na Figura 4, manteve-se notavelmente estável em todos os cenários — com variações inferiores a 0,1 MB. Este comportamento reflete o padrão de operações da aplicação testada, caracterizado por operações de banco de dados de baixo impacto em memória. O *endpoint* da API implementa uma consulta SELECT que retorna um conjunto de dados pequeno e uma operação INSERT com um payload mínimo, não exercitando operações que tipicamente carregam grandes volumes de memória.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

A análise acima responde à segunda questão de pesquisa: 'Como cada abordagem impacta o consumo de CPU e memória?', além de referenciar o objetivo específico ii).

4.5. Implicações práticas para ambientes com recursos de *hardware* restritos

Para contextos práticos de ambientes restritos, como edge computing e dispositivos IoT, similares aos investigados por Breno e Aletéia [2], a abordagem cumulativa demonstra especial valor. A progressão desde técnicas básicas até a Amostragem Adaptativa mostra que é possível alcançar ganhos de eficiência substanciais — como a redução de 45% no tempo de resposta e 43% no uso de CPU — sem comprometer completamente a capacidade de monitoramento. A capacidade de ajustar dinamicamente o volume de dados de telemetria (como na Amostragem Adaptativa) é particularmente vantajosa nesses ambientes, onde as condições da rede e a disponibilidade de recursos podem flutuar.

4.6. Considerações sobre complexidade, manutenção e *trade-offs*

Vale ressaltar que a implementação cumulativa dessas estratégias de observabilidade introduz complexidade adicional. Cada camada exige configuração específica, como definir taxas de amostragem adequadas e limiares para dinâmicas adaptativas, além de manutenção contínua para calibrar esses parâmetros. A Amostragem Adaptativa, em particular, requer uma lógica temporal mais sofisticada, utilizando múltiplas taxas baseadas em intervalos predefinidos. No entanto, os ganhos de performance obtidos justificam essa complexidade incremental em ambientes onde a eficiência de recursos é crítica. O desafio se desloca, portanto, da simples coleta de dados para o projeto inteligente de quando e o que coletar, otimizando o custo-benefício da observabilidade.

Em resposta à terceira questão de pesquisa (Quais são os *trade-offs* entre qualidade de monitoramento e custo computacional?), os resultados demonstram um

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

trade-off entre a profundidade do monitoramento e a eficiência computacional. A coleta completa de dados, sem otimizações, oferece máxima visibilidade do sistema para *debugging*, porém consome cerca de 45% a mais de CPU e aumenta o tempo médio de resposta na mesma proporção, representando um *overhead* significativo.

Em contrapartida, estratégias de otimização como a amostragem e a redução de cardinalidade reduzem drasticamente o custo computacional, mas introduzem limitações na qualidade do monitoramento. A amostragem pode omitir eventos raros críticos, enquanto a redução de cardinalidade simplifica excessivamente as métricas, comprometendo a detecção de anomalias específicas. A escolha ideal depende, portanto, do balanceamento entre a necessidade de detalhamento para análise e a disponibilidade de recursos no ambiente-alvo.

4.7. Síntese das respostas às questões de pesquisa

Tabela 2: Vereditos finais das perguntas de pesquisa

Pergunta	**Veredito**
(i) Qual estratégia oferece melhor desempenho em tempo de resposta?	Aglutinação de estratégias até a Amostragem Adaptativa (menor tempo médio e P95)
(ii) Como cada abordagem impacta o consumo de CPU e memória?	De maneira significativa, práticas de amostragem podem reduzir o uso de CPU em até 43%. Já para memória, todas as estratégias apresentaram um consumo estável
(iii) Quais são os *trade-offs* entre qualidade de monitoramento e custo computacional?	Maior otimização = menor custo computacional, porém menor visibilidade (eventos raros podem ser perdidos; métricas simplificadas)

Fonte: Autores (2026).

4.8. Ameaças à validade

Estudos experimentais em Engenharia de *software* estão propícios a diferentes ameaças à validade, que necessitam ser postas para deixar as conclusões confiáveis.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

Esta seção descreve as principais ameaças à validade identificadas na pesquisa, bem como as estratégias empregadas para mitigá-las.

4.8.1. Validade interna

No que se refere à validade interna, que avalia se os resultados podem ser efetivamente atribuídos às variáveis manipuladas, uma ameaça considerada foi a possível interferência de fatores diversos nos resultados observados, como variações inerentes ao ambiente virtualizado e alocação incontrolada de recursos. Para mitigar este risco, o experimento foi conduzido em um ambiente controlado e reproduzível, utilizando contêineres Docker com limites estritos de CPU e memória. Adicionalmente, o banco de dados era limpo antes de cada execução de teste, e os dados utilizados no corpo das requisições eram gerados aleatoriamente, assegurando que cada teste partisse de uma condição igualitária e evitando viés proveniente de dados ou padrões de carga previsíveis.

4.8.2. Validade externa

"A validade externa diz respeito à generalização dos resultados para outros contextos. Neste estudo, a principal limitação reside no foco em uma API backend típica de ambientes web, executada em uma distribuição Linux. Consequentemente, os resultados podem não ser diretamente generalizáveis para sistemas com características radicalmente distintas, como aplicações de IoT e outros sistemas operacionais. No entanto, a metodologia adotada, que incluiu a simulação de um ambiente com recursos limitados e a aplicação de cargas de trabalho variáveis e aleatórias, buscou criar um cenário representativo de restrições comuns em ambientes de produção de baixo recurso, aumentando a potencial aplicabilidade dos resultados em contextos análogos.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

4.8.3. Validade de construção

Quanto à validade de construção, que verifica se as métricas e ferramentas usadas realmente medem o que se propuseram a medir, uma preocupação foi a forma como colocamos em prática as diretrizes de observabilidade. Conceitos como “Amostragem Adaptativa” foram implementados usando regras específicas criadas para este trabalho, e outras formas de implementar essas mesmas ideias poderiam levar a resultados diferentes. Para aumentar a confiança nas medições, utilizamos ferramentas consolidadas no mercado (Grafana k6, Prometheus e a biblioteca hyperf/metric), garantindo que a coleta de dados sobre consumo de recursos e desempenho fosse feita de maneira padronizada e automática em todos os cenários testados. Assim, assegura-se que quaisquer diferenças observadas nos resultados pudessem ser atribuídas às estratégias de observabilidade em teste, e não a variações na forma de medição.

4.8.4. Validade de conclusão

A validade de conclusão refere-se à robustez das inferências estatísticas extraídas dos dados. Reconhece-se que usar apenas gráficos de *box plot*, embora eficaz para identificar tendências centrais e valores atípicos, apresenta uma limitação importante: esse método não comprova se as diferenças observadas entre os cenários são estatisticamente significativas ou apenas fruto de variações aleatórias. Para um estudo futuro, seria recomendável complementar essa análise visual com testes estatísticos que poderiam quantificar o nível de confiança nas diferenças encontradas.

5. Considerações finais

Este trabalho avaliou o impacto de quatro cenários de observabilidade de forma cumulativa (Sem Diretrizes, Redução da Cardinalidade, Amostragem Simples e Amostragem Adaptativa) em uma API com *hardware* controlado. A aplicação com cada diretriz foi analisada em cima de quatro métricas: Média de tempo de resposta de

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

requisições HTTP (ms), percentil 95 dos tempos de resposta das requisições HTTP (ms), Média de memória utilizada (MB) e Média de CPU utilizada (µs/s).

Ao avaliar e comparar as métricas, os resultados demonstraram que a aplicação cumulativa de estratégias de observabilidade produz ganhos de desempenho superiores à ausência de práticas otimizadas. Especificamente, o cenário de Amostragem Adaptativa alcançou a maior redução no tempo médio de resposta (aproximadamente 45%) e no consumo de CPU (cerca de 43%) em comparação com o cenário base Sem Diretrizes. Esses ganhos substanciais indicam que a combinação de técnicas é altamente eficaz para otimizar a eficiência operacional em ambientes com recursos limitados.

Além disso, também quanto ao uso de recursos de *hardware*, o consumo de memória manteve-se notavelmente estável em todos os cenários, com variações inferiores a 0,1 MB, refletindo o perfil de baixo impacto das operações da API testada. Em contraste, o consumo de CPU mostrou-se sensível às estratégias adotadas, com a Redução de Cardinalidade isolada introduzindo um pequeno *overhead* adicional que foi drasticamente mitigado pela introdução das amostragens.

Os resultados obtidos ressaltam a importância de um projeto inteligente de observabilidade, no qual o desafio se desloca da simples coleta de dados para a decisão estratégica de quando e o que coletar. A complexidade incremental introduzida pelas estratégias cumulativas — especialmente pela Amostragem Adaptativa — mostra-se justificável pelos ganhos de eficiência alcançados, mas exige configuração e manutenção contínuas para calibrar parâmetros como taxas de amostragem e limiares adaptativos. Todos os *insights* obtidos e discutidos propõem cenários diversos para a aplicação das diretrizes de observabilidade vistas, referenciando o terceiro objetivo específico.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

6. Indicação de trabalhos futuros

Para trabalhos futuros, recomenda-se a investigação de mais técnicas de amostragem, como a amostragem baseada em cauda (*tail-based sampling*), na qual as decisões de amostragem são tomadas após o trace ter sido concluído, permitindo decisões mais fundamentadas com base no contexto completo do trace. Adicionalmente, sugerem-se estudos em ambientes com *hardware* mais restrito e com maiores cargas de trabalho. Por fim, a exploração de técnicas de machine learning para automação da calibração dos parâmetros de observabilidade representa um caminho promissor para reduzir a complexidade operacional de manutenção dos mesmos.

7. Declaração de direitos

Os autores declaram ser detentores dos direitos autorais da presente obra, que o artigo não foi publicado anteriormente e que não está sendo considerado por outra(o) Revista/Journal. Declaram que as imagens e textos publicados são de responsabilidade dos autores, e não possuem direitos autorais reservados a terceiros. Textos e/ou imagens de terceiros são devidamente citados ou devidamente autorizados com concessão de direitos para publicação quando necessário. Declaram respeitar os direitos de terceiros e de Instituições públicas e privadas. Declaram não cometer plágio ou autoplágio e não ter considerado/gerado conteúdos falsos e que a obra é original e de responsabilidade dos autores.

8. Referências

1. BASILI, V. R. Software modeling and measurement: the goal/question/metric paradigm. 1992.
2. COSTA, B.; ARAUJO, A. Análise do overhead da observabilidade na computação em névoa. *In*: Anais da VII Escola Regional de Alto Desempenho do Centro-Oeste. Porto Alegre: SBC, 2024. p. 11-15. Disponível em: https://www.elastic.co/elasticsearch.
3. GOMES, F. *et al*. Observabilidade de desempenho de arquiteturas monolíticas e microsserviços com OpenTelemetry. In: Anais do LI Seminário Integrado de Software e Hardware (SEMISH). Brasília: SBC, 2024a. p. 121-132.

Scientific Society Journal
ISSN: 2595-8402
DOI: https://doi.org/10.61411/rsc31879

4. GOMES, F.; REGO, P.; TRINTA, F. Rumo a uma taxonomia de observabilidade para aplicações baseadas em microsserviços. *In*: Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software. Porto Alegre: SBC, 2024b. p. 234-245.
5. KALMAN, R. On the general theory of control systems. IRE Transactions on Automatic Control, v. 4, n. 3, p. 110-110, 1959.
6. LI, Z. *et al*. Practical root cause localization for microservice systems via trace analysis. *In*: 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS). [S.l.]: IEEE/ACM, 2021. p. 1-10.
7. MADUPATI, B. Observability in microservices architectures: Leveraging logging, metrics, and distributed tracing in large-scale systems. Zenodo, 2023.
8. MAZAREI, A. *et al*. Online boxplot derived outlier detection. International Journal of Data Science and Analytics, v. 19, n. 1, p. 83-97, 2025.
9. PICORETI, R. *et al*. Multilevel observability in cloud orchestration. *In*: 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech). [S.l.]: IEEE, 2018. p. 776-784.
10. RODRIGUES, K. B. C. *et al*. Uma investigação empírica sobre observabilidade em sistemas 5G nativos de nuvem. *In*: Anais do Simpósio Brasileiro de Software (SBSOFT). Goiânia: SBC, 2023. p. 1.
11. SANTANA, A. C. d. A.; NARCISO, R.; FERNANDES, A. B. Explorando as metodologias científicas: tipos de pesquisa, abordagens e aplicações práticas. Caderno Pedagógico, v. 22, n. 1, e13333, 2025.
12. USMAN, M. *et al*. A survey on observability of distributed edge & container-based microservices. IEEE Access, v. 10, p. 86904-86919, 2022.