Custo Experimental e Valor Diagnóstico em Estratégias de Teste de Desempenho para Aplicações Web

Eric Santos de Moraes
LESSE/UNIPAMPA
Alegrete, RS, Brazil
ericmoraes.aluno@unipampa.edu.br

Luan Evander Pimentel Tavares
LESSE/UNIPAMPA
Alegrete, RS, Brazil
luantavares.aluno@unipampa.edu.br

Anielle Andrade
LESSE/PPGES/UNIPAMPA
Alegrete, RS, Brazil
anielleandrade@unipampa.edu.br

Elder de Macedo Rodrigues
LESSE/PPGES/UNIPAMPA
Alegrete, RS, Brazil
elderrodrigues@unipampa.edu.br

Resumo

Garantir o desempenho de aplicações web sob carga é uma preocupação central da engenharia de software. Diferentes estratégias de teste, ao combinarem perfis de *workload* e níveis de monitoramento, podem produzir custos experimentais e capacidades diagnósticas distintas, porém essa relação ainda é pouco investigada empiricamente. Este trabalho apresenta uma avaliação comparativa de seis estratégias de teste de desempenho aplicadas a uma aplicação de comércio eletrônico baseada no TPC-W, utilizando a ferramenta k6. As estratégias combinam três perfis de *workload* (*browsing*, *shopping* e *ordering*) com dois níveis de monitoramento (básico e estendido), sendo avaliadas por latência (p50, p95, p99), *throughput*, taxa de erro e gargalos identificados. Os resultados de 60 execuções revelam diferenças substanciais entre os perfis avaliados. A análise inferencial identificou efeito significativo do perfil de *workload* sobre o *throughput* e forte interação entre *workload* e nível de monitoramento. Enquanto estratégias baseadas apenas em leitura indicaram um sistema saudável, os perfis transacionais expuseram um gargalo crítico de concorrência na camada de escrita. O monitoramento estendido apresentou efeito pequeno no perfil *browsing*, mas esteve associado a reduções de aproximadamente 50% no *throughput* dos perfis transacionais, ao mesmo tempo em que forneceu evidências adicionais para identificação da causa raiz. A combinação de monitoramento básico com um perfil de leitura e um perfil transacional detectou o principal gargalo com 90 minutos de preparação, oferecendo o melhor compromisso entre custo experimental e informação diagnóstica no cenário avaliado.

Palavras-chave

Teste de Desempenho, Workload, Monitoramento, Estratégias de Teste, Engenharia de Software Empírica, TPC-W, k6

1 Introdução

Aplicações web modernas são submetidas a múltiplos usuários simultâneos, e o desempenho sob carga afeta diretamente a experiência do usuário final e a confiabilidade do serviço [10]. Problemas de desempenho frequentemente só se manifestam quando o sistema é submetido a padrões de uso realistas, e suas causas podem residir em camadas distintas da arquitetura: consultas ineficientes ao banco de dados, contenção de recursos no servidor de aplicação ou saturação de rede [9, 10]. Apesar dessa relevância, estudos sobre teste de software tendem a concentrar-se na correção funcional das interfaces, deixando em segundo plano a avaliação sistemática do comportamento sob carga [4].

Identificar essas causas não é trivial e depende, fundamentalmente, da estratégia de teste adotada. Uma estratégia de teste de desempenho compreende um conjunto de decisões interdependentes: o perfil de *workload* simulado, que define quais operações são executadas e em que proporção; as métricas coletadas durante a execução, que determinam o que pode ser observado; e a profundidade do monitoramento, que estabelece o nível de detalhe disponível para diagnóstico [10]. Estratégias mais simples, como a aplicação de carga uniforme com coleta apenas de métricas agregadas, são rápidas de implementar, mas podem falhar em revelar gargalos que só se manifestam sob padrões de uso específicos. Estratégias mais elaboradas, que combinam *workloads* realistas com monitoramento de infraestrutura, tendem a produzir diagnósticos mais ricos, porém exigem maior investimento na preparação do ambiente, na modelagem do cenário e na análise dos resultados [9].

Essa relação entre o esforço de preparação e a qualidade do diagnóstico é central para a prática de engenharia de software, mas permanece pouco investigada empiricamente. A literatura recente sobre os testes de APIs avançou em técnicas de geração automatizada, incluindo *fuzzing* guiado por algoritmos genéticos [7], aprendizagem por reforço para exploração adaptativa [6] e avaliação de ferramentas em contexto industrial [2], demonstrando que a forma como o espaço de teste é explorado influencia diretamente os resultados obtidos. Contudo, esses trabalhos concentram-se na correção funcional e nenhum deles compara sistematicamente estratégias completas de teste de *desempenho*, isto é, combinações de perfil de *workload* e granularidade de monitoramento, nem quantifica o *trade-off* entre custo de preparação e capacidade de detecção de gargalos.

Neste contexto, o presente trabalho conduz um *quasi*-experimento para responder à seguinte questão de pesquisa (QP) (Figura 1):

QP. *Quais estratégias de teste de desempenho oferecem melhor relação entre custo experimental e valor diagnóstico?*

Figura 1: Questões de Pesquisa (QP).

Para isso, seis estratégias são definidas a partir de um desenho fatorial 3×2, combinando três perfis de *workload* com dois níveis de monitoramento, e aplicadas a uma aplicação web de comércio eletrônico baseada na especificação TPC-W [8], utilizando a ferramenta k6 [5]. Cada configuração é executada dez vezes, totalizando 60 execuções, e os resultados são analisados por meio de estatística descritiva, testes inferenciais e medidas de tamanho de efeito.

A contribuição deste trabalho é dupla. Para pesquisadores, o estudo oferece evidências empíricas, apoiadas por análise inferencial e medidas de tamanho de efeito, sobre como o perfil de *workload*, o nível de monitoramento e a interação entre esses fatores influenciam o comportamento observado e a capacidade de diagnosticar problemas de desempenho em aplicações web. Para profissionais, os resultados fundamentam uma estratégia incremental de teste, na qual diferentes perfis de carga são inicialmente exercitados com monitoramento básico e a instrumentação estendida é aplicada seletivamente para aprofundar a investigação de gargalos previamente identificados.

O estudo está organizado da seguinte forma: a Seção 2 apresenta os conceitos de teste de desempenho, a ferramenta k6 e a aplicação-alvo TPC-W; a Seção 3 descreve o desenho do *quasi*-experimento e as seis estratégias avaliadas; a Seção 4 apresenta os resultados das 60 execuções; a Seção 5 discute o efeito do perfil de *workload*, do monitoramento e a relação custo-benefício; e a Seção 6 conclui o trabalho e aponta trabalhos futuros.

2 Referencial Teórico

2.1 Teste de Desempenho

O teste de desempenho é uma categoria de teste não funcional cujo objetivo é avaliar como um sistema se comporta sob diferentes condições ou cargas de trabalho, verificando se requisitos de tempo de resposta, capacidade de processamento e utilização de recursos são satisfeitos em cenários que se aproximam do uso real [9, 10]. Diferentemente do teste funcional, que verifica se as saídas são corretas para entradas dadas, o teste de desempenho investiga a qualidade do comportamento: quão rápido, estável e escalável o sistema é quando submetido a múltiplas requisições simultâneas.

A disciplina compreende subtipos com objetivos distintos, cada um voltado a uma faceta do comportamento do sistema sob pressão [10]. O **teste de carga** (*load testing*) submete o sistema à carga esperada em produção para verificar se os tempos de resposta e a taxa de erro permanecem dentro dos limites aceitáveis. O **teste de estresse** (*stress testing*) aumenta progressivamente a carga além da capacidade projetada para identificar o ponto de ruptura, isto é, a condição a partir da qual o desempenho degrada de forma abrupta ou o sistema deixa de responder. O **teste de resistência** (*soak testing*) mantém uma carga constante por períodos prolongados para detectar problemas que só se manifestam ao longo do tempo, como vazamentos de memória, acúmulo de conexões e degradação gradual do throughput. O **teste de pico** (*spike testing*) avalia a capacidade do sistema de absorver e se recuperar de aumentos súbitos e temporários de carga, simulando eventos como campanhas promocionais. O presente trabalho adota o **teste de carga** como modalidade principal, avaliando o comportamento da aplicação sob carga constante de 50 usuários virtuais simultâneos [10].

A eficácia de um teste de desempenho depende de três fatores interdependentes [10]. O primeiro é o *perfil de workload*: define quais operações são executadas, em que proporção e com que padrão de chegada; *workloads* uniformes são simples de modelar, mas podem não representar o comportamento real dos usuários, enquanto *workloads* derivados de padrões de uso observados tendem a revelar gargalos que cenários artificiais não alcançam. O segundo é a *granularidade das métricas coletadas*: métricas agregadas fornecem uma visão geral rápida, enquanto métricas por interação e por percentil (p50, p95, p99) permitem identificar variações que afetam subconjuntos específicos de requisições. O terceiro é a *profundidade do monitoramento*: determina se o diagnóstico se limita às métricas do gerador de carga ou inclui dados de infraestrutura CPU, memória, I/O e logs de consultas lentas que permitem correlacionar sintomas com causas.

2.2 Grafana k6

O k6 é uma ferramenta de código aberto para teste de desempenho, mantida pela Grafana Labs e escrita em Go [5]. Os cenários de teste são definidos em JavaScript (ES6+), executados por um motor embarcado (Goja) que dispensa a instalação de Node.js ou de qualquer runtime externo. Cada usuário virtual (VU) é implementado como uma *goroutine* leve, o que permite simular milhares de VUs concorrentes com consumo de recursos significativamente inferior ao de ferramentas baseadas em *threads* do sistema operacional, como Apache JMeter [1]. Essa diferença arquitetural é relevante porque, em ferramentas baseadas em *threads*, o gerador de carga pode se tornar um gargalo antes da aplicação-alvo, comprometendo a validade das medições.

A ferramenta oferece funcionalidades relevantes para experimentação. O k6 coleta automaticamente latência por requisição, *throughput*, taxa de erro e volume de dados trafegados, além de suportar métricas customizadas (tipos Trend, Rate, Counter e Gauge). O conceito de *thresholds* permite definir critérios de aprovação diretamente no *script*, automatizando a verificação de objetivos de nível de serviço (SLOs). O suporte a *stages* possibilita a modelagem de perfis de carga com *ramp-up*, carga estável e *ramp-down* programáticos, enquanto o agrupamento de requisições via `group()` permite segmentar métricas por interação [5].

Para integração com ecossistemas de observabilidade, o k6 oferece exportação nativa de métricas para Prometheus [11], InfluxDB e Grafana Cloud, além de saída em JSON e CSV para análise offline. Essa integração é particularmente útil para correlacionar métricas do gerador de carga com dados de infraestrutura coletados por ferramentas como node_exporter, viabilizando o monitoramento estendido proposto neste estudo.

O k6 foi selecionado por sua capacidade de modelar com precisão os perfis de workload da especificação TPC-W e por sua integração nativa com o ecossistema Prometheus/Grafana, que viabiliza o monitoramento estendido descrito na Seção 3.

2.3 Aplicação-Alvo: TPC-W

O TPC-W é uma especificação de benchmark de desempenho proposta pelo *Transaction Processing Performance Council* que define uma loja virtual de livros com 14 interações web, grafos de navegação entre páginas, tabelas de transição probabilísticas e esquema

de banco de dados [8]. A especificação padroniza três perfis de *workload* que diferem na proporção entre interações de navegação (leitura) e interações transacionais (escrita): *browsing* (~95% navegação), *shopping* (~52% navegação, ~48% transações) e *ordering* (~37% navegação, ~63% transações). A métrica principal definida pela especificação é o WIPS (*Web Interactions Per Second*). Embora a especificação tenha sido retirada de circulação pelo TPC em 2005, sua implementação de referência permanece amplamente utilizada como aplicação-alvo em estudos empíricos de desempenho [3], função distinta da comparação de infraestrutura para a qual o benchmark foi originalmente concebido.

Neste trabalho, a especificação TPC-W é utilizada não como benchmark para comparação de hardware ou infraestrutura, mas como **aplicação-alvo** do *quasi*-experimento. A implementação Java disponível no repositório TPC-W-Benchmark-R é composta por *servlets* executados sobre Apache Tomcat com PostgreSQL, e fornece uma aplicação web completa com operações de leitura e escrita, autenticação simplificada, consultas de complexidade variável e um esquema de banco de dados relacional. Essas características permitem avaliar como diferentes estratégias de teste exercitam diferentes camadas do sistema e produzem diagnósticos distintos.

A escolha dessa aplicação se justifica por três razões. Primeiro, os três perfis de *workload* padronizados permitem isolar o efeito do perfil de carga sobre o diagnóstico, mantendo a aplicação constante. Segundo, a complexidade variável das 14 interações web oferece oportunidades para que gargalos se manifestem em interações específicas, e não apenas no desempenho global. Terceiro, o uso consolidado do TPC-W na literatura de desempenho [3, 8] confere reprodutibilidade ao experimento e permite comparação com estudos anteriores.

2.4 Trabalhos Relacionados

Golmohammadi, Zhang e Arcuri [4] reuniram 92 estudos sobre teste de APIs REST, categorizando abordagens por estratégia de geração, tipo de oráculo e métricas de avaliação. O *survey* identifica que a maioria das ferramentas opera em modo *black-box* e aponta a ausência de comparações padronizadas entre abordagens sob condições controladas. Contudo, o foco permanece em teste funcional e de conformidade, sem investigar a dimensão de desempenho ou o custo experimental envolvido.

Arcuri *et al.* [2] avaliaram ferramentas de geração de testes (OpenAPI Generator, EvoMaster, StarCoder) em contexto industrial na Volkswagen AG, propondo critérios que incluem facilidade de integração e custo operacional. Embora o trabalho considere aspectos de custo, a avaliação é voltada a testes funcionais e não mede a capacidade de cada abordagem em revelar problemas de desempenho sob carga.

Lee, Kuo e Chen [7] utilizaram algoritmos genéticos para guiar *fuzzing* em APIs web, buscando entradas que causem latência elevada. O método superou o *fuzzing* aleatório em cinco APIs reais, identificando entradas com tempos de resposta até 26 vezes superiores. O estudo aborda especificamente a dimensão de desempenho, mas avalia apenas uma técnica de geração de entradas, sem comparar estratégias completas que incluam decisões sobre *workload* e monitoramento.

Kim, Sinha e Orso [6] propuseram o ARAT-RL, que usa aprendizagem por reforço para priorizar operações durante a exploração de APIs REST. A técnica superou ferramentas como RESTler e EvoMaster em cobertura e detecção de falhas, mas, assim como Lee *et al.*, concentra-se na estratégia de exploração sem avaliar o custo experimental ou comparar estratégias de teste de desempenho de ponta a ponta.

Os trabalhos analisados investigam técnicas sofisticadas para geração e exploração de testes, mas nenhum deles compara sistematicamente estratégias que variem em perfil de *workload* e granularidade de monitoramento, nem avalia a relação entre custo de preparação e capacidade diagnóstica. Vale observar que trabalhos diretamente relacionados ao tema deste estudo, teste de desempenho de aplicações web com análise de custo experimental, são escassos na literatura recente, o que reforça a lacuna que este trabalho busca preencher. Os trabalhos selecionados representam as abordagens mais próximas disponíveis, ainda que com foco predominante em correção funcional. A Tabela 1 sintetiza esse posicionamento segundo três critérios: (i) *foco em desempenho* indica se o trabalho avalia comportamento sob carga; (ii) *custo experimental* indica se o custo de preparação dos testes é considerado; e (iii) *comparação de estratégias* indica se múltiplas estratégias de teste completas são comparadas entre si.

Tabela 1: Posicionamento em relação aos trabalhos relacionados

Trabalho	Foco desemp.	Custo exp.	Comp. estrat.
[2] Arcuri	Não	Parcial	Não
[4] Golmohammadi	Não	Não	Não
[6] Kim	Não	Não	Parcial
[7] Lee	Sim	Não	Parcial
Este trabalho	**Sim**	**Sim**	**Sim**

3 Metodologia

Para responder à pergunta de pesquisa, o estudo segue o fluxo metodológico ilustrado na Figura 2, organizado em quatro fases: planejamento, execução dos testes, análise comparativa e síntese dos resultados.

3.1 Desenho de Pesquisa

O estudo adota um desenho de *quasi*-experimento, uma vez que as variáveis independentes são manipuladas deliberadamente, mas não é possível garantir controle absoluto sobre variáveis intervenientes do ambiente de execução, como estado de caches, comportamento do compilador JIT e ordem de execução das estratégias [10].

Para respondê-la de forma operacional, o trabalho segue três passos analíticos, cada um associado a uma dimensão da questão de pesquisa:

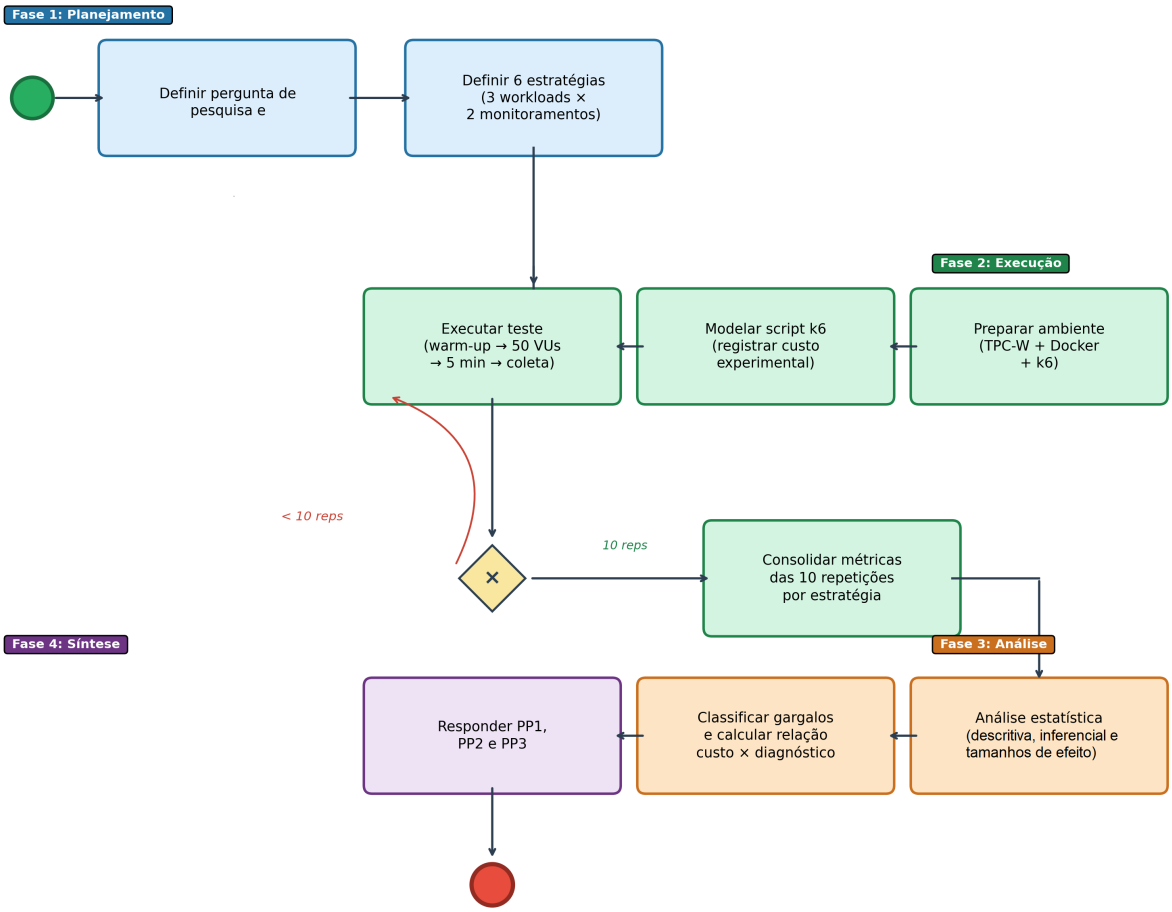

Figura 2: Fluxo metodológico do *quasi*-experimento

Passo 1 (PP1): Isolar o efeito do perfil de *workload*. Os três perfis (browsing, shopping, ordering) são aplicados à mesma aplicação com monitoramento básico. As métricas são comparadas entre os perfis para determinar se *workloads* com proporções distintas de leitura e escrita produzem diagnósticos qualitativamente diferentes.

Passo 2 (PP2): Isolar o efeito do monitoramento. Para cada perfil, os resultados com monitoramento básico e estendido são comparados para avaliar se a adição de métricas de infraestrutura permite identificar a causa raiz dos gargalos detectados no passo anterior, e a que custo adicional.

Passo 3 (PP3): Calcular a relação custo-benefício. Para cada estratégia, o custo experimental total é relacionado com o valor diagnóstico produzido. As estratégias são comparadas considerando conjuntamente o custo incremental de preparação e a informação diagnóstica obtida, buscando identificar o melhor equilíbrio entre essas duas dimensões.

3.2 Sistema-Alvo e Ambiente

O *quasi*-experimento utiliza a implementação Java da aplicação web disponível no repositório TPC-W-Benchmark-R[1], composta por *servlets* sobre Apache Tomcat 9.0.118 com PostgreSQL 15, implantados em contêineres Docker. O banco é populado com 1.000 itens, uma escala inferior à recomendada pela especificação original (10.000), adotada para viabilizar o volume de execuções dentro das restrições do ambiente experimental. A aplicação não é utilizada como benchmark de comparação de infraestrutura, mas como sistema-alvo dos testes de desempenho.

O ambiente de execução é composto por uma máquina com processador Intel Core i7-1255U (10 núcleos, 12 threads), 16 GB de RAM, armazenamento NVMe de 473 GB, sistema operacional CachyOS Linux (kernel 7.0.11 - sistema operacional de uso geral baseado em Arch Linux) e Docker 29.2.1. O k6 v2.0.0 é instalado nativamente. Para o monitoramento estendido, são utilizados Prometheus [11]

[1]https://github.com/lowgue/TPC-W-Benchmark-R

com *node_exporter* (métricas de sistema) e *postgres_exporter* (métricas do banco), visualizados via Grafana.

3.3 Estratégias de Teste

Cada estratégia combina um perfil de *workload* com um nível de monitoramento, resultando em 6 configurações experimentais.

Os perfis de *workload* seguem as tabelas de transição probabilísticas da especificação TPC-W [8], que definem como os usuários virtuais navegam entre as 14 interações da aplicação. O perfil **browsing** (E1, E2) concentra a carga em interações de leitura: *home* (29%), *product detail* (23%), *new products* (20%), *search* (17%) e *best sellers* (11%). Nenhuma operação de escrita é realizada, o que permite avaliar o desempenho do sistema em condições de consulta pura. O perfil **shopping** (E3, E4) introduz operações transacionais de compra: além das interações de navegação (∼52%), o *workload* inclui: *shopping cart* (19%), *buy request* (16%) e *buy confirm* (13%). Essas operações envolvem escrita no banco de dados (inserção de itens no carrinho, criação de pedidos, atualização de estoque) e dependências entre requisições (o *buy confirm* requer um SHOPPING_ID e um C_ID obtidos em interações anteriores). O perfil ***ordering*** (E5, E6) maximiza as operações transacionais (∼63%), reduzindo a navegação a ∼37% e adicionando a interação *order inquiry* (12%), que consulta pedidos realizados.

O nível de monitoramento determina a profundidade das informações disponíveis para diagnóstico. O **monitoramento básico** utiliza exclusivamente as métricas nativas do k6: latência por requisição (com segmentação por interação via `group()`), *throughput* agregado, taxa de erro e volume de dados transferidos. O **monitoramento estendido** adiciona três fontes de dados: *node_exporter* para métricas de CPU, memória, disco e rede da máquina anfitriã; *postgres_exporter* para métricas de conexões ativas, locks e estatísticas de tabelas do PostgreSQL; e *slow query log* do PostgreSQL configurado com limiar de 50 ms, que registra todas as consultas com tempo de execução acima desse valor. Todas as métricas de infraestrutura são coletadas por Prometheus [11] com intervalo de 5 segundos e visualizadas via Grafana.

As seis configurações resultantes são: E1 (*browsing*, básico), E2 (*browsing*, estendido), E3 (*shopping*, básico), E4 (*shopping*, estendido), E5 (*ordering*, básico) e E6 (*ordering*, estendido).

3.4 Variáveis e Métricas

As variáveis independentes são o perfil de *workload* e o nível de monitoramento. As variáveis dependentes capturam duas dimensões. O *valor diagnóstico* é avaliado por meio das seguintes métricas: latência por interação em percentis (p50, p95, p99), *throughput* em interações por segundo (WIPS), taxa de erro e, no monitoramento estendido, utilização de recursos (CPU, memória, I/O de disco e rede). Operacionalizamos o valor diagnóstico como a capacidade de uma estratégia de detectar degradação de desempenho, localizar as interações problemáticas e fornecer evidências para classificar a natureza do gargalo. Uma estratégia é considerada indicativa de degradação quando sua taxa de erro global ultrapassa 10%. Uma interação é considerada problemática quando sua latência p95 excede em mais de cinco vezes a mediana das demais interações do mesmo perfil de *workload*. A classificação da natureza do gargalo é realizada a partir das evidências fornecidas pelas métricas, logs e informações de infraestrutura disponíveis em cada nível de monitoramento.

Os gargalos são classificados em três tipos: gargalo de banco de dados (latência elevada em consultas ou escritas), gargalo de concorrência (contenção por *locks* ou pool de conexões) e gargalo de aplicação (*timeout* em lógica de negócio). O *custo experimental* é medido por tempo de modelagem do *script* (minutos), complexidade (linhas de código) e tempo de configuração do monitoramento (minutos), registrados por cronômetro durante a execução real e incluindo consulta à documentação e depuração.

3.5 Procedimento de Execução

Cada configuração é executada 10 vezes, totalizando 60 execuções. As dez repetições por configuração foram adotadas para reduzir a influência de flutuações pontuais do ambiente e permitir avaliar a variabilidade dos resultados entre execuções. Cada execução segue cinco etapas: reinício dos contêineres Docker; *warm-up* de 2 minutos com 10 VUs (dados descartados); execução com 50 VUs, *ramp-up* de 1 minuto, 5 minutos de carga estável e *ramp-down* de 30 segundos; coleta de métricas em JSON; e classificação dos gargalos identificados.

3.6 Análise Estatística

Além da análise descritiva baseada em medianas, desvios e diferenças percentuais, foram aplicados testes estatísticos inferenciais para avaliar se as diferenças observadas entre as estratégias poderiam ser atribuídas aos fatores manipulados no *quasi*-experimento. O nível de significância adotado foi $\alpha = 0{,}05$.

Para o *throughput* (WIPS), inicialmente foram avaliados os pressupostos para análise paramétrica. A normalidade foi verificada separadamente para cada uma das seis estratégias pelo teste de Shapiro–Wilk, enquanto a homogeneidade das variâncias foi avaliada pelo teste de Levene. Como não foram observadas violações desses pressupostos para WIPS, aplicou-se uma ANOVA fatorial 3×2, considerando como fatores o perfil de *workload* (*browsing*, *shopping* e *ordering*) e o nível de monitoramento (básico e estendido). A magnitude dos efeitos foi estimada pelo eta quadrado parcial (η_p^2).

Como as demais métricas apresentaram distribuições assimétricas, heterogeneidade de variâncias ou valores limitados pelo *timeout*, foram utilizados procedimentos não paramétricos. O efeito do perfil de *workload* foi analisado pelo teste de Kruskal–Wallis, considerando as estratégias com monitoramento básico, e sua magnitude foi estimada pelo epsilon quadrado (ε^2).

Para avaliar o efeito simples do nível de monitoramento dentro de cada perfil de *workload*, foram realizadas comparações entre E1–E2, E3–E4 e E5–E6 utilizando o teste de Mann–Whitney. Nessas comparações, o tamanho de efeito foi estimado pelo delta de Cliff (δ), sendo valores absolutos inferiores a 0,147 considerados desprezíveis, entre 0,147 e 0,329 pequenos, entre 0,330 e 0,473 médios e iguais ou superiores a 0,474 grandes. Os valores de p das comparações múltiplas foram ajustados pelo procedimento de Holm.

A interpretação dos resultados considerou conjuntamente significância estatística e magnitude do efeito. Essa distinção é particularmente importante para métricas de latência limitadas pelo *timeout* de 60 segundos, nas quais pequenas diferenças numéricas próximas

ao limite podem ser estatisticamente detectáveis sem representar diferenças de relevância prática.

4 Resultados

Esta seção apresenta os resultados consolidados das 60 execuções, incluindo o desempenho nos diferentes perfis, o efeito do monitoramento, a análise estatística inferencial, o custo experimental e a resposta à pergunta de pesquisa. Os dados brutos, *scripts* e logs estão disponíveis no repositório de artefatos[2].

4.1 Visão Geral

A Tabela 2 apresenta as métricas globais de cada estratégia, calculadas como mediana das 10 repetições, juntamente com o custo de preparação registrado durante o experimento.

Os resultados revelam uma separação acentuada entre os perfis de *workload*. As estratégias de *browsing* (E1, E2) apresentaram desempenho estável: *throughput* mediano de 6,2 WIPS com desvio padrão inferior a 0,16, latência p95 inferior a 8 ms e taxa de erro de 0% em todas as 20 execuções. As estratégias que introduziram operações transacionais (E3 a E6) apresentaram degradação severa, com taxas de erro entre 15% e 51% e valores de p95 atingindo o limite de *timeout* de 60 segundos configurado no k6.

4.2 Perfil *Browsing* (E1 e E2)

No perfil *browsing*, todas as cinco interações de navegação operaram com latências consistentemente baixas ao longo das 20 execuções (10 por estratégia). As interações *Product Detail* e *Search* apresentaram as menores latências p95 (5,85 ms e 5,29 ms em E1, respectivamente), seguidas de *Home* (6,41 ms). As interações *New Products* e *Best Sellers* registraram latências ligeiramente superiores (8,28 ms e 9,47 ms em E1), o que é compatível com a complexidade das consultas ao banco envolvidas, que incluem agregações e ordenações por assunto. Todas as interações ficaram muito abaixo do *threshold* de 3.000 ms definido no *script*.

A comparação entre E1 e E2 revela que o monitoramento estendido não introduziu *overhead* perceptível: as latências p95 por interação variaram menos de 0,3 ms entre as duas estratégias (por exemplo, *Home* com 6,41 ms em E1 e 6,18 ms em E2, e *Best Sellers* com 9,47 ms em E1 e 9,19 ms em E2). Esse resultado indica que, quando o sistema opera sem contenção, a adição de Prometheus, *node_exporter* e *postgres_exporter* não afeta as métricas de desempenho da aplicação.

4.3 Perfis *Shopping* e *Ordering* (E3 a E6)

A introdução de operações transacionais alterou radicalmente o comportamento do sistema. O gargalo concentrou-se integralmente na interação *Buy Confirm*, que atingiu o *timeout* de 60 segundos em todas as 40 execuções dos perfis transacionais (E3 a E6), sem exceção. Esse *timeout* indica que a operação de confirmação de compra, que envolve múltiplas escritas no banco de dados (atualização de estoque, registro de pedido, processamento de pagamento), não conseguiu ser concluída dentro do limite configurado quando submetida a 50 usuários virtuais simultâneos.

[2]https://github.com/moraeseriic/teste-api-rest

As demais interações transacionais apresentaram latências elevadas em comparação ao *browsing*, mas dentro de limites operacionais. No perfil *shopping* (E3), *Shopping Cart* registrou p95 de 21,27 ms e *Buy Request* ficou em 11,48 ms. No perfil *ordering* (E5), os valores foram similares: Shopping Cart com 21,56 ms e *Buy Request* com 11,29 ms. A interação *Order Inquiry*, exclusiva do perfil *ordering*, apresentou latência de apenas 2,07 ms, confirmando que a consulta de pedidos é uma operação leve que não contribui para a contenção.

Um efeito colateral relevante é que a contenção gerada pelo *Buy Confirm* degradou também as interações de navegação. No perfil shopping (E3), *Home* registrou p95 de 11,69 ms e *New Products* de 15,50 ms, valores entre 2 e 4 vezes superiores aos observados no *browsing* puro (6,41 ms e 8,28 ms, respectivamente). No perfil *ordering* (E5), essa degradação foi ainda mais acentuada em algumas interações, como *Best Sellers*, que subiu de 9,47 ms no *browsing* para 19,45 ms no *ordering*. Esse comportamento sugere que a contenção no banco de dados causada pelo *Buy Confirm* afeta o desempenho global do sistema, e não apenas a interação problemática. Dados detalhados por interação e por repetição estão disponíveis no repositório de artefatos.

4.4 Efeito do Monitoramento nos Perfis Transacionais

Nos perfis transacionais, as estratégias com monitoramento estendido (E4, E6) apresentaram *throughput* inferior às estratégias básicas correspondentes. No perfil *shopping*, a mediana caiu de 4,11 WIPS em E3 para 2,02 WIPS em E4, uma redução de aproximadamente 51%. No perfil *ordering*, a redução foi de 4,01 para 2,03 WIPS, aproximadamente 49%. As análises inferenciais apresentadas na Seção 4.5 confirmam que essas diferenças possuem elevada magnitude ($\delta = 1{,}00$) e são estatisticamente significativas ($p_{\text{Holm}} < 0{,}001$).

Ao mesmo tempo, a taxa de erro diminuiu de 51,29% em E3 para 16,37% em E4 e de 45,02% em E5 para 15,70% em E6, também com efeitos máximos ($\delta = 1{,}00$; $p_{\text{Holm}} < 0{,}001$). Esse comportamento ocorre simultaneamente à redução da vazão: com menos requisições sendo processadas durante o intervalo experimental, uma proporção menor alcança o *Buy Confirm* e atinge o *timeout*. A associação observada, contudo, não permite atribuir integralmente a redução de *throughput* ao custo do monitoramento, pois o efeito da ordem de execução permanece como ameaça à validade interna.

4.5 Análise Estatística

A análise inferencial foi utilizada para verificar se as diferenças observadas entre as estratégias eram consistentes ao longo das dez repetições de cada configuração. Para WIPS, os testes de Shapiro–Wilk não indicaram evidência de desvio de normalidade em nenhuma das seis estratégias ($p > 0{,}23$). O teste de Levene também não rejeitou a hipótese de homogeneidade das variâncias ($F(5, 54) = 1{,}54$, $p = 0{,}194$), permitindo a aplicação da ANOVA fatorial.

Como apresentado na Tabela 3, foram observados efeitos estatisticamente significativos do perfil de *workload* ($F(2, 54) = 2724{,}25$, $p < 0{,}001$, $\eta_p^2 = 0{,}990$), do nível de monitoramento ($F(1, 54) = 1131{,}29$, $p < 0{,}001$, $\eta_p^2 = 0{,}954$) e, principalmente, da interação entre os dois fatores ($F(2, 54) = 262{,}33$, $p < 0{,}001$, $\eta_p^2 = 0{,}907$). Todos os efeitos apresentaram magnitude elevada.

Tabela 2: Métricas globais por estratégia (mediana de 10 repetições) e custo de preparação

ID	Configuração	WIPS	p95 (ms)	Média (ms)	Erro (%)	Reqs	Custo (min)	Script (LoC)
E1	Browsing / Básico	6,22	7,23	4,82	0,00	2552	60	103
E2	Browsing / Estendido	6,10	7,40	4,62	0,00	2528	45	0
E3	Shopping / Básico	4,11	60.000	3.056	51,29	1974	30	192
E4	Shopping / Estendido	2,02	60.000	9.827	16,37	1170	0	0
E5	Ordering / Básico	4,01	28.711	3.017	45,02	1993	15	206
E6	Ordering / Estendido	2,03	60.000	9.430	15,70	1200	0	0

Tabela 3: ANOVA fatorial para *throughput* (WIPS).

Efeito	gl	F	p	η_p^2
Workload	2	2724,25	$< 0,001$	0,990
Monitoramento	1	1131,29	$< 0,001$	0,954
Workload × Monitoramento	2	262,33	$< 0,001$	0,907

A interação indica que o efeito do monitoramento sobre o *throughput* não é constante entre os perfis de *workload*. Portanto, os efeitos principais não devem ser interpretados isoladamente. Para investigar essa interação, foram realizadas comparações entre monitoramento básico e estendido dentro de cada perfil.

Tabela 4: Efeito do nível de monitoramento sobre WIPS e taxa de erro.

Métrica	Comparação	Básico	Estendido	δ	p_{Holm}
WIPS	E1–E2	6,22	6,10	0,24	0,385
WIPS	E3–E4	4,11	2,02	1,00	$< 0,001$
WIPS	E5–E6	4,01	2,03	1,00	$< 0,001$
Erro (%)	E1–E2	0,00	0,00	0,00	1,000
Erro (%)	E3–E4	51,29	16,37	1,00	$< 0,001$
Erro (%)	E5–E6	45,02	15,70	1,00	$< 0,001$

Os efeitos simples apresentados na Tabela 4 esclarecem a interação identificada pela ANOVA. No perfil *browsing*, a diferença de *throughput* entre monitoramento básico e estendido foi pequena: a mediana passou de 6,22 para 6,10 WIPS, uma redução de aproximadamente 2%, com tamanho de efeito pequeno ($\delta = 0,24$) e sem significância estatística ($p_{Holm} = 0,385$).

Nos perfis transacionais, entretanto, o comportamento foi distinto. No perfil *shopping*, o *throughput* caiu de 4,11 para 2,02 WIPS, uma redução de aproximadamente 51%, enquanto no perfil *ordering* caiu de 4,01 para 2,03 WIPS, redução de aproximadamente 49%. Em ambos os casos, todas as observações obtidas com monitoramento básico foram superiores às observações correspondentes do grupo com monitoramento estendido, resultando em um delta de Cliff máximo ($\delta = 1,00$; $p_{Holm} < 0,001$).

A taxa de erro apresentou comportamento complementar. Não houve diferença entre E1 e E2, pois nenhuma execução do perfil *browsing* apresentou erros. Já nos perfis transacionais, as estratégias com monitoramento estendido apresentaram taxas de erro substancialmente menores, com efeito máximo tanto em E3–E4 quanto em E5–E6 ($\delta = 1,00$; $p_{Holm} < 0,001$). Esse resultado não indica melhoria do sistema: ele ocorre simultaneamente à redução de aproximadamente 50% do *throughput*, diminuindo a quantidade de requisições que alcançam a interação problemática durante o intervalo de execução.

Para complementar a análise de PP1, o efeito do perfil de *workload* foi analisado considerando apenas as estratégias com monitoramento básico (E1, E3 e E5). Essa comparação evita que o efeito do monitoramento interfira diretamente na avaliação dos diferentes perfis. Como algumas métricas apresentaram assimetria e heterogeneidade de variâncias, foi utilizado o teste não paramétrico de Kruskal–Wallis.

Tabela 5: Efeito do perfil de workload nas estratégias com monitoramento básico.

Métrica	H(2)	p	ε^2
WIPS	20,39	$< 0,001$	0,681
p95 global	20,97	$< 0,001$	0,703
Latência Média	19,94	$< 0,001$	0,664
Taxa de Erro	25,05	$< 0,001$	0,854

Como mostra a Tabela 5, o perfil de *workload* apresentou efeito estatisticamente significativo e de grande magnitude sobre todas as métricas analisadas. Para WIPS, obteve-se $H(2) = 20,39$, $p < 0,001$ e $\varepsilon^2 = 0,681$. Resultados semelhantes foram encontrados para p95 ($\varepsilon^2 = 0,703$), latência média ($\varepsilon^2 = 0,664$) e taxa de erro ($\varepsilon^2 = 0,854$).

As comparações pós-hoc para WIPS mostraram diferenças entre *browsing* e *shopping* ($\delta = 1,00$, $p_{Holm} < 0,001$) e entre *browsing* e *ordering* ($\delta = 1,00$, $p_{Holm} < 0,001$). Por outro lado, a diferença entre *shopping* e *ordering* não foi estatisticamente significativa ($\delta = 0,40$, $p_{Holm} = 0,140$). Assim, para *throughput*, a principal mudança de comportamento ocorre entre o perfil exclusivamente de leitura e os dois perfis que introduzem operações transacionais, e não entre os dois perfis transacionais entre si.

A interpretação inferencial do p95 nos perfis transacionais requer cautela. Diversas execuções atingiram o limite de *timeout* de 60 segundos, produzindo observações concentradas próximas a 60.000 ms. Nessa situação, diferenças numéricas entre estratégias podem resultar em efeitos estatísticos elevados apesar de representarem essencialmente o mesmo evento: uma requisição que atingiu o limite de tempo. Por esse motivo, as conclusões sobre os perfis transacionais priorizam *throughput*, taxa de erro e a ocorrência do *timeout*, em vez de interpretar isoladamente diferenças no p95 global.

4.6 Custo Experimental

O custo de preparação variou entre as estratégias. A estratégia E1 demandou o maior investimento inicial (60 minutos), incluindo a escrita do script de browsing (103 linhas, 45 minutos) e a configuração do ambiente Docker (15 minutos). A estratégia E3 exigiu 30 minutos adicionais para implementar o fluxo transacional de compra (192 linhas), que envolveu a extração dinâmica de SHOPPING_ID e C_ID das respostas HTML da aplicação. A estratégia E5 adicionou 15 minutos sobre E3 para incluir a interação *Order Inquiry* (206 linhas). O monitoramento estendido consumiu 45 minutos de configuração em E2 (setup de Prometheus, *node_exporter*, *postgres_exporter* e *slow query log*) e foi reutilizado sem custo adicional em E4 e E6.

4.7 Resposta à Pergunta de Pesquisa

Os dados permitem responder à pergunta central do estudo: *realizar testes com monitoramento básico vale a pena em relação ao esforço de configurar testes com monitoramento estendido?*

No contexto experimental avaliado, os resultados favorecem uma estratégia incremental que inicia com monitoramento básico e utiliza perfis de *workload* distintos antes de adicionar instrumentação mais detalhada. As estratégias com monitoramento básico (E1, E3, E5), que utilizaram exclusivamente as métricas nativas do k6, foram suficientes para identificar o gargalo mais severo da aplicação (*timeout* de 60 segundos na interação *Buy Confirm*) e para distinguir o comportamento saudável do perfil *browsing* da degradação provocada pelos perfis transacionais. Essas três estratégias, somadas, demandaram 105 minutos de preparação e nenhuma dependência de infraestrutura adicional.

As estratégias com monitoramento estendido (E2, E4, E6) exigiram 45 minutos adicionais de configuração (Prometheus, *exporters*, *slow query log*). Nas execuções realizadas, as estratégias com monitoramento estendido foram associadas a uma redução de aproximadamente 50% no *throughput* dos perfis transacionais. Essa diferença apresentou efeito máximo ($\delta = 1{,}00$, $p_{\text{Holm}} < 0{,}001$), embora a ordem fixa das execuções não permita atribuir toda a redução exclusivamente ao *overhead* do monitoramento. O monitoramento estendido agregou informação diagnóstica ao permitir investigar a causa raiz dos gargalos identificados, incluindo contenção no banco, *slow queries* e esgotamento de conexões. No cenário avaliado, entretanto, sua aplicação mostrou-se mais vantajosa como segundo passo, após a identificação inicial das interações problemáticas por meio do monitoramento básico.

Em termos práticos, o monitoramento básico com dois perfis de *workload* distintos (E1 + E3, 90 minutos) detectou o mesmo gargalo que as seis estratégias combinadas (150 minutos), com 40% menos esforço. O monitoramento estendido se justifica como segundo passo, aplicado seletivamente sobre as interações problemáticas já identificadas.

5 Discussão

5.1 PP1: Efeito do Perfil de Workload

Os resultados confirmam que o perfil de *workload* exerce forte influência sobre o comportamento observado da aplicação. Considerando as estratégias com monitoramento básico, foram identificados efeitos significativos e de grande magnitude sobre WIPS ($H(2) = 20{,}39$, $p < 0{,}001$, $\varepsilon^2 = 0{,}681$), latência e taxa de erro. Em particular, as comparações pós-hoc evidenciaram diferenças máximas entre o perfil *browsing* e os dois perfis transacionais ($\delta = 1{,}00$), enquanto *shopping* e *ordering* não diferiram significativamente em throughput. O perfil *browsing*, que exercita exclusivamente operações de leitura, apresentou um sistema aparentemente saudável, com taxa de erro nula e latências consistentemente baixas em todas as interações. Esse diagnóstico seria a única conclusão disponível caso apenas esse perfil fosse utilizado, e indicaria, de forma equivocada, que a aplicação está pronta para operar em produção sem restrições.

Os perfis *shopping* e *ordering*, ao introduzirem operações transacionais de escrita, revelaram um gargalo crítico na interação *Buy Confirm* que compromete toda a camada transacional da aplicação. A latência dessa interação atingiu o *timeout* de 60 segundos em 100% das execuções de E3 a E6, com taxas de erro globais entre 15% e 51%. Esse achado demonstra que a escolha do perfil de *workload* determina não apenas *quanto* se mede, mas *o que* se consegue observar: um teste baseado apenas em leitura é incapaz de diagnosticar falhas que ocorrem na camada de escrita.

Para WIPS, as comparações pós-hoc identificaram efeitos máximos entre *browsing* e os perfis *shopping* e *ordering* ($\delta = 1{,}00$, $p_{\text{Holm}} < 0{,}001$ em ambos os casos). Em contraste, *shopping* e *ordering* não apresentaram diferença estatisticamente significativa de *throughput* ($\delta = 0{,}40$, $p_{\text{Holm}} = 0{,}140$). Esse resultado sugere que a introdução de operações transacionais, mais do que o aumento adicional de sua proporção, é responsável pela principal mudança observada nesse cenário experimental.

Outro achado relevante é o efeito de propagação: a contenção gerada pelo *Buy Confirm* degradou também as interações de navegação. No perfil *ordering*, a interação *Best Sellers* registrou p95 de 19,45 ms, o dobro do valor observado no *browsing* (9,47 ms), embora a operação de leitura em si não tenha mudado. Isso indica que o gargalo no banco de dados afeta o desempenho global do sistema por meio de contenção de conexões ou *locks* que competem com as consultas de leitura.

5.2 PP2: Efeito do Monitoramento

No perfil *browsing*, o monitoramento estendido não produziu valor diagnóstico adicional, uma vez que o sistema operou sem contenção e não havia gargalos a investigar. As métricas de latência foram equivalentes entre E1 e E2, com diferenças inferiores a 0,3 ms por interação. Esse resultado, em conjunto com a diferença pequena e não significativa de WIPS entre E1 e E2, não fornece evidências de impacto relevante da instrumentação sobre o desempenho no perfil *browsing*.

Os resultados inferenciais mostram, entretanto, que o efeito do monitoramento depende do perfil exercitado. A ANOVA identificou forte interação entre *workload* e monitoramento ($F(2, 54) = 262{,}33$, $p < 0{,}001$, $\eta_p^2 = 0{,}907$). No perfil *browsing*, a diferença entre E1 e E2 foi pequena e não significativa para WIPS ($\delta = 0{,}24$, $p_{\text{Holm}} = 0{,}385$). Nos perfis *shopping* e *ordering*, em contraste, foram observados efeitos máximos entre as configurações básica e estendida ($\delta = 1{,}00$, $p_{\text{Holm}} < 0{,}001$). Assim, os dados não sustentam a interpretação de um custo uniforme do monitoramento: seu efeito observado

sobre o *throughput* depende das condições de carga e do estado de contenção da aplicação.

Nos perfis transacionais, o monitoramento estendido apresentou um *trade-off* relevante. Por um lado, os *slow query logs* coletados em E4 e E6 permitiram identificar a causa raiz do gargalo: a interação *Buy Confirm* executa um padrão `SELECT MAX(id)+1` seguido de `INSERT`, que não é *thread-safe* sob concorrência de 50 VUs, gerando violações de chave primária (*duplicate key value violates unique constraint "cc_xacts_pkey"*). Além disso, os *logs* revelaram mensagens de `too many clients already` no PostgreSQL, indicando exaustão do pool JDBC do Tomcat após execuções transacionais sucessivas. Sem o monitoramento estendido, esses 45–55% de erro seriam atribuídos a um "*timeout* genérico" sem identificação da causa. Por outro lado, as estratégias estendidas (E4, E6) apresentaram WIPS reduzido pela metade em relação às básicas (2,02 vs. 4,11 no shopping; 2,03 vs. 4,01 no *ordering*). Esse comportamento é compatível com a hipótese de que a instrumentação adicional compete por recursos com a aplicação sob contenção. Contudo, o desenho experimental não permite atribuir exclusivamente ao monitoramento a redução observada, uma vez que o efeito da ordem de execução não pode ser descartado.

A diminuição da taxa de erro nas estratégias estendidas (de 51% em E3 para 16% em E4) reforça essa interpretação: com menos requisições por segundo chegando ao *Buy Confirm*, menos delas atingem o *timeout*, o que reduz a taxa de erro aparente sem que o gargalo subjacente tenha sido resolvido.

5.3 PP3: Relação Custo-Benefício

A análise conjunta do custo experimental e das evidências diagnósticas indica que E3 (*shopping* com monitoramento básico) apresentou o melhor compromisso entre esforço adicional e informação diagnóstica obtida. Com 30 minutos adicionais de preparação sobre E1, ela expôs o gargalo crítico de *Buy Confirm*, que é completamente invisível ao perfil *browsing*. A estratégia E5 (*ordering* básico), embora tenha confirmado o mesmo gargalo e adicionado a observação sobre *Order Inquiry* (operação leve, p95 de 2 ms), acrescentou pouca informação incremental a um custo de 15 minutos. A estratégia E1 (*browsing* básico), com o maior investimento inicial (60 min para *script* e ambiente), é indispensável como linha de base, mas insuficiente como estratégia única de teste.

As estratégias com monitoramento estendido (E2, E4, E6) agregam valor quando a causa raiz de um gargalo já identificado precisa ser determinada, mas não se justificam como abordagem inicial. A configuração do monitoramento estendido (45 min em E2) é um custo fixo que se amortiza nas execuções seguintes (E4, E6 reutilizaram a mesma infraestrutura), porém a redução de *throughput* observada nas configurações estendidas, ainda que não possa ser atribuída exclusivamente ao monitoramento, recomenda cautela em sua aplicação em cenários de alta contenção.

A recomendação prática derivada dos dados é uma abordagem em camadas: iniciar com monitoramento básico em pelo menos dois perfis de *workload* distintos (um de leitura e um transacional) para identificar *onde* os gargalos estão, e aplicar monitoramento estendido seletivamente sobre as interações problemáticas para determinar *por quê*. No contexto deste estudo, a combinação E1 + E3 (90 minutos de preparação) foi suficiente para detectar e localizar o gargalo mais severo do sistema, enquanto o custo total das seis estratégias foi de 150 minutos.

5.4 Ameaças à Validade

Validade interna. A principal ameaça é a variabilidade introduzida pelo ambiente de execução. Embora os contêineres Docker forneçam isolamento a nível de processo, fatores como estado de caches, comportamento do compilador JIT do Tomcat, escalonamento de *goroutines* do k6 e processos de fundo do sistema operacional podem influenciar as medições. Para mitigar essa ameaça, cada configuração foi executada 10 vezes com reinício completo dos contêineres entre execuções, e um *warm-up* de 2 minutos com 10 VUs foi realizado antes de cada teste (com dados descartados) para estabilizar o JIT e o *connection pool*. Uma ameaça adicional é o viés de ordem de execução: as estratégias foram executadas na sequência E1→E3→E5→E2→E4→E6, e as execuções transacionais (E3, E5) podem ter deixado o sistema em estado degradado (conexões JDBC não liberadas, transações órfãs) antes de E4 e E6, contribuindo para a degradação observada nessas estratégias. O fator de escala do banco (1.000 itens) é inferior ao recomendado pela especificação original (10.000), o que pode alterar o perfil de contenção observado.

Validade externa. Os resultados foram obtidos sobre uma única implementação da aplicação web (TPC-W-Benchmark-R) e com uma única ferramenta de teste (k6 v2.0.0), executados em uma máquina específica (Intel i7-1255U, 16 GB RAM). O gargalo de 60 segundos na interação *Buy Confirm* pode ser específico dessa implementação dos *servlets* e não necessariamente representativo de aplicações de comércio eletrônico reais. Além disso, os resultados refletem o comportamento sob 50 VUs com *think time* de 1 a 7 segundos; cargas superiores ou inferiores poderiam revelar padrões diferentes de degradação. A generalização dos achados para outros sistemas, ferramentas e escalas de carga deve ser feita com cautela.

Validade de construto. A operacionalização de "valor diagnóstico" envolve julgamento dos autores na identificação e classificação dos gargalos. Para minimizar a subjetividade, foram adotados critérios explícitos: uma estratégia é considerada indicativa de degradação quando sua taxa de erro global ultrapassa 10%, enquanto uma interação é considerada problemática quando sua latência p95 excede em mais de cinco vezes a mediana das demais interações do mesmo perfil. A natureza do gargalo é então classificada a partir das métricas, logs e informações de infraestrutura disponíveis em cada nível de monitoramento. Quanto ao "custo experimental", os tempos de preparação foram registrados por cronômetro durante a execução real, incluindo consulta à documentação e depuração, o que introduz variabilidade pessoal. Todos os critérios e dados brutos estão documentados no repositório de artefatos para viabilizar replicação independente.

Validade de conclusão. Cada configuração foi executada dez vezes, resultando em 60 execuções. Além das estatísticas descritivas, foram utilizados testes inferenciais e medidas de tamanho de efeito. Para WIPS, os pressupostos de normalidade e homogeneidade das variâncias foram avaliados antes da aplicação da ANOVA fatorial. Para métricas que não satisfizeram esses pressupostos, foram empregados testes não paramétricos de Kruskal–Wallis e Mann–Whitney, acompanhados por ε^2 e delta de Cliff. Comparações múltiplas tiveram seus valores de p ajustados pelo método de Holm. Apesar disso,

o tamanho de dez repetições por configuração limita a capacidade de detectar efeitos sutis. Adicionalmente, métricas de latência próximas ao *timeout* de 60 segundos apresentam efeito de teto e foram interpretadas em conjunto com *throughput* e taxa de erro, evitando atribuir relevância prática apenas à significância estatística.

6 Conclusão e Trabalhos Futuros

Este trabalho comparou seis estratégias de teste de desempenho aplicadas a uma aplicação web de comércio eletrônico, variando o perfil de *workload* e o nível de monitoramento. Os resultados de 60 execuções conduziram a três achados principais.

A análise inferencial reforçou esses achados ao identificar forte interação entre perfil de *workload* e nível de monitoramento sobre o *throughput* ($F(2, 54) = 262{,}33$, $p < 0{,}001$, $\eta_p^2 = 0{,}907$). Enquanto o monitoramento apresentou efeito pequeno e não significativo no perfil *browsing* ($\delta = 0{,}24$), os perfis transacionais apresentaram efeitos máximos ($\delta = 1{,}00$), evidenciando que o impacto observado da instrumentação depende das condições de carga exercitadas.

Primeiro, no cenário experimental avaliado, o perfil de *workload* exerceu forte influência sobre a qualidade do diagnóstico. Estratégias que exercitam apenas operações de leitura indicaram um sistema saudável, enquanto a introdução de operações transacionais revelou um gargalo crítico na interação *Buy Confirm*, causado por um *bug* de concorrência no padrão `SELECT MAX(id)+1/INSERT` dos *servlets*. Segundo, o monitoramento estendido agrega valor concreto ao permitir identificar essa causa raiz por meio dos *slow query logs*. Nos perfis transacionais, sua utilização esteve associada a uma redução de aproximadamente 50% no throughput, efeito que deve ser interpretado com cautela devido à possível influência da ordem de execução. Terceiro, no cenário avaliado, a combinação de monitoramento básico com pelo menos dois perfis de *workload* distintos apresentou o melhor compromisso entre esforço de preparação e informação diagnóstica, detectando o gargalo mais severo com 90 minutos de preparação.

Como trabalhos futuros, sugere-se replicar o *quasi*-experimento com outras aplicações web para verificar a generalização dos achados, corrigir o *bug* de concorrência identificado e reavaliar o desempenho da aplicação após a correção, e ampliar a comparação para outras ferramentas de teste de desempenho. Adicionalmente, pretende-se repetir as configurações com ordem de execução randomizada ou contrabalanceada, de modo a isolar com maior precisão o efeito da instrumentação sobre o desempenho observado.

Disponibilidade de Artefatos

Os *scripts* k6, configurações de monitoramento, dados brutos de 60 execuções, *logs* e documentação estão disponíveis em repositório https://github.com/moraeseriic/teste-api-rest. Complementarmente, disponibilizamos os arquivos e demais dados de análise e orientações do repositório no repositório aberto do Zenodo sob o DOI https://doi.org/10.5281/zenodo.21879528.

Agradecimentos

Os autores agradecem à UNIPAMPA pelo apoio durante o desenvolvimento deste trabalho. Maicon Bernardino é financiado pelo CNPq (Bolsa #307969/2026-6).

Referências

[1] Apache Software Foundation. 2025. Apache JMeter. https://jmeter.apache.org/ Accessed: 2026-06-01.

[2] Andrea Arcuri, Robert Kogler, Christian Schwartz, and Man Zhang. 2025. Technology Adoption Performance Evaluation Applied to Testing Industrial REST APIs. *Automated Software Engineering* 32, 1 (2025), 5. doi:10.1007/s10515-024-00477-2

[3] Daniel F. Garcia and Javier Garcia. 2003. Throughput Analysis of E-commerce Servers Using the TPC-W Benchmark. In *Proceedings of the 12th International Conference on World Wide Web (WWW)*. ACM, 255–256.

[4] Amid Golmohammadi, Man Zhang, and Andrea Arcuri. 2023. Testing RESTful APIs: A Survey. *ACM Transactions on Software Engineering and Methodology* 33, 1, Article 27 (2023), 41 pages. doi:10.1145/3617175

[5] Grafana Labs. 2025. k6 Documentation. https://grafana.com/docs/k6/latest/ Accessed: 2026-06-01.

[6] Myeongsoo Kim, Saurabh Sinha, and Alessandro Orso. 2023. Adaptive REST API Testing with Reinforcement Learning. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 446–458. doi:10.1109/ASE56229.2023.00116

[7] Shin-Jie Lee, Chien-Hsun Kuo, and Jia-Ying Chen. 2024. Revealing Inputs Causing Web API Performance Latency Using Response-Time-Guided Genetic Algorithm Fuzzing. *Artificial Life and Robotics* 29, 4 (2024), 459–472. doi:10.1007/s10015-024-00957-4

[8] Daniel A. Menascé. 2002. TPC-W: A Benchmark for E-Commerce. In *IEEE Internet Computing*, Vol. 6. IEEE, 83–87.

[9] Deepti Mishra, Aditi Sharan, and Shailja Marvi. 2022. RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions. *Applied Sciences* 12, 9 (2022), 4369. doi:10.3390/app12094369

[10] Ian Molyneaux. 2014. *The Art of Application Performance Testing* (2 ed.). O'Reilly Media.

[11] Prometheus Authors. 2025. Prometheus – Monitoring System and Time Series Database. https://prometheus.io/ Accessed: 2026-06-01.