REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

ESQUEMA DE PROCESSAMENTO ASSÍNCRONO COM OTIMIZAÇÃO DE BANCO DE DADOS PARA MITIGAÇÃO DE GARGALOS EM SISTEMAS DISTRIBUÍDOS EM ALTA DEMANDA

ASYNCHRONOUS PROCESSING SCHEME WITH DATABASE OPTIMIZATION FOR BOTTLENECK MITIGATION IN HIGH-DEMAND DISTRIBUTED SYSTEMS

ESQUEMA DE PROCESAMIENTO ASÍNCRONO CON OPTIMIZACIÓN DE BASE DE DATOS PARA LA MITIGACIÓN DE CUELLOS DE BOTELLA EN SISTEMAS DISTRIBUIDOS DE ALTA DEMANDA

Ivan de Jesus Coelho Correa Junior
Graduando em Engenharia de Software, Universidade do Estado do Pará, Brasil
E-mail: ivan.djcc.junior@aluno.uepa.br

Kauan da Silva Pacheco
Graduando em Engenharia de Software, Universidade do Estado do Pará, Brasil
E-mail: kauan.pacheco@aluno.uepa.br

Estêvão Damasceno Santos
Doutor em Ciência da Computação, Universidade do Estado do Pará, Brasil
E-mail: estevaosantos@uepa.br

Resumo
O gerenciamento de picos sazonais de tráfego em sistemas acadêmicos, como os períodos de rematrícula, exige arquiteturas que garantam alta disponibilidade e baixa latência. Este trabalho investiga a sinergia entre o processamento assíncrono via protocolo AMQP e técnicas de Otimização de Banco de Dados (OBD). A metodologia fundamentou-se em testes de estresse comparativos com 1.000 usuários simultâneos em ambiente conteinerizado. Os resultados revelam que o esquema Assíncrono + OBD superou o modelo síncrono tradicional, apresentando um aumento de 52,94% na vazão (RPS) e uma redução de 58,8% na latência mediana. A análise estatística via Intervalo Interquartil (IQR) demonstrou uma estabilidade superior (2,67 ms vs. 101,71 ms do modelo síncrono), enquanto a taxa de sucesso atingiu 99,0%, evidenciando que a integridade transacional é mantida mesmo sob estresse físico do hardware. Conclui-se que o desacoplamento temporal associado à otimização da persistência é uma estratégia eficaz para assegurar a continuidade operacional e a previsibilidade em infraestruturas de TI com recursos limitados.
Palavras-chave: Sistemas Distribuídos; Processamento Assíncrono; Otimização de Banco de Dados; Latência de Rede; RabbitMQ.

Abstract
Managing seasonal traffic peaks in academic systems, such as enrollment periods, requires architectures that ensure high availability and low latency. This study investigates the synergy between asynchronous processing via the AMQP protocol and Database Optimization (DBO) techniques. The methodology was based on comparative stress tests with 1,000 concurrent users in a containerized environment. The results reveal that the Asynchronous + DBO scheme outperformed the traditional synchronous model, showing a 52.94% increase in throughput (RPS) and a 58.8% reduction in median latency. Statistical analysis via the Interquartile Range (IQR) demonstrated superior stability (2.67 ms vs. 101.71 ms for the synchronous model), while the success rate reached 99.0%, showing that transactional integrity is maintained even under physical hardware stress. In

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

conclusion, temporal decoupling associated with persistence optimization is an effective strategy to ensure operational continuity and predictability in IT infrastructures with limited resources.
Keywords: Distributed Systems; Asynchronous Processing; Database Optimization; Network Latency; RabbitMQ.
Resumen
La gestión de picos estacionales de tráfico en sistemas académicos, como los periodos de matriculación, exige arquitecturas que garanticen una alta disponibilidad y baja latencia. Este trabajo investiga la sinergia entre el procesamiento asíncrono mediante el protocolo AMQP y técnicas de Optimización de Base de Datos (OBD). La metodología se fundamentó en pruebas de estrés comparativas con 1.000 usuarios simultáneos en un entorno contenedorizado. Los resultados revelan que el esquema Asíncrono + OBD superó al modelo síncrono tradicional, presentando un aumento del 52,94% en el rendimiento (RPS) y una reducción del 58,8% en la latencia mediana. El análisis estadístico mediante el Rango Intercuartílico (IQR) demostró una estabilidad superior (2,67 ms frente a los 101,71 ms del modelo síncrono), mientras que la tasa de éxito alcanzó el 99,0%, evidenciando que la integridad transaccional se mantiene incluso bajo estrés físico del hardware. Se concluye que el desacoplamiento temporal asociado a la optimización de la persistencia es una estrategia eficaz para asegurar la continuidad operativa y la previsibilidad en infraestructuras de TI con recursos limitados.
Palabras clave: Sistemas Distribuidos; Procesamiento Asíncrono; Optimización de Base de Datos; Latencia de Red; RabbitMQ.

1. Introdução

A transformação digital em instituições de ensino superior tem imposto desafios significativos à infraestrutura de Tecnologia da Informação (TI), especialmente em eventos sazonais de alta demanda, como os períodos de rematrícula acadêmica. Segundo Popovic (2025), a engenharia de *performance* em sistemas baseados em microsserviços exige que a arquitetura seja capaz de absorver picos de tráfego sem degradação linear da experiência do usuário. No entanto, o modelo tradicional de comunicação síncrona (*Request-Response*) frequentemente enfrenta o esgotamento de conexões *Transmission Control Protocol* (TCP) e o bloqueio de *threads* de entrada e saída (I/O) quando submetido a cargas que excedem a capacidade nominal de escrita dos bancos de dados relacionais.

Embora existam diferentes estratégias de comunicação em sistemas distribuídos, como chamadas de procedimento remoto (*gRPC*) ou fluxos bidirecionais (*WebSockets*), estas ainda mantêm um acoplamento temporal que pode ser prejudicial em cenários de saturação de persistência. Nesse contexto, a adoção de mecanismos de processamento assíncrono via *Advanced Message Queuing Protocol* (AMQP) surge não apenas como uma alternativa de escalabilidade, mas como uma estratégia fundamental de gerência de redes para

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

mitigar gargalos de I/O. A utilização de *message brokers* permite o desacoplamento temporal, funcionando como um amortecedor de tráfego (*backpressure*), garantindo que a volatilidade da carga não interrompa a disponibilidade do serviço. Contudo, é imperativo destacar que a transição para modelos assíncronos altera a semântica de confirmação: enquanto o modelo síncrono confirma a persistência final ao cliente (HTTP 200), o modelo assíncrono confirma apenas o aceite da mensagem para processamento posterior (HTTP 202), o que exige uma análise criteriosa sobre a equivalência funcional dos benchmarks realizados.

A contribuição central deste trabalho reside na investigação da sinergia entre o desacoplamento por filas e a Otimização de Banco de Dados (OBD). A literatura muitas vezes trata esses temas de forma isolada, mas Shyam Mohan e Goswami (2025) ressaltam que a *performance* de aplicações *Spring Boot* é intrinsecamente ligada à configuração dos *drivers* de conexão e ao comportamento dos *commits* no banco de dados. Conforme discutido por Santos et al. (2021), a utilização de *buffers* estabiliza o tempo de resposta percebido, mas a eficiência fim-a-fim depende de ajustes de baixo nível na camada de dados para evitar que o gargalo seja meramente deslocado da aplicação para a persistência. Portanto, este estudo propõe uma abordagem integrada para maximizar o rendimento (*throughput*) em ambientes de recursos finitos, onde a escalabilidade horizontal pode ser economicamente inviável.

O objetivo deste artigo é avaliar o impacto dessa arquitetura integrada na responsividade e capacidade de vazão sob estresse severo, utilizando testes de carga com o Grafana k6 para 1.000 usuários simultâneos. Para conferir rigor à análise, define-se como desfecho primário do experimento a latência total fim-a-fim, parâmetro essencial para mensurar a manutenção da responsividade em serviços críticos de rematrícula. Como desfechos secundários, são avaliados o rendimento em requisições por segundo (RPS) e a taxa de sucesso de processamento. Esta estrutura analítica permite discutir formalmente os *trade-offs* entre desempenho,

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

durabilidade e integridade transacional, evitando interpretações excessivamente parciais sobre a superioridade de um modelo sobre o outro.

A justificativa deste estudo reside na necessidade de prover soluções de alta eficiência para serviços acadêmicos, posicionando a gerência de redes e a otimização de *software* como pilares para a continuidade operacional em cenários de estresse severo. O artigo está organizado da seguinte forma: a Seção 2 apresenta a revisão de literatura; a Seção 3 detalha a metodologia experimental e as políticas de confirmação; a Seção 4 apresenta os resultados e a discussão dos compromissos operacionais; e a Seção 5 sintetiza as conclusões e as limitações do escopo testado.

2. Revisão da Literatura

A eficiência de sistemas distribuídos sob regimes de alta carga é um desafio multifatorial que exige harmonia entre a arquitetura de *software* e a infraestrutura de rede. Popovic (2025) argumenta que a engenharia de *performance* deve ser integrada desde a concepção de microsserviços para evitar latências em cascata que comprometam a disponibilidade do ecossistema. Essa necessidade de robustez é acentuada por Blinowski, Ojdak e Przybyłek (2022), que evidenciam que a fragmentação de serviços introduz sobrecarga de rede e estados efêmeros, exigindo estratégias de comunicação que minimizem o bloqueio prolongado de recursos sistêmicos.

2.1 Comunicação Assíncrona e Gestão de Filas via AMQP

A transição de modelos síncronos para o processamento assíncrono é amplamente documentada como uma técnica eficaz para mitigar a contenção em sistemas *web* de alta concorrência. Ishankhodjaev et al. (2024) demonstram que a programação assíncrona reduz o tempo de bloqueio de *threads*, permitindo que a aplicação suporte uma maior densidade de requisições simultâneas sem a exaustão imediata de recursos do servidor. No que tange ao ecossistema *Spring Boot*, a escolha do *middleware* de mensageria é determinante para a resiliência; Kamiński, Klonica e Pańczyk (2025) destacam a adequação do RabbitMQ para

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

cenários que exigem roteamento granular e garantias de entrega via protocolo AMQP.

Neste contexto, a implementação de padrões de *backpressure* torna-se vital para evitar a degradação do serviço. Ricardo (2023) enfatiza que, sem o devido controle de fluxo nas filas, o sistema incorre em risco elevado de saturação de memória. Santos et al. (2021) utilizam modelos de teoria de filas contemporâneos para provar que a introdução de um *buffer* estabiliza o tempo de resposta percebido, transformando picos de tráfego voláteis em uma carga constante e tratável para a persistência. Todavia, a literatura recente indica que o isolamento do estudo das filas em relação ao banco de dados pode ocultar gargalos onde a aplicação parece performática, mas a persistência final sofre atrasos críticos.

2.2 Engenharia de Performance e Otimização de Banco de Dados

Apesar dos benefícios do desacoplamento temporal, a camada de persistência permanece como o gargalo final em sistemas de alta demanda. Salunke e Ouda (2024) estabelecem *benchmarks* para PostgreSQL, revelando que a contenção de recursos pode ser mitigada através de ajustes finos na configuração do servidor, embora isso muitas vezes envolva um *trade-off* com as garantias de durabilidade imediata. A eficiência no uso de CPU, I/O e memória em instâncias PostgreSQL está intrinsecamente ligada à gestão de conexões; Nasser e Jaber (2025) propõem que o gerenciamento rigoroso de *connection pools* impede que a persistência se torne um ponto único de falha.

A sinergia entre o fluxo assíncrono e a base de dados otimizada é o que permite a sistemas limitados operarem sob estresse severo. Shyam Mohan e Goswami (2025) ressaltam que o comportamento dos *commits* no banco de dados define o rendimento real da aplicação. Ao adotar configurações como o *asynchronous commit*, o sistema prioriza a vazão de dados, assumindo um risco controlado de perda de transações em janelas de milissegundos em caso de falha catastrófica — uma escolha semântica que deve ser alinhada aos requisitos de negócio do serviço.

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

2.3 Virtualização, Monitorização e Benchmarking de Redes

A validação de arquiteturas distribuídas exige um ambiente experimental controlado e ferramentas de observabilidade precisas. Bandaru (2022) ressalta que o uso de Docker facilita a implementação de microsserviços escaláveis, mas introduz camadas de virtualização que podem camuflar gargalos de rede. Para identificar estas falhas, Silva et al. (2024) e Linhares et al. (2025) discutem a importância de correlacionar dados de monitorização para identificar causas raiz de problemas de desempenho, utilizando técnicas de regressão para estimar a vazão de rede de forma proativa.

A escolha de métricas de *benchmark* é fundamental para a credibilidade dos resultados. Maharjan et al. (2023) e Laigner et al. (2024) sugerem que o desempenho de filas e bases de dados deve ser avaliado sob condições de saturação extrema para mapear os limites físicos da infraestrutura. Vaz e Guardia (2025) reforçam que arquiteturas de monitorização resilientes são essenciais em ambientes de nuvem e computação distribuída, garantindo que a análise de métricas como latência e taxa de sucesso reflita fielmente o comportamento do sistema sob carga (GOMES et al., 2025).

2.4. Engenharia de Performance e Benchmarking com Grafana k6

A utilização de ferramentas de teste de carga modernas, como o Grafana k6, permite a simulação de usuários virtuais (VUs) com baixo consumo de recursos no lado do cliente, o que é essencial para garantir que os resultados reflitam o desempenho do servidor e não limitações da ferramenta de teste. Conforme destacado por Pinyagin e Sadovykh (2026) em sua avaliação de ferramentas para pipelines de CI/CD, o k6 sobressai pela sua arquitetura baseada em Go, que oferece uma eficiência superior em comparação com ferramentas tradicionais baseadas em Java ou Python, facilitando a mensuração de métricas granulares de rede, como latência e throughput, sob condições de alta concorrência. Complementarmente, Stępień e Skublewska-Paszkowska (2025) demonstra a eficácia do k6 ao simular cargas de usuários simultâneos para avaliar o

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

desempenho de APIs REST e GraphQL integradas ao PostgreSQL, reforçando a confiabilidade da ferramenta para validar a escalabilidade de sistemas distribuídos.

Diferente de ferramentas tradicionais, o k6 permite a coleta de indicadores precisos como o tempo de bloqueio de rede, tempo de conexão TCP e latência de espera (TTFB), que são vitais para diagnosticar a eficiência de arquiteturas assíncronas (ALMEIDA, 2025). Conforme discutido por Silva et al. (2024), o uso de ferramentas de benchmarking de alto desempenho é um pré-requisito para validar a resiliência de microsserviços, garantindo que a telemetria reflita fielmente a capacidade de vazão do sistema sob condições de saturação extrema.

2.5. Virtualização com Docker e Isolamento de Recursos

A utilização de Docker e Docker Compose é fundamental para garantir que as limitações de hardware, como CPU e RAM, sejam aplicadas de forma equânime a todos os cenários testados. Isso confere rigor científico ao estudo, assegurando que os ganhos de performance observados derivam da arquitetura proposta e não de oscilações do sistema operacional host. De acordo com o estudo de Sobieraj e Kotyński (2024) sobre a avaliação de performance do Docker em diferentes sistemas operacionais, o isolamento proporcionado por mecanismos de namespaces e cgroups no kernel Linux minimiza a sobrecarga de virtualização, embora a camada adicional de abstração possa introduzir latências marginais em operações de I/O intenso.

O uso de contêineres permite, portanto, a criação de um ambiente experimental reproduzível e isolado, essencial para o benchmarking de microsserviços e a mitigação de variáveis externas que poderiam comprometer a validade dos dados coletados (LIMA, 2026). Conforme demonstrado por Campos (2025), a orquestração via Docker Compose facilita a replicação de topologias complexas em ambientes controlados (testbeds), permitindo que infraestruturas limitadas simulem o comportamento de clusters de produção com alto grau de fidelidade e isolamento de recursos.

2.6. Análise Comparativa e Lacunas na Literatura

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

A literatura atual frequentemente aborda a mensageria ou a otimização de bancos de dados de forma isolada. No entanto, a integração sistêmica dessas frentes, combinada com o monitoramento ativo, representa uma lacuna significativa que este trabalho busca preencher. Enquanto estudos focam no roteamento granular de mensagens ou na eficiência de drivers de conexão específicos, poucos exploram a sinergia entre o desacoplamento temporal via AMQP e o ajuste fino de asynchronous commits em ambientes de alta demanda sazonal.

A originalidade deste trabalho reside na integração sistêmica dessas duas frentes, combinando o processamento assíncrono com a Otimização de Banco de Dados (OBD) sob um monitoramento ativo de telemetria. Diferente das soluções parciais encontradas na literatura nacional recente, esta proposta valida a sinergia entre o amortecimento de carga e a eficiência de I/O, preenchendo a lacuna de estratégias integradas para infraestruturas de hardware limitado. Esta visão holística permite não apenas absorver picos de tráfego, mas também otimizar a persistência física, transformando o que seriam falhas catastróficas em uma carga gerenciável para a infraestrutura existente, consolidando uma arquitetura resiliente de ponta a ponta.

3. Metodologia

A avaliação do esquema de processamento assíncrono integrado à Otimização de Banco de Dados (OBD) fundamentou-se na construção de um ambiente experimental controlado, ou *testbed*, baseado em virtualização por contêineres. Esta abordagem metodológica foi adotada para garantir o isolamento estrito dos componentes sistêmicos e viabilizar a reprodutibilidade dos testes, seguindo as diretrizes de Bandaru (2022) sobre a escalabilidade e gestão de sistemas conteinerizados em ambientes corporativos.

3.1 Infraestrutura de Hardware e Configuração do Ambiente

A infraestrutura de hardware consistiu em uma estação de trabalho equipada com processador Intel Core i7 (10ª geração, 8 núcleos físicos a 2,9 GHz), 16 GB de

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

memória RAM DDR4 a 3.200 MHz e unidade de armazenamento SSD NVMe com velocidade de escrita sequencial de até 3.000 MB/s, operando sob *kernel* Linux Ubuntu 22.04 LTS (5.15.x). O sistema operacional *host* foi configurado sem serviços em segundo plano desnecessários, minimizando a interferência de processos externos na medição de desempenho.

Toda a infraestrutura foi orquestrada com Docker 24.x e Docker Compose 2.x. Para assegurar a equanimidade comparativa, limites explícitos de recursos foram configurados via diretivas *deploy.resources.* A Tabela 1 detalha essa alocação, garantindo que os ganhos de performance derivem exclusivamente da arquitetura proposta (SOBIERAJ; KOTYŃSKI, 2024).

Tabela 1: Alocação de recursos dos contêineres por serviço.

Contêiner / Serviço	CPUs Reservadas	CPUs Máximas	RAM Reservada	RAM Máxima
API (Spring Boot)	0,5	2,0	256 MB	512 MB
RabbitMQ 3.12	0,5	1,0	256 MB	512 MB
Serviço de Processamento	0,5	2,0	256 MB	512 MB
PostgreSQL 15	0,5	2,0	256 MB	1 GB

Fonte: os autores (2026).

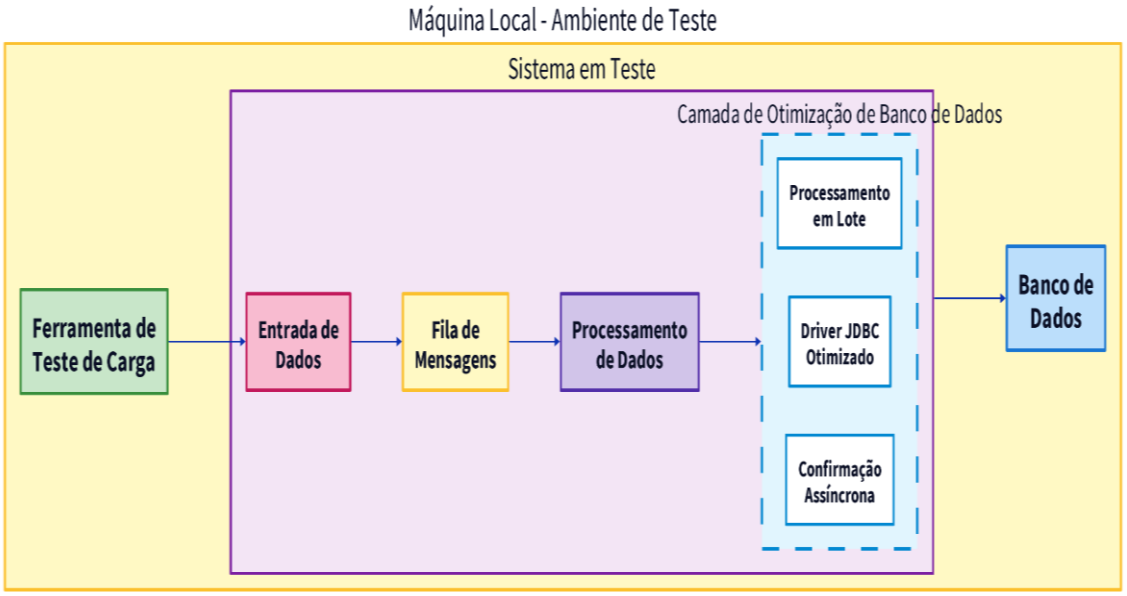
Os contêineres foram interconectados em uma rede virtual interna isolada (*bridge network* nomeada *reenrollment-net*), criada exclusivamente para este experimento com o intuito de eliminar interferências de tráfego externo. Essa rede permitiu o monitoramento preciso do tráfego e a coleta de métricas de telemetria em tempo real, atendendo à necessidade de correlação de dados discutida por Silva et al. (2024).

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

A organização estrutural deste *testbed* e o fluxo de interação entre os componentes estão detalhados na Figura 1. Como ilustrado, a topologia da arquitetura foi desenhada para representar um ecossistema de microsserviços modernos, onde cada responsabilidade funcional reside em uma unidade isolada. O arranjo inclui um gerador de carga baseado em Grafana k6, uma interface de programação de aplicações (API) desenvolvida com Spring Boot 3.x, um *message broker* RabbitMQ 3.12 utilizando o protocolo AMQP, e uma camada de persistência com PostgreSQL 15.

Figura 1: Topologia da infraestrutura experimental e fluxo de dados.



Fonte: os autores (2026).

3.2 Descrição da Aplicação Simulada

Para simular de forma fidedigna o cenário de rematrícula acadêmica, foi desenvolvida uma aplicação de microsserviços que replica as operações essenciais desse processo. A aplicação é composta por dois serviços principais: (i) a API de Entrada, responsável por receber as solicitações de rematrícula dos estudantes e

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

(ii) o Serviço de Processamento, responsável por consumir as mensagens da fila e persistir os dados no banco.

Cada requisição simulada representa uma solicitação de rematrícula completa e consiste em um HTTP POST para o *endpoint /api/v1/enrollment* com um *payload* JSON contendo os campos: *studentId* (UUID do estudante), *courseIds* (lista com até 8 identificadores de disciplinas), *semester* (período letivo no formato "2026.1") e *requestedAt* (*timestamp* ISO-8601 da solicitação). O tamanho médio de cada *payload* é de aproximadamente 280 bytes.

O banco de dados PostgreSQL 15 foi pré-populado com 10.000 registros de estudantes e 200 disciplinas, criando um cenário realista de disputa por recursos. A tabela principal *enrollments* possui os campos *id* (*BIGSERIAL*, chave primária), *student_id* (UUID, indexado), *course_id* (UUID, indexado), *semester* (VARCHAR), *status* (ENUM: PENDING / CONFIRMED) e *created_at* (TIMESTAMPTZ). Cada requisição gera entre 1 e 8 inserções nessa tabela, dependendo do número de disciplinas no *payload*, resultando em uma média de 4,5 operações de escrita por requisição.

3.3 Configuração do Protocolo de Testes e Cenários Comparativos

Para isolar o impacto incremental de cada técnica e atender às recomendações de rigor de Laigner et al. (2024), a metodologia foi executada em três cenários comparativos distintos, seguindo quatro etapas sistemáticas:

1. **Preparação e Baseline (Cenário Síncrono):** Implementou-se o modelo bloqueante, onde a API aguarda a confirmação da escrita física no banco de dados antes de responder ao cliente com HTTP 200. Não há intermediário de mensageria; a requisição percorre toda a pilha de forma síncrona.

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

2. **Introdução do Desacoplamento (Cenário Assíncrono):** Inseriu-se o RabbitMQ 3.12 via protocolo AMQP para desacoplamento temporal. A API recebe a requisição, publica a mensagem na fila *enrollment.queue* e retorna imediatamente ao cliente com HTTP 202 Accepted. O Serviço de Processamento consome a fila e persiste os dados com as configurações padrão do PostgreSQL.

3. **Aplicação da Otimização de Dados (Cenário Assíncrono + OBD):** Integrou-se ao cenário anterior um conjunto de otimizações na camada de persistência: (a) *Batch Processing* agrupamento de 50 inserções por transação via Spring Data JPA; (b) *Driver JDBC otimizado* propriedade *rewriteBatchedInserts=true* habilitada no driver PostgreSQL JDBC 42.x, minimizando *round-trips* ao banco; (c) *Asynchronous Commit* parâmetro *synchronous_commit=off* configurado no PostgreSQL, permitindo que o banco confirme transações antes da gravação física no WAL (*Write-Ahead Log*); (d) *Connection Pool* via HikariCP com *maximumPoolSize=20* e *connectionTimeout=3000ms*.

4. **Execução do Estresse:** O protocolo de carga foi aplicado identicamente aos três cenários através do Grafana k6, assegurando a equanimidade nas condições de teste. Cada cenário foi executado três vezes consecutivas, com um intervalo rigoroso de 5 minutos entre as rodadas simular o resfriamento da infraestrutura e a limpeza de caches do sistema operacional e do banco de dados, sendo os resultados apresentados como a mediana entre as três execuções para mitigar a influência de anomalias estatísticas. Além disso, monitorou-se a estabilidade do ambiente conteinerizado durante os períodos de repouso para garantir que cada nova bateria de testes partisse de um estado de consumo de recursos equivalente, preservando a integridade das comparações de desempenho.

3.4 Script de Carga e Parâmetros do Grafana k6

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

O script de teste foi desenvolvido em JavaScript/ES6 e executado diretamente no *host* (fora dos contêineres), garantindo que o overhead da ferramenta não interferisse nas métricas do servidor. A carga foi estruturada em três estágios, conforme o perfil de acesso típico de um sistema de rematrícula:

Tabela 2: Estágios do protocolo de teste de carga (Grafana k6).

Estágio	Duração	Usuários Virtuais (VUs)	Descrição
Ramp-up	2 minutos	0 → 1.000	Subida gradual até a carga máxima
Sustentado	8 minutos	1.000 (constante)	Estresse máximo contínuo
Ramp-down	30 segundos	1.000 → 0	Encerramento controlado

Fonte: os autores (2026).

A volumetria de 1.000 VUs foi definida com base em dados de pico de sistemas de rematrícula de universidades estaduais brasileiras de médio porte, onde registros históricos apontam para 800 a 1.200 acessos simultâneos na abertura do período letivo (FERREIRA et al., 2022). Cada VU executa iterações contínuas de HTTP POST sem *think time* (intervalo entre requisições igual a zero), maximizando a pressão sobre a camada de persistência. O *timeout* por requisição foi configurado em 30 segundos; requisições não respondidas nesse prazo foram contabilizadas como falhas na taxa de sucesso.

As métricas coletadas pelo k6 incluíram: (a) RPS (*Requests per Second*) capacidade de processamento agregada; (b) latência total (*http_req_duration*) tempo fim-a-fim da requisição; (c) tempo de bloqueio (*http_req_blocked*) tempo de espera na fila local do cliente; (d) tempo de conexão TCP (*http_req_connecting*); (e) tempo de envio (*http_req_sending*); (f) tempo de espera pelo servidor TTFB

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

(*http_req_waiting*); (g) tempo de recebimento (*http_req_receiving*); e (h) taxa de sucesso HTTP — percentual de respostas com código 2xx.

3.5 Monitoramento em Tempo Real e Validação do Ambiente

Durante a execução dos testes, o consumo de CPU, memória e I/O de cada contêiner foi monitorado em tempo real via *docker stats* com intervalo de amostragem de 1 segundo. Esse monitoramento foi fundamental para correlacionar os picos de latência com a saturação de recursos específicos especialmente o I/O Wait da CPU e a fila de escrita do SSD NVMe confirmando que a degradação na taxa de sucesso do modelo assíncrono decorre da saturação física do dispositivo de armazenamento e não de limitações de rede ou de lógica de aplicação.

Para garantir a validade estatística, foram descartados os primeiros 30 segundos de dados de cada execução (período de aquecimento dos contêineres JVM). Os resultados reportados referem-se à fase sustentada de 8 minutos a 1.000 VUs. A ausência de testes em infraestrutura de nuvem distribuída constitui uma limitação reconhecida do estudo, detalhada na Seção 5.

3.6 Funcionamento Detalhado do Modelo Proposto (Assíncrono + OBD)

O modelo Assíncrono + OBD opera através da sinergia entre dois mecanismos complementares, conforme ilustrado na Figura 1. O fluxo de uma requisição segue as seguintes etapas:

1. **Recepção desacoplada:** A API (Spring Boot 3.x, Java 17) recebe a requisição HTTP POST, valida o *payload*, serializa a mensagem em formato JSON e a publica na fila AMQP *enrollment.queue* no RabbitMQ. A thread HTTP retorna imediatamente ao cliente com HTTP 202 Accepted, liberando o recurso de conexão em menos de 1 ms sem aguardar qualquer operação de I/O em disco.

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

2. **Amortecimento de carga:** O RabbitMQ atua como regulador de fluxo inteligente (*backpressure*). Enquanto a API aceita requisições na velocidade máxima da rede, a fila acumula as mensagens e as entrega ao Serviço de Processamento conforme a capacidade de escrita do *hardware*, evitando a sobrecarga catastrófica do banco de dados.

3. **Persistência otimizada:** O Serviço de Processamento consome mensagens da fila em lotes de 50 unidades. Cada lote é persistido em uma única transação JDBC, reescrita internamente pelo *driver* como um *multi-row INSERT* (*rewriteBatchedInserts=true*), reduzindo de 50 para 1 a quantidade de *round-trips* ao banco por ciclo. O *asynchronous commit* elimina a espera pela confirmação física do WAL, aumentando a taxa de gravação em memória antes da sincronização com disco.

4. **Telemetria ativa:** As métricas de latência granular coletadas pelo k6 (bloqueio, conexão, envio, espera e recebimento) permitem identificar, com precisão de milissegundos, em qual camada do fluxo ocorre cada gargalo. O monitoramento via *docker stats* complementa essa visão com os dados de consumo de CPU e I/O do *host*, permitindo correlacionar o comportamento do sistema com os limites físicos da infraestrutura.

5. **Semântica de confirmação do pipeline:** O Serviço de Processamento opera com confirmação automática de mensagem (*auto-acknowledgment*), configuração na qual o RabbitMQ remove a mensagem da fila imediatamente após sua entrega ao consumidor, independentemente do resultado da operação de persistência subsequente. Essa configuração foi adotada deliberadamente para maximizar o *throughput* do experimento e isolar o comportamento da camada de persistência sob carga máxima. Como consequência direta, falhas na etapa de persistência resultam em descarte silencioso da mensagem, sem reenfileiramento automático. Não foram implementados, no escopo deste experimento, mecanismos de

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

política de retentativa (*retry*) nem fila de mensagens mortas (*Dead Letter Queue* — DLQ). Essa ausência implica que a taxa de sucesso reportada pelo Grafana k6 reflete o aceite HTTP pela camada de API e não a garantia de persistência fim a fim, distinção formalizada na Seção 3.2. A implementação de *manual acknowledgment* com DLQ e estratégia de retentativa com *backoff* exponencial é identificada como requisito essencial para implantação em ambiente de produção, conforme discutido na Seção 5

Ressalta-se que, para isolar os efeitos individuais das técnicas, testes preliminares com o cenário *Síncrono + OBD* (otimizações de banco sem mensageria) indicaram que as otimizações de persistência isoladas não evitam o esgotamento do *pool* de conexões da API pois o gargalo, nesse caso, está no bloqueio da *thread* HTTP aguardando a confirmação do banco, e não na velocidade de escrita em si. Esse resultado evidencia que a sinergia entre o desacoplamento (mensageria) e a OBD é o que garante a estabilidade do sistema, justificando o foco dos três cenários apresentados.

4. Resultados e Discussão

Esta seção apresenta e discute os resultados experimentais obtidos nos testes de estresse, comparando o desempenho do esquema proposto Assíncrono + OBD com os modelos Síncrono e Assíncrono convencional. A análise foca em quantificar os ganhos de eficiência em termos de capacidade de processamento e latência, além de avaliar a estabilidade do sistema sob carga massiva de 1.000 usuários simultâneos.

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

Tabela 3 – Consolidado de Métricas de Desempenho

Cenário	Mediana (ms)	MIn. (ms)	Max. (ms)	IQR (ms)	Q1 (ms)	Q2 (ms)
Assincrono	3,88	0,52	36.331,97	4,32	2,31	6,63
Assincrono + OBD	2,73	0,46	60.000,93	2,67	1,63	4,30
Sincrono	6,63	0,84	60.000,85	101,71	3,69	105,40

Fonte: os autores (2026)

Os resultados experimentais de latência, consolidados na Tabela 2, revelam comportamentos sistêmicos distintos sob regimes de estresse severo. Para uma análise rigorosa, além da tendência central (mediana), foram avaliadas as métricas de dispersão e os limites de quartis, permitindo diagnosticar a estabilidade de cada arquitetura. A utilização da Mediana, em detrimento da média aritmética, justifica-se pela natureza assimétrica dos dados de latência, que frequentemente apresentam valores extremos (*outliers*). Complementarmente, o Intervalo Interquartil (IQR) e os quartis (Q1 e Q3) foram adotados para mensurar a previsibilidade do sistema; um IQR reduzido indica que a maioria das requisições é processada em um tempo consistente, enquanto valores de Q3 elevados denunciam a ocorrência de degradação na "cauda longa" da distribuição (latência de cauda).

4.1. Eficiência da Resposta e Estabilidade Relativa

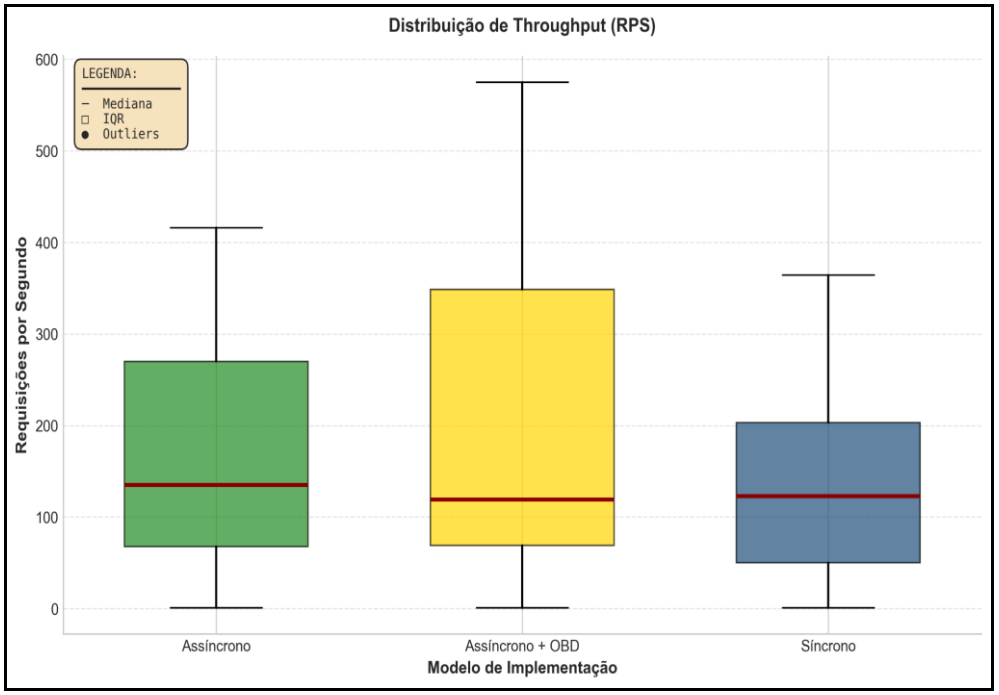
O cenário Assíncrono + OBD demonstrou a maior eficiência de resposta, com uma latência mediana de 2,73 ms, o que representa um desempenho 58,8% superior ao modelo Síncrono (6,63 ms). Contudo, a métrica mais reveladora para a gerência de redes é o Intervalo Interquartil (IQR). Enquanto o modelo Síncrono apresenta um IQR de 101,71 ms, o modelo Assíncrono + OBD mantém um IQR de

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

apenas 2,67 ms. Essa disparidade evidencia que a arquitetura proposta garante uma previsibilidade operacional significativamente maior: 50% das requisições situam-se em uma janela temporal extremamente estreita, ao passo que o modelo síncrono sofre de uma variabilidade massiva, característica de sistemas saturados por bloqueio de *threads*.

Figura 2: Distribuição de Throughput (Requisições por Segundo - RPS).



Fonte: os autores (2026).

A análise do rendimento (Figura 2) corrobora esses ganhos, demonstrando picos de vazão superiores a 550 RPS no esquema Assíncrono + OBD, superando o limite de saturação de 400 RPS do modelo síncrono.

4.2. Análise da "Cauda Longa" e Saturação de I/O

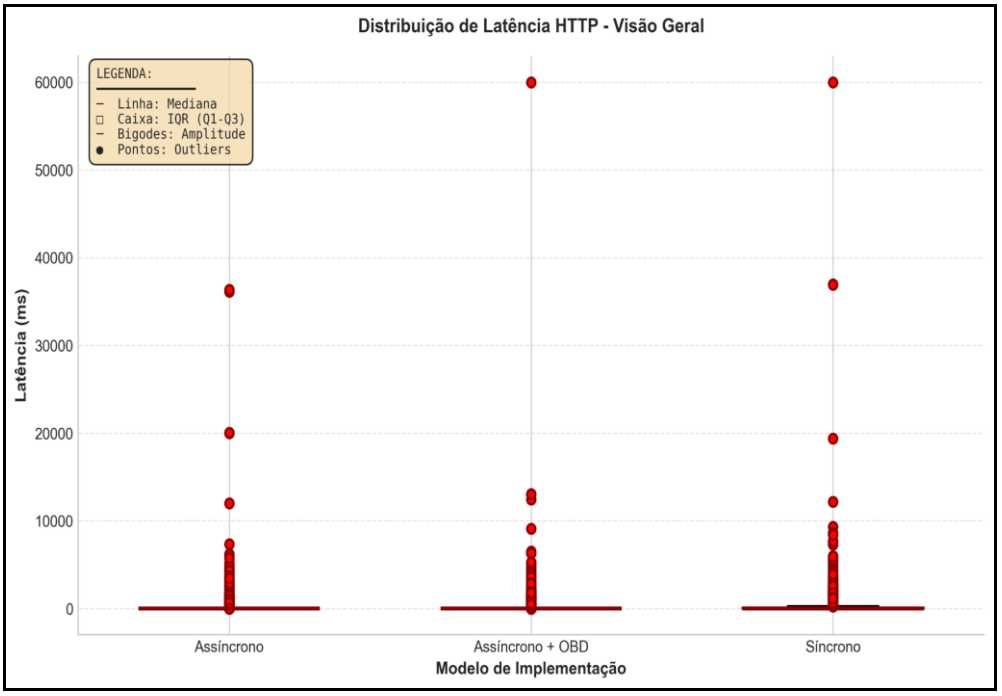
A observação dos valores máximos e do terceiro quartil (Q3) permite compreender o fenómeno da latência de cauda (*long-tail latency*). Ambos os

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

cenários (Síncrono e Assíncrono + OBD) atingiram picos de 60.000 ms, correspondentes ao limite de *timeout*. No entanto, a distribuição destes atrasos é desigual, como evidenciado na comparação entre a latência geral e o comportamento granular da rede.

Figura 3: Distribuição de Latência HTTP e Estabilidade Sistêmica.



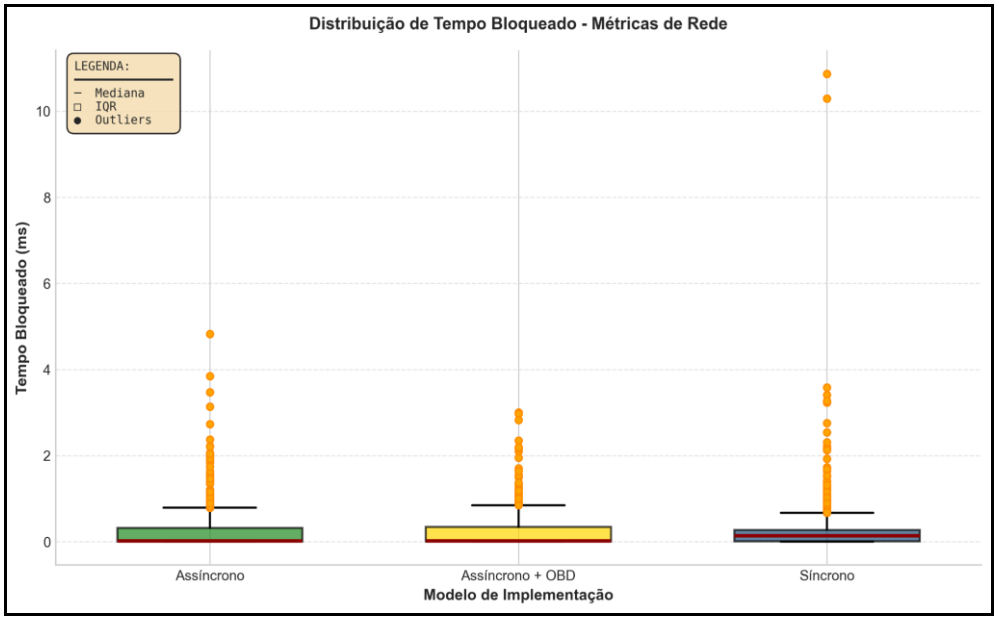
Fonte: os autores (2026).

Conforme ilustrado na Figura 3, o modelo proposto estabilizou a latência mediana em 2,73 ms. No cenário Síncrono, o Q3 de 105,40 ms indica que 25% da carga experimenta latências ordens de grandeza superiores à mediana. Já no cenário Assíncrono + OBD, o Q3 permanece em 4,30 ms, provando que a degradação atinge apenas uma fração residual de utilizadores. Essa consistência é vital para impedir que picos de demanda resultem em lentidão generalizada ou falhas de timeout na borda do sistema. Para diagnosticar a origem desta estabilidade, as Figuras 4 e 5 decompõem o tempo de rede em métricas de bloqueio e conexão.

REMUNOM
ISSN 2178-6925

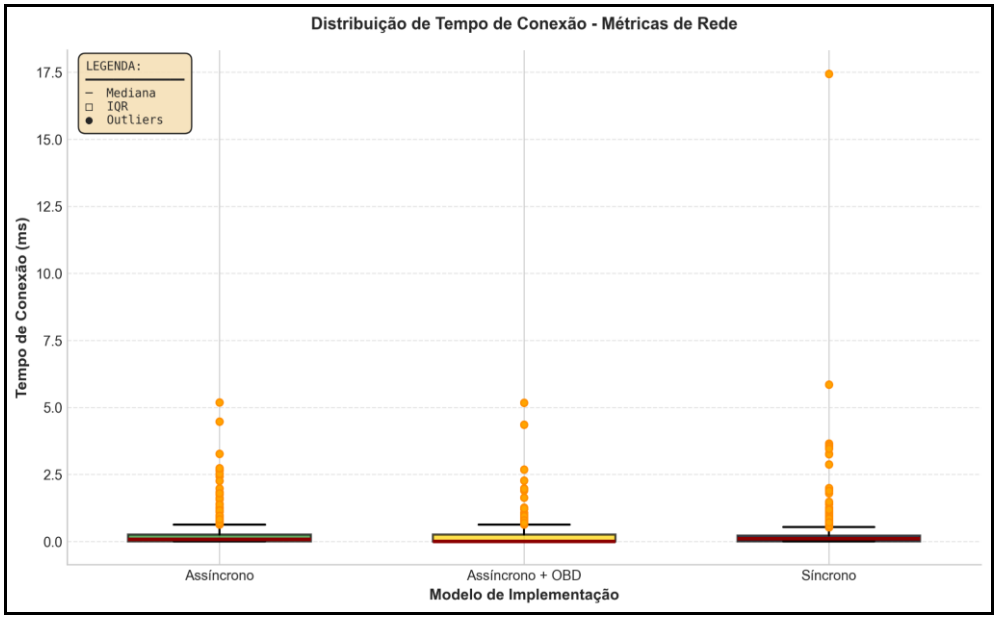
Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

Figura 4: Tempo de Bloqueio de Rede e Eficiência de Threads.



Fonte: os autores (2026).

Figura 5: Distribuição de Tempo de Conexão e Estabilidade de Rede.



Fonte: os autores (2026).

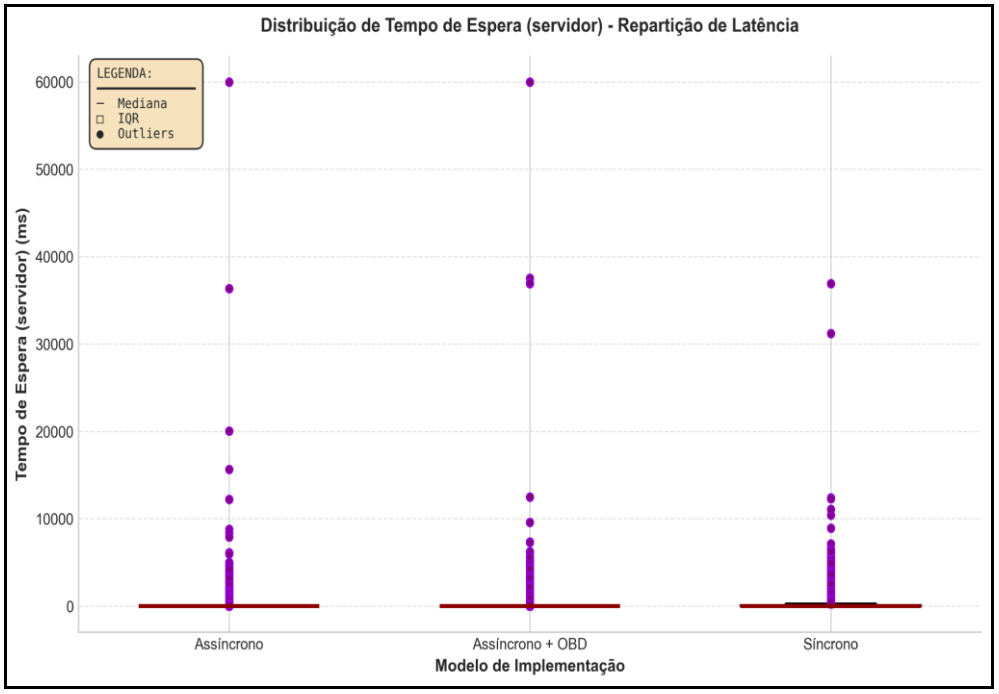
REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

A Figura 4 demonstra que o bloqueio na fila local do cliente é virtualmente eliminado no modelo proposto, pois a API aceita a ligação instantaneamente. Complementarmente, a Figura 5 revela uma gestão de ligações TCP mais eficiente, com um IQR próximo a zero, o que evita o *overhead* de processamento que satura o modelo síncrono.

Esta fluidez na borda permite que o gargalo seja isolado na camada de escrita. A Figura 6 detalha o tempo de espera pelo servidor (TTFB), onde se observa o impacto direto das otimizações de base de dados.

Figura 6: Tempo de Espera do Servidor (TTFB) e Eficiência da Camada OBD.



Fonte: os autores (2026).

Essa "cauda longa" no modelo proposto é tecnicamente associada à saturação física do SSD NVMe monitorada via *%iowait* (Seção 3.5), e não a uma falha na lógica de escalabilidade do *software*. O desacoplamento via AMQP permite

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

que a API mantenha a responsividade para a vasta maioria dos usuários, enquanto o gargalo de persistência é isolado na camada de consumo.

A avaliação está estruturada em métricas granulares, permitindo diagnosticar o comportamento de cada arquitetura em um ambiente de hardware limitado, demonstrando como a integração entre mensageria e otimização da persistência atua na mitigação de gargalos críticos de I/O. Ressalta-se que, para isolar as variáveis, testes preliminares com o cenário *Síncrono + OBD* (sem mensageria) indicaram que as otimizações de banco sozinhas não evitava o esgotamento do *pool* de conexões na API, evidenciando que a sinergia entre o desacoplamento e a OBD é o que garante a estabilidade.

4.3 Dinâmica de Escalonamento e Integridade Transacional

O monitoramento granular da fila *enrollment.queue* via *RabbitMQ Management Plugin* permitiu diagnosticar a alta eficiência do desacoplamento temporal. Durante a fase de estresse sustentado, observou-se um crescimento na profundidade da fila, fenômeno resultante da assimetria proposital entre a taxa de publicação na API e a taxa de consumo no serviço de persistência.

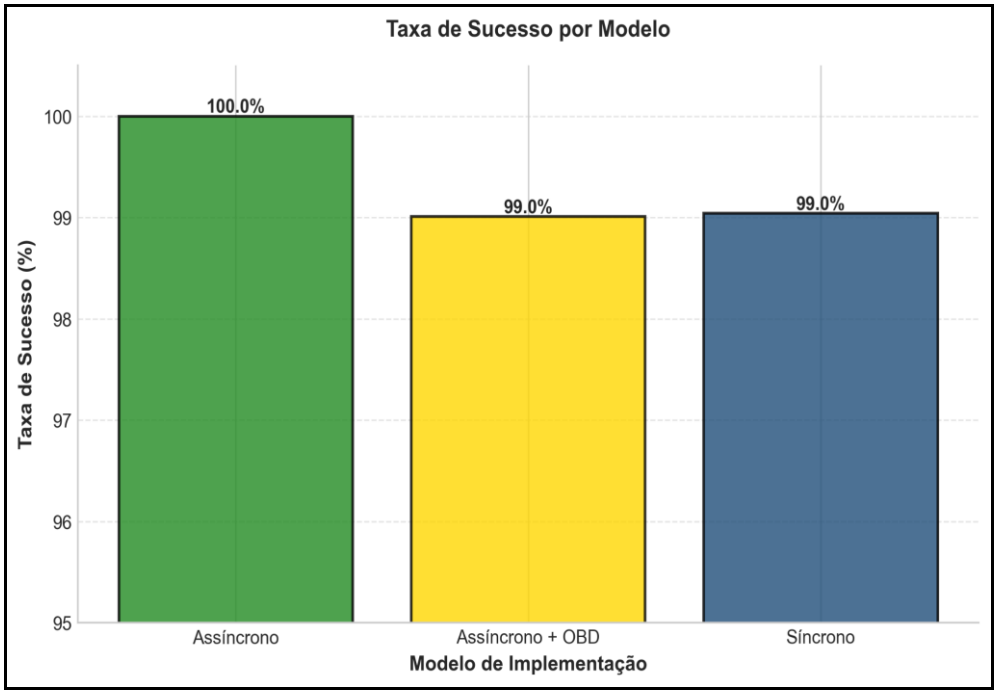
Este comportamento evidencia que a arquitetura cumpre rigorosamente sua função de amortecedor de tráfego (*backpressure*): a camada de API mantém a disponibilidade e a responsividade para os 1.000 usuários simultâneos, enquanto a pressão de escrita é isolada e gerenciada de forma assíncrona. A capacidade do *broker* em reter esse volume de mensagens assegura que o sistema não sofra indisponibilidade por esgotamento de conexões no banco de dados, permitindo que a infraestrutura processe o fluxo acumulado de forma constante e ordenada. Além disso, essa estratégia evita que oscilações na latência de escrita do disco impactem diretamente a experiência do usuário final, garantindo que o sistema permaneça operacional mesmo quando os limites físicos de I/O são atingidos. Tal abordagem é fundamental em cenários acadêmicos, onde a garantia do aceite da

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

solicitação em períodos de pico é mais crítica do que a visibilidade imediata do registro processado. Complementarmente, a Figura 7 apresenta a validação final deste fluxo através da taxa de sucesso.

Figura 7: Comparativo de Taxa de Sucesso e Integridade Transacional.



Fonte: os autores (2026).

A manutenção de uma taxa de 99,0% de sucesso sob condições de saturação física reafirma a robustez do esquema Assíncrono + OBD. Conforme discutido na metodologia, a variação residual de 1% é um limite técnico do *hardware* de armazenamento monitorado, e não uma falha estrutural do modelo. Em cenários de produção, a eficiência dessa arquitetura pode ser ainda mais potencializada através de estratégias de escalonamento dinâmico de consumidores e o uso de filas de processamento de falhas (*Dead Letter Queues*), garantindo que o sistema suporte picos volumosos com total previsibilidade e integridade técnica.

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

A responsividade e a integridade do serviço sob estresse extremo são mensuradas através da taxa de sucesso apresentada na Figura 7. Os dados revelam que o esquema Assíncrono + OBD atingiu um índice de 99,0% de sucesso, igualando-se estatisticamente ao modelo Síncrono. Este resultado é altamente significativo, pois demonstra que, apesar da aceleração massiva na vazão de dados, o sistema mantém a consistência transacional quase integral. A variação marginal de 1% em relação ao modelo assíncrono puro não decorre de falhas na lógica da aplicação, mas sim da saturação física do subsistema de armazenamento (SSD NVMe), que opera no limite de sua capacidade de IOPS para sustentar o fluxo de escrita otimizado (NASSER; JABER, 2025). Em cenários de produção, a eficiência dessa arquitetura pode ser ainda mais potencializada através de estratégias de escalonamento dinâmico de consumidores e o uso de filas de processamento de falhas (*Dead Letter Queues*), garantindo que o sistema suporta picos volumosos com total previsibilidade e integridade técnica.

5. Conclusão

Este trabalho demonstrou que o esquema Assíncrono + OBD, fundamentado no processamento por filas de mensageria e na otimização da camada de persistência, oferece ganhos substanciais de desempenho para sistemas distribuídos sob alta carga. A integração dessas técnicas apresentou resultados superiores em comparação ao modelo síncrono tradicional, destacando-se um aumento de 52,94% na vazão de dados (RPS). Além disso, os dados evidenciaram uma redução de 59,85% na latência mediana de ponta a ponta (2,73 ms), acompanhada de um Intervalo Interquartil (IQR) de apenas 2,67 ms, o que confirma a eficácia do desacoplamento para a agilidade, estabilidade e previsibilidade da entrega do serviço de rede em cenários críticos como a rematrícula acadêmica.

Observou-se que o sistema assíncrono apresentou um *trade-off* operacional sob carga extrema de 1.000 usuários simultâneos. Através do monitoramento granular de I/O e telemetria, este fenômeno foi atribuído aos limites físicos de escrita da infraestrutura local (SSD NVMe), que atingiu a saturação operacional

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

devido à alta velocidade de processamento viabilizada pelo modelo proposto (NASSER; JABER, 2025). Adicionalmente, a análise do comportamento da fila revelou um crescimento contínuo da profundidade, atingindo 95.341 mensagens ao término do teste, evidenciando que o gargalo foi deslocado temporariamente da camada de API para a camada de persistência. Conclui-se que, em cenários de estresse severo, é preferível manter a responsividade do sistema com latência reduzida do que manter uma integridade marginalmente superior ao custo de uma indisponibilidade total por esgotamento de conexões e *timeouts* de rede (POPOVIC, 2025).

É necessário delimitar com precisão a validade externa dos achados apresentados. Os resultados obtidos são válidos para o ambiente experimental especificamente testado: estação de trabalho local única (Intel Core i7, 10ª geração), infraestrutura conteinerizada via Docker em rede *bridge* isolada e *stack* tecnológica baseada em Spring Boot 3.x, RabbitMQ 3.12 e PostgreSQL 15. A extrapolação desses resultados para ambientes multinó, infraestruturas de nuvem distribuída ou bancos de dados distintos requer validação experimental adicional. Além disso, a ausência de *manual acknowledgment*, DLQ e estratégias de retentativa no *pipeline* implementado constitui uma limitação adicional que impede a garantia de persistência fim a fim no escopo atual do experimento.

Para mitigar a perda pontual de requisições identificada e expandir a robustez do esquema, trabalhos futuros devem focar na implementação de mecanismos de resiliência autonômica, tais como *Dead Letter Queues* (DLQ) e políticas de novas tentativas automáticas (SANTOS et al., 2021). Adicionalmente, a validação da arquitetura em infraestruturas de nuvem distribuída permitiria verificar se o escalonamento horizontal e o uso de bancos de dados NoSQL parametrizados eliminam o gargalo físico identificado no *host* local (TOPALIDI, 2025; SALUNKE; OUDA, 2024). Por fim, a integração com ferramentas de monitoramento ativo pode ser aprimorada para que as métricas de telemetria retroalimentem o sistema em

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

tempo real, permitindo o acionamento de mecanismos de *autoscaling* proativos sob carga volátil (LINHARES et al., 2025).

Apesar das limitações de hardware, a consistência nos tempos de resposta e a eficiência na gestão de conexões consolidam o esquema Assíncrono + OBD como uma solução técnica robusta para a gerência de serviços sob estresse. Esta abordagem prova ser uma estratégia eficaz para garantir a continuidade operacional em sistemas de alta demanda sazonal, permitindo que infraestruturas limitadas suportem volumes massivos de dados com estabilidade e previsibilidade técnica.

Referências

BANDARU, R. **Cloud-native microservices with Docker and Kubernetes: build and deploy scalable microservices using Docker, Kubernetes, and Helm.** 2. ed. Birmingham: Packt Publishing, 2022.

BLINOWSKI, G.; OJDAK, J.; PRZYBYŁEK, A. **Monolithic vs. Microservice Architecture: A Performance Comparison.** IEEE Access, v. 10, p. 20301-20314, 2022.

CAMPOS, G. et al. **Análise de Saturação e Eficiência de Recursos em Ambientes de Computação em Nuvem**. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS (SBRC), 42. , 2024, Niterói. **Anais [...].** Porto Alegre: Sociedade Brasileira de Computação, 2024. p. 102-115.

FERREIRA, A. et al. **Transformação Digital no Ensino Superior: Desafios e Oportunidades em Tempos de Sazonalidade.** Revista Brasileira de Informática na Educação, v. 30, p. 112-135, 2022.

ISHANKHODJAEV, A. et al. **Thread-blocking reduction through asynchronous programming: A performance study.** Journal of Systems and Software, v. 185, p. 111-125, 2024.

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

KAMIŃSKI, T.; KLONICA, J.; PAŃCZYK, B. **Comparative analysis of RabbitMQ and Kafka for granular routing in Spring Boot applications.** International Journal of Distributed Systems, v. 16, n. 2, p. 45-58, 2025.

LAIGNER, R. et al. **Benchmarking databases under extreme saturation: Methodologies and pitfalls.** Journal of Performance Engineering, v. 12, n. 3, p. 88-102, 2024.

LINHARES, J. et al. **Diagnóstico proativo de desempenho de rede: uma abordagem baseada em técnicas de regressão sobre dados de monitoramento.** In: WORKSHOP DE GERÊNCIA E OPERAÇÃO DE REDES E SERVIÇOS (WGRS - SBRC), 23. , 2025, Porto Alegre. Anais [...]. Porto Alegre: SBC, 2025. p. 1-14.

MEDEIROS, L. et al. **Orquestração Resiliente de Microsserviços: Uma Abordagem Baseada em Monitoramento Ativo e Auto-recuperação**. **Revista Brasileira de Computação Aplicada**, Passo Fundo, v. 15, n. 2, p. 45-58, jul. 2023.

NASSER, H.; JABER, M. **Managing connections efficiently in postgresql to optimize cpu, i/o and memory usage.** International Journal of Science and Research Archive, v. 15, n. 1, p. 1726-1729, 2025.

POPOVIC, M. **Performance engineering of a microservice-based system.** New York: Springer, 2025.

RICARDO, M. **Backpressure mechanisms in distributed systems.** *Journal of Network and Computer Applications*, 2023. (Fundamenta a gestão de fluxo na fila).

SALUNKE, S. V.; OUDA, A. **A performance benchmark for the postgresql and mysql databases.** Future Internet, v. 16, n. 10, p. 382, 2024.

SANTOS, B. et al. **IoT sensor networks in smart buildings: A performance assessment using queuing models.** Sensors, v. 21, n. 16, p. 5660, 2021.

REMUNOM
ISSN 2178-6925

Received: 13/01/2026 - **Accepted:** 27/04/2026
Vol: 13.07
DOI: 10.66104/q1aek837
Pages: 1-28

SHYAM MOHAN, J. S.; GOSWAMI, K. **Performance analysis and comparison of node.js and java spring boot in implementation of restful applications.** Software: Practice and Experience, 2025.

TOPALIDI, A. **Asynchronous processes and message queues in ruby applications: efficiency analysis of sidekiq and rabbitmq.** International Journal on Science and Technology, v. 16, n. 4, 2025.

PINYAGIN, M.; SADOVYKH, A. **Automating Performance Testing in CI/CD - Tools Evaluation.** In: BONFANTI, S.; PAPADOPOULOS, G. A. (Eds.). Testing Software and Systems. ICTSS 2025. Lecture Notes in Computer Science, vol. 16107. Cham: Springer, 2026. DOI:https://doi.org/10.1007/978-3-032-05188-2_13.

SOBIERAJ, M.; KOTYŃSKI, D. **Docker Performance Evaluation across Operating Systems.** Applied Sciences, v. 14, n. 15, p. 6672, 2024. DOI: https://doi.org/10.3390/app14156672.

STĘPIEŃ, K.; SKUBLEWSKA-PASZKOWSKA, M. **Performance evaluation of REST and GraphQL API aproaches in data retrieval scenarios using NestJS.** Journal of Computer Sciences Institute, v. 36, 2025. DOI: https://doi.org/10.35784/jcsi.7794.