

## REVISÃO

# Uma Análise Comparativa Reprodutível de Apache IoTDB, InfluxDB e TimescaleDB para Cargas de Trabalho IoT

Luiz Fernando Klein  [Universidade Federal da Fronteira Sul - UFFS | [luizkleinfernando@gmail.com](mailto:luizkleinfernando@gmail.com)]

 *Curso de Ciência da Computação, Universidade Federal da Fronteira Sul – UFFS, Rodovia SC 484, Km 02, Chapecó, SC, 89815-899, Brasil.*

**Resumo.** O rápido crescimento das implantações de Internet das Coisas gerou volumes sem precedentes de dados de sensores indexados por tempo, expondo as limitações dos bancos de dados tradicionais e impulsionando a adoção de Bancos de Dados de Séries Temporais (BDSTs) especializados. Apesar do crescente interesse, a maioria dos benchmarks de BDSTs publicados carece de scripts e configurações suficientes para replicação independente.

Este artigo apresenta um benchmark controlado e reprodutível comparando Apache IoTDB 1.3.0, InfluxDB 2.7 e TimescaleDB (PostgreSQL 15) em nove tipos de carga de trabalho IoT em três escalas de volume de dados com variação de cem vezes no tamanho dos dados, utilizando um ambiente totalmente automatizado construído sobre Docker Compose e *iot-benchmark*. Os resultados mostram que nenhum banco de dados domina todas as cargas de trabalho. O IoTDB lidera em vazão de escrita (23,6 Mpts/s na escala grande) e escalonamento de filtro de valor (crescimento de latência de  $5,2\times$  para dados cem vezes maiores). O TimescaleDB fornece subamostragem abaixo de um milissegundo em todas as escalas e as consultas de último valor mais rápidas, com suporte completo a SQL. O InfluxDB oferece um meio-termo equilibrado com menores requisitos de memória. Uma descoberta fundamental é que ambientes de benchmark com recursos compartilhados produzem resultados enganosos: duas conclusões das execuções iniciais foram revertidas por reexecuções isoladas, demonstrando que o isolamento de recursos é um pré-requisito para comparações válidas.

**Abstract.** The rapid growth of Internet of Things deployments has generated unprecedented volumes of time-indexed sensor data, exposing the limitations of traditional databases and driving the adoption of specialized Time Series Databases (TSDBs). Despite growing interest, most published TSDB benchmarks lack sufficient scripts and configurations for independent replication. This paper presents a controlled, reproducible benchmark comparing Apache IoTDB 1.3.0, InfluxDB 2.7, and TimescaleDB (PostgreSQL 15) across nine IoT workload types at three data-volume scales spanning a hundredfold range in data size, using a fully scripted infrastructure built on Docker Compose and *iot-benchmark*. Results show that no single database dominates all workloads. IoTDB leads on write throughput (23.6 Mpts/s at large scale) and value-filter scaling ( $5.2\times$  latency growth for a hundredfold increase in data). TimescaleDB provides sub-millisecond downsampling across all scales and the fastest latest-point lookups, backed by full SQL compatibility. InfluxDB offers a balanced middle ground with lower memory requirements. A key finding is that shared-resource benchmark environments produce misleading results: two conclusions from the initial runs were reversed by isolated reruns, underscoring that resource isolation is a prerequisite for valid comparisons.

**Keywords:** Bancos de Dados de Séries Temporais, Benchmark, Apache IoTDB, InfluxDB, TimescaleDB, IoT

**Recebido:** DD Mês AAAA • **Aceito:** DD Mês AAAA • **Publicado:** DD Mês AAAA

## 1 Introdução

A adoção generalizada das tecnologias de Internet das Coisas (IoT), sistemas de monitoramento industrial e aplicações financeiras em tempo real gerou um volume sem precedentes de dados temporais, criando desafios críticos para as infraestruturas de armazenamento e análise. Conforme projetado em 2015 por Thierer and Castillo [2015], o número de dispositivos IoT conectados vem crescendo exponencialmente e deve continuar aumentando nos próximos anos, conforme ilustrado na Figura 1.

O gráfico apresenta dados históricos até 2015 e projeções de crescimento até 2019, estimativas que estudos subsequentes, incluindo relatórios da Cisco, viriam a confirmar.

Tal crescimento foi posteriormente reafirmado por Zikria *et al.* [2021], que aponta que a Cisco prevê 500 bilhões de dispositivos conectados à IoT até 2030, enquanto a Telefonica estima que 90% dos veículos estarão conectados e cada pessoa terá, em média, 15 dispositivos conectados nesse mesmo

horizonte. A proliferação de aplicações de próxima geração, abrangendo saúde inteligente, cidades inteligentes, agricultura inteligente e IoT industrial, confirma que as projeções feitas em 2015 não apenas se concretizaram, mas continuam a se ampliar.

Como observado por Palma [2016], inúmeros setores passaram a depender da capacidade de processar fluxos contínuos de dados. Entre eles: o setor financeiro, que utiliza essa capacidade para analisar flutuações de ações e volumes de transações; os serviços de transporte, que a empregam para monitorar viagens e migrações sazonais; a medicina, que a utiliza para rastrear a disseminação de doenças; e a análise meteorológica, que aproveita dados históricos para previsões mais precisas.

Esses fluxos de dados são sequências de eventos ou medições naturalmente ordenadas e indexadas pelo tempo, frequentemente com precisão de milissegundos. Como observado por Dunning and Friedman [2014], o que costumava levar dez minutos para ser coletado em alguns sistemas muito ativos é

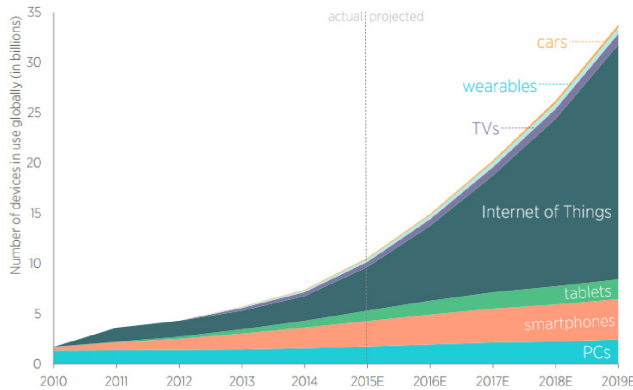


**Figura 1.** IoT: dispositivos em uso globalmente. Fonte: Thierer and Castillo [2015].

agora gerado a cada segundo. Essa explosão de dados sequenciais indexados por tempo expôs as limitações fundamentais dos bancos de dados relacionais tradicionais quando confrontados com padrões de acesso temporais, alta velocidade de ingestão e armazenamento histórico eficiente de longo prazo.

Nesse cenário, os Bancos de Dados de Séries Temporais (BDSTs) emergiram como soluções especializadas, arquitetadas desde a concepção para atender às demandas exclusivas dos dados temporais [InfluxData, 2025]. Os bancos de dados relacionais tradicionais sofrem de ineficiências de indexação e limitações de memória ao lidar com grandes volumes de dados temporais. Os BDSTs, por sua vez, mantêm alto desempenho por terem sido construídos com o tempo como preocupação central [Naqvi *et al.*, 2017]. Além da vazão, os BDSTs oferecem mecanismos especializados de compressão, co-localização de dados em intervalos de tempo e políticas automáticas de retenção, que reduzem os custos operacionais [Dunning and Friedman, 2014]. Esses recursos são importantes para aplicações IoT, nas quais medições contínuas de sensores demandam escalabilidade e uso econômico de recursos.

O cenário dos BDSTs ampliou-se nos últimos anos, com sistemas adotando abordagens arquiteturais distintas para o mesmo problema. O *Apache IoTDB* [Wang *et al.*, 2023], desenvolvido na Universidade Tsinghua, foi concebido especificamente para IoT industrial, utilizando um formato de arquivo colunar personalizado (TsFile) e um protocolo binário Remote Procedure Call (RPC) que possibilita taxas de ingestão extremamente elevadas. O *InfluxDB*, avaliado em quatro dos cinco estudos de desempenho revisados neste trabalho [Naqvi *et al.*, 2017; Shah *et al.*, 2022; Lima and Lichtnow, 2024; Wang *et al.*, 2023], usa um motor de armazenamento Time-Structured Merge Tree (TSM) exposto via API HTTP com protocolo de linha. O *TimescaleDB* estende o PostgreSQL com capacidades de séries temporais, oferecendo compatibilidade com SQL e o ecossistema relacional completo como contrapartida a uma menor vazão de escrita. Cada abordagem possui pontos fortes e fracos distintos que não são aparentes apenas a partir de descrições e exigem medições controladas sob cargas de trabalho IoT realistas.

O campo de benchmarking de BDSTs, no entanto, sofre de uma lacuna metodológica crítica: a falta de reprodutibilidade. Uma análise dos estudos comparativos recentes revela que grande parte dos benchmarks publicados não fornece scripts, configurações ou versões exatas de software

suficientes para permitir replicação independente [de Leal Meireles Pereira, 2023; Shah *et al.*, 2022; Naqvi *et al.*, 2017]. Sem metodologia reprodutível, os resultados não podem ser validados de forma independente, são difíceis de comparar entre estudos e os experimentos não podem ser adaptados a novos hardwares ou versões de software. Este trabalho aborda diretamente essa lacuna tornando ambiente completo de benchmark (i.e., todos os scripts de orquestração, definições de contêineres, ambiente de construção e clientes de benchmark parametrizados) publicamente disponível junto com os resultados.

Este artigo apresenta uma comparação controlada e reprodutível do Apache IoTDB, InfluxDB 2.7 e TimescaleDB em nove padrões de carga de trabalho representativos de implantações IoT e três escalas de volume de dados. Os bancos de dados foram selecionados com base em sua proeminência na literatura: InfluxDB como o padrão de referência estabelecido, Apache IoTDB como o líder emergente nos dois estudos mais recentes [de Leal Meireles Pereira, 2023; Wang *et al.*, 2023], e TimescaleDB como a alternativa baseada em SQL que oferece ampla compatibilidade com ecossistemas.

Os resultados mostram que nenhum banco de dados domina em todas as dimensões: o IoTDB lidera em vazão de escrita sequencial e filtragem de valor, o TimescaleDB oferece latência constante em subamostragem e compatibilidade SQL completa, e o InfluxDB ocupa um meio-termo prático entre os dois.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica cobrindo bancos de dados, bancos de dados de séries temporais e conceitos de benchmarking. A Seção 3 revisa estudos comparativos relacionados e identifica as lacunas que motivam este trabalho. A Seção 4 descreve a metodologia de benchmark, o projeto das cargas de trabalho e o ambiente reprodutível. A Seção 5 apresenta e analisa os resultados experimentais. A Seção 6 conclui o artigo com uma comparação final e direções para trabalhos futuros.

## 2 Fundamentação Teórica

Esta seção apresenta os conceitos fundamentais que embasam esta pesquisa. Está organizada em quatro partes: conceitos gerais de banco de dados e o modelo relacional; sistemas de bancos de dados distribuídos; bancos de dados de séries temporais; e metodologia de benchmarking.

### 2.1 Bancos de Dados

De acordo com Ramakrishnan and Gehrke [2003], um banco de dados pode ser definido como uma coleção de dados que tipicamente descreve as atividades de uma ou mais organizações relacionadas. Isso abrange desde informações sobre entidades (como estudantes e cursos em uma universidade) até os relacionamentos entre essas entidades, como as matrículas de alunos em disciplinas específicas. SILBERSCHATZ [2020] acrescenta que o objetivo primário de um sistema de banco de dados deve ser fornecer uma maneira de armazenar e recuperar informações que seja ao mesmo tempo conveniente e eficiente.

O gerenciamento desses dados é realizado por um Sistema Gerenciador de Banco de Dados (SGBD), que é um software projetado para auxiliar na manutenção e uso de gran-

des coleções de dados [Ramakrishnan and Gehrke, 2003]. A necessidade de tais sistemas cresceu rapidamente, pois a alternativa de armazenar dados em arquivos de dados simples e escrever código de gerenciamento específico para cada aplicação produz soluções que não são transferíveis entre aplicações e não oferecem garantias de integridade ou de acesso concorrente seguro.

Como enfatizado por Ramakrishnan and Gehrke [2003], o sucesso de uma organização depende cada vez mais de sua capacidade de adquirir dados precisos e oportunos sobre suas operações, gerenciar esses dados de forma eficaz e utilizá-los para analisar e guiar suas atividades. Isso posiciona o banco de dados como um ativo organizacional estratégico, e não meramente como um componente técnico. SILBERSCHATZ [2020] acrescenta que um banco de dados deve fornecer aos usuários uma visão abstrata dos dados, ocultando os detalhes de armazenamento e representação para que possa servir como uma abstração prática para resolução de problemas.

Ramakrishnan and Gehrke [2003] enumeram diversas vantagens dos SGBDs em relação aos sistemas de arquivos tradicionais: independência de dados (i.e., os programas de aplicação são isolados dos detalhes de representação e armazenamento de dados); acesso eficiente aos dados (i.e., por meio de técnicas sofisticadas de indexação e otimização de consultas); integridade e segurança dos dados (i.e., o SGBD impõe restrições e restringe acessos não autorizados); e suporte para acesso concorrente e recuperação de falhas.

O modelo relacional, proposto por Codd em 1970, é simples e elegante: um banco de dados é visto como uma coleção de relações (i.e., tabelas com linhas e colunas), uma representação tão intuitiva que mesmo usuários não especialistas conseguem entender o conteúdo armazenado [Ramakrishnan and Gehrke, 2003]. A estrutura de cada relação consiste em um esquema (i.e., nomes dos campos e seus domínios) e uma instância (i.e., o conjunto de tuplas, ou registros). Uma propriedade essencial é que cada relação é um conjunto de tuplas únicas, garantindo a ausência de registros duplicados. Restrições de integridade, ou seja, condições especificadas no esquema que restringem os dados armazenáveis, garantem que apenas instâncias válidas do banco de dados possam ser persistidas. Entre elas, as restrições de chave primária identificam unicamente as tuplas, enquanto as restrições de chave estrangeira mantêm a consistência quando informações em uma relação referenciam outra.

O SQL tornou-se o padrão comercial para interação com bancos de dados relacionais. Como observam Ramakrishnan and Gehrke [2003], o SQL é a linguagem comercial mais amplamente utilizada para SGBDs relacionais, dando suporte não apenas à definição e manipulação de dados, mas também a consultas complexas. O projeto de banco de dados tipicamente começa com um diagrama Entidade-Relacionamento (ER) como representação de alto nível, que é então traduzido em um esquema relacional.

## 2.2 Bancos de Dados Distribuídos

Os bancos de dados distribuídos (BDDs) são a convergência das tecnologias de sistemas de banco de dados e redes de computadores [Özsu and Valduriez, 2011]. Os sistemas de banco de dados fornecem os mecanismos de armazenamento, consulta e controle de integridade dos dados, enquanto as

redes de computadores viabilizam a distribuição física entre múltiplos nós. A tecnologia dos BDDs combina esses dois campos, demonstrando que a integração de dados não implica necessariamente centralização física.

Para SILBERSCHATZ [2020], um sistema de banco de dados distribuído é composto por múltiplas instâncias com bancos de dados locais, cada uma capaz de processar transações locais, ao mesmo tempo em que executa transações globais que acessam dados em múltiplos locais, exigindo comunicação entre sítios.

Um banco de dados distribuído é definido como uma coleção de múltiplos bancos de dados logicamente inter-relacionados, distribuídos por uma rede de computadores [Özsu and Valduriez, 2011]. O SGBD distribuído (SGBDD) é o software que gerencia esse banco de dados distribuído, tornando a distribuição transparente para os usuários. Um BDD não é simplesmente uma coleção de arquivos armazenados em diferentes nós de rede; é um sistema estruturado com um esquema definido, acessível por meio de uma interface comum.

A arquitetura de um SGBDD pode assumir diversas formas, incluindo cliente/servidor, ponto-a-ponto e sistemas multibanco [Özsu and Valduriez, 2011]. Em sistemas cliente/servidor, as funções são divididas entre clientes (i.e., os responsáveis pela interface do usuário) e servidores (i.e., os responsáveis pelo gerenciamento de dados). Em sistemas ponto-a-ponto, todas as máquinas possuem funcionalidade completa de SGBD e podem se comunicar diretamente. Os sistemas multibanco lidam com a integração de bancos de dados heterogêneos e autônomos.

Um benefício fundamental dos BDDs é o desempenho melhorado por meio da localidade de dados e do paralelismo. Como observam Özsu and Valduriez [2011], armazenar dados próximos aos seus pontos de uso reduz a contenção de CPU e entrada/saída, enquanto o paralelismo pode ser explorado tanto entre consultas (i.e., interqueries) quanto dentro de uma única consulta (i.e., intraquery). No entanto, isso traz desafios: consistência da replicação, sincronização de transações distribuídas e detecção de deadlock distribuído, como destacado por Özsu and Valduriez [2011] e SILBERSCHATZ [2020]. O objetivo mais importante da tecnologia de banco de dados, como argumentam Özsu and Valduriez [2011], é a integração, não a centralização.

## 2.3 Bancos de Dados de Séries Temporais

Os Bancos de Dados de Séries Temporais (BDSTs) são uma categoria especializada de sistemas de armazenamento projetados para lidar com dados registrados como uma sequência de valores, cada um associado a um instante de tempo específico. De acordo com Özsu and Valduriez [2011], uma série temporal é um conjunto de valores de atributos ao longo de um período de tempo. Essa característica fundamental diferencia os BDSTs dos bancos de dados de uso geral: eles são otimizados para consultas baseadas em intervalos de tempo, oferecendo desempenho superior para esse padrão de acesso.

Registros meteorológicos, indicadores econômicos e métricas de saúde de pacientes são todos dados de séries temporais. Os dados de séries temporais também incluem métricas de servidores, monitoramento de desempenho de aplicações, dados de rede, dados de sensores, eventos, cliques e muitos ou-

tros tipos de dados analíticos [InfluxData, 2025]. A Figura 2 ilustra uma série temporal de medições meteorológicas. Cada ponto no gráfico corresponde a uma leitura realizada em um instante específico. O eixo horizontal representa a progressão do tempo, enquanto o eixo vertical representa o valor medido.

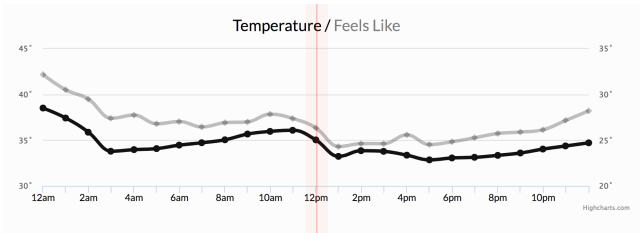


**Figura 2.** Exemplo de dados temporais: condições climáticas. O ponto em destaque representa um único registro de dados capturado em um instante específico.

| # date                       | # p                         | # T                  | # rh                  | # sh                     |
|------------------------------|-----------------------------|----------------------|-----------------------|--------------------------|
| Date and time of observation | Atmospheric pressure (mbar) | Air temperature (°C) | Relative humidity (%) | Specific humidity (g/kg) |
| 2020-01-01 00:10:00          | 1008.89                     | 0.71                 | 86.1                  | 3.42                     |
| 2020-01-01 00:20:00          | 1008.76                     | 0.75                 | 85.2                  | 3.39                     |
| 2020-01-01 00:30:00          | 1008.66                     | 0.73                 | 85.1                  | 3.39                     |
| 2020-01-01 00:40:00          | 1008.64                     | 0.37                 | 86.3                  | 3.35                     |
| 2020-01-01 00:50:00          | 1008.61                     | 0.33                 | 87.4                  | 3.38                     |
| 2020-01-01 01:00:00          | 1008.44                     | 0.34                 | 87.1                  | 3.37                     |
| 2020-01-01 01:10:00          | 1008.49                     | 0.19                 | 87.4                  | 3.34                     |
| 2020-01-01 01:20:00          | 1008.4                      | 0.83                 | 88.1                  | 3.33                     |
| 2020-01-01 01:30:00          | 1008.45                     | 0.44                 | 88.4                  | 3.32                     |

**Figura 3.** Exemplo de tabela em um BDST representando a série temporal climática. Cada linha corresponde a uma medição indexada por tempo.

A Figura 3 mostra a tabela correspondente em um BDST, onde cada linha é um registro indexado por tempo que captura pressão atmosférica, temperatura, umidade e outras leituras em um momento específico.

Como explicado por Özsu and Valduriez [2011], uma série temporal normalmente consiste apenas em valores numéricos, contínuos ou discretos. Isso torna possível modelar fluxos de dados que contêm apenas valores numéricos como séries temporais, permitindo que técnicas de análise de séries temporais sejam aplicadas a certos tipos de fluxos de dados. Uma característica fundamental é que os conjuntos de dados de séries temporais são tipicamente utilizados em situações em que as medições, uma vez realizadas, não são revisadas ou atualizadas, mas acumuladas: novos dados são adicionados para cada parâmetro medido a cada novo instante de tempo, como observado por Dunning and Friedman [2014].

A significância histórica da análise de séries temporais é notável, com exemplos que remontam ao século XIX. Dunning and Friedman [2014] citam o trabalho pioneiro de Matthew Fontaine Maury, que analisou diários de bordo de navios para criar cartas de vento e correntes, demonstrando como a análise coletiva de dados de séries temporais acumulados ao longo de muitos anos em muitos navios pode revelar padrões valiosos para a navegação marítima. Esse exemplo histórico ilustra como os conjuntos de dados de séries temporais podem revelar tendências e padrões não aparentes em medições pontuais.

No contexto moderno, os BDSTs enfrentam novos desafios devido ao volume, velocidade e variedade dos dados gerados. O que costumava ser coletado em 10 minutos para alguns sistemas muito ativos é agora gerado a cada segundo [Dun-

ning and Friedman, 2014]. Essa explosão no volume de dados exige novas abordagens de armazenamento e acesso, tornando os bancos de dados NoSQL particularmente adequados para essa escala. Para consultas baseadas em tempo em escalas muito grandes, esses acessos podem ser implementados como varreduras grandes e contíguas, extremamente eficientes quando os dados estão armazenados adequadamente em um banco de dados de séries temporais.

A arquitetura dos BDSTs modernos evoluiu consideravelmente. Dunning and Friedman [2014] descrevem três abordagens principais para o armazenamento de séries temporais: (1) tabelas com muitas colunas que armazenam dados ponto a ponto; (2) um design híbrido misturando tabelas com muitas colunas com armazenamento em bloco binário de dados (BLOB, do inglês *Binary Large Object*); e (3) inserção direta de BLOB a partir de um cache em memória, que pode atingir taxas de ingestão até 1.000 vezes mais rápidas que as abordagens ponto a ponto. Os BDSTs armazenam múltiplas séries temporais de forma que consultas que recuperam dados de uma ou poucas séries para um intervalo de tempo específico sejam particularmente eficientes [Dunning and Friedman, 2014]. Essa eficiência é alcançada por meio de estratégias de armazenamento que agrupam fisicamente dados temporalmente próximos, habilitando operações eficientes de varredura sequencial.

As aplicações dos BDSTs são diversas e estão em expansão. No setor financeiro, a velocidade das negociações e a necessidade de latência extremamente baixa tornam os BDSTs de alto desempenho essenciais [Dunning and Friedman, 2014]. No contexto IoT, os BDSTs ocupam uma posição central no fluxo de dados de sensor à percepção que está mudando rapidamente como percebemos e entendemos o mundo [Dunning and Friedman, 2014].

## 2.4 Benchmarks

A avaliação de desempenho é um componente crítico do processo de seleção de banco de dados. Como destacado por Ramakrishnan and Gehrke [2003], quando um banco de dados cresce além da capacidade do SGBD atual de fornecer desempenho satisfatório, a migração para um novo sistema deve ser considerada. Os benchmarks são as principais ferramentas que permitem a comparação objetiva entre os diferentes produtos disponíveis. Um único teste é insuficiente; como argumenta SILBERSCHATZ [2020], um benchmark deve simular cargas de trabalho reais, pois cada produto pode ter desempenho diferente dependendo da tarefa.

A complexidade dos SGBDs e as diferentes abordagens adotadas pelos fornecedores tornam os benchmarks particularmente valiosos. Como observam Ramakrishnan and Gehrke [2003], um SGBD é um software complexo, e diferentes fornecedores podem ter como alvo diferentes segmentos de mercado, otimizando certas partes de seus sistemas ou escolhendo projetos arquiteturais distintos. Essa diversidade de abordagens e otimizações específicas torna a comparação direta difícil sem instrumentos padronizados.

Quanto às características fundamentais de um bom benchmark, Ramakrishnan and Gehrke [2003] enumeram três qualidades essenciais. Primeiro, portabilidade: a capacidade de executar o benchmark em diferentes plataformas e sistemas. Segundo, clareza: garantir fácil compreensão do que

está sendo realmente medido. Terceiro, escalabilidade: a adaptação natural a instâncias maiores do problema. Além desses atributos básicos, bons benchmarks devem medir tanto o desempenho máximo (expresso em métricas como transações por segundo) quanto a relação custo-eficácia (dólares por transação por segundo) para cargas de trabalho típicas de um domínio de aplicação específico. Foi precisamente para atender a essas necessidades que o Transaction Processing Council (TPC) foi criado, com a missão de desenvolver benchmarks padronizados para processamento de transações e sistemas de banco de dados.

SILBERSCHATZ [2020] acrescenta uma observação importante sobre o cálculo correto de desempenho: usar a média simples de vazões pode ser enganoso. O método correto é calcular o tempo total para uma carga de trabalho mista e derivar a vazão média a partir disso, ou usar a média harmônica, que reflete melhor a realidade quando há diferentes tipos de operações.

A literatura de benchmarks categoriza seus instrumentos em vários grupos [Ramakrishnan and Gehrke, 2003]. Os benchmarks de Processamento de Transações Online (TPC-A, TPC-B, TPC-C) definem medidas padrão de transações por segundo e custo por transação. O TPC-C, o mais amplamente utilizado, modela um ambiente complexo de armazém com cinco tipos de transação, exercitando uma gama muito mais ampla de capacidades do sistema do que seus predecessores. O TPC-C foi concebido para cargas OLTP (Online Transaction Processing), enquanto o TPC-H foca em cargas OLAP (Online Analytical Processing), evidenciando que benchmarks distintos são necessários para avaliar diferentes perfis de acesso a dados. Os benchmarks de desempenho de consultas (Wisconsin, AS3AP, TPC-D) avaliam o desempenho de consultas relacionais, de simples a complexas para suporte a decisões. Os benchmarks de banco de dados orientado a objetos (001, 007, Bucky) medem sistemas orientados a objetos e objeto-relacionais.

Para aplicação prática, os benchmarks devem ser interpretados com cuidado. Ramakrishnan and Gehrke [2003] alertam que métricas de desempenho de número único podem ser demasiadamente simplistas para softwares complexos como um SGBD. Um bom benchmark deve incluir um conjunto abrangente de tarefas cuidadosamente selecionadas para cobrir um domínio de aplicação específico e para testar as capacidades do SGBD críticas para esse domínio. Os analistas devem pesar as tarefas do benchmark de acordo com o quanto se assemelham às operações críticas da aplicação pretendida; e, quando os benchmarks padrão não correspondem suficientemente à carga de trabalho real, benchmarks personalizados que adaptem ou substituam tarefas padrão por operações mais representativas devem ser considerados.

Shapira and Chen [2016] catalogam cinco armadilhas recorrentes em benchmarks de sistemas de big data: comparar configurações que diferem em múltiplos parâmetros simultaneamente, testar em escala insuficiente e extrapolar os resultados, aceitar resultados suspeitos sem questionar sua origem, usar cargas de trabalho artificiais desconectadas do uso real, e comunicar resultados de forma incompleta ou enganosa. Essas armadilhas são particularmente relevantes para benchmarks de bancos de dados de séries temporais, onde diferenças de configuração, isolamento de recursos e escolha

de carga de trabalho podem inverter conclusões.

A reprodutibilidade é uma propriedade essencial, mas frequentemente negligenciada, dos benchmarks de banco de dados. Como observado nos seis estudos de desempenho revisados neste trabalho, apenas um fornece os scripts e configurações exatas necessárias para replicação independente [de Leal Meireles Pereira, 2023]. Isso limita a capacidade de validar resultados, comparar descobertas entre estudos e construir sobre trabalhos anteriores, uma lacuna que este estudo aborda diretamente por meio de seu ambiente de benchmark totalmente automatizado e publicamente disponível. Além do compartilhamento de código, o isolamento de recursos entre os sistemas avaliados é igualmente crítico: executar múltiplos motores de banco de dados simultaneamente no mesmo hardware introduz artefatos de contenção de cache de páginas e entrada/saída que podem inverter conclusões reportadas [Shapira and Chen, 2016].

### 3 Trabalhos Relacionados

Para a elaboração deste estudo, foram revisados trabalhos comparativos sobre bancos de dados de séries temporais, com foco em benchmarks de desempenho. A busca foi conduzida utilizando palavras-chave como “data streaming”, “timeseries” e “timeseries databases”, com os resultados ordenados por contagem de citações para priorizar análises mais estabelecidas e desenvolvidas. A partir dessa busca, 23 resultados foram selecionados para leitura inicial, a fim de verificar sua relevância para os objetivos deste estudo.

Dentre os 23 trabalhos, alguns não apresentaram relação direta suficiente para uma análise aprofundada, embora tenham fornecido contexto relevante. Zubaroglu and Atalay [2021], por exemplo, apresenta uma lista de conjuntos de dados aplicáveis a análises de bancos de dados de séries temporais. Gaber *et al.* [2005] contribuiu com informações relevantes sobre os algoritmos subjacentes às implementações de BDSTs, apesar de não estar diretamente focado em avaliação comparativa.

Da leitura inicial, seis trabalhos foram selecionados para estudo detalhado. Com base nestes trabalhos, foram identificadas duas linhas de análise: (i) Desempenho e (ii) Comparativa. Desta forma, este capítulo está organizado em duas seções. A seção de Análise de Desempenho cobre estudos comparativos de benchmark. A seção de Análise Comparativa sintetiza e critica todos os trabalhos selecionados por meio de uma tabela comparativa, destacando padrões, lacunas e tendências identificadas na literatura.

#### 3.1 Análise de Desempenho

Seis trabalhos no formato de benchmarks controlados ou comparações em cenários reais foram selecionados para análise detalhada.

Lima and Lichtnow [2024] empregaram testes de inserção (variando de 1.000 a 5 milhões de registros) e consultas de agregação com dados sintéticos (medições indexadas por tempo, localização, temperatura e umidade) para comparar o desempenho de SGBDs utilizando o Apache JMeter. A mesma estrutura de dados foi replicada em todos os bancos de dados. Os testes foram realizados em um ambiente baseado em Docker em uma máquina com AMD Ryzen 5 4500 (6 núcleos), 16 GB de RAM e armazenamento SSD, testando

volumes de 500.000 e 5 milhões de registros. A reprodutibilidade foi parcial, pois contêineres Docker foram utilizados, mas com detalhes incompletos para replicação total. O InfluxDB foi o destaque do estudo.

Shah *et al.* [2022] compararam o desempenho do InfluxDB v2.1, TimescaleDB v2.6, Druid v0.22.1 e Cassandra v3.11.12 em operações de inserção e oito tipos de consulta (e.g., consultas pontuais, agregações e outras), avaliando tempo de carga, consumo de espaço e latência. Foram utilizados tanto conjuntos de dados reais (dados IoT, dados financeiros, NYC Taxi) quanto sintéticos (gerados por um gerador Python de séries temporais). O ambiente de teste consistia em um Intel Core i5-9500 (6 núcleos), 8 GB de RAM, 500 GB SSD e Ubuntu 20.04. O InfluxDB se destacou não apenas no desempenho de consultas, mas também no consumo de espaço, ficando em segundo lugar apenas após o Druid. O estudo foi classificado como não reprodutível, pois nenhum script ou configuração foi compartilhado.

Naqvi *et al.* [2017] compararam InfluxDB, SQL Server, Cassandra, Elasticsearch e OpenTSDB em três métricas primárias: taxa de ingestão de dados, requisitos de armazenamento e tempo médio de resposta a consultas. Foram utilizados tanto dados sintéticos (simulando métricas de servidor) quanto um conjunto de dados real de viagens de táxi em NYC (2009–2017), processados com ferramentas nativas do SGBD e scripts Java personalizados. Os testes foram realizados em um Intel Core i5-6300HQ (4 núcleos), 16 GB de RAM e SSD, usando versões específicas de cada banco de dados (InfluxDB 1.3, Elasticsearch 5.6.3). O estudo concluiu que o InfluxDB é ideal para cenários de ingestão de dados temporais de alta velocidade, como IoT e monitoramento, mas pode não ser adequado para aplicações que exigem operações complexas de JOIN ou atualizações frequentes. A reprodutibilidade foi classificada como inexistente, sem scripts ou configurações compartilhadas.

Grzesik and Mrozek [2020] avaliaram cinco bancos de dados de séries temporais (TimescaleDB, InfluxDB, Riak TS, PostgreSQL e SQLite) em um Raspberry Pi 4 (ARM Cortex-A72, 4 GB de RAM), escolhido como dispositivo de borda de baixo consumo representativo, sendo o único estudo na literatura revisada direcionado a implantações com recursos restritos. Os testes cobriram inserções pontuais e em lote (grupos de 10) e três cenários de consulta de agregação executados 5.000 vezes cada, utilizando 28.800 leituras sintéticas de sensores (temperatura, pressão, umidade e métricas de qualidade do ar de 10 sensores simulados em uma janela de 24 horas). O PostgreSQL foi o sistema mais rápido tanto para inserção quanto para consulta nessas restrições de hardware; o InfluxDB apresentou o maior ganho relativo com o uso de lotes (melhoria de 8,65 vezes na vazão). TimescaleDB e InfluxDB foram destacados como escolhas mais adequadas quando recursos nativos de séries temporais, como políticas de retenção e agregados contínuos, são necessários em implantações de borda em produção. A reprodutibilidade foi classificada como inexistente, sem scripts ou configurações publicados.

de Leal Meireles Pereira [2023] avaliaram o desempenho do InfluxDB 2.7 e Apache IoTDB 1.1.1 em três cenários: somente nuvem, 2 borda + nuvem e 4 borda + nuvem, testando cargas de trabalho de inserção, consulta e mistas, cada uma

executada três vezes após reinicialização do sistema. As ferramentas utilizadas incluíram MulletBench para o benchmark, Ansible e Docker para orquestração, PCP para monitoramento e tcconfig para simulação de rede. Os testes foram conduzidos em nós servidores com Intel i5-9500 (16 GB de RAM, NVMe SSD) e nós clientes com Intel i3-4170 (16 GB de RAM, SATA SSD) em uma rede comutada Gigabit. Este é o único trabalho na literatura revisada classificado como totalmente reprodutível, com todas as configurações e código publicamente disponíveis no GitHub. O Apache IoTDB foi o destaque, superando o InfluxDB em todos os cenários testados.

Wang *et al.* [2023] compararam IoTDB v0.13.0 contra InfluxDB, TimescaleDB, KairosDB, Parquet e ORC em operações de escrita e leitura, custo de armazenamento e tipos de consulta incluindo consultas de intervalo, agregações e subamostragem, medindo vazão, latência e tempo de resposta. Foram utilizados tanto conjuntos de dados industriais reais (dados de veículos, trens e máquinas) quanto conjuntos sintéticos gerados pelo TSBS. Os testes foram conduzidos em ambientes standalone (Intel i7-10700, 32 GB de RAM, HDD de 10 TB) e distribuídos (Alibaba Cloud com Xeon Platinum, 256 GB de RAM), com fator de replicação 3 e particionamento temporal na configuração distribuída. Embora os detalhes de configuração sejam descritos no artigo, nenhum guia completo de reprodução foi fornecido, levando a uma classificação de reprodutibilidade parcial. O Apache IoTDB demonstrou superioridade tanto em ambientes locais quanto distribuídos, especialmente em grandes escalas industriais envolvendo trilhões de pontos de dados.

## 3.2 Análise Comparativa

A Tabela 1 agrega os principais atributos dos seis estudos de desempenho, permitindo uma análise comparativa nas dimensões mais relevantes para a metodologia de benchmark.

Os anos de publicação variam de 2017 a 2024. A maioria dos estudos é recente, com densidade particular em 2023, refletindo o crescente interesse na avaliação de BDSTs. A presença de estudos de 2017 destaca que esta não é uma preocupação emergente; no entanto, a rápida evolução dos próprios bancos de dados significa que descobertas mais antigas podem não se aplicar mais às versões atuais.

### 3.2.1 Cobertura de Bancos de Dados

O InfluxDB está presente em todos os estudos revisados, servindo como o ponto de referência de facto para comparações de BDSTs, mesmo em trabalhos onde outro sistema é o assunto principal, como em [Wang *et al.*, 2023]. A presença do TimescaleDB em quatro dos seis estudos reflete sua crescente relevância como alternativa compatível com SQL. A tendência mais notável é a emergência do Apache IoTDB: aparecendo apenas nos dois estudos mais recentes (2023), ele lidera em ambos [de Leal Meireles Pereira, 2023; Wang *et al.*, 2023], sugerindo melhorias de desempenho nas versões recentes, viés de seleção dos benchmarks mais recentes para cargas de trabalho IoT, ou um crescente reconhecimento de seus pontos fortes para esse domínio; os dados disponíveis não permitem distinguir entre essas explicações.

**Tabela 1.** Comparação de estudos de benchmark de BDSTs relacionados.

| Autor                           | Ano  | Bancos de Dados | Reprod. | Datasets         | Ferramentas                   | Líder      |
|---------------------------------|------|-----------------|---------|------------------|-------------------------------|------------|
| Lima and Lichtnow [2024]        | 2024 | SQL + NoSQL     | Parcial | Sintético        | Apache JMeter                 | InfluxDB   |
| de Leal Meireles Pereira [2023] | 2023 | NoSQL           | Total   | Sintético        | MulletBench, Ansible, Docker  | IoTDB      |
| Wang <i>et al.</i> [2023]       | 2023 | SQL + NoSQL     | Parcial | Real + Sintético | TSBS                          | IoTDB      |
| Shah <i>et al.</i> [2022]       | 2022 | SQL + NoSQL     | Nenhuma | Real + Sintético | timeseries-generator (Python) | InfluxDB   |
| Grzesik and Mrozek [2020]       | 2020 | SQL + NoSQL     | Nenhuma | Sintético        | Manual                        | PostgreSQL |
| Naqvi <i>et al.</i> [2017]      | 2017 | SQL + NoSQL     | Nenhuma | Real + Sintético | Telegraf, Kapacitor           | InfluxDB   |

### 3.2.2 Reprodutibilidade

Esta é a lacuna metodológica mais significativa na literatura revisada. Dos seis estudos de desempenho, apenas um ([de Leal Meireles Pereira, 2023]) é totalmente reprodutível, com todo o código e configurações publicamente disponíveis no GitHub. Dois estudos ([Lima and Lichtnow, 2024] e [Wang *et al.*, 2023]) oferecem reprodutibilidade parcial por meio de contêineres Docker ou configurações descritas, enquanto os dois restantes ([Shah *et al.*, 2022] e [Naqvi *et al.*, 2017]) não fornecem nenhum meio de replicação. Isso limita a validação independente dos resultados e dificulta atribuir diferenças de desempenho à arquitetura do banco de dados em vez de às condições experimentais. Este trabalho foi projetado com reprodutibilidade como requisito central. Enquanto a maioria dos estudos revisados se limita a descrever o ambiente de maneira superficial no artigo ou a publicar somente ferramentas auxiliares, como o gerador de dados de [Shah *et al.*, 2022], todos os componentes necessários para replicação estão publicamente disponíveis, incluindo scripts de orquestração, imagens Docker com versões fixadas e clientes de benchmark parametrizados. Em relação a [de Leal Meireles Pereira, 2023], único estudo totalmente reprodutível da literatura, este trabalho acrescenta um Nix flake com arquivo de lock que garante dependências determinísticas além do Docker.

### 3.2.3 Conjuntos de Dados

Todos os seis estudos utilizam conjuntos de dados sintéticos, refletindo a necessidade de dados controlados e escaláveis em volume. Três estudos incluem adicionalmente conjuntos de dados reais, como dados de táxi em NYC, leituras de sensores IoT e dados industriais de veículos [Shah *et al.*, 2022; Naqvi *et al.*, 2017; Wang *et al.*, 2023], proporcionando relevância prática ao lado do controle experimental. O TSBS (Time Series Benchmark Suite) aparece nos estudos mais recentes, sugerindo uma tendência potencial de padronização na comunidade.

### 3.2.4 Configurações de Hardware

Os ambientes testados variam consideravelmente: os processadores variam de Intel i3 a Intel i7, a RAM de 8 GB a 32 GB (mais nuvem com 256 GB em um caso [Wang *et al.*, 2023]), e o armazenamento de SATA SSD a NVMe e HDD. A maioria dos testes utilizou máquinas de médio porte, consistente com foco em cenários de implantação realistas. A inclusão de testes em nuvem em estudos mais recentes [de Leal Meireles Pereira, 2023; Wang *et al.*, 2023] reflete o crescente interesse em arquiteturas borda-nuvem.

### 3.2.5 Tendências de Resultados

O InfluxDB lidera em três dos seis estudos (2017, 2022 e 2024). O Apache IoTDB lidera em ambos os estudos de 2023. O PostgreSQL lidera no estudo de computação de

borda [Grzesik and Mrozek, 2020], resultado atribuível ao hardware restrito do Raspberry Pi em vez de características gerais de desempenho de BDSTs. O padrão temporal entre os benchmarks de propósito geral permanece: pode indicar melhorias de desempenho nas versões recentes do IoTDB, ou pode refletir o fato de que benchmarks mais recentes incluem cargas de trabalho específicas de IoT nas quais o modelo de dados nativo do IoTDB oferece uma vantagem estrutural. A mudança não invalida necessariamente o InfluxDB para outras cargas de trabalho; no entanto, motiva a inclusão de ambos os bancos de dados em qualquer avaliação comparativa contemporânea.

A literatura revisada revela três lacunas principais que o presente trabalho aborda: (1) reprodutibilidade insuficiente: apenas um dos seis estudos é totalmente replicável; (2) cobertura limitada do IoTDB junto ao InfluxDB e TimescaleDB no mesmo benchmark controlado; e (3) a ausência de uma análise sistemática de como as escolhas metodológicas (isolamento de recursos, ajuste de banco de dados) afetam as conclusões reportadas. Este estudo contribui para fechar todas as três lacunas.

## 4 Metodologia

Este trabalho adota uma abordagem experimental controlada para comparar três bancos de dados de séries temporais em nove padrões distintos de carga de trabalho IoT e três escalas de volume de dados. A avaliação cobre vazão de escrita, latência por operação (i.e., média e percentil 99) e consumo de recursos (i.e., CPU e memória) para cada combinação de banco de dados, tipo de carga de trabalho e escala. O objetivo é caracterizar quais compromissos arquiteturais favorecem quais perfis de aplicação.

Um objetivo central de projeto, além da medição de desempenho, é a reprodutibilidade. A literatura comparativa revisada neste trabalho revela uma lacuna sistêmica: dos cinco estudos de desempenho recentes de BDSTs analisados, apenas um publicou scripts e configurações suficientes para replicação independente [de Leal Meireles Pereira, 2023]. Os demais omitiram as configurações completamente ou as descreveram com poucos detalhes para serem seguidas mecanicamente. Este trabalho trata a reprodutibilidade como prioridade central: cada componente necessário para replicar os resultados tem sua versão fixada e está publicamente disponível junto com este artigo.

### 4.1 Seleção dos Bancos de Dados

Os três bancos de dados foram selecionados para representar abordagens arquiteturais distintas para o armazenamento de séries temporais, sendo cada um deles relevante na literatura revisada.

#### 4.1.1 InfluxDB 2.7

O InfluxDB serve como uma base comum na pesquisa comparativa de BDSTs, aparecendo em quatro dos cinco estudos de desempenho revisados [Naqvi *et al.*, 2017; Shah *et al.*, 2022; Lima and Lichtnow, 2024; Wang *et al.*, 2023]. Ele usa um motor de armazenamento TSM (Time-Structured Merge Tree) que organiza dados em blocos comprimidos ordenados por tempo, com escritas absorvidas em cache em memória e compactadas periodicamente em arquivos imutáveis em disco. A ingestão ocorre via API HTTP com protocolo de linha; consultas analíticas usam a linguagem de expressão Flux. O InfluxDB mantém um Índice de Séries Temporais (TSI) para localização de séries; a memória em processo é dominada pelo cache TSM, pelo TSI e pela sobrecarga do runtime Go. A presença frequente do InfluxDB em benchmarks anteriores o torna um ponto de referência útil: qualquer novo estudo que o omita perde comparabilidade com a literatura existente.

#### 4.1.2 Apache IoTDB 1.3.0

O Apache IoTDB [Wang *et al.*, 2023] emergiu como líder de desempenho nos dois estudos comparativos mais recentes [de Leal Meireles Pereira, 2023; Wang *et al.*, 2023], representando uma mudança em relação à dominância do InfluxDB reportada em benchmarks mais antigos. Desenvolvido na Universidade Tsinghua especificamente para IoT industrial, usa o TsFile, um formato de arquivo colunar personalizado que armazena dados em blocos ordenados com estatísticas de mínimo/máximo por bloco. As escritas são absorvidas em buffer no heap gerenciado pela JVM e descarregadas assincronamente para segmentos TsFile, em uma estratégia similar a uma Log-Structured Merge Tree (LSM) [O'Neil *et al.*, 1996]. A ingestão nativa usa um protocolo binário Thrift RPC (Session API), submetendo tablets colunares tipados diretamente no buffer de escrita. As estatísticas por bloco permitem a eliminação antecipada de blocos irrelevantes em consultas de filtro de valor, vantagem que cresce com o volume de dados. Sua rápida ascensão em benchmarks recentes, combinada com essa abordagem arquitetural distinta, tornou-o adequado para este estudo.

#### 4.1.3 TimescaleDB

O TimescaleDB estende o PostgreSQL 15 com capacidades de séries temporais: particionamento automático de hipertabela por intervalo de tempo, a função de agregação `time_bucket()` para subamostragem e compressão opcional por bloco. Ele mantém o motor relacional completo com durabilidade baseada em WAL, controle de concorrência MVCC, índices B-tree e compatibilidade completa com SQL. Os índices BRIN (Block Range INdex) fornecem eliminação baseada em intervalos para predicados de valor. Para organizações com infraestrutura PostgreSQL existente, ou para aplicações que exigem junções entre dados de séries temporais e relacionais, ele fornece um caminho para a capacidade de séries temporais sem abandonar o ecossistema relacional. Aparece em quatro dos seis benchmarks revisados [Shah *et al.*, 2022; Lima and Lichtnow, 2024; Wang *et al.*, 2023; Grzesik and Mrozek, 2020].

O PostgreSQL, sem a extensão TimescaleDB, foi explicitamente excluído. O TimescaleDB tende a superar o PostgreSQL neste domínio; incluir ambos duplicaria os tes-

tes. Conclusões sobre configurações gerais do PostgreSQL, como dimensionamento de `shared_buffers`, são transferíveis, mas resultados que dependem de otimizações específicas do TimescaleDB, como hipertabelas e eliminação de blocos por estatísticas de mínimo/máximo, não se aplicam diretamente ao PostgreSQL sem a extensão.

A Tabela 2 sumariza as principais características dos três bancos selecionados.

#### 4.2 Ferramenta de Benchmark

O gerador de carga de trabalho é o *iot-benchmark* [Tsinghua University IoTDB Team, 2023], que vem com clientes específicos para os três bancos de dados em avaliação. Esse suporte nativo a múltiplos bancos de dados significa que cada carga de trabalho é expressa no protocolo correto para cada alvo: o modo `SESSION_BY_TABLET` do *iot-benchmark*, que usa a Session API Thrift RPC do IoTDB, o protocolo de linha HTTP para InfluxDB e uma conexão JDBC para TimescaleDB. Uma interface consistente e parametrizada garante que as diferenças de desempenho medidas reflitam os próprios bancos de dados, e não a camada de benchmarking. O *iot-benchmark* é fundamentado no YCSB (*Yahoo! Cloud Serving Benchmark*), adaptando sua filosofia de cargas de trabalho parametrizadas ao contexto de séries temporais IoT [Apache IoTDB Community, 2024a].

Ferramentas alternativas foram avaliadas. O Time Series Benchmark Suite (TSBS) oferece um framework maduro para múltiplos bancos de dados, mas carece de suporte nativo ao IoTDB no momento desta escrita. O Apache JMeter lida bem com sistemas baseados em HTTP, mas requer scripts personalizados para interfaces Thrift e JDBC, introduzindo potencial sobrecarga específica da ferramenta. A interface uniforme do *iot-benchmark* e a semântica de carga de trabalho específica para IoT (i.e., clientes concorrentes organizados em hierarquias de dispositivos e sensores, tamanhos de lote configuráveis e geração de dados fora de ordem distribuída por Poisson) tornaram-no a escolha mais adequada para este estudo. Cabe registrar, no entanto, que o *iot-benchmark* é desenvolvido pelo mesmo grupo de pesquisa responsável pelo Apache IoTDB [Tsinghua University IoTDB Team, 2023], o que representa um potencial viés em favor desse banco de dados; os resultados relativos ao IoTDB devem ser interpretados com essa ressalva.

A ferramenta produz duas matrizes de saída estruturadas que são analisadas pela camada de orquestração. A Matriz de Resultado reporta operações totais, tempo decorrido e vazão agregada (pts/s). A Matriz de Latência reporta latências média, mínima, máxima e por percentil por tipo de operação, fornecendo os dados de comportamento de cauda por operação necessários para avaliar cargas de trabalho IoT em tempo real.

#### 4.3 Projeto das Cargas de Trabalho

O benchmark cobre nove tipos de carga de trabalho em duas categorias, projetados para abranger o espectro de padrões encontrados em sistemas IoT em produção.

##### 4.3.1 Testes de Escrita

Quatro testes de escrita exercitam padrões de ingestão distintos:

**Tabela 2.** Comparação entre os bancos de dados de séries temporais avaliados.

| Banco de Dados | Arquitetura                             | Armazenamento                    | Protocolo                  | Consistência  |
|----------------|-----------------------------------------|----------------------------------|----------------------------|---------------|
| IoTDB          | TSDB nativa orientada a colunas         | TsFile (formato proprietário)    | IoTDB-SQL, MQTT, REST      | Eventual      |
| InfluxDB       | TSDB nativa com mecanismo TSM           | TSM (Time-Structured Merge Tree) | Line Protocol, HTTP/Flux   | Eventual      |
| TimescaleDB    | Extensão do PostgreSQL com hipertabelas | Herda estruturas do PostgreSQL   | Protocolo PostgreSQL, HTTP | ACID completa |

**write** Ingestão sequencial de linha de base com registros ordenados cronologicamente e tamanho de lote padrão de 100 linhas por requisição. Mede a vazão bruta de escrita em condições ideais e ordenadas.

**out-of-order** Vinte por cento dos pontos de dados chegam indexados com tempo anterior à fronteira de ingestão atual, modelado como um atraso distribuído por Poisson. Simula sensores IoT que perdem conectividade e depois reproduzem leituras armazenadas em buffer, um cenário típico em implantações industriais e de campo onde os sensores operam sobre redes celulares ou LPWAN não confiáveis.

**batch-small** BATCH\_SIZE=1: um ponto de dado por requisição de escrita. Isola a sobrecarga de protocolo por requisição: uma rodada HTTP mais análise do protocolo de linha para InfluxDB, um ciclo completo cliente-servidor PostgreSQL mais flush do WAL para TimescaleDB e uma única chamada Thrift RPC para IoTDB.

**batch-large** BATCH\_SIZE=1000: 1.000 pontos de dados por requisição. Testa a eficiência de carregamento em lote e revela o teto prático da arquitetura de lote de cada banco de dados. A razão entre a vazão de **batch-small** e **batch-large** quantifica diretamente quanto a sobrecarga por requisição domina sobre o custo de transferência de dados.

Cada teste de escrita limpa e repopula o banco de dados antes de ser executado, garantindo medições independentes e sem histórico.

### 4.3.2 Testes de Leitura

Cinco testes de leitura cobrem os padrões de consulta mais comuns em pipelines de análise IoT:

**latest-point** Recupera o valor mais recente para cada sensor. Esta é a consulta fundamental de monitoramento em tempo real: um painel ou sistema de alertas que consulta continuamente o estado atual do dispositivo.

**downsample** Executa consultas GROUP BY de janela de tempo calculando médias em janelas de tempo fixas. Modela o caso de uso padrão de agregação em painel, onde leituras brutas de sensores são consolidadas em resumos de nível de minuto ou hora.

**range-query** Recupera todos os pontos de dados dentro de uma janela de tempo especificada, sem filtragem por valor. Modela exportação de dados históricos, pipelines de reprodução e ingestão para retraining de modelos.

**value-filter** Aplica predicados de valor como `temperature > 85.0`, incluindo agregações sobre intervalos filtrados por valor. Modela alertas por limite: “encontrar todas as leituras de sensores que excedem o nível de alarme na última hora.” O filtro de valor é o tipo de consulta mais discriminatório arquiteturalmente neste benchmark porque não pode ser respondido a partir de índices baseados em tempo: cada

banco de dados deve manter um índice secundário ou varrer todos os valores armazenados.

**read** Executa todos os dez tipos de consulta suportados pelo `iot-benchmark` com peso igual, fornecendo uma linha de base abrangente de carga de trabalho mista.

Todos os testes de leitura operam no conjunto de dados carregado pelo teste de escrita de linha de base anterior.

## 4.4 Parâmetros de Escala

Três escalas de volume de dados são definidas para expor como as características de desempenho evoluem com o tamanho do conjunto de dados. A Tabela 3 resume os parâmetros.

A escala grande contém aproximadamente 20 vezes mais dados que a média e cem vezes mais que a pequena, impulsionada tanto por uma contagem de laços maior quanto por uma topologia expandida (10 clientes  $\times$  50 dispositivos  $\times$  20 sensores versus  $5 \times 10 \times 10$ ). A medição em múltiplas escalas é essencial porque vários fenômenos – como a saturação de compilação JIT da JVM, o esgotamento de buffer, as paralisações de checkpoint e o escalonamento de varredura de índice – só se manifestam ou se tornam dominantes em volumes de dados mais elevados, como mostrado nos resultados da Seção 5.4. Um benchmark de escala única arrisca tirar conclusões que se aplicam apenas a um intervalo estreito de dados.

A escolha de cada nível reflete cenários de implantação IoT. A escala pequena representa ambientes com poucos sensores e baixa frequência de amostragem, comuns em provas de conceito ou instalações residenciais e serve principalmente para verificar a correção funcional e medir a sobrecarga base de cada banco de dados. A escala média corresponde a instalações industriais de porte moderado, em que o volume acumulado começa a pressionar os mecanismos de compressão e os tamanhos de buffer internos. A escala grande simula ambientes de alta densidade, como cidades inteligentes ou plantas industriais, onde o volume de dados é um gargalo e diferenças arquiteturais entre os bancos de dados se tornam mais pronunciadas.

## 4.5 Ambiente e Reprodutibilidade

A reprodutibilidade neste projeto repousa em três componentes.

### 4.5.1 Docker Compose

Os experimentos foram conduzidos de forma sequencial, com apenas um banco de dados ativo por vez, garantindo isolamento completo entre as medições e eliminando interferência de cargas concorrentes. O Docker Compose [Docker Inc., 2025] gerencia os contêineres de banco de dados com versões de imagem fixadas: `apache/iotdb:1.3.0-standalone`, `influxdb:2.7-alpine` e `timescale/timescaledb:latest-pg15`. Os contêineres são iniciados e parados sequencialmente: apenas um banco de dados está ativo a qualquer momento. Esse design,

**Tabela 3.** Parâmetros de escala do benchmark. Todas as escalas usam um tamanho de lote padrão de 100 linhas por requisição. A escala grande contém aproximadamente cem vezes mais dados que a pequena.

| Escala  | Clientes | Dispositivos | Sensores | Laços | Duração |
|---------|----------|--------------|----------|-------|---------|
| Pequena | 5        | 10           | 10       | 1.000 | ~5 min  |
| Média   | 5        | 10           | 10       | 5.000 | ~30 min |
| Grande  | 10       | 50           | 20       | 5.000 | ~2 h    |

introduzido após as execuções iniciais revelarem artefatos de contenção de recursos entre bancos de dados, dá a cada banco de dados acesso exclusivo à CPU, largura de banda de memória e entrada/saída do host.

#### 4.5.2 Nix Flake

O Nix flake [NixOS Contributors, 2025] é uma ferramenta opcional que fornece um ambiente de desenvolvimento declarativo com arquivo de lock fixado. Executar `nix develop` inicia um shell com versões exatamente especificadas de Java 17, Maven, Python 3 e ferramentas Docker. Executar `nix run .#build` compila todos os três clientes de benchmark em uma única etapa. Toda dependência transitiva de construção é registrada no arquivo de lock do flake, eliminando as falhas de variação de ambiente que tipicamente prejudicam replicações de benchmark. Usuários sem Nix podem replicar os experimentos instalando manualmente as dependências listadas (Java 17, Maven, Python 3 e Docker) em qualquer host Linux compatível.

#### 4.5.3 CLI Python

A CLI Python (`benchmark.py`) orquestra o fluxo de trabalho completo do benchmark programaticamente. Ele inicia o contêiner alvo, escreve os parâmetros corretos no arquivo `conf/config.properties` do cliente de benchmark, invoca o cliente, consulta `docker stats` em intervalos de um segundo para coletar amostras de CPU e memória, e acrescenta uma linha de resultado estruturada à tabela de saída após cada teste. Uma execução completa de nove testes e três bancos de dados em qualquer escala se resume a:

```
python3 benchmark.py --scale large
```

Esse nível de automação torna os resultados rastreáveis a um ambiente de execução específico e versionado. Qualquer pesquisador com o repositório e o Docker disponíveis pode reproduzir o conjunto de dados completo do zero.

## 4.6 Pacote de Replicação

O ambiente completo de benchmark está publicamente disponível em <https://github.com/LuizFerK/iot-benchrunner>. Instruções detalhadas de configuração e execução encontram-se no repositório.

## 4.7 Métricas

A Tabela 4 descreve as métricas de desempenho coletadas para cada execução.

## 4.8 Ajuste de Configuração

Após completar as execuções iniciais de benchmark, duas melhorias metodológicas foram aplicadas antes de reexecutar todas as escalas:

1. **Ciclo de vida sequencial de contêineres.** A configuração original iniciava todos os três contêineres de banco

de dados simultaneamente. Embora cada teste fosse executado em um banco de dados por vez, os contêineres ociosos competiam pelo cache de páginas do SO e largura de banda de entrada/saída [Shapira and Chen, 2016]. O TimescaleDB acumulou vários gigabytes de cache de páginas durante as sessões de escrita do IoTDB, produzindo medições de memória enganosas. A JVM do IoTDB pré-alocou seu heap máximo configurado durante o tempo ocioso pela mesma razão. A configuração corrigida inicia cada contêiner imediatamente antes de seus testes e o para imediatamente após.

2. **Ajuste no nível do banco de dados.** O TimescaleDB recebeu uma configuração `pgtune` [Vasiliev, 2025] para PostgreSQL 15 direcionada ao hardware desta máquina: `shared_buffers=8GB`, `work_mem=27413kB`, `maintenance_work_mem=2GB`, `effective_io_concurrency=1000` e `checkpoint_completion_target=0.9`. Essas configurações são a linha de base padrão para qualquer implantação PostgreSQL em produção nesta classe de hardware; os padrões são ajustados para ambientes com RAM mínima e produzem desempenho artificialmente ruim em máquinas capazes. O teto do cache de escrita TSM do InfluxDB foi elevado do padrão de 1 GB para 8 GB para corresponder à RAM disponível. Um limite de 28 GB de memória foi aplicado a cada contêiner.

Tanto os resultados originais quanto os ajustados são reportados e analisados na escala pequena, onde o contraste é mais instrutivo. Os resultados de escala média e grande utilizam a metodologia ajustada ao longo. A comparação entre as execuções original e ajustada serve a um duplo propósito: quantifica a magnitude dos artefatos de contenção e valida quais características de desempenho são propriedades arquiteturais genuínas de cada banco de dados.

# 5 Experimentos

Todos os benchmarks foram executados em uma única máquina host com as especificações listadas na Tabela 5. Os três contêineres de banco de dados e o cliente Java de benchmark compartilharam essa máquina sem cargas de trabalho concorrentes durante as sessões de teste.

Os valores de utilização de CPU são expressos como porcentagem de um único núcleo lógico, consistente com as convenções de reporte do `docker stats`. Um valor de 600% indicaria utilização total de todos os seis núcleos físicos; valores por banco de dados na faixa de 20–100% indicam carga compatível com o orçamento computacional disponível.

## 5.1 Desempenho de Escrita

Esta seção analisa o desempenho de escrita em três cenários: ingestão sequencial de linha de base, que mede a vazão má-

**Tabela 4.** Métricas de desempenho coletadas por execução de benchmark.

| Métrica                 | Fonte                                        |
|-------------------------|----------------------------------------------|
| Vazão (pts/s)           | Matriz de Resultado do <i>iot-benchmark</i>  |
| Latência média (ms)     | Matriz de Latência do <i>iot-benchmark</i>   |
| Latência P99 (ms)       | Matriz de Latência do <i>iot-benchmark</i>   |
| CPU média/pico (%)      | <code>docker stats</code> (intervalo de 1 s) |
| Memória média/pico (MB) | RSS do <code>docker stats</code>             |
| Tempo de parede (s)     | Duração da execução                          |

**Tabela 5.** Hardware e versões de software do ambiente de teste.

| Componente  | Detalhes                                                                        |
|-------------|---------------------------------------------------------------------------------|
| CPU         | Intel Core i5-11400F (6c/12t, 2,60–4,40 GHz)                                    |
| RAM         | 32 GB DDR4                                                                      |
| Disco       | 500 GB NVMe SSD                                                                 |
| SO          | NixOS (Linux 6.12)                                                              |
| IoTDB       | <code>apache/iotdb:1.3.0-standalone</code>                                      |
| InfluxDB    | <code>influxdb:2.7-alpine</code>                                                |
| TimescaleDB | <code>timescale/timescaledb:latest-pg15</code>                                  |
| Benchmark   | <code>iot-benchmark</code> [Tsinghua University IoTDB Team, 2023] (Java, Maven) |

xima sustentada; sensibilidade ao tamanho do lote, que avalia o impacto do agrupamento de pontos por requisição; e ingestão fora de ordem, que testa o comportamento sob chegada temporalmente desordenada de dados.

### 5.1.1 Ingestão Sequencial de Linha de Base

A Tabela 6 mostra o desempenho de escrita de linha de base na escala pequena sob a configuração ajustada. O IoTDB atinge 2,16 M pts/s, aproximadamente 5 vezes o InfluxDB e 12 vezes o TimescaleDB.

Os resultados são consistentes com três fatores arquitetais cumulativos.

**Profundidade do protocolo.** O *iot-benchmark* se conecta ao IoTDB via Session API Thrift RPC no modo `SESSION_BY_TABLET`, submetendo tablets colunares tipados (i.e., agrupamentos de múltiplos pontos de séries temporais em estrutura colunar) diretamente por uma conexão TCP persistente. Uma `syscall` submete um lote completo diretamente no buffer de escrita do `TsFile`. O InfluxDB requer uma requisição HTTP por escrita, incluindo gerenciamento de conexão, análise de texto do protocolo de linha e pesquisa de índice TSI. O TimescaleDB roteia cada escrita pela pilha completa do PostgreSQL: TCP, análise SQL, planejamento de consulta, roteamento de hipertabela, escrita no WAL e formatação de tupla MVCC. Cada camada adiciona latência; a pilha PostgreSQL adiciona mais.

**Buffering em memória e compromissos de durabilidade.** O IoTDB acumula tablets em seu heap JVM e reconhece escritas antes da persistência em disco, descarregando para `TsFile` assincronamente quando o memtable se enche. Isso troca explicitamente a garantia de durabilidade por vazão, um compromisso que deve ser avaliado por implantação. O cache TSM do InfluxDB opera de forma similar, mas é limitado (8 GB na configuração ajustada). O TimescaleDB segue a semântica padrão do PostgreSQL: cada escrita é descarregada pelo WAL antes do reconhecimento, fornecendo a garantia de durabilidade mais forte ao custo de entrada/saída síncrona por escrita.

**Manutenção de índices.** O IoTDB não mantém índices

síncronos no caminho de escrita, os dados vão diretamente para o buffer em memória sem atualização de estruturas auxiliares. O InfluxDB atualiza o Índice de Séries Temporais (TSI) a cada escrita. O TimescaleDB mantém índices B-tree e BRIN que crescem com o volume de dados, e o processo de autovacuum do PostgreSQL, responsável por recuperar espaço de registros obsoletos, adiciona sobrecarga à medida que a hipertabela se expande.

A combinação desses três fatores sugere que a vantagem de escrita do IoTDB tem base arquitetural: o protocolo binário, o buffering assíncrono e a ausência de manutenção de índice síncrona atuam de forma cumulativa, não sendo eliminados por ajuste de configuração. Essa hierarquia de desempenho pode ser observada na Figura 4, que consolida os resultados de todas as quatro cargas de trabalho de escrita na escala pequena.

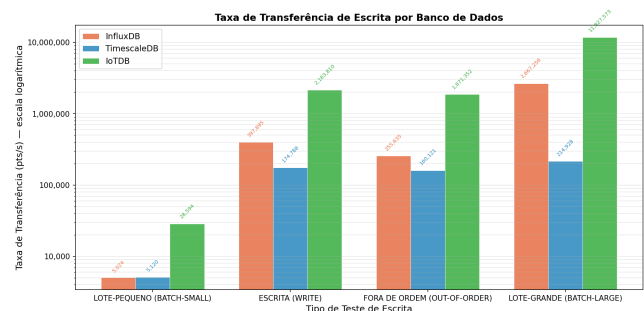


**Figura 4.** Vazão de escrita (pts/s, escala logarítmica) para todas as quatro cargas de trabalho de escrita na escala pequena (ajustado). O IoTDB lidera em todos os tipos de teste.

### 5.1.2 Sensibilidade ao Tamanho do Lote

A Tabela 7 mostra como a vazão muda entre `BATCH_SIZE=1` (uma requisição de escrita por ponto) e `BATCH_SIZE=1000` (1.000 pontos por requisição).

IoTDB e InfluxDB mostram aceleração de 413–531 vezes, refletindo o alto custo da sobrecarga por requisição. Para o InfluxDB, trata-se de uma rodada HTTP mais sobrecarga de conexão TCP; para o IoTDB, é o custo fixo de uma chamada

**Tabela 6.** Desempenho de escrita de linha de base na escala pequena (ajustado). Vazão em pts/s.

| BD          | pts/s     | Lat. Média | Lat. P99  | CPU Média |
|-------------|-----------|------------|-----------|-----------|
| IoTDB       | 2.163.809 | 1,73 ms    | 1,18 ms   | 5,1%      |
| InfluxDB    | 397.894   | 12,00 ms   | 190,50 ms | 8,5%      |
| TimescaleDB | 174.787   | 27,70 ms   | 223,00 ms | 22,5%     |

**Tabela 7.** Sensibilidade ao tamanho do lote na escala pequena (ajustado). Vazão em pts/s.

| BD          | LOTE=1 | LOTE=1000  | Razão |
|-------------|--------|------------|-------|
| IoTDB       | 28.593 | 11.827.572 | 413×  |
| InfluxDB    | 5.024  | 2.667.256  | 531×  |
| TimescaleDB | 5.120  | 214.927    | 42×   |

Thrift por tablet. A razão de 42 vezes do TimescaleDB é marcadamente menor porque o flush do WAL do PostgreSQL já domina no LOTE=1: a sobrecarga por linha é tão alta que lotes maiores fornecem benefício proporcionalmente menor; o gargalo passa da sobrecarga de protocolo para a entrada/saída de armazenamento.

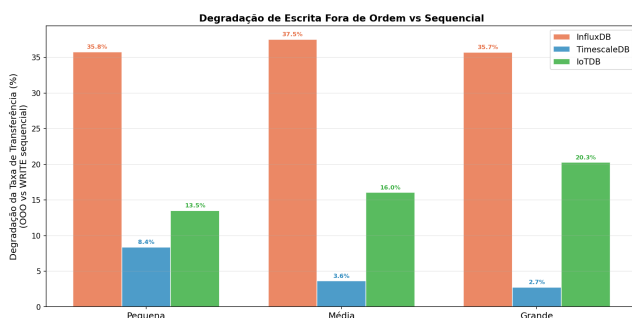
Este resultado tem significado prático direto. Uma aplicação IoT que envia uma leitura de sensor por requisição HTTP ao InfluxDB atingirá vazão 531 vezes menor do que aquela que armazena localmente em buffer e escreve em lotes. Para InfluxDB e IoTDB, agregar escritas em lotes é a otimização de maior impacto disponível.

### 5.1.3 Ingestão Fora de Ordem

A Tabela 8 mostra como cada banco de dados lida com a carga de trabalho de 20% de dados com chegada atrasada.

O TimescaleDB mostra a menor degradação relativa, resultado consistente com a independência de ordem das atualizações de índice B-tree do PostgreSQL. A penalidade de 13,5% do IoTDB é compatível com o custo de compactação de entradas desordenadas na árvore LSM para restaurar a ordem exigida pelos blocos TsFile. A maior penalidade relativa do InfluxDB (35,7%) é consistente com regravações de blocos TSM necessárias para acomodar pontos fora de sequência.

Em vazão absoluta, o IoTDB ainda lidera amplamente sob condições fora de ordem (1,87 M vs. 255 K pts/s para InfluxDB). Portanto, duas dimensões devem ser consideradas separadamente: vazão absoluta fora de ordem (IoTDB lidera em cada escala) e degradação relativa por desordem de escrita (TimescaleDB é o mais resiliente com a menor penalidade fracional). A Figura 5 ilustra essas degradações relativas em todas as três escalas.

**Figura 5.** Degradação de vazão fora de ordem (%) relativa à escrita sequencial de linha de base, em todas as três escalas. O B-tree do TimescaleDB permanece estável; as regravações de blocos TSM do InfluxDB consistentemente incorrem na maior penalidade relativa.

## 5.2 Desempenho de Leitura

A Tabela 9 apresenta os resultados consolidados de escrita (vazão em pts/s) e leitura de último valor (latência em ms) por banco de dados e escala, permitindo uma visão comparativa rápida das duas dimensões de desempenho avaliadas.

A vazão em pts/s varia conforme o número de pontos retornados por consulta, o que torna esse valor incomparável entre tipos de leitura diferentes. A latência é usada como métrica primária nesta seção. Vale notar também que os testes de leitura são executados sequencialmente no mesmo conjunto de dados, de modo que testes posteriores se beneficiam de caches de páginas do SO mais quentes. Quando esse aquecimento de cache é o próprio mecanismo por trás do desempenho de um banco de dados, como ocorre com o `shared_buffers` do TimescaleDB na subamostragem, o resultado é esperado e reproduzível em produção (ver Seção 6.1).

### 5.2.1 Consultas de Último Valor

A Tabela 10 mostra a latência média de último valor em todas as três escalas para as execuções ajustadas.

TimescaleDB e IoTDB mantêm latência abaixo de um milissegundo ou próxima disso em todo o crescimento de cem vezes no volume de dados. Ambos exploram atalhos específicos do banco de dados para recuperação do último valor. O comportamento do TimescaleDB é consistente com seu particionamento de hipertabela ordenado por tempo [Timescale, Inc., 2024], onde consultas de último valor precisam acessar apenas o bloco mais recente, independentemente do volume total armazenado. O IoTDB mantém um cache de último valor em memória dedicado por série, servindo a consulta inteiramente da RAM sem acessar o TsFile [Apache IoTDB Community, 2024b].

O InfluxDB não tem atalho equivalente, assim, deve buscar o final dos blocos de arquivo TSM de cada série, e esse custo de busca cresce à medida que mais arquivos TSM se acumulam ao longo do tempo de vida do conjunto de dados. A progressão resultante (9,24 ms na escala pequena, 21,72 ms na média, 23,24 ms na grande) confirma uma relação aproximadamente linear entre dados armazenados e custo de último valor. Na janela de escalas avaliada, a latência de último valor do InfluxDB mostrou crescimento aproximadamente linear com o volume de dados armazenados. Em implantações sem política ativa de retenção ou downsampling, esse padrão pode torná-lo desvantajoso para monitoramento em tempo real de alta frequência; implantações que apliquem essas políticas podem mitigar esse comportamento. Esse comportamento divergente entre os bancos de dados é visualizado na Figura 6.

### 5.2.2 Subamostragem

A Tabela 11 mostra a latência de subamostragem por janela de tempo com GROUP BY.

A agregação `time_bucket()` do TimescaleDB permanece abaixo de 1 ms em todo o crescimento de cem vezes no

**Tabela 8.** Penalidade fora de ordem: vazão (pts/s) e degradação relativa em relação à escrita de linha de base, escala pequena (ajustado).

| BD          | ESCRITA (pts/s) | FORA-ORDEM (pts/s) | Penalidade |
|-------------|-----------------|--------------------|------------|
| IoTDB       | 2.163.809       | 1.871.351          | −13,5%     |
| InfluxDB    | 397.894         | 255.634            | −35,7%     |
| TimescaleDB | 174.787         | 160.121            | −8,4%      |

**Tabela 9.** Resultados consolidados de escrita e leitura por banco de dados e escala.

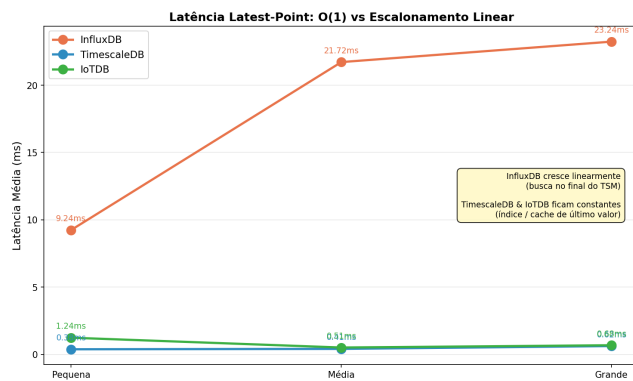
| BD                                                            | Tipo    | Pequena   | Média     | Grande     |
|---------------------------------------------------------------|---------|-----------|-----------|------------|
| <i>Escrita: vazão de linha de base (pts/s)</i>                |         |           |           |            |
| IoTDB                                                         | Escrita | 2.163.809 | 7.991.224 | 23.607.475 |
| InfluxDB                                                      | Escrita | 397.894   | 392.975   | 1.256.602  |
| TimescaleDB                                                   | Escrita | 174.787   | 175.685   | 400.455    |
| <i>Leitura: latência de último valor (ms), menor é melhor</i> |         |           |           |            |
| TimescaleDB                                                   | Leitura | 0,38      | 0,41      | 0,62       |
| IoTDB                                                         | Leitura | 1,24      | 0,51      | 0,68       |
| InfluxDB                                                      | Leitura | 9,24      | 21,72     | 23,24      |

**Tabela 10.** Latência média de último valor (ms) por escala (ajustado). Menor é melhor.

| BD          | Pequena | Média | Grande |
|-------------|---------|-------|--------|
| TimescaleDB | 0,38    | 0,41  | 0,62   |
| IoTDB       | 1,24    | 0,51  | 0,68   |
| InfluxDB    | 9,24    | 21,72 | 23,24  |

**Tabela 11.** Latência média de subamostragem (ms) por escala (ajustado). Menor é melhor.

| BD          | Pequena | Média | Grande |
|-------------|---------|-------|--------|
| TimescaleDB | 0,61    | 0,52  | 0,73   |
| IoTDB       | 1,81    | 0,64  | 0,95   |
| InfluxDB    | 5,71    | 5,65  | 9,22   |

**Figura 6.** Latência média de último valor (ms) por escala. TimescaleDB e IoTDB permanecem próximos de zero (caminhos de acesso O(1)); o InfluxDB cresce linearmente à medida que a contagem de arquivos TSM aumenta.

volume de dados, resultado que se repete de forma consistente em todas as escalas testadas. Esse resultado é consistente com o mecanismo de exclusão de blocos de hipertabela do TimescaleDB [Timescale, Inc., 2024], pelo qual apenas os blocos que se sobrepõem à janela de tempo consultada são acessados, independentemente do volume total armazenado. O `shared_buffers` aquecido nas execuções ajustadas reforça esse comportamento, mantendo os blocos relevantes em memória. Essa combinação torna o TimescaleDB adequado para cargas de trabalho de painel que consultam consistentemente janelas de tempo recentes.

A latência de subamostragem do IoTDB melhora da escala pequena para a média (1,81 ms → 0,64 ms) à medida que a execução mais longa na escala média dá ao compilador JIT da JVM tempo para otimizar o caminho quente de agregação de blocos [Barrett *et al.*, 2017]. Na escala grande, permanece abaixo de 1 ms. O InfluxDB incorre em maior sobrecarga por consulta (5–9 ms). Esse comportamento é ilustrado de forma comparativa na Tabela 11.

### 5.2.3 Consultas de Filtro de Valor

Consultas de filtro de valor (predicados sobre valores de sensores como `temperature > 85.0`) são a carga de trabalho mais discriminatória arquiteturalmente neste estudo. Índices baseados em tempo não podem respondê-las diretamente; cada motor deve varrer todos os valores armazenados ou explorar uma estrutura de dados secundária para eliminar blocos de armazenamento que não correspondem.

A Tabela 12 mostra a latência de filtro de valor e seu comportamento de escalonamento por escala.

A vantagem do IoTDB é grande e melhora com a escala. Cada bloco `TsFile` armazena estatísticas de mínimo/máximo por série. Um predicado de valor pode pular blocos inteiros cujo intervalo armazenado não intercepta o limiar do predicado, fornecendo indexação secundária leve na camada de armazenamento. Na escala grande, com mais dados por bloco, uma fração maior de blocos pode ser eliminada, produzindo a razão de escalonamento sublinear (5,2 vezes para dados cem vezes maiores). Esse é o comportamento esperado de uma estrutura de salto baseada em estatísticas: a razão de eliminação melhora à medida que os blocos ficam mais densos.

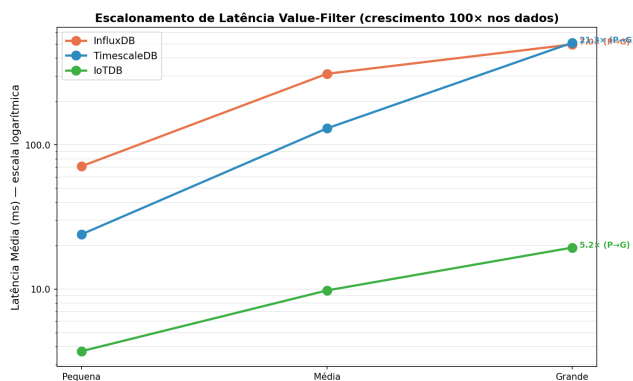
O TimescaleDB usa o BRIN (Block Range INdex) do PostgreSQL, que registra valores de mínimo/máximo por intervalo de bloco em disco. O BRIN fornece eliminação com granularidade mais ampla do que as estatísticas por bloco do IoTDB e degrada em taxa superlinear (21,3 vezes para dados cem vezes maiores). Na escala grande, o TimescaleDB fica em média 510 ms por consulta de filtro de valor, uma desvantagem de 26 vezes relativa ao IoTDB. Essa carga de trabalho também leva o TimescaleDB à sua maior utilização de CPU (77% médio, 83% de pico), indicando que o executor de consultas é o gargalo.

O InfluxDB não tem estrutura de eliminação baseada em estatísticas, mas se beneficia de alguma filtragem de granularidade reduzida no nível de arquivo baseada em metadados

**Tabela 12.** Latência média de filtro de valor (ms) por escala (ajustado). A coluna de razão mostra o crescimento de latência da escala pequena para a grande (dados com vezes maiores).

| BD          | Pequena | Média  | Grande | Razão P→G |
|-------------|---------|--------|--------|-----------|
| IoTDB       | 3,72    | 9,80   | 19,42  | 5,2×      |
| TimescaleDB | 23,98   | 130,08 | 510,05 | 21,3×     |
| InfluxDB    | 71,09   | 310,75 | 496,81 | 7,0×      |

de intervalo de tempo no índice TSM, produzindo uma razão de escalonamento de 7 vezes, significativamente melhor do que a do TimescaleDB, mas ainda 10 vezes pior do que a do IoTDB em termos absolutos na escala grande. Conforme ilustrado na Figura 7, a divergência entre os bancos de dados se acentua com o crescimento da escala.

**Figura 7.** Latência média de filtro de valor (ms, escala logarítmica) por escala. As estatísticas de mínimo/máximo por bloco do IoTDB produzem crescimento sublinear (5,2 vezes para dados com vezes maiores); o índice BRIN do TimescaleDB degrada superlinearmente (21,3 vezes).

### 5.2.4 Consultas de Intervalo

As consultas de intervalo recuperam todos os pontos de dados dentro de uma janela de tempo sem filtragem por valor. A Tabela 13 consolida as latências medidas por banco de dados e escala.

**Tabela 13.** Latências de consultas de intervalo por banco de dados e escala (em ms).

| BD          | Pequena | Média | Grande |
|-------------|---------|-------|--------|
| TimescaleDB | 0,54    | —     | 0,66   |
| IoTDB       | 1,19    | —     | 1,21   |
| InfluxDB    | 5,41    | —     | 8,68   |

O TimescaleDB é o mais rápido, com 0,54 ms na escala pequena e 0,66 ms na grande, beneficiando-se da eliminação de blocos de hipertabela (i.e., apenas blocos que sobrepõem a janela de consulta são abertos). O IoTDB atinge 1,19–1,21 ms e permanece estável entre as escalas dentro deste intervalo de dados. O InfluxDB atinge 5,41–8,68 ms, onde o scanner de blocos TSM suporta maior sobrecarga por arquivo do que a varredura sequencial no nível de bloco do TimescaleDB.

### 5.3 Footprint de Memória

As medições de memória requerem interpretação cuidadosa porque as execuções originais do benchmark com início compartilhado produziram valores substancialmente inflados por um artefato metodológico.

O TimescaleDB reportou aproximadamente 5,4 GB desde o primeiro teste nas execuções originais. Isso não ocor-

reu devido ao pool de buffers do PostgreSQL, mas sim ao cache de páginas do sistema operacional, que havia sido acumulado durante as sessões de escrita anteriores do IoTDB. O Linux contabiliza o cache de páginas do sistema de arquivos como parte do RSS de um contêiner no `docker stats`, e o contêiner ocioso do TimescaleDB estava em execução há horas enquanto o IoTDB gerava gigabytes de entrada/saída. Nas execuções sequenciais ajustadas, o TimescaleDB começa em 239 MB e cresce para aproximadamente 1,8 GB após os testes de escrita aquecerem o `shared_buffers`. O valor de 1,8 GB é o footprint operacional preciso para esse tamanho de conjunto de dados sob a configuração ajustada.

Já o IoTDB reportou aproximadamente 12,5 GB fixo ao longo das execuções originais. Durante as horas em que ficou ocioso enquanto InfluxDB e TimescaleDB eram testados, a JVM expandiu o heap até próximo do teto máximo configurado e não devolveu essas páginas ao SO, comportamento padrão do gerenciador de heap da JVM. Nas execuções ajustadas, a JVM começa em 1,7 GB e cresce para aproximadamente 10 GB ao final da fase de leitura, refletindo a demanda real do conjunto de trabalho por fase de carga.

O InfluxDB, por sua vez, é o mais eficiente em memória entre as cargas de trabalho de escrita: 190–940 MB em testes de escrita ajustados na escala pequena. No entanto, essa vantagem não se mantém incondicionalmente. Na escala grande, o LOTE-GRANDE atinge pico de 2,9 GB à medida que o cache de escrita TSM mantém grandes volumes de lotes pendentes junto com buffers de compactação concorrente. O limite padrão de 1 GB no InfluxDB aplica-se apenas ao teto do cache TSM. O RSS total do processo inclui o índice de séries, buffers WAL e a sobrecarga do runtime Go, componentes que, juntos, podem exceder esse valor sob alta carga de escrita.

A implicação prática para o planejamento de implantação é que a vazão de escrita dominante do IoTDB requer um compromisso substancial de memória (~10 GB na escala média, crescendo para ~15 GB na grande). Para ambientes com memória restrita (gateways de borda IoT, sistemas embarcados ou instâncias de nuvem com RAM limitada), o footprint mais enxuto do InfluxDB pode ser o fator decisivo mesmo quando a vazão de escrita é a principal preocupação. A evolução do footprint de memória por escala é apresentada na Figura 8.

### 5.4 Efeitos de Escala

A Figura 9 sintetiza os padrões de degradação de latência de leitura entre as escalas, fornecendo uma visão consolidada dos efeitos de crescimento de volume sobre cada banco de dados e tipo de carga de trabalho.

O mapa de calor mostra que o IoTDB mantém células predominantemente verdes em quase todas as transições de escala. A exceção está no filtro de valor entre as escalas pequena e média, em que as estatísticas de mínimo/máximo dos blocos TsFile ainda não são suficientes para eliminar blocos

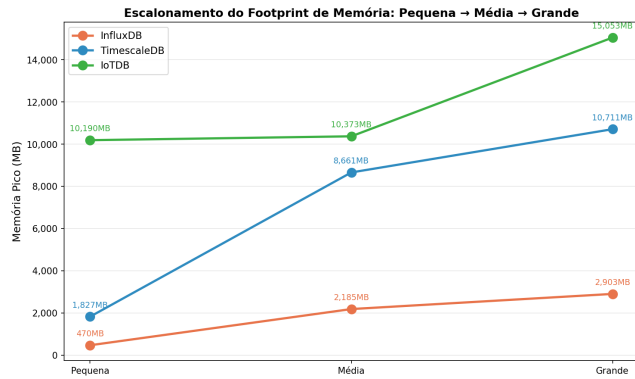


**Figura 8.** Footprint de memória de pico (MB) por escala (execuções ajustadas). O conjunto de trabalho JVM do IoTDB cresce para 15 GB na escala grande. O InfluxDB permanece a opção mais eficiente em memória em cada escala.

com eficiência nesse volume. O TimescaleDB concentra as células mais vermelhas no filtro de valor em ambas as transições, confirmando que o índice BRIN não acompanha o crescimento do volume de dados. O InfluxDB apresenta degradação moderada e distribuída na maioria das cargas, sem picos tão severos quanto os do TimescaleDB no filtro de valor e sem a estabilidade do IoTDB nas demais.

#### 5.4.1 Escalonamento da Vazão de Escrita

A Tabela 14 mostra como a vazão de escrita de linha de base evolui por escalas nas execuções ajustadas.

**Tabela 14.** Vazão de escrita de linha de base (pts/s) por escala (ajustado).

| BD          | Pequena   | Média     | Grande     |
|-------------|-----------|-----------|------------|
| IoTDB       | 2.163.809 | 7.991.224 | 23.607.475 |
| InfluxDB    | 397.894   | 392.975   | 1.256.602  |
| TimescaleDB | 174.787   | 175.685   | 400.455    |

A vazão de escrita do IoTDB escala superlinearmente: 2,2 M pts/s na escala pequena, 8,0 M na média, 23,6 M na grande. Esse crescimento é consistente com o efeito de aquecimento da compilação JIT da JVM: benchmarks curtos não dão ao JIT tempo suficiente para otimizar totalmente os caminhos quentes [Barrett *et al.*, 2017]. Nas escalas média e grande, as execuções mais longas permitiriam que o JIT atingisse a compilação de pico, saturando o caminho de escrita Thrift. Os números de escrita do IoTDB na escala pequena, portanto, provavelmente subestimam seu teto em estado estacionário; o valor na escala grande é o mais representativo da vazão de produção sustentada.

O InfluxDB permanece aproximadamente constante entre as escalas pequena e média, padrão sugestivo de saturação pelo WAL e pela compactação TSM independentemente do volume. Na escala grande, o ganho observado é consistente com a amortização da sobrecarga de compactação em volumes maiores. O TimescaleDB mostra um padrão similar: plano da escala pequena para a média, com melhora na escala grande sugestiva de maior aproveitamento dos intervalos de checkpoint com execuções mais longas. Esses padrões de escalonamento estão consolidados na Figura 10.

Esse efeito JIT é ainda mais evidente na carga de trabalho batch-small: a vazão do IoTDB por escalas (28 K → 81 K → 446 K pts/s) é uma clara demonstração de escalonamento impulsionado por JIT no conjunto de dados. A latência

média simultaneamente diminui de 1,22 ms para 0,40 ms à medida que a escala aumenta, o que só é possível se a melhoria vier de otimização de código em vez de redução de carga. O valor na escala grande (446 K pts/s, 0,40 ms de latência média) representa a vazão de ingestão de ponto único em estado estacionário do IoTDB uma vez que o JIT compilou completamente o caminho quente de serialização Thrift [Barrett *et al.*, 2017].

Quanto ao TimescaleDB, o teste batch-large atinge o timeout de 3.600 segundos do benchmark na escala grande tanto nas execuções original quanto nas ajustadas. Com 50 dispositivos e um tamanho de lote de 1.000 linhas, cada requisição de escrita toca 50 partições de hipertabela simultaneamente e dispara até 50 flushes WAL concorrentes. O custo cumulativo de entrada/saída esgota o orçamento de tempo antes que a contagem total de laços seja completada. Isso não representa uma anomalia, sendo na verdade o resultado esperado do fan-out de partições na hipertabela sob cenários de alta concorrência e lotes volumosos de escrita. Para cargas de trabalho de carregamento em lote direcionadas ao TimescaleDB em escala, a contagem de partições deve ser ponderada contra a amplificação de flush WAL.

Entre as tendências multiescala, a progressão de latência de último valor do InfluxDB (9,24 → 21,72 → 23,24 ms) merece atenção particular porque revela um problema estrutural de escalabilidade. Cada arquivo TSM incremental escrito em disco adiciona à profundidade de busca que uma consulta de último valor deve percorrer. TimescaleDB e IoTDB mostram latência de último valor plana ou até decrescente (devido ao cache em memória do IoTDB [Apache IoTDB Community, 2024b] e ao aquecimento JIT [Barrett *et al.*, 2017]) ao longo do mesmo crescimento de cem vezes no volume de dados. Qualquer implantação que acumulará meses ou anos de dados verá o custo de último valor do InfluxDB continuar crescendo, enquanto os custos do TimescaleDB e do IoTDB permanecem otimizados.

## 5.5 Impacto do Ajuste de Configuração

### 5.5.1 Ganhos do TimescaleDB com pg tune

A Tabela 15 compara o desempenho de escrita do TimescaleDB antes e depois do pg tune na escala pequena.

O principal driver é `shared_buffers=8GB`, que fornece ao PostgreSQL um pool de buffers grande o suficiente para evitar a substituição de buffers, que forçaria escritas extras no WAL. O `work_mem=27413kB` elimina a gravação em disco durante a filtragem baseada em ordenação, o que explica a redução de 40% na latência do filtro de valor — uma carga de trabalho que exige a ordenação dos resultados filtrados. O `checkpoint_completion_target=0.9` distribui a entrada/saída de checkpoint de forma mais uniforme ao longo do intervalo de checkpoint, reduzindo os picos de paralisação de escrita que inflam a latência P99 (o P99 do LOTE-GRANDE caiu de 1.069 ms para 812 ms). Essas melhorias indicam que as configurações padrão do PostgreSQL são intencionalmente conservadoras para garantir funcionamento em hardware de baixo recurso, e que o ajuste via pg tune ou ferramenta equivalente deve ser aplicado antes de reportar resultados em hardware de médio e alto porte.

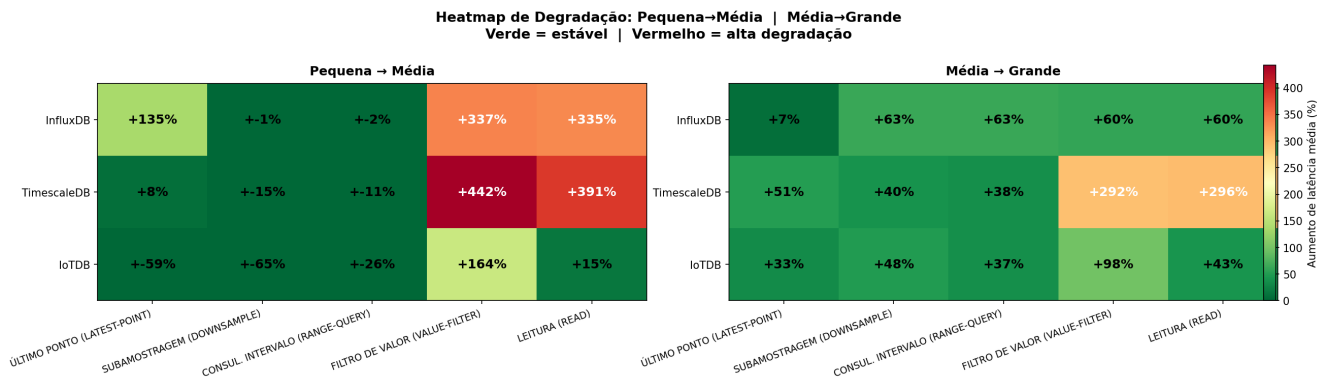


**Figura 9.** Mapa de calor de degradação de latência por tipos de carga de trabalho de leitura e transições de escala (execuções ajustadas). Verde indica latência estável; vermelho indica alto crescimento. O IoTDB é consistentemente verde exceto para filtro de valor (pequena→média); o TimescaleDB mostra degradação severa no filtro de valor em ambas as transições.

**Tabela 15.** Desempenho de escrita do TimescaleDB: original vs. ajustado (escala pequena).  $\Delta$  mostra variação percentual; negativo significa melhoria para latência.

| Teste         | pts/s ant. | pts/s dep. | $\Delta$ vazão | $\Delta$ lat |
|---------------|------------|------------|----------------|--------------|
| ESCRITA       | 133.677    | 174.787    | +30%           | -25%         |
| LOTE-GRANDE   | 160.546    | 214.927    | +34%           | -25%         |
| FORA-DE-ORDEM | 132.189    | 160.121    | +21%           | -21%         |

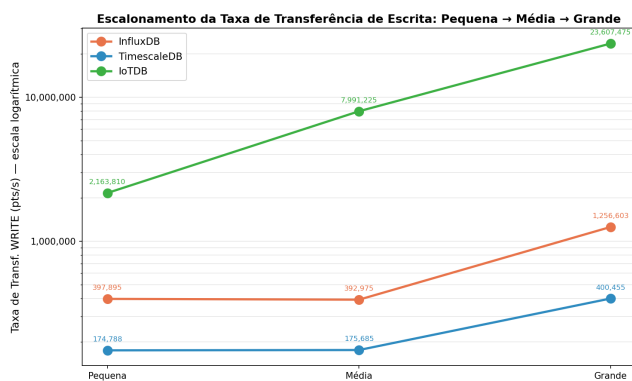


**Figura 10.** Vazão de escrita de linha de base (pts/s, escala logarítmica) por escala (ajustado). O crescimento superlinear do IoTDB reflete a saturação JIT da JVM; InfluxDB e TimescaleDB permanecem planos até a escala grande.

### 5.5.2 Ganhos do InfluxDB com Isolamento

O InfluxDB não recebeu alterações de configuração nas execuções ajustadas. Suas melhorias vêm inteiramente da execução sem bancos de dados concorrentes. Todas as latências de leitura melhoraram 15–21%, consistente com melhor utilização do cache de arquivos TSM uma vez que o cache de páginas do TimescaleDB não compete mais pelo espaço de buffer do SO. O ganho mais expressivo foi o P99 do LOTE-GRANDE: 205 ms → 60 ms (−71%). Na configuração original, o TimescaleDB estava escrevendo em disco durante a sessão de teste do InfluxDB, privando periodicamente o motor de compactação TSM do InfluxDB de largura de banda de entrada/saída. Essas paralisações raras, mas severas, de entrada/saída produziram picos extremos de latência que, embora não tenham afetado a vazão média, aumentaram consideravelmente o P99. O isolamento removeu essas paralisações.

### 5.5.3 Correção de Dois Falsos Positivos na Escala Grande

As execuções originais na escala grande produziram duas descobertas que as reexecuções ajustadas reverteram.

**Penalidade fora de ordem do TimescaleDB.** A exe-

cução original na escala grande mostrou uma penalidade de 23% na vazão fora de ordem para o TimescaleDB, atribuída a ciclos de descompressão-recompressão de inserções fora de ordem em blocos comprimidos mais antigos. A reexecução ajustada mostra apenas 2,7% de degradação, consistente com o padrão nas escalas pequena e média. A hipótese mais provável: inserções fora de ordem em blocos mais antigos requerem leitura do disco para descompressão. Na sessão original, a expansão do heap JVM do IoTDB (acima de 15 GB) provavelmente deslocou as páginas do cache do SO que continham esses blocos, forçando leituras frias em cada inserção fora de ordem em um bloco mais antigo. Com acesso exclusivo, os blocos permaneceriam quentes no cache e os custos de descompressão seriam negligíveis.

**Saturação de CPU do TimescaleDB.** A carga de trabalho READ original na escala grande apresentou média de 97,7% de utilização de CPU com latência P99 de 4.232 ms, levando à conclusão de que o TimescaleDB “ficou sem capacidade computacional” e não poderia ser melhorado sem hardware de maior capacidade ou fragmentação horizontal. A reexecução ajustada mostra 77,4% de CPU médio e P99 de 1.850 ms. O valor original de 97,7% resultou da disputa simultânea de três bancos de dados e do cliente de benchmark pelos mesmos seis núcleos de CPU. Portanto, o desempenho do TimescaleDB não era limitado pela capacidade física do hardware, mas sim pela concorrência excessiva por processamento causada por processos paralelos.

Essas duas reversões ressaltam a importância de uma metodologia de benchmark isolada e corretamente ajustada. Sem ciclos de vida sequenciais de contêineres e configuração padrão de banco de dados, ambas as afirmações teriam sido publicadas como descobertas validadas. A Figura 11 compara diretamente as execuções original e ajustada em termos de utilização de CPU e latência P99 para a carga de trabalho READ.

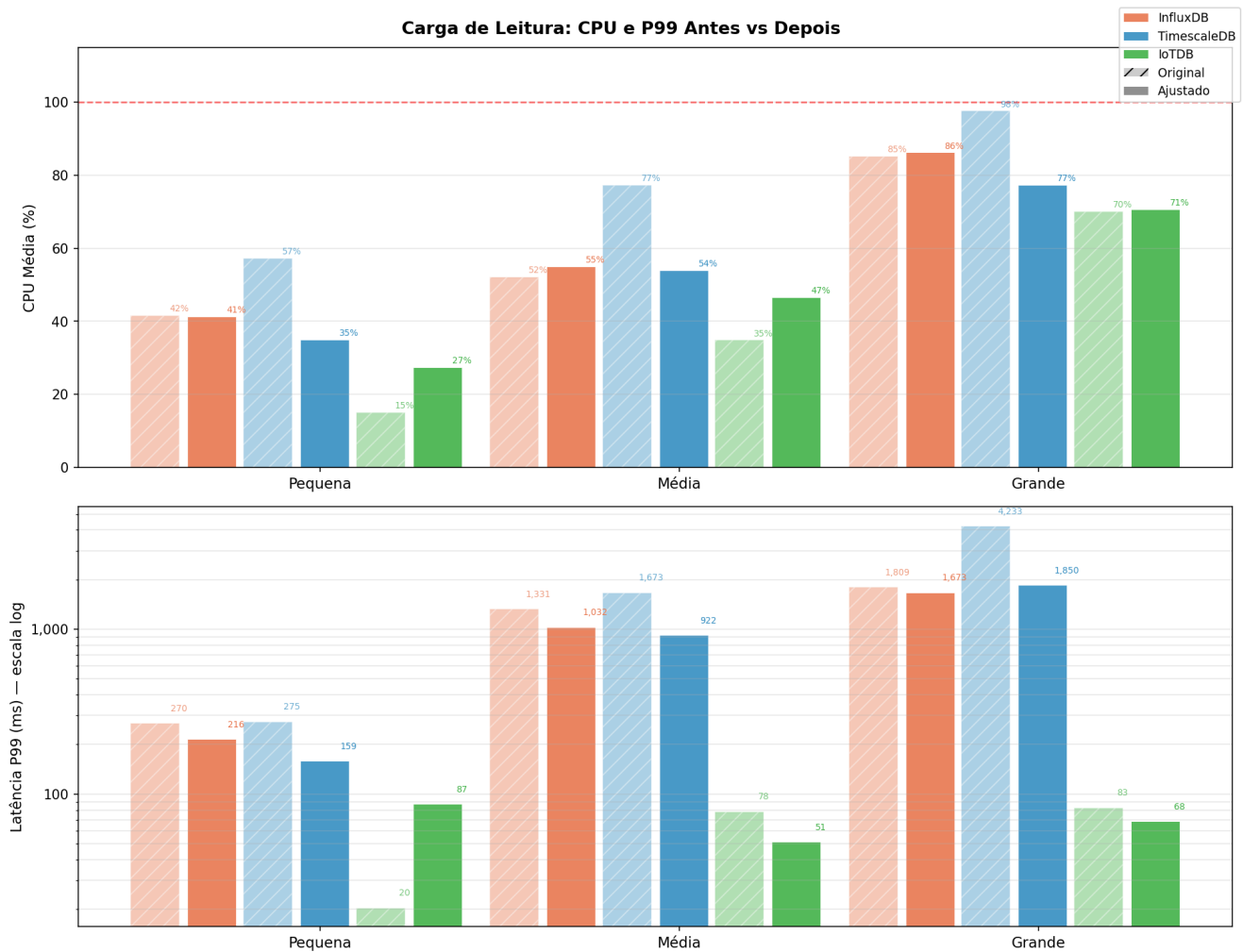


**Figura 11.** Utilização média de CPU da carga de trabalho READ (acima) e latência P99 (abaixo, escala logarítmica) para execuções original (hachurado) vs. ajustado (sólido) em todas as três escalas. A CPU na escala grande do TimescaleDB cai de 98% para 77% e o P99 de 4.233 ms para 1.850 ms uma vez que os contêineres concorrentes são removidos.

## 5.6 Carga de Trabalho Mista de Leitura

A carga de trabalho read executa todos os dez tipos de consulta do *iot-benchmark* com peso igual, fornecendo uma linha de base de leitura composta que inclui consultas de intervalo, agregações, subamostragem, consultas de último valor e consultas de predicado de valor simultaneamente. Na escala pequena (ajustado), o TimescaleDB tem média de 0,67 ms, o IoTDB de 2,11 ms e o InfluxDB de 14,28 ms por operação no conjunto combinado. Esses valores são consistentes com os resultados individuais de consulta apresentados acima: o TimescaleDB domina para consultas de janela de tempo, que constituem a maioria dos dez tipos, enquanto a vantagem do IoTDB nas consultas de filtro de valor é diluída pela ponderação igual.

O papel analítico primário da carga de trabalho mista neste estudo é como benchmark de ajuste: a latência de leitura do TimescaleDB melhorou 38% após o *pgtune*, confirmando que múltiplos tipos de consulta se beneficiam de um pool de buffers dimensionado corretamente, não apenas o filtro de valor.

## 5.7 Resumo

A Tabela 16 consolida o mapeamento de vencedor por carga de trabalho em todos os cenários testados.

A Tabela 17 complementa o resumo qualitativo com os valores numéricos mais representativos de cada tipo de carga de trabalho avaliado.

O IoTDB é a escolha clara para cargas de trabalho dominadas por escrita, mas requer compromisso substancial de memória (~10 GB na escala média, crescendo para ~15 GB na grande) e não suporta junções relacionais. O TimescaleDB troca velocidade bruta de escrita por compatibilidade com SQL, o ecossistema completo do PostgreSQL e desempenho de leitura consistentemente forte para consultas de janela de tempo uma vez que os dados estão armazenados em cache. O InfluxDB ocupa o meio-termo: melhor vazão de escrita do que o TimescaleDB, melhor eficiência de memória do que o IoTDB, mas mais fraco em consultas complexas de predicado de valor e consultas de último valor em escala. Nenhum banco de dados único domina em todas as cargas de trabalho.

## 6 Conclusão

Os resultados mostram que nenhum banco de dados domina em todas as dimensões: o IoTDB lidera em vazão de escrita sequencial e filtro de valor, o TimescaleDB oferece latência constante em subamostragem e compatibilidade SQL completa, e o InfluxDB ocupa um meio-termo prático entre de-

**Tabela 16.** Resumo de recomendação por carga de trabalho em todos os cenários testados.

| Carga de Trabalho                      | Melhor BD   | Razão arquitetural principal                                                         |
|----------------------------------------|-------------|--------------------------------------------------------------------------------------|
| Ingestão sequencial (ordenada)         | IoTDB       | Protocolo de tablet binário + buffering em memória antes do flush para TsFile        |
| Carregamento em lote (LOTE=1000)       | IoTDB       | Caminho de escrita colunar vetorizado; 11,8 M pts/s na escala pequena                |
| Ingestão de ponto único (LOTE=1)       | IoTDB       | Menor sobrecarga por chamada Thrift vs. rodada HTTP                                  |
| Fora de ordem (vazão absoluta)         | IoTDB       | 1,87 M pts/s OOO vs. 255 K (InfluxDB); lidera em cada escala                         |
| Fora de ordem (tolerância relativa)    | TimescaleDB | Atualizações de B-tree são independentes de ordem                                    |
| Subamostragem em painel                | TimescaleDB | <code>time_bucket()</code> em <code>shared_buffers</code> quente; latência constante |
| Exportação de intervalo de tempo       | TimescaleDB | Eliminação de blocos + cache de páginas quente do SO                                 |
| Alertas por limite (filtro de valor)   | IoTDB       | Estatísticas de mín/máx por bloco; escalonamento sublinear com volume de dados       |
| Consulta de último valor em tempo real | TimescaleDB | Varredura B-tree reversa sub-ms no bloco mais recente                                |
| Eficiência de memória (escrita)        | InfluxDB    | Cache TSM limitado; sem pré-alocação JVM                                             |

**Tabela 17.** Resumo quantitativo dos resultados por tipo de carga de trabalho.

| Carga de Trabalho                   | Melhor BD   | Melhor Valor     | Pior BD     | Pior Valor    |
|-------------------------------------|-------------|------------------|-------------|---------------|
| Escrita (linha de base, grande)     | IoTDB       | 23.607.475 pts/s | TimescaleDB | 400.455 pts/s |
| Escrita (lote, LOTE=1000, pequena)  | IoTDB       | 11.827.572 pts/s | TimescaleDB | 214.927 pts/s |
| Fora de ordem (absoluto, pequena)   | IoTDB       | 1.871.351 pts/s  | TimescaleDB | 160.121 pts/s |
| Fora de ordem (penalidade relativa) | TimescaleDB | −8,4%            | InfluxDB    | −35,7%        |
| Último valor (pequena, ms)          | TimescaleDB | 0,38 ms          | InfluxDB    | 9,24 ms       |
| Subamostragem (pequena, ms)         | TimescaleDB | 0,61 ms          | InfluxDB    | 5,71 ms       |
| Filtro de valor (grande, ms)        | IoTDB       | 19,42 ms         | TimescaleDB | 510,05 ms     |
| Consulta de intervalo (pequena, ms) | TimescaleDB | 0,54 ms          | InfluxDB    | 5,41 ms       |
| Memória de pico (escrita, pequena)  | InfluxDB    | 190 MB           | IoTDB       | 1.700 MB      |

sempenho e consumo de memória. As escolhas arquiteturais de cada sistema, e não o ajuste de configuração, determinam o comportamento de escalonamento, e o isolamento de recursos durante o benchmark afeta criticamente a validade dos resultados.

## 6.1 Limitações

Os experimentos foram realizados em nó único, o que não captura a sobrecarga de rede real de implantações distribuídas. Os testes de leitura compartilham o estado do cache de páginas entre cargas de trabalho consecutivas, o que pode favorecer testes posteriores. A vazão de escrita do IoTDB reflete reconhecimento em memória antes da persistência em disco; configurações de durabilidade estrita podem reduzir esse valor. Por fim, o aquecimento da JVM faz com que benchmarks curtos subestimem a vazão em estado estacionário do IoTDB.

## 6.2 Trabalhos Futuros

Direções relevantes incluem avaliar implantações distribuídas (e.g., IoTDB multi-nó, InfluxDB Clustered, TimescaleDB com Citus), testar com conjuntos de dados industriais de escala real e reavaliar os compromissos após atualizações arquiteturais significativas, como o novo motor colunar do InfluxDB 3.0. Medir o custo de replicação e failover também seria essencial para decisões de implantação em produção.

## Declarações complementares

### Contribuições dos autores

L.F.K. concebeu o estudo, implementou o ambiente de benchmark, realizou todos os experimentos, analisou os resultados e redigiu o manuscrito.

### Conflitos de interesse

Os autores declaram não ter interesses conflitantes.

### Disponibilidade de artefatos de pesquisa

Os scripts de benchmark, as configurações do Docker Compose, o ambiente de construção declarativo (Nix flake) e os conjuntos

de dados de resultados criados e utilizados neste estudo estão disponíveis publicamente em <https://github.com/LuizFerK/iot-benchrunner>.

## Uso de Inteligência Artificial

O Claude Code (Anthropic, claude-sonnet-4-6) foi utilizado exclusivamente para auxiliar na geração dos scripts Python para execução dos benchmarks e análise de dados com Matplotlib. O autor realizou de forma independente todo o planejamento experimental, a interpretação dos resultados e a redação do manuscrito. Todas as conclusões analíticas e decisões finais foram tomadas exclusivamente pelo autor, que assume plena responsabilidade pelo conteúdo.

## Referências

- Apache IoTDB Community (2024a). Iotdb user guide – benchmark tool. Disponível em: <https://iotdb.apache.org/UserGuide/V1.2.x/Tools-System/Benchmark.html>. Acesso em: 8 jun. 2026.
- Apache IoTDB Community (2024b). Query data – last query. Disponível em: <https://iotdb.apache.org/UserGuide/V1.2.x/User-Manual/Query-Data.html#last-query>. Acesso em: 8 jun. 2026.
- Barrett, E., Bolz-Tereick, C. F., Killick, R., Mount, S., and Tratt, L. (2017). Virtual machine warmup blows hot and cold. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA). Disponível em: <https://arxiv.org/pdf/1602.00602>. Acesso em: 8 jun. 2026. DOI: 10.1145/3133876.
- de Leal Meireles Pereira, P. M. (2023). Mulletbench: Multi-layer edge time series database benchmark. Master's thesis, University of Minho, School of Engineering. Disponível em: <https://core.ac.uk/download/619581927.pdf>. Acesso em: 31 maio 2026.
- Docker Inc. (2025). Docker: Accelerated container application development. Disponível em: <https://www.docker.com>. Acesso em: 31 maio 2026.
- Dunning, T. and Friedman, E. (2014). *Time Series Databases*:

- New Ways to Store and Access Data*. O'Reilly Media, first edition.
- Gaber, M. M., Zaslavsky, A., and Krishnaswamy, S. (2005). Mining data streams: A review. *ACM SIGMOD Record*, 34(2):18–26. Disponível em: <https://sigmodrecord.org/publications/sigmodRecord/0506/p18-survey-gaber.pdf>. Acesso em: 31 maio 2026. DOI: 10.1145/1083784.1083789.
- Grzesik, P. and Mrozek, D. (2020). Comparative analysis of time series databases in the context of edge computing for low power sensor networks. In *Computational Science – ICCS 2020*, volume 12141 of *Lecture Notes in Computer Science*, pages 371–383. Springer. Disponível em: <https://pmc.ncbi.nlm.nih.gov/articles/PMC7302557/>. Acesso em: 31 maio 2026. DOI: 10.1007/978-3-030-50426-7\_28.
- InfluxData (2025). What is time series data? Disponível em: <https://www.influxdata.com/what-is-time-series-data/>. Acesso em: 31 maio 2026.
- Lima, M. C. d. and Lichtnow, D. (2024). Avaliando a performance de sgbd's na inserção e consulta de dados de séries temporais. *Anais da XIX Escola Regional de Banco de Dados (ERBD)*, pages 170–173. Disponível em: <https://sol.sbc.org.br/index.php/erbd/article/view/28507>. Acesso em: 31 maio 2026.
- Naqvi, S. N. Z., Yfantidou, S., and Zimányi, E. (2017). Time series databases and influxdb. Disponível em: [https://cs.ulb.ac.be/public/\\_media/teaching/influxdb\\_2017.pdf](https://cs.ulb.ac.be/public/_media/teaching/influxdb_2017.pdf). Acesso em: 31 maio 2026.
- NixOS Contributors (2025). NixOS: Reproducible builds and deployments. Disponível em: <https://nixos.org>. Acesso em: 31 maio 2026.
- O'Neil, P. E., Cheng, E., Gawlick, D., and O'Neil, E. J. (1996). The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385. Disponível em: <https://www.cs.umb.edu/~poneil/lsmtree.pdf>. Acesso em: 31 maio 2026. DOI: 10.1007/s002360050048.
- Özsu, M. T. and Valduriez, P. (2011). *Principles of Distributed Database Systems*. Springer, New York, third edition.
- Palma, W. (2016). *Time Series Analysis*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, New Jersey.
- Ramakrishnan, R. and Gehrke, J. (2003). *Database Management Systems*. McGraw-Hill, New York, NY, USA, 2nd edition.
- Shah, B., Jat, P. M., and Sasidhar, K. (2022). Performance study of time series databases. *arXiv preprint arXiv:2208.13982v1*. Disponível em: <https://arxiv.org/abs/2208.13982v1>. Acesso em: 31 maio 2026.
- Shapira, G. and Chen, Y. (2016). Common pitfalls of benchmarking big data systems. *IEEE Transactions on Services Computing*, 9(1):152–160. Disponível em: <http://yanpeichen.com/professional/BenchmarkingPitfallsIEEE.pdf>. Acesso em: 31 maio 2026. DOI: 10.1109/TSC.2015.2494595.
- SILBERSCHATZ, A. (2020). *Sistema de banco de dados*. GEN LTC, 7. ed. edition.
- Thierer, A. and Castillo, A. (2015). Projecting the growth and economic impact of the internet of things. Technical report, Mercatus Center at George Mason University, Arlington, VA. Disponível em: [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=2618794](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2618794). Acesso em: 31 maio 2026.
- Timescale, Inc. (2024). TimescaleDB: An open-source time-series SQL database. Disponível em: <https://docs.timescale.com/>. Acesso em: 31 maio 2026.
- Tsinghua University IoTDB Team (2023). iot-benchmark: A comprehensive benchmarking tool for iot time series databases. GitHub repository. Disponível em: <https://github.com/thulab/iot-benchmark>. Acesso em: 31 maio 2026.
- Vasiliev, A. (2025). Pgtune: Postgresql configuration tool. Disponível em: <https://pgtune.leopard.in.ua>. Acesso em: 31 maio 2026.
- Wang, C., Qiao, J., Huang, X., Song, S., Hou, H., Jiang, T., Rui, L., Wang, J., and Sun, J. (2023). Apache iotdb: A time series database for iot applications. *Proc. ACM Manag. Data*, 1(2):27. Disponível em: <https://sxsong.github.io/doc/23sigmod-iotdb.pdf>. Acesso em: 31 maio 2026. DOI: 10.1145/3589775.
- Zikria, Y. B., Ali, R., Afzal, M. K., and Kim, S. W. (2021). Next-generation internet of things (IoT): Opportunities, challenges, and solutions. *Sensors*, 21(4):1174. Disponível em: <https://www.mdpi.com/1424-8220/21/4/1174>. Acesso em: 31 maio 2026. DOI: 10.3390/s21041174.
- Zubaroglu, A. and Atalay, V. (2021). Data stream clustering: A review. *arXiv preprint arXiv:2007.10781*. Disponível em: <https://arxiv.org/abs/2007.10781>. Acesso em: 31 maio 2026.