Análise de Desempenho em Banco de Dados Temporais Aplicado no Cenário de Dados de Potência Elétrica

Vitor O. Santos[1], Viviane Dal Molin[1], Jefferson Tales Oliva[1], Ives Renê Venturini Pola[1]

[1]Departamento Acadêmico de Informática
Universidade Tecnológica Federal do Paraná (UTFPR) – Pato Branco, PR – Brasil

vitorsantos.2018@alunos.utfpr.edu.br

{vivianemolin,jeffersonoliva,ivesr}@utfpr.edu.br

Abstract. *This study evaluates the performance of three temporal database management systems (TDBMS) — TimescaleDB, TDengine, and Druid — in managing electrical energy data. The results of read and write tests indicate that, although no single solution outperforms the others in all tests, each offers unique advantages depending on the specific workload requirements. TimescaleDB excels in high-concurrency environments, TDengine is more efficient in scenarios with less complex relational data management needs, and Druid is better suited for situations where analytical query performance is crucial.*

Resumo. *Este estudo avalia o desempenho de três sistemas gerenciadores bancos de dados temporais (SGBDTs) – TimescaleDB, TDengine e Druid – no gerenciamento de dados de energia elétrica. Por meio de testes de leitura e de escrita, os resultados indicam que, embora nenhuma solução supere as outras em todos os testes, cada uma oferece vantagens únicas dependendo das exigências específicas de carga de trabalho. O TimescaleDB sobressaiu em ambientes de alta concorrência, o TDengine é mais eficiente em cenários com menores necessidades de gestão de dados relacionais complexos, e o Druid é mais adequado para situações onde o desempenho de consultas analíticas é crucial.*

1. Introdução

Estimativas sugerem que, até 2025, haverá mais de 75 bilhões de dispositivos conectados na Internet das Coisas (*Internet of Things* – IoT) [Statista 2024]. Esses dispositivos geram dados que frequentemente estão relacionados a eventos com selos de tempo, conhecidos como timestamps [Esling and Agon 2012]. Exemplos disso são as medições de sensores em uma rede elétrica. Esses conjuntos de dados coletados ao longo do tempo são representados por séries temporais, que são cruciais para análises em diversos domínios, incluindo o setor de energia elétrica. Nos sistemas de potência elétrica, as séries temporais desempenham um papel vital, pois capturam o comportamento dinâmico de redes e equipamentos, fornecendo informações essenciais para monitoramento, controle e otimização [Liu et al. 2024].

Para armazenar e lidar com desafios do gerenciamento de séries temporais, como altas taxas de inserção de dados, consultas complexas e a necessidade de armazenamento eficiente, diversos Sistemas Gerenciadores de Banco de Dados (SGBD) foram

desenvolvidos, tais como TimescaleDB [1], Prometheus [2], Apache Druid [3] e Amazon Timestream[4] [5]. Dada a ampla variedade de SGBDs para dados temporais e suas diferentes funcionalidades, é importante identificar a melhor aplicação para cada ferramenta em cenários específicos, como o mercado de ações, dados médicos, dados industriais e dados de potência elétrica. Nesse cenário, diversos *benchmarks* foram propostos, tais como TS-Benchmark [Hao et al. 2021], SimpleMetric [Sychev et al. 2020] e IoTDB-Benchmark [Liu and Yuan 2019]. Contudo, esses *benchmarks* são amplos e não cobrem o domínio de medição de potência elétrica. Nesse sentido, Visperas e Chodpathumwan [Visperas and Chodpathumwan 2021] apresentam um conjunto de testes e dados para o domínio. Esse conjunto foi revisitado no presente trabalho, no qual adicionamos dois novos sistemas de gerenciamento de bancos de dados temporais e atualizamos a versão de um dos SGBDs já utilizados anteriormente. Também implementamos melhorias nos componentes de execução.

2. Trabalhos relacionados

A escolha de um SGBD adequado para armazenar e gerenciar uma base de dados requer uma avaliação minuciosa das necessidades do usuário, que pode incluir uma alta taxa de escrita e/ou leitura, a manutenção de um extenso histórico de dados em um espaço reduzido de armazenamento, entre outras características [Taipalus 2024]. Nesse contexto, diversos testes de comparação foram desenvolvidos, como o trabalho de [Bader et al. 2017], que ampliou os dados e metodologias do Yahoo! Cloud Serving Benchmark [Cooper et al. 2010], analisando diferentes cenários com variados números de *clusters*. No estudo foram avaliados 11 SGBDs, categorizando-os de acordo com características como código aberto ou solução proprietária, baseados em modelo relacional clássico ou NoSQL, alta disponibilidade, escalabilidade, balanceamento de carga, tipos de operações nativas (inserção, soma, média, etc.), uso de *tags*, menor granularidade de armazenamento, e facilidade de integração com APIs, interfaces, bibliotecas de cliente e *plugins*. Além disso, no estudo foi mensurado o tempo gasto em operações de escrita e leitura. A conclusão do trabalho indica que, embora nenhuma solução de SGBD temporal atenda a todos os critérios em cada grupo de características, o Apache Druid se destacou em maior grau, seguido pelo InfluxDB[6], MonetDB[7], MySQL Community Server[8] e PostgreSQL[9]. Além disso, os autores ressaltam que, quando os recursos disponíveis não são suficientes para diferenciar os bancos de dados temporais, o desempenho pode se tornar um fator decisivo.

Destacando a necessidade de avaliar a característica dos SGBDs temporais em cenários de escrita fora de ordem e com diferentes distribuições de dados o IoTDB-Benchmark, introduzido por [Liu and Yuan 2019], propõe um cenário de dados IoT para aprimorar a avaliação desses bancos. O *benchmark* inclui testes de carregamento e con-

[1]https://www.timescale.com/
[2]https://prometheus.io/
[3]https://druid.apache.org/
[4]Solução nativa de serviço em nuvem.
[5]https://aws.amazon.com/pt/timestream/
[6]https://www.influxdata.com/products/influxdb-overview/
[7]https://www.monetdb.org/
[8]https://www.mysql.com/products/community/
[9]https://www.postgresql.org/

sulta, além de monitoramento contínuo, com as métricas de desempenho sendo armazenadas em um banco de dados separado para facilitar análises estatísticas. A configuração dos testes pode ser ajustada para incluir dados gerados e inseridos tanto cronologicamente quanto de forma aleatória, variando o número de dispositivos e sensores por dispositivo, o que aumenta a quantidade de dados aplicados no teste. O *benchmark* é usado para comparar quatro sistemas gerenciadores de banco de dados de séries temporais sob diversas cargas de trabalho: InfluxDB, KairosDB [10], OpenTSDB [11] e TimescaleDB. Os resultados mostraram diferenças significativas em desempenho e eficiência, dependendo do sistema e das configurações operacionais utilizadas, indicando que não existe uma ferramenta universalmente superior, mas que a escolha ideal depende das exigências específicas da carga de trabalho em que o SGBD será empregado.

Por outro lado, o TS-Benchmark, apresentado por [Hao et al. 2021], avalia bancos de dados de séries temporais em cenários como o monitoramento de turbinas eólicas, utilizando um modelo de geração de dados baseado em rede adversária generativa (*generative adversarial network* – GAN) para produzir grandes volumes de dados de séries temporais. O *benchmark* é executado em dois servidores interligados por uma rede LAN e testa a capacidade dos sistemas em realizar carga de dados em lote, injeção de dados em fluxo contínuo e acesso a dados históricos. Como resultado, a comparação entre quatro bancos de dados de séries temporais, sendo estes o InfluxDB, TimescaleDB, Druid e OpenTSDB, revelou que o desempenho varia de acordo com as operações realizadas. Nesse estudo, o InfluxDB alcançou, frequentemente, melhores resultados em métricas como taxa de leitura e taxa de escrita. Durante os testes, a ferramenta de visualização Grafana[12] e o Prometheus foram usadas para realizar o monitoramento dos testes.

Para avaliar o desempenho dos bancos de dados ClickHouse [13], InfluxDB, TimescaleDB e PostgreSQL em cenários de dados de experimentos científicos e IoT industrial, [Mostafa et al. 2022] realizou testes de carregamento em cenários de escrita em lote e concorrente, observando o consumo de recursos dos SGBDs à medida que o banco de dados crescia. Além disso, o teste executou consultas de obtenção de dados brutos, consultas fora de alcance, agregação de dados, amostragem reduzida de dados, e operações em dois sensores reduzidos, comparando-os por meio de funções de comparação. Os testes foram conduzidos usando duas máquinas interconectadas por um switch Ethernet de 1 Gbit/s, com monitoramento contínuo do uso de recursos. Os resultados indicaram que os SGBDs de séries temporais, especialmente o ClickHouse, superaram o PostgreSQL em eficácia. O ClickHouse se destacou tanto nas taxas de ingestão de dados quanto na rapidez das consultas, beneficiando-se de sua arquitetura otimizada para operações intensivas de escrita e consultas eficientes em grandes volumes de dados.

Desenvolvida por Sychev *et al* [Sychev et al. 2020], a ferramenta de linha de comando SimpleMetric avaliou os seguintes SGBDs: InfluxDB, KDB+ [14], Graphite [15], TimescaleDB, KairosDB e CrateDB [16]. A ferramenta é composta por três componentes: o

[10]https://kairosdb.github.io/
[11]http://opentsdb.net/
[12]https://grafana.com/
[13]https://clickhouse.com/
[14]https://kx.com/products/kdb/
[15]https://graphiteapp.org/
[16]https://cratedb.com

Data Generator, que adapta o conjunto de dados para cada um dos SGBDs testados; o *Latency Meter*, que coleta as métricas de desempenho durante as transações; e o *Runner*, que funciona como o *frontend* da ferramenta, permitindo configuração e execução adaptadas ao SGBD alvo. Os testes foram realizados utilizando containers Docker, e a latência foi medida em milissegundos. O estudo também examinou o comportamento dos SGBDs sob múltiplas conexões, com configurações de 1, 5 e 10 conexões. Os resultados indicaram que não existe um SGBD temporal superior em todos os cenários. Embora o InfluxDB tenha apresentado resultados promissores em alguns testes, o Graphite mostrou desempenho comparável em testes de escrita.

No setor de energia elétrica, o *benchmark* de [Visperas and Chodpathumwan 2021] avaliou os SGBDs OpenTSDB, TimescaleDB e InfluxDB, utilizando suas respectivas imagens de contêiner e o pacote Airspeed Velocity da linguagem Python para testar a ingestão e recuperação de dados. A base de dados inicial continha 648 mil registros de leituras de unidades de medição fasorial, que foram duplicados para aumentar o volume e o período de teste. Foram realizados testes de escrita em lote, leitura e escrita concorrente, com o número de conexões variando entre 1, 2, 4, 8, 16 e 20. Em todos os testes, foram coletados dados sobre o tempo de execução e o uso de memória. De forma geral, o InfluxDB apresentou os melhores tempos de resposta, mas também registrou o maior consumo de memória.

Reconhecendo que novas versões de softwares podem trazer melhorias significativas no desempenho e proporcionar comparações com novas soluções disponíveis, o presente trabalho revisitou a metodologia apresentada por Visperas e Chodpathumwan [Visperas and Chodpathumwan 2021], por considerar mais soluções de software além de analisar as otimizações nas versões mais recentes comparadas. Dessa forma, o presente trabalho difere na avaliação das ferramentas em um novo cenário com novos SGBDs em ambiente de execução ligados ao contexto de dados de energia elétrica., nesse teste específico,

3. Metodologia

3.1. Seleção de SGBD

Para a seleção dos SGBDs utilizados nos testes, o critério principal foi a posição de cada um no ranking de SGBDs temporais [solid 2024]. Uma exceção a esse critério foi o TDengine, que, apesar de não estar entre os dez primeiros colocados, demonstrou uma taxa de crescimento significativa no momento da consulta ao gráfico de tendências mostrado no *ranking*. Assim, os escolhidos foram:

- **Apache Druid 27.0.0**: Solução de código aberto com arquitetura baseada em serviços voltada para consultas analíticas.
- **TDengine 3.0.5.0**: SGBD colunar de código aberto e com arquitetura distribuída.
- **TimescaleDB 2.11.2** (baseado no PostgreSQL 14): Ferramenta de código aberto com armazenamento misto entre linha e coluna.

3.2. Materiais

Os testes foram realizados em um ambiente Docker versão 24.0.5, em uma instância EC2 C6a da AWS equipada com processador AMD EPYC de 3ª Geração, 32 *threads* e 64 Gb de memória, utilizando o sistema operacional Ubuntu Server Pro 22.04. Os testes aplicados neste trabalho foram implementados por meio da linguagem Python 3.10.12.

3.3. Fluxo de execução do *Benchmark*

O *benchmark* proposto é representado em 5 fases e pode ser visto na Figura 1. Em todas as fases de teste nos experimentos, foram realizadas três repetições ao final, obtendo-se uma média das métricas coletadas, embora não tenha sido definida no trabalho original.

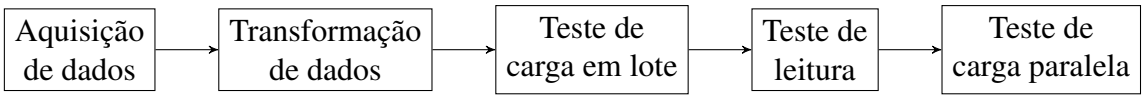

Figura 1. Fluxograma do *Benchmark*.

3.4. Aquisição de dados

Essa etapa do foi responsável por realizar a aquisição dos dados de forma manual no site do Laboratório Nacional de Energia Renovável (National Renewable Energy Laboratory - NREL) [Allen et al. 2014]. O conjunto de dados utilizado é originalmente composto por 19 colunas e contém 108 mil linhas. A primeira coluna representa o horário em que a amostra foi coletada e as demais colunas são um conjunto de ângulo, magnitude e frequência da rede de sincrofasores do Texas, contendo dados de 6 estações de coletas. A Tabela 1 exibe um recorte das medidas coletadas em Austin em um momento aleatório dos dados presentes na base original.

Tabela 1. Amostra de Dados.

Timestamp	Austin Magnitude	Austin Angle	Austin Frequency
2012/01/03 01:00:00.000	78807.3672	9.3185	60.0218
2012/01/03 01:00:00.033	78797.2422	9.5815	60.0218

3.5. Transformação de dados

Nessa fase, os dados foram segmentados utilizando a biblioteca pandas em lotes de 1 mil, 5 mil, 10 mil, 50 mil e 100 mil registros. A segmentação iniciou pelo registro com o menor *timestamp* disponível na base de dados original. Para obter um conjunto de dados maior que o original e garantir continuidade, as métricas foram replicadas com *timestamps* da mesma granularidade que os registros originais, especialmente em conjuntos que excediam o tamanho da base original, resultando em novos conjuntos de 500 mil, 648 mil e 1 milhão de registros.

Adicionalmente, os conjuntos de dados foram estruturados para otimizar o armazenamento em diferentes SGBDs. Especificamente, para o TimescaleDB e o Apache Druid, os dados foram reorganizados através de um mapeamento de colunas, transformando as colunas em linhas. A transformação incluiu a criação de uma nova coluna numérica para localização. Dessa forma, como um instante de coleta da tabela original apresentava seis localidades diferentes na mesma tupla, obtivemos o mesmo instante de coleta seis vezes. Essas coletas foram organizadas em tuplas, cujas chaves eram o instante

Tabela 2. Esquema de tabela com localização.

Timestamp	Magnitude	Ângulo	Frequência	Localização
2012/01/03 01:00:00.000	78807.3672	9.3185	60.0218	1
2012/01/03 01:00:00.000	78797.2422	9.5815	60.0218	2

e o código da localidade. Um exemplo dessa transformação para duas cidades é apresentado na Tabela 2. Como a cidade não era o foco principal do teste, optou-se por numerar as localidades de forma sequencial, conforme apresentado no conjunto de dados original.

Para o TDengine, decidiu-se separar os dados em novos arquivos por localidade, assegurando uniformidade no número de registros para todas as localidades durante o mesmo período de tempo.

O procedimento descrito foi aplicado uma única vez para todos os testes subsequentes.

3.6. Teste de carga em Lote

Os testes de carga em lote seguiram a seguinte organização:

1. Realizar conexão com SGBD;
2. Criação de tabelas;
3. Envio de arquivos segmentados para escrita em lote;
4. Coleta do tempo de execução.

Mais detalhadamente, os passos podem ser descritos da seguinte forma. Inicialmente, foi estabelecida uma conexão com o SGBD alvo. A preparação do ambiente subsequente incluiu a criação de tabelas, conforme as recomendações dos manuais de cada SGBD, que especificam esquemas que podem otimizar seu desempenho. Uma vez configurado o banco de dados, os arquivos segmentados na fase anterior foram enviados para o SGBD utilizando os comandos específicos para escrita em lote, como indicado em cada manual. O tempo de execução dessas operações foi medido com a biblioteca *timeit*, focando no tempo de resposta do comando executado pelo cursor. No caso do Apache Druid, esse método apenas realizava o agendamento de uma tarefa no SGBD, retornando instantaneamente um código identificador, mas não executando de fato a inserção dos dados. Para obter o tempo de execução da tarefa de inserção, foi coletado, via Interface de Programação de Aplicações, o tempo (em segundos) registrado para o identificador da tarefa.

3.7. Teste de leitura

Esse teste teve como objetivo responder aos tipos mais comuns de consultas em bancos de dados temporais, conforme definido por [Visperas and Chodpathumwan 2021]. Os tipos de buscas definidos nesse teste foram:

- Busca por um *timestamp* específico;
- Busca com agregação de dados;
- Busca com abrangência de todo o período de tempo da base;
- Busca em um intervalo de tempo específico.

Para manter a consistência, um intervalo e um *timestamp* específicos foram escolhidos arbitrariamente, assegurando que estivessem presentes em todos os segmentos de dados, o que garantia que as consultas fossem idênticas em todos os cenários. Utilizando a biblioteca *timeit*, o tempo de resposta do cursor para cada uma das consultas foi registrado.

3.8. Teste de ingestão Paralela

Após a conclusão do teste de leitura, os dados foram removidos das tabelas utilizando comandos de deleção específicos para cada SGBD, mantendo, no entanto, os esquemas das tabelas para o teste de ingestão paralela.

O teste de ingestão paralela, como pode ser visto na Figura 2, foi realizado dividindo-se aleatoriamente os registros do conjunto de dados em N partições de igual tamanho, processadas em paralelo por múltiplas *threads*, cujas quantias variaram entre 1, 2, 4, 8, 16 e 32. Cada *thread* estabelece uma conexão independente com o SGBD, sendo responsável por executar inserções de registros individualmente a partir de seu respectivo subconjunto de dados.

O tempo total de execução foi medido com a biblioteca *timeit*, registrando a duração combinada necessária para todas as *threads* completarem suas tarefas de inserção. Neste ponto, vale ressaltar que conforme descrito na subseção 3.6, o tempo coletado para o Apache Druid difere dos demais SGBDs. A dinâmica deste teste é ilustrada na Figura 2.

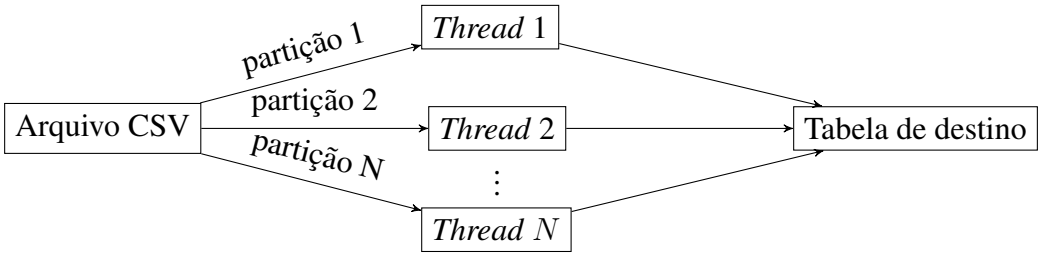

Figura 2. Fluxograma do teste de ingestão ponto-a-ponto.

4. Resultados e Discussão

4.1. Teste de carga em lote

Nesta subseção, apresentamos os resultados do teste de ingestão em lote exibidos na Tabela 3, que apresenta os tempos médios obtidos e, no caso do Druid, os tempos médios de execuções retornados pela tarefa. A tabela também apresenta a taxa de escrita em megabytes por segundo (MB/s), que foi calculada a partir do volume de dados em MB fornecido pelo sistema hospedeiro do teste. Os tamanhos dos arquivos utilizados nos testes são 0,3068, 1,6, 3,2, 15,8, 31,5, 157,6, 204,3 e 315,3 MB, ordenados da menor para a maior quantidade de *timestamps* distintos, respectivamente.

Para o Apache Druid, duas questões principais afetam os resultados atípicos. A primeira é o método de medição: os tempos são relatados pela duração da tarefa no banco de dados, tornando inviável compará-los com outros sistemas que utilizam a função *timeit* para medir diretamente o tempo de processamento das chamadas de função. A segunda questão decorre de sua arquitetura baseada em microsserviços, que envolve trocas

Tabela 3. *Benchmark* de escrita em lote de acordo com a quantidade de registros do arquivo.

DBT	Número de *timestamps*	Tempo médio de escrita (s)	Desvio padrão	Taxa de escrita (MB/s)
Druid	1.000	4,539	0,73	0,07
TDengine		0,028	0,013	10,96
TimescaleDB		0,052	0,00017	5,9
Druid	5.000	4,831	0,49	0,33
TDengine		0,049	0,00047	32,65
TimescaleDB		0,199	0,00032	8,04
Druid	10.000	5,035	0,027	0,64
TDengine		0,0995	0,00057	32,16
TimescaleDB		0,399	0,0018	8,02
Druid	50.000	6,191	0,058	2,55
TDengine		0,483	0,00047	32,71
TimescaleDB		2,104	0,0066	7,51
Druid	100.000	7,567	0,049	4,16
TDengine		0,992	0,0018	31,75
TimescaleDB		4,250	0,020	7,41
Druid	500.000	16,225	0,20	9,17
TDengine		4,926	0,0027	31,99
TimescaleDB		22,063	0,096	7,14
Druid	648.000	19,224	0,19	10,63
TDengine		6,307	0,016	32,39
TimescaleDB		27,408	0,14	7,45
Druid	1.000.000	26,573	0,19	11,87
TDengine		9,751	0,14	32,34
TimescaleDB		44,130	0,25	7,14

frequentes de mensagens durante cada tarefa executada ou agendada, prejudicando o desempenho com volumes de dados menores. No entanto, ao final dos dados tabulados, observa-se que o desempenho do Druid eventualmente supera o do TimescaleDB, apesar de ambos usarem PostgreSQL para armazenamento profundo na configuração padrão. Essa diferença de desempenho pode ser atribuída ao fato de que, no Apache Druid, os dados são primeiramente salvos em um arquivo mutável, o qual disponibiliza os dados para consulta (informando o fim da tarefa). Após, de forma periódica, o Apache Druid envia os dados para o serviço de armazenamento profundo. Por fim, é possível observar que o TDengine é a ferramenta que possui o menor tempo médio de escrita ao longo de todo o teste.

4.2. Teste de leitura

Os resultados das consultas realizadas no teste de leitura são apresentados na Tabela 4, onde é apresentado o tempo despendido em cada uma das consultas mencionadas na seção 3.7, além da média de tempo para as quatro consultas, conforme a variação no número total de registros. A taxa de leitura calculada foi baseada no tamanho em MB dos arquivos no sistema hospedeiro e no tempo médio de consulta.

Tabela 4. *Benchmark* de leitura variando de acordo com o tamanho de arquivo.

DBT	Número de *timestamps*	Tempo consulta 1 (s)	Tempo consulta 2 (s)	Tempo consulta 3 (s)	Tempo consulta 4 (s)	Tempo médio de consulta (s)	Taxa de leitura (MB/s)
Druid	1.000	0,53	0,088	0,075	0,044	0,18	1,70
TDengine		0,0014	0,00097	0,0010	0,0010	0,0011	278,91
TimescaleDB		0,0027	0,00050	0,0011	0,00065	0,0012	247,92
Druid	5.000	0,15	0,031	0,026	0,046	0,063	25,40
TDengine		0,0020	0,0013	0,0019	0,0020	0,0018	888,89
TimescaleDB		0,015	0,00073	0,0027	0,0026	0,0053	304,33
Druid	10.000	0,14	0,025	0,027	0,033	0,056	56,89
TDengine		0,0024	0,0013	0,0029	0,0020	0,0022	1488,37
TimescaleDB		0,016	0,00069	0,0053	0,0033	0,0063	505,93
Druid	50.000	0,14	0,025	0,025	0,038	0,057	277,19
TDengine		0,0034	0,0017	0,0087	0,0022	0,0040	4077,42
TimescaleDB		0,019	0,00070	0,022	0,0046	0,012	1365,01
Druid	100.000	0,14	0,027	0,025	0,033	0,056	560,00
TDengine		0,0036	0,0016	0,019	0,0026	0,0067	4701,49
TimescaleDB		0,021	0,00052	0,040	0,0056	0,017	1877,79
Druid	500.000	0,12	0,024	0,035	0,034	0,053	2959,62
TDengine		0,0034	0,0015	0,075	0,0028	0,021	7622,73
TimescaleDB		0,020	0,00049	0,12	0,0068	0,037	4279,70
Druid	648.000	0,13	0,023	0,029	0,027	0,052	3910,05
TDengine		0,0043	0,0017	0,13	0,0036	0,035	5853,87
TimescaleDB		0,022	0,00054	0,18	0,0077	0,053	3886,80
Druid	1.000.000	0,13	0,020	0,033	0,027	0,053	6005,71
TDengine		0,0047	0,0021	0,23	0,0034	0,060	5250,62
TimescaleDB		0,024	0,00049	0,26	0,0090	0,073	4297,10

Os resultados mostram que o TDengine e o TimescaleDB tiveram desempenhos bastante semelhantes ao longo do teste. No entanto, conforme o espaço de busca aumentava, o TDengine demonstrou melhores resultados, uma vantagem que pode ser atribuída ao uso de sub-tabelas. Isso significa que, para o TDengine, o espaço de busca efetivo foi de apenas 1/6 dos registros em comparação ao mesmo período de tempo (número de *timestamps*) de seus concorrentes. Por outro lado, o Druid apresentou um desempenho geralmente inferior ao das outras soluções, mas manteve uma consistência notável durante o teste, com tempos de resposta próximos aos dos outros sistemas no final.

Existem várias configurações que podem otimizar as buscas em cada sistema testado. Por exemplo, enquanto o Druid suporta configurações de agregações, o TDengine se beneficia de buscas em sub-tabelas. Por fim, o TimescaleDB tira proveito de compressão de dados e do uso de índices. Portanto, uma configuração adequada e uma modelagem eficaz do SGBD podem fazer uma grande diferença no desempenho, algo que deve ser considerado cuidadosamente durante a realização dos testes.

4.3. Teste de ingestão Paralela

Por fim, a Tabela 5 apresenta os resultados do último teste realizado. Assim como no teste de carga em lote, os valores do Druid são atípicos em relação aos demais, devido às particularidades do método de coleta das métricas. Portanto, esses resultados do Druid

não devem ser considerados para comparações diretas com os outros sistemas.

Tabela 5. *Benchmark* de escrita variando o número de *threads*

DBT	Número de *threads*	Tempo médio de escrita (s)	Desvio Padrão	Taxa de escrita (MB/s)
Druid	1	7,63	0,19	95,58
TDengine		190,53	8,69	3,83
TimescaleDB		154,73	0,61	4,72
Druid	2	11,97	0,018	60,93
TDengine		92,37	1,72	7,90
TimescaleDB		81,14	0,062	8,99
Druid	4	12,00	0,043	60,82
TDengine		47,70	0,93	15,30
TimescaleDB		42,51	0,12	17,16
Druid	8	12,04	0,95	60,60
TDengine		31,13	0,20	23,44
TimescaleDB		23,52	0,17	31,02
Druid	16	12,08	0,19	60,42
TDengine		38,09	0,048	19,16
TimescaleDB		15,29	0,071	47,71
Druid	32	12,04	0,031	60,61
TDengine		37,46	0,070	19,48
TimescaleDB		17,66	0,045	41,31

Para facilitar a comparação, o gráfico na Figura 3 omite o Apache Druid e demonstra que, ao longo do teste de concorrência, o TimescaleDB supera o TDengine em desempenho. Conforme descrito na documentação do TDengine, a estratégia de dividir dados em sub-tabelas é geralmente adotada para melhorar o tempo de escrita, minimizando problemas de concorrência e bloqueios. No entanto, nesse teste específico, o TimescaleDB apresentou melhores resultados, o que pode ser atribuído à sua base robusta no PostgreSQL. Este último permite ajustes relacionados a bloqueios que, embora não tenham sido explorados neste trabalho, poderiam ser considerados em avaliações futuras.

É possível observar que há uma redução ou estabilização no desempenho ao aumentar o número de *threads* de 8 para 16. Esse fenômeno pode ser um reflexo da falta de configuração adequada dos parâmetros nos SGBDs, o que restringe a abertura de mais conexões.

5. Conclusão

Através dos testes realizados, observou-se que não há uma solução definitivamente superior para o cenário de dados de potência elétrica. As diferenças de desempenho entre os Sistemas Gerenciadores de Banco de Dados Temporais (SGBDTs) testados, incluindo TimescaleDB, TDengine e Druid, foram relativamente pequenas. Essa similaridade de desempenho sugere que a escolha do SGBDT ideal pode depender mais de características específicas de cada sistema e de como essas características se alinham às necessidades operacionais e de integração do ambiente em questão.

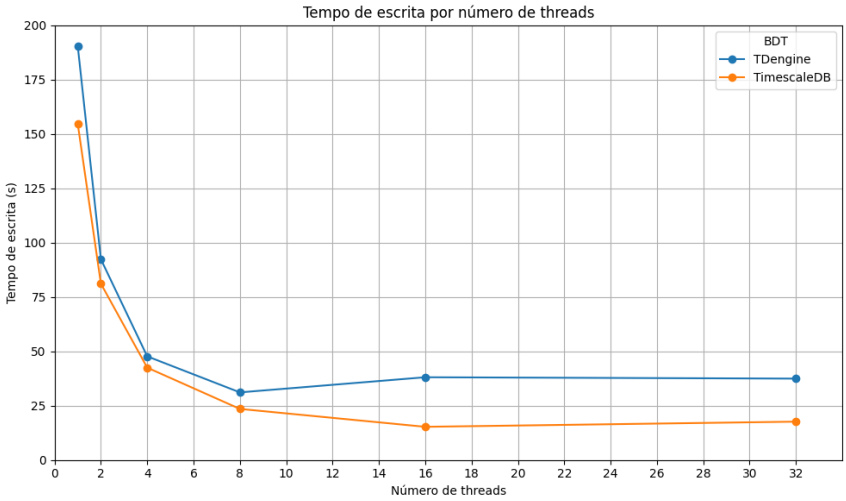

Figura 3. Teste de escrita concorrente.

Especificamente no setor de energia elétrica, os SGBDTs são cruciais para o gerenciamento eficiente de grandes volumes de dados de medição que são gerados continuamente. O Druid, por exemplo, mostrou-se adequado para cenários nos quais a análise é prioritária e a rapidez na inserção de dados não é crítica. Por outro lado, o TDengine destacou-se em ambientes que não requerem uma gestão complexa de relacionamentos entre entidades e onde a escrita não concorrente é uma vantagem. O TimescaleDB, que se baseia na robustez do PostgreSQL, provou ser eficaz em situações que exigem alta concorrência, oferecendo uma ampla gama de opções de compressão e rotinas de gerenciamento bem estabelecidas.

Nesse estudo foi constatado que que enquanto algumas soluções de gerenciamento de dados podem oferecer melhor desempenho em determinadas condições, a adaptabilidade e a capacidade de integrar eficientemente com outros sistemas existentes também são consideradas importantes para sua escolha. Portanto, a decisão final deve considerar não apenas o desempenho isolado dos SGBDTs mas também como eles se encaixam no ecossistema tecnológico mais amplo.

Em suma, este estudo contribuiu para o entendimento de como diferentes SGBDs temporais podem ser empregados eficazmente em cenários de dados de potência elétrica. Além disso, ampliamos a pesquisa original utilizando uma máquina de hospedagem com um número maior de *threads* físicas e empregando novas versões de softwares.

Para trabalhos futuros, recomenda-se a inclusão de outros SGBDTs e a atualização das versões testadas para explorar avanços recentes na tecnologia de bancos de dados. Além disso, ajustes finos na configuração dos SGBDs, como a personalização do tamanho do cache, o ajuste das configurações de índice e a otimização dos parâmetros de concorrência, poderiam ser explorados para entender melhor como aprimorar o desempenho específico para o domínio de dados de potência elétrica. A realização de testes adicionais para validar a compressão de dados e o impacto do custo em ambientes de nuvem também podem fornecer *insights* valiosos para a adoção dessas tecnologias em escala industrial.

Referências

Allen, A., Singh, M., and Muljadi, E. (2014). PMU Data Event Detection: A User Guide for Power Engineers, 2014. - https://www.nrel.gov/docs/fy15osti/61664.pdf. National Renewable Energy Laboratory.

Bader, A., Kopp, O., and Falkenthal, M. (2017). Survey and Comparison of Open Source Time Series Databases. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017) - Workshopband*, pages 249–268. Gesellschaft für Informatik e.V., Bonn.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, page 143–154, New York, NY, USA. Association for Computing Machinery.

Esling, P. and Agon, C. (2012). Time-Series Data Mining. *ACM Comput. Surv.*

Hao, Y., Qin, X., Chen, Y., Li, Y., Sun, X., Tao, Y., Zhang, X., and Du, X. (2021). TS-Benchmark: A Benchmark for Time Series Databases. *2021 IEEE 37th International Conference on Data Engineering (ICDE)*.

Liu, Q., Huang, Y., Guan, K., Xiong, L., Zhang, J., Fan, W., and Guan, R. (2024). Design of High Voltage Power Management System Based on IoTDB Time Series Database. *2024 IEEE 4th International Conference on Power, Electronics and Computer Applications (ICPECA)*.

Liu, R. and Yuan, J. (2019). Benchmarking Time Series Databases with IoTDB-Benchmark for IoT Scenarios. *arXiv e-prints*.

Mostafa, J., Wehbi, S., Chilingaryan, S., and Kopmann, A. (2022). SciTS: A Benchmark for Time-Series Databases in Scientific Experiments and Industrial Internet of Things. *arXiv e-prints*.

solid, I. (2024). DB-engine ranking - popularity ranking of time series DBMS.

Statista (2024). Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025.

Sychev, I., Abdelkader, A., Kozak, W., Bonetto, R., and Fitzek, F. H. P. (2020). Closed Loop Benchmark for Timeseries Databases. *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*.

Taipalus, T. (2024). Database management system performance comparisons: A systematic literature review. *Journal of Systems and Software*, 208:111872.

Visperas, L. K. and Chodpathumwan, Y. (2021). Time-Series Database Benchmarking Framework for Power Measurement Data. *2021 Research, Invention, and Innovation Congress: Innovation Electricals and Electronics (RI2C)*.