
**Universidade Federal de Pernambuco**

Centro de Informática

Curso de Engenharia da Computação

# Ambiente de Medição de Consumo de Energia para SGBDs NoSQL

Trabalho de Conclusão de Curso de Graduação

por

João Pedro Souza Pereira de Moura

Orientador: Prof. Eduardo Tavares

Recife, Abril / 2025

João Pedro Souza Pereira de Moura

# Ambiente de Medição de Consumo de Energia para SGBDs NoSQL

Monografia apresentada ao Curso de Engenharia da Computação, como requisito parcial para a obtenção do Título de Bacharel em Engenharia da Computação, Centro de Informática da Universidade Federal de Pernambuco.

Orientador: Prof. Eduardo Tavares

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Moura, João Pedro Souza Pereira de.
Ambiente de medição de consumo de energia para SGBDs NoSQL / João Pedro Souza Pereira de Moura. - Recife, 2025.
59 p : il., tab.

Orientador(a): Eduardo Antônio Guimarães Tavares
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2025.
Inclui referências.

1. SGBD. 2. Eficiência energetica. 3. Ambiente de medição. 4. Medição de consumo energetico. 5. Otimização de recursos computacionais. 6. Computação sustentável. I. Tavares, Eduardo Antônio Guimarães. (Orientação). II. Título.

000 CDD (22.ed.)

João Pedro Souza Pereira de Moura

Ambiente de Medição de Consumo de Energia para SGBDs NoSQL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Engenharia da Computação.

Aprovado em: 03 de Abril de 2025

Banca Examinadora:

---

Prof. Dr. Eduardo Tavares (Orientador)
Centro de Informatica da UFPE

---

Prof. Dr. Jamilson Ramalho (Examinador Interno)
Centro de Informatica da UFPE

Recife

2025

**Agradecimentos**

Primeiramente gostaria de agradecer toda a minha família por todo o apoio durante todos esses anos, em especial aos meus pais, Jane e Márcio, que sempre me apoiaram e fizeram tanto por mim para que essa conquista fosse possível, aos meus primos, Hayla, Hialy e Hierlan, com quem convivi todos esses anos.

Aos amigos que fiz no curso durante toda a minha jornada, em especial aos amigos do Team Grande: Edinaldo Filho, Enrique Laborão, Felipe Lin, Gabriel Evangelista, João Gabriel Vasconcelos, Lucas Calabria, Marcelo Valois, Matheus Machado, Maycon Cunegundes, Rafael Bernardo e Vinícius Revoredo, nossa trajetória foi árdua, mas vencemos, aos amigos do RobôCIn. A Milena, minha namorada, obrigado por todo apoio e suporte.

Por fim, gostaria de agradecer aos Docentes do CIn que fizeram parte da minha graduação, por todas as lições e ensinamentos, em especial ao Professor Eduardo, pela amizade e orientação durante toda a graduação, sem dúvidas suas contribuições foram inestimáveis.

*I do not know whether I shall make progress, but I should prefer to lack success rather than to lack faith.*

Seneca, Epistulae Morales ad Lucilium, Letter 25

## RESUMO

O crescimento exponencial do volume de dados processados por aplicações computacionais exige soluções eficientes em termos de consumo energético e desempenho. Os Sistemas de Gerenciamento de Bancos de Dados NoSQL desempenham um papel fundamental no armazenamento e recuperação de excesso de informação, especialmente em infraestruturas distribuídas e *data centers*. No entanto, a influência dessas tecnologias no consumo energético ainda é um desafio pouco explorado.

Este trabalho apresenta o desenvolvimento e validação de um ambiente de medição de consumo de energia para SGBDs NoSQL. Para isso, foi projetado um servidor de medição baseado em Arduino, utilizando sensores especializados para capturar dados de corrente e tensão. A infraestrutura desenvolvida possibilita a avaliação do impacto energético de diferentes bancos de dados, permitindo a comparação de seu consumo energético em diferentes cargas de trabalho.

Os experimentos foram conduzidos utilizando o *Yahoo! Cloud Serving Benchmark* (YCSB) para simular operações comuns em bancos de dados NoSQL. Além disso, foram aplicadas técnicas estatísticas, como Bootstrap e T-Student, para garantir a precisão e confiabilidade dos resultados obtidos. Os dados coletados foram validados por meio da comparação com um medidor comercial MINIPA ET-4091, evidenciando a robustez da abordagem adotada.

Os resultados indicam que a arquitetura do banco de dados influencia diretamente o consumo de energia. Bancos de dados chave-valor, como o Redis, demonstraram menor consumo energético, enquanto bancos orientados a documentos, como o MongoDB, apresentaram maior demanda energética, especialmente em operações de escrita e atualização. Observou-se também que a eficiência energética varia conforme a carga de trabalho e a complexidade das operações realizadas.



Palavras-chave: SGBD, Consumo de energia, NoSQL, Eficiência energética, Ambiente de medição, Banco de Dados Distribuído, Benchmarking Energético, Medição de Consumo Elétrico, Computação Sustentável, Otimização de Recursos Computacionais.

## ABSTRACT

The exponential growth in the volume of data processed by computational applications requires efficient solutions in terms of energy consumption and performance. NoSQL Database Management Systems play a fundamental role in the storage and retrieval of excess information, especially in distributed infrastructures and data centers. However, the influence of these technologies on energy consumption is still an underexplored challenge.

This work presents the development and validation of an energy consumption measurement environment for NoSQL DBMSs. For this purpose, an Arduino-based measurement server was designed, using specialized sensors to capture current and voltage data. The developed infrastructure allows the evaluation of the energy impact of different databases, allowing the comparison of their energy consumption in different workloads.

The experiments were conducted using the Yahoo! Cloud Serving Benchmark (YCSB) to simulate common operations in NoSQL databases. In addition, statistical techniques, such as Bootstrap and T-Student, were applied to ensure the accuracy and reliability of the results obtained. The collected data were validated by comparison with a commercial MINIPA ET-4091 meter, demonstrating the robustness of the adopted approach.

The results indicate that the database architecture directly influences energy consumption. Key-value databases, such as Redis, demonstrated lower energy consumption, while document-oriented databases, such as MongoDB, presented higher energy demand, especially in write and update operations. It was also observed that energy efficiency varies according to the workload and the complexity of the operations performed.



Keywords: DBMS, Energy consumption, NoSQL, Energy efficiency, Measurement environment, Distributed Database, Energy Benchmarking, Electricity Consumption Measurement, Sustainable Computing, Optimization of Computational Resources.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE SIGLAS

| | |
|---|---|
| NoSQL | Not Only SQL |
| YCSB | Yahoo! Cloud Serving Benchmark |
| SGBD | Sistema de Gerenciamento de Banco de Dados |
| IoT | Internet das Coisas |
| SQL | Structured Query Language |
| API | Application Programming Interface |
| JSON | JavaScript Object Notation |
| kWh | Quilowatt-hora |
| TWh | Terawatt-hora |
| CA | Corrente Alternada |
| W | Watt |
| P | Potência Ativa |
| Q | Potência Reativa |
| S | Potência Aparente |
| VA | Volt-ampère |
| FP | Fator de Potência |
| SI | Sistema Internacional de Unidades |
| E | Energia |
| J | Joule |
| V | Volt |
| SO | Sistema Operacional |
| DoE | Design of Experiments |
| DOC | Documento |
| KEY | Chave |
| s | Segundo |

**SUMÁRIO**

# 1 INTRODUÇÃO

## 1.1 Contexto

A importância e a presença dos aplicativos na rotina diária, tanto para usuários quanto para empresas, têm aumentado de forma constante. Esses recursos móveis atendem à demanda por competitividade, interação e até diversão, destacando uma tendência rumo à melhoria das interações, resultando em comunicações mais eficientes. Com a frequência no lançamento de novos produtos e sensores cada vez mais compactos, essas soluções estão prontamente disponíveis, tornando-se uma parte vital do cotidiano digital. [1]

O desenvolvimento tecnológico tem avançado rapidamente, e as formas de incorporar conteúdo em dispositivos como *desktops*, *laptops*, *tablets*, *smartphones*, dispositivos vestíveis e TVs inteligentes precisam acompanhar essa evolução. Isso demanda uma infraestrutura de *software* e *hardware* que proporcione uma experiência eficiente e flexível, otimizando o aproveitamento dos recursos computacionais e assegurando que os sistemas funcionem com o menor gasto energético possível.

Em um panorama no qual a quantidade de dados aumenta de maneira exponencial, a eficiência energética dos sistemas computacionais se tornou um aspecto crucial, tanto pela sustentabilidade quanto pelo desempenho. O tratamento, armazenamento e recuperação de grandes volumes de informação requerem um consumo elevado de energia, o que impacta diretamente a escalabilidade das aplicações e as despesas operacionais das infraestruturas de TI. Portanto, é vital examinar como diferentes métodos de gestão de dados influenciam o consumo de energia dos sistemas. [2]

Dentre os desafios enfrentados pela comunidade científica e pela indústria de *software*, um dos mais significativos é a criação de estratégias que visem diminuir o impacto energético dos bancos de dados, especialmente no contexto de aplicações que tratam de *Big Data* [3]. Embora bancos de dados NoSQL e relacionais sejam amplamente empregados para diversas finalidades, o efeito energético de suas operações ainda não é totalmente esclarecido [4]. Aspectos como a eficiência na recuperação de dados, otimização de operações de escrita e leitura, além do custo energético associado a cada ação, são elementos essenciais para garantir um melhor uso dos recursos computacionais.

Este trabalho de graduação visa contribuir para a pesquisa sobre o impacto do consumo de energia em sistemas de gestão de bancos de dados, explorando como diferentes

métodos podem afetar a eficiência energética das aplicações. Para isso, foi utilizado o *Yahoo! Cloud Serving Benchmark (YCSB)*, uma ferramenta desenvolvida para avaliar o desempenho e a energia consumida por diversos bancos de dados. Esse *framework* simula o comportamento de uma aplicação real, permitindo medir o efeito das operações de inserção, leitura e atualização no consumo energético de cada Sistema de Gerenciamento de Banco de Dados (SGBD) analisado.

Ao examinar o uso de energia desses sistemas, pretende-se oferecer uma compreensão mais precisa sobre as práticas ideais para reduzir o impacto ambiental e aprimorar a utilização de recursos computacionais em aplicações que gerenciam abundancia de dados. Dessa forma, esta pesquisa poderá ajudar programadores e cientistas a fazer escolhas mais eficazes sobre quais tecnologias implementar, levando em conta não apenas a escalabilidade e o desempenho, mas também a sustentabilidade dos sistemas computacionais.

## 1.2 **Motivação**

Atualmente, estamos imersos em um período caracterizado pela aceleração significativa na produção de dados oriundos de múltiplas fontes. Aparelhos móveis, sensores, câmeras, sistemas de geolocalização, plataformas de mídias sociais, operações comerciais e aparelhos ligados à Internet das Coisas (IoT) geram um incessante e volumoso fluxo de informações. Esses dados podem ser classificados como estruturados, semiestruturados ou não estruturados, evidenciando a variedade e complexidade dos sistemas tecnológicos atuais.

Dentro desse cenário, a avaliação e o manuseio eficaz dessas informações tornaram-se essenciais para aplicações em várias disciplinas, incluindo ciência, economia e setores industriais. Porém, um dos principais obstáculos enfrentados tanto pela comunidade acadêmica quanto pelo setor tecnológico é o alto custo computacional relacionado ao armazenamento, manipulação e administração desses dados. Esse gasto abrange não apenas questões financeiras, mas também envolve o consumo energético, uma preocupação cada vez mais relevante à medida que a necessidade por infraestrutura digital cresce.

A questão do consumo de energia associado ao manejo de abundantes de dados tornou-se fundamental. *Data centers*, servidores dispersos e sistemas de computação em nuvem exigem elevadas quantidades de eletricidade [5], o que afeta não apenas os custos operacionais, mas também o meio ambiente. A eficiência energética dessas instalações

se tornou um critério essencial na seleção de tecnologias e estruturas computacionais, estimulando investigações para criar soluções que busquem diminuir o consumo de energia sem prejudicar o desempenho e a escalabilidade.

Os métodos tradicionais de gestão de dados, como os Bancos de Dados Relacionais, lidam com dificuldades particulares neste contexto. O grande custo adicional relacionado à manutenção da consistência, a dificuldade na replicação de dados e a exigência de uma infraestrutura sólida impactam diretamente o consumo de energia. Em resposta, novas soluções estão sendo desenvolvidas, como os bancos de dados NoSQL e abordagens de armazenamento distribuído, que visam melhorar a eficiência dos recursos computacionais e reduzir o consumo energético.

A busca por soluções mais eficientes tem gerado pesquisas sobre modelos de armazenamento e processamento de dados que integrem escalabilidade e eficiência energética, por exemplo, Prokhorenko e Babar [6](2024) que avalia quatro soluções de bancos de dados SQL e NoSQL em termos de consumo de energia: Cassandra, MongoDB, Redis e MySQL. Também pode se citar Lella et al. [3](2024) que investiga o consumo de energia de consultas em quatro bancos de dados: MySQL, PostgreSQL, MongoDB e Couchbase.

A distribuição de carga, a otimização de algoritmos de indexação e recuperação de informações, além da adoção de infraestruturas híbridas, são algumas das estratégias utilizadas para mitigar o impacto energético. Ademais, a escolha de *hardware* especializado e a aplicação de técnicas de computação verde tornam-se fatores cruciais para o futuro dos sistemas de gerenciamento de dados.

## 1.3 **Objetivos**

### 1.3.1 Objetivo Geral

O objetivo desse trabalho é desenvolver e validar um ambiente de medição de consumo de energia para Sistemas de Gerenciamento de Bancos de Dados (SGBDs) NoSQL, permitindo a análise do impacto energético das operações realizadas por diferentes bancos de dados.

### 1.3.2 Objetivos Específicos

Para atingir o objetivo principal, foram definidos os seguintes objetivos específicos:

- Projeto e implementação de um ambiente de medição capaz de monitorar o consumo de energia de SGBDs NoSQL, utilizando sensores e dispositivos de medição compatíveis.
- Avaliação do consumo energético de diferentes SGBDs NoSQL por meio de testes experimentais e estatísticos.
- Comparação dos resultados obtidos com medições realizadas utilizando equipamentos comerciais, garantindo a precisão dos dados coletados.
- Aplicação de técnicas estatísticas para análise e validação dos resultados, possibilitando inferências confiáveis sobre o impacto energético das operações dos bancos de dados analisados.
- Proposição de diretrizes e recomendações para a escolha e otimização de SGBDs NoSQL com base em critérios de eficiência energética.

## 1.4 **Estrutura do trabalho**

O Capítulo 2 apresenta o referencial teórico essencial para a compreensão deste estudo. São explorados conceitos fundamentais sobre Sistemas de Gerenciamento de Bancos de Dados (SGBDs) NoSQL, consumo de energia em sistemas computacionais e técnicas estatísticas aplicadas à análise do consumo energético. Além disso, são abordados métodos de medição de potência e eficiência energética, bem como estratégias de benchmarking para avaliar o impacto energético dos bancos de dados analisados.

O Capítulo 3 detalha o ambiente de medição desenvolvido para a realização dos experimentos. São descritos os principais componentes do sistema, incluindo o servidor de medição baseado em Arduino, os sensores utilizados para a captura de corrente e tensão, e a arquitetura geral do sistema de coleta e análise dos dados. Adicionalmente, são explicados os métodos empregados para garantir a precisão das medições e a confiabilidade dos resultados obtidos.

No Capítulo 4, são apresentados os resultados experimentais obtidos ao longo do estudo. Inicialmente, são descritos os procedimentos de validação do ambiente de medição, comparando os dados coletados pelo sistema desenvolvido com aqueles obtidos por um medidor comercial. Em seguida, são analisados os padrões de consumo energético dos

bancos de dados avaliados, considerando diferentes cenários de carga de trabalho. Os resultados são interpretados com base em técnicas estatísticas, permitindo inferências sobre o impacto energético das operações realizadas.

Por fim, o Capítulo 5 apresenta as conclusões do estudo, sintetizando as principais descobertas e contribuições da pesquisa. Além disso, são discutidas as limitações do trabalho e sugeridas direções para pesquisas futuras, incluindo a ampliação do escopo da análise para outros bancos de dados, a diversificação dos cenários de carga de trabalho e o aprimoramento das estratégias de otimização do consumo energético.

# 2 REFERENCIAL TEÓRICO

## 2.1 NoSQL

O Not Only SQL (NoSQL) é uma categoria de Banco de dados que surgiu visando gerenciar grandes volumes de dados semi-estruturados ou não estruturados, que necessitam de alta disponibilidade e escabilidade, essa categoria surgiu com a ineficiência dos bancos de dados relacionais para lidar com essas necessidades. O NoSQL se viu imprescindível com o grande volume de dados gerado por aplicações web que precisam de escabilidade sob demanda e o elevado grau de disponibilidade. Devido à impossibilidade de adequar os dados a um modelo rígido e a restrição na escabilidade necessária para lidar com a quantidade de informações. [7]

Um dos movimentos de código aberto mais influentes que ajudaram o NoSQL a crescer foi o movimento iniciado pelo Google com o *BigTable* [8] em 2004, para atender à necessidade da empresa de armazenar seus grandes dados e alcançar as características de alto desempenho, escalabilidade e disponibilidade que o NoSQL oferece.

### 2.1.1 Principais características dos bancos de dados NoSQL

- **Flexibilidade:** Bancos de dados NoSQL geralmente oferecem um esquema que permite um desenvolvimento mais rápido e iterativo. Devido ao modelo de dados mais flexível, bancos de dados NoSQL são bem adequados para dados semi-estruturados e não estruturados.

- **Escalabilidade:** Bancos de dados NoSQL são amplamente construídos para escalabilidade horizontal em clusters de hardware distribuído em vez de escalabilidade vertical usando servidores caros e poderosos. Certos provedores em nuvem gerenciam essas operações em segundo plano como um serviço totalmente gerenciado.

- **Alto Desempenho:** Bancos de dados NoSQL são otimizados para modelos de dados específicos e padrões de acesso que lhes permitem alcançar desempenho superior em comparação com tentar obter a mesma funcionalidade usando bancos de dados relacionais.

- **Funcional:** Bancos de dados NoSQL oferecem APIs funcionais e tipos de dados

projetados para corresponder a cada um de seus modelos de dados.

### 2.1.2 Tipos de bancos de dados NoSQL

Atualmente, existem mais de 225 bancos de dados NoSQL, categorizados pelo modelo de dados e estrutura que utilizam, conforme delineado em [9]. Dos 225, podemos mencionar 4 exemplos mais usados: Chave-valor, Documento, Orientado a Grafos e Colunar, onde estes, exceto o orientado a grafos, manipulam seus dados como unidades, o que é uma estrutura mais complexa do que apenas n tuplas.

- **Chave-Valor:** Os bancos de dados de chave-valor são altamente particionáveis, permitindo escalabilidade horizontal em uma escala que outros tipos de bancos de dados não conseguem. Particularmente propício ao modelo de dados chave-valor, jogos, tecnologia de publicidade e IoT parecem como casos de uso. Eles são principalmente usados como um sistema de armazenamento embutido e todos os tipos de cache de dados.

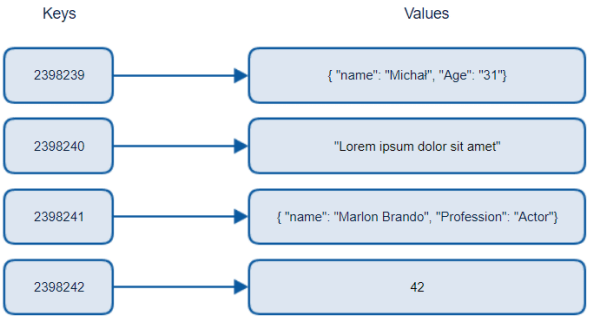


Figura 1: Exemplo de estrutura SGBD chave-valor

Modelos de DBMS Chave-valor são como um dicionário encontrado em algumas linguagens de programação de alto nível; eles usam uma tabela *hash* onde as chaves correspondem aos valores. Uma técnica popular para bancos de dados *Key-Value* é o Redis [1] [10], que difere dos outros porque o Redis nos permite manipular seus dados apenas por meio de operações simples: inserção, deleção, atualização e busca, baseando-se apenas no campo de chave, a qual é o campo usado para operar com o elemento. Como este paradigma é simples, uma vantagem é que tal abordagem permite recuperação de informações em alta velocidade, o que é altamente necessário

[1] Redis: https://redis.io/pt/

em aplicativos. Além disso, este modelo é fácil de implementar e adicionar novos dados em tempo de execução, mas não suporta relações de dados.

- **Documento:** Os dados são tipicamente representados como um objeto ou um documento do tipo JSON (*JavaScript Object Notation*) no código de aplicação para fazer sentido para os desenvolvedores, já que este é um modelo de dados eficiente e familiar para lidar, mas pode estar em qualquer outro formato, estruturado, semi-estruturado ou não estruturado. Os bancos de dados de documentos permitem que os desenvolvedores armazenem dados no mesmo formato de modelo de documentos que eles usam no código de aplicativo e também consultem.

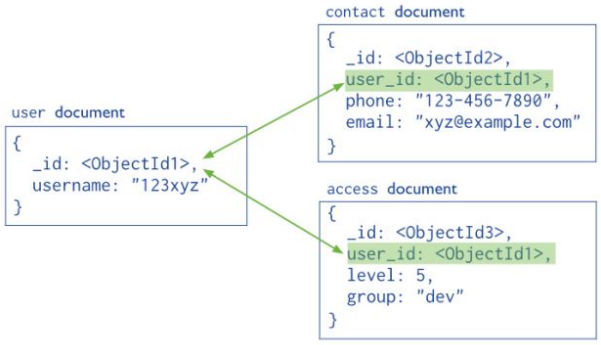


Figura 2: Exemplo de estrutura do SGBD baseado em documento

Bancos de dados de documentos evoluem conforme as necessidades do aplicativo mudam, devido às suas características semi-estruturadas, flexíveis e hierárquicas [11]. É adequado para catálogos, perfis de usuário e sistemas de gerenciamento de conteúdo, onde cada documento é diferente e desenvolvido de maneira gradual. Uma das abordagens mais poderosas que pode ser utilizada é o MongoDB [2] [12].

Analogamente aos bancos de dados relacionais, podemos dizer que uma coleção de documentos corresponde a uma tabela, e um documento na coleção corresponde a um registro na tabela. Uma das principais características distintivas de ambos é a capacidade de os documentos manterem uma relação entre eles sem chaves estrangeiras, como você faria em um modelo relacional. Vale a pena mencionar, mais sobre isso em um momento, essa semelhança que ambos têm, mas também, ao contrário da interface rígida de um modelo orientado a banco de dados relacional, o modelo de documentos é tal que o distingue com alta flexibilidade de modelos

[2]MongoDb: https://www.mongodb.com/

relacionais.

- **Orientado a Grafos:** Um banco de dados orientado a grafos é projetado para permitir a criação e execução de aplicativos que requerem trabalhar com um conjunto rico de interconexões entre dados. Projetados para armazenar relações e navegar por elas, eles permitem o armazenamento de dados por meio de nós que representam entidades de dados e arestas que representam relações entre entidades. É uma categoria de banco de dados projetada para superar algumas das limitações de bancos de dados relacionais. Visando lidar principalmente com dados altamente conectados e dinâmicos em volume, os exemplos mais comuns que podem se beneficiar de bancos de dados NoSQL (como grafos) incluem redes sociais, motores de recomendação e outros dados de formas complexas. Ex. Neo4j[3] [13].

  Com relações entre dados armazenados como arestas, os modelos de DBMS orientados a grafos fornecem um meio para modelar relações complexas entre duas ou mais entidades. O conjunto de dados está interconectado, e o modelo orientado a grafos também é mais eficiente em consultas que envolvem conexão indireta, já que as relações no grafo são diretamente representadas pelas arestas, anulando assim a necessidade de uso de junções custosas dos modelos relacionais.

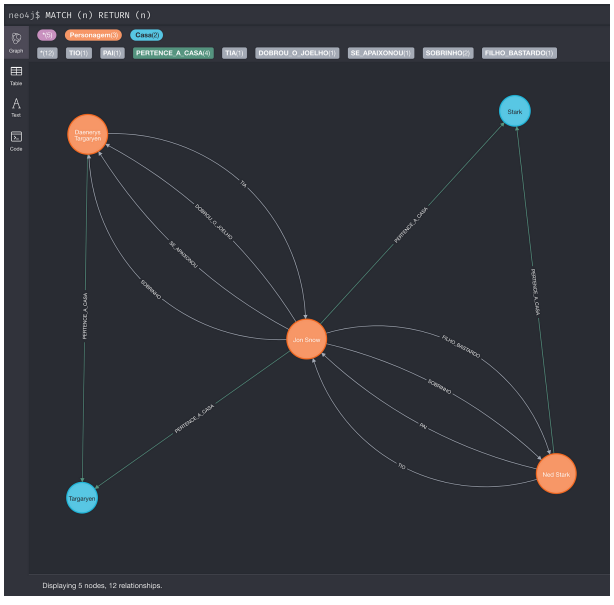


Figura 3: Exemplo de estrutura de SGBD orientado a grafo

- **Colunares:** Em bancos de dados colunar, os dados são armazenados em colu-

[3]Neo4j: https://neo4j.com/

nas apontadas ao mesmo tempo, o que melhora drasticamente seu desempenho ao reduzir a quantidade de informação que você precisará carregar no disco. Eles representam as entidades dentro das tabelas, escrevem-nas no disco e agrupam os dados por diferentes colunas. Este método beneficia os leitores, uma vez que fornece mais anotações de dados semânticos aos sistemas que precisam ler e realizar operações de consulta em grandes volumes de dados. Cassandra[4] [14], por exemplo.

Devido às consultas analíticas, este paradigma é mais adequado para cargas de trabalho de suporte à decisão, na verdade, uma vez que o acesso é direto às colunas que importam, modelos colunares geralmente são mais eficientes em consultas que realizam operações de agregação.

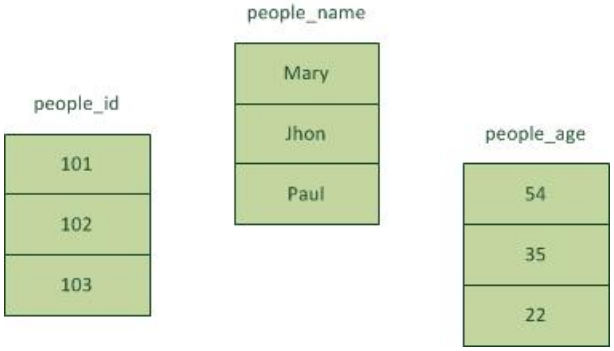


Figura 4: Exemplo de estrutura de SGBD Colunar

## 2.2 **Consumo de Energia**

### 2.2.1 Consumo de energia em sistemas computacionais

O crescimento exponencial do tráfego de dados e dos serviços e aplicações digitais reflete-se diretamente no consumo de energia em sistemas computacionais. Como infraestrutura essencial para computação em nuvem, serviços de big data, treinamento de modelos de IA e aplicações online, os Centros de Dados consomem uma quantidade significativa de energia. Eles se tornaram computadores em escala de armazém, nos quais o desempenho agregado e a eficiência de milhares de servidores têm prioridade. [5].

O relatório mostra um efeito rebote para todos os dados do consumo global de eletricidade de 2007 a 2012, onde as melhorias na eficiência reduzem os custos, aumentando

[4]Cassandra: https://cassandra.apache.org/*index.html*

o uso e o consumo total de energia à medida que a demanda global por dados e serviços aumenta, apesar das melhorias na eficiência energética e dos avanços tecnológicos. [15]

Em 2011, representavam cerca de 1,5% de toda a energia consumida no planeta pelos centros de dados. Nos Estados Unidos, o consumo de energia dos centros de dados naquele ano foi de cerca de 100 bilhões de quilowatt-hora (kWh). As emissões de carbono aumentaram cerca de 6% ao ano e representavam cerca de 12% das emissões globais [16]. O consumo de energia dentro dos centros de dados era de aproximadamente 4,4% de toda a energia consumida em 2023 (Departamento de Energia dos EUA DOE 2021), e até 2028, estima-se que consomem entre 6,7% e 12% de toda a energia consumida nos Estados Unidos. Segundo o Departamento de Energia dos EUA, o consumo total de energia dos centros de dados foi de 58 terawatts-hora (TWh) em 2014, 176 TWh em 2023 e projeta-se estar entre 325 e 580 TWh até 2028. [17].

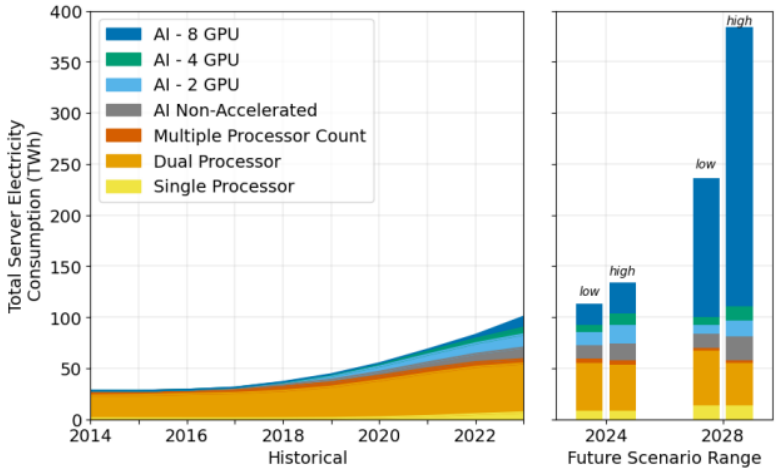


Figura 5: Uso anual de eletricidade do servidor por tipo.

O consumo de energia dissipado pelos servidores, redes e sistemas de resfriamento compõe o uso total de energia, tornando, assim, a eficiência energética uma consideração significativa de design para o centro de dados moderno, e a dissipação e o consumo de energia tornaram-se um objetivo primordial de design para processadores e sistemas. Os meros avanços na tecnologia de semicondutores não são suficientes para superar esses desafios; inovações arquitetônicas são necessárias para a computação eficiente em termos de energia, por meio de técnicas como escalonamento dinâmico de tensão e frequência, processadores multicores e aceleradores de *hardware*. [18]

A eficiência energética não apenas em termos de *hardware*, mas também em termos de *software* precisa ser co-projetada para um TI ecoeficiente com uma abordagem holística,

pois meios tradicionais que tratam apenas de melhorias no *hardware* não são suficientes. Em vez disso, o aproveitamento dessas práticas eficientes em termos de energia em toda a pilha de TI deve ser um dado [19]. Um dos grandes desafios que os campeões do consumo de energia nos centros de dados precisam enfrentar é encontrar o equilíbrio certo entre desempenho e eficiência. A eficiência energética não deve ser em detrimento da confiabilidade e responsividade do serviço.

### 2.2.2 Fundamentos de potência elétrica

Antes de lidar com o consumo de energia em sistemas computacionais, devemos entender o básico da eletricidade. Em sistemas de corrente alternada (CA), a Potência elétrica é composto por três partes [20]:

- Potência Real (P): É medida em watts (W) e representa a energia usada pelos dispositivos para trabalhar eficientemente em uma tarefa (como os servidores processam informações).
- Potência Reativa (Q): Potência reativa é medida em volt-ampere reativo (var) e está relacionada à energia trocada entre a fonte e os elementos reativos (capacitores e indutores), não realizando trabalho útil, mas necessária para manter campos magnéticos.
- Potência Aparente (S), medida em volt-amperes (VA), é a soma vetorial da Potência Real e Reativa.

  A relação entre essas potências é expressa por:

$$S^2 = P^2 + Q^2 \tag{2.1}$$

Além disso, temos o fator de potência (FP)2.2, definido como a razão entre a potência real e a potência aparente, é um parâmetro critico que indica a eficiência do uso da energia elétrica. Nos data centers, a correção do fator de potência é uma prática adotada para melhora a eficiência energética. Um fator de potência baixo resulta em aumento do fluxo de corrente para a mesma quantidade de potência real, levando a maiores perdas devido ao aquecimento resistivo dos condutores, isso reduz a eficiência da distribuição de energia. [21]

$$FP = \frac{P}{S} = cos(\theta) \tag{2.2}$$

onde theta o ângulo de defasagem entre a tensão e corrente. O FP varia entre 0 e 1, sendo:

- FP = 1: toda energia fornecida é convertida em trabalho útil
- FP <1, há desperdício de energia, exigindo maior corrente para fornecer a mesma potência real, o que causa perdas térmicas e sobrecarga da rede elétrica.

#### 2.2.2.1 Medição do consumo de energia

Considerando corrente alternada, o consumo de energia (E)2.3 em um sistema computacional pode ser calculado a partir da potência real (P)2.4 consumida ao longo do tempo. No Sistema internacional de Unidades (SI), a energia elétrica é medida em Joules (J), onde 1 joule equivale a 1 watt multiplicado por 1 segundo [22]:

$$E = P \times t \tag{2.3}$$

Enquanto a Potência real deve ser escrita levando em consideração o fator de potência, pois cargas com baixo FP podem desperdiçar energia, além de depender da tensão e da corrente [23]:

$$P = V \times I \times cos(\theta) \tag{2.4}$$

desta forma o consumo total de energia é determinado pela soma de potência ao longo do tempo:

$$E = \sum P_i \times \Delta t \tag{2.5}$$

## 2.3 **Tecnicas de Estatística**

### 2.3.1 Bootstrap

O bootstrap é uma técnica estatística que permite estimar a distribuição de uma amostra de dados, sem assumir que ela segue uma distribuição específica. A idéia principal

é gerar várias amostras aleatórias com reposição da amostra original, e então usar essas amostras para calcular estatísticas como a média e a variância.

O bootstrap foi desenvolvido pelo estatístico Bradley Efron em 1979 [24] e é amplamente utilizado em diversas áreas, como análise de dados, inferência estatística, modelagem e ciência da computação. Ele é especialmente útil quando a distribuição subjacente é desconhecida ou é difícil de se calcular analiticamente.

O Bootstrap funciona criando múltiplas amostras com reposição da amostra original, chamadas de amostras bootstrap. A partir dessas amostras, é possível estimar a distribuição de parâmetros de interesse, como a média, a mediana e a variância. Essa distribuição é chamada de distribuição bootstrap. Detalhado a seguir:

1. Inicialmente, uma lista de amostras $x = (x_1, x_2, ..., x_n)$ com $n$ medições são obtidas através do ambiente.

2. Então, $B$ *amostras bootstrap* $(x^{*1}, x^{*2}, ..., x^{*B})$ são geradas. Uma *amostra bootstrap* $x^* = (x_1^*, x_2^*, ..., x_n^*)$ é obtida por meio de uma amostragem aleatória feita $n$ vezes, com reposição dos dados originais contidos em $x$. A notação estrela indica que $x^*$ é a versão reamostrada de $x$. Por exemplo, assumindo $n = 4$, uma amostra bootstrap típica poderia ser $x^* = (x_4, x_2, x_2, x_3)$ no qual $x_1^* = x_4$, $x_2^* = x_3^* = x_2$ e $x_4^* = x_3$. É importante frisar que cada elemento x tem a mesma probabilidade de ser selecionada: $\frac{1}{n}$.

3. Para cada conjunto de dados bootstrap $x^{*b}$, $b = 1, 2, ...B$, existe uma *replicação bootstrap* $\hat{\theta}^*(b) = s(x^{*b})$ no qual $s$ é a estatística de interesse, mais especificamente, a média dos conjuntos de dados:

$$s(x^{*b}) = \overline{x}^{*b} = \frac{(\sum_{i=1}^{n} x_i^{*b})}{n} \tag{2.6}$$

4. Logo após, as replicações são ordenadas, de tal modo que $\hat{\theta}^*_{(1)}$ seja a menor replicação e $\hat{\theta}^*_{(B)}$ seja a maior. Considerando o grau de confiança de $1 - \alpha$, no qual $\alpha$ é o nível de significância, o intervalo de confiança é dado por $[\hat{\theta}^*_{(B\alpha/2)}, \hat{\theta}^*_{B[1-\alpha/2]}]$. Por exemplo, assumindo $B = 1000$ e um grau de confiança de 95% ou 0.095 ($\alpha = 0.05$), o intervalo de confiança é representado por $[\hat{\theta}^*_{(25)}, \hat{\theta}^*_{(975)}]$

5. Por fim, o intervalo médio ($mr$) é calculado, que representa o valor estimado final

$$mr = \frac{\hat{\theta}^*_{(B\alpha/2)} + \hat{\theta}^*_{B[1-\alpha/2]}}{2} \tag{2.7}$$

Uma das principais vantagens do Bootstrap é que ele não requer nenhuma suposição sobre a distribuição da população, tornando-se útil em casos onde a distribuição é desconhecida ou difícil de modelar. Além disso, ele permite estimar a incerteza associada aos parâmetros estimados, através da construção de intervalos de confiança.

### 2.3.2 T-Student

A técnica estatística paramétrica T-Student é utilizada para testar hipóteses sobre a média de uma população normalmente distribuída, quando a variância da população não é conhecida. Ela é baseada na distribuição t-student, que se aproxima da distribuição normal quando o tamanho da amostra é grande.

O teste T-student é composto de duas hipóteses: a hipótese nula e a hipótese alternativa. A hipótese nula é geralmente que a média populacional é igual a um valor especificado (por exemplo, média populacional é igual a zero) e a hipótese alternativa é geralmente que a média populacional é diferente desse valor (por exemplo, média populacional é diferente de zero).

Para realizar o teste $t-student$, é necessário calcular o valor $t$ e então comparar com o valor calculado com o valor crítico na tabela $t-student$ com o grau de liberdade $(n-1)$ e o nível de significância escolhido. Se o valor calculado é maior que o valor crítico, rejeita-se a hipótese nula e aceita-se a hipótese alternativa, ou seja, existe evidência suficiente para sugerir que a média populacional é diferente do valor especificado na hipótese nula. Caso contrário, não há evidência suficiente para rejeitar a hipótese nula. Detalhado a seguir:

1. Inicialmente um conjunto de amostras são obtidas a partir do ambiente de medição. a quantidade de amostras iniciais é denotado por $B$ e cada amostra $x = (x_1, x_2, ..., x_n)$ contem n medições.

2. Para cada amostra $x^b$, $b = 1, 2, ...B$, a média ($\overline{x}^b$) é calculada:

$$\overline{x}^b = \frac{(\sum_{i=1}^{n} x_i^b)}{n} \tag{2.8}$$

3. Após, a média de todas as amostras ($\overline{\overline{x}}$) é obtida, junto de seu desvio padrão ($s$) e do erro associado ($se$):

$$\overline{\overline{x}} = \frac{(\sum_{b=1}^{B} \overline{x}^b)}{B} \tag{2.9}$$

$$s = \sqrt{\frac{\sum_{b=1}^{B} (\overline{x}^b - \overline{\overline{x}})^2}{B-1}} \tag{2.10}$$

$$se = t_{1-\alpha/2,n-1} \times \frac{s}{\sqrt{B}} \tag{2.11}$$

4. A precisão relativa ($rp$) é então calculada. Se a precisão relativa for igual ou menor que a precisão especificada pelo designer ($dp$), o processo de medida para. Se não, um número adicional de amostras é obtida ($B'$), e os passos 2 a 4 são repetidos. É importante afirmar que, no passo 3, as amostras previas também são consideradas no cálculo. equações apresentadas a seguir:

$$rp = \frac{se}{\overline{\overline{x}}} \tag{2.12}$$

$$B' = (\frac{t_{1-\alpha/2,n-1} \times s}{dp \times \overline{\overline{x}}})^2 \tag{2.13}$$

É importante notar que o teste $t-student$ pressupõe que a amostra é normalmente distribuída. Se a amostra não for normalmente distribuída, é recomendado usar testes não paramétricos, como o teste Wilcoxon [25].

# 3 AMBIENTE DE MEDIÇÃO

## 3.1 Visão Geral

Construímos um Servidor de Medição [26], que pode coletar informações sobre o consumo de energia de um sistema computacional e seu tempo de execução. O servidor adota a plataforma Arduino e o circuito sensor ACS712-20A, que é um circuito sensor linear baseado no efeito Hall e possui um isolamento de tensão de 2,1 kV RMS e um condutor de corrente de baixa resistência [27], sendo assim capaz de medir corrente contínua ou alternada para capturar os valores da corrente. Ele possui cancelamento de ruído com tempo de resposta muito alto.

O sensor ZMPT101B [28] foi utilizado para medir os valores de tensão, suportando até 250 Volts (V). Permite uma leitura precisa de até ±1% e também pode ser ajustado e calibrado diretamente com o Arduino, possuindo um potenciômetro soldado na placa para ajustar a tensão de saída exata. A partir dessas medições, a potência real pode ser derivada e, combinando esses dados com o tempo de execução, é possível determinar o uso de energia [29]. Alternativamente, o sistema pode ser ligado ao dispositivo MINIPA [30], permitindo assim obter valores de corrente e tensão de um medidor comercial.

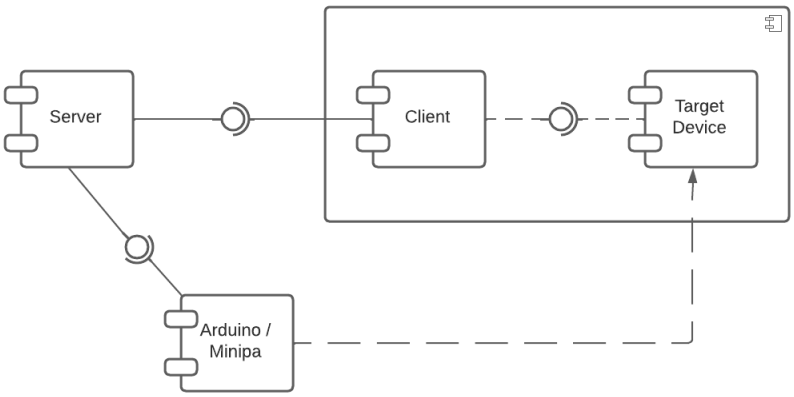


Figura 6: Estrutura de Medição

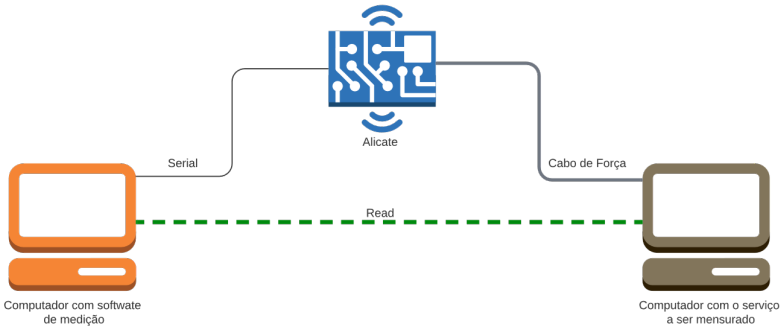


Figura 7: Arquitetura de Medição

A arquitetura do sistema (Figura 6) compreende um Servidor de Medição que media todas as comunicações entre dispositivos sensores e o cliente. O servidor se comunica com o Arduino usando comunicação serial e com o cliente usando um socket que interage com o banco de dados NoSQL alvo para executar comandos de leitura e escrita (Figura 7).

Além de coletar dados, o Servidor de Medição realiza cálculos estatísticos paramétricos e não-paramétricos para analisar o resultado das medições. A análise paramétrica infere a média e o desvio padrão sob uma determinada distribuição de dados, enquanto a análise não-paramétrica ajuda a reconhecer padrões e tendências sem uma suposição sobre seguir uma determinada distribuição estatística e estimando valores extremos ou médios em relação ao tempo de execução e ao consumo de energia.

## 3.2 Descrição dos Principais Módulos e Implementação

### 3.2.1 Server implementation

A principal classe do Servidor de Medição está representada em Figura 8 e chamada *ServerImplementation*. Foi desenvolvida para gerir as interações entre o cliente e o aparelho de medição, recuperar e processar os dados e utilizar métodos estatísticos para entregar resultados precisos. O servidor deve ser configurado antes de começar a coletar dados; ele é invocado na linha de comando com parâmetros que determinam como funcionará. Parâmetros de configuração: Informações do dispositivo de medição, (n) (tamanho das amostras), (b) (tamanho do lote), $\alpha$ (nível de significância) e método de avaliação. Os valores são mantidos em variáveis internas para uso futuro.

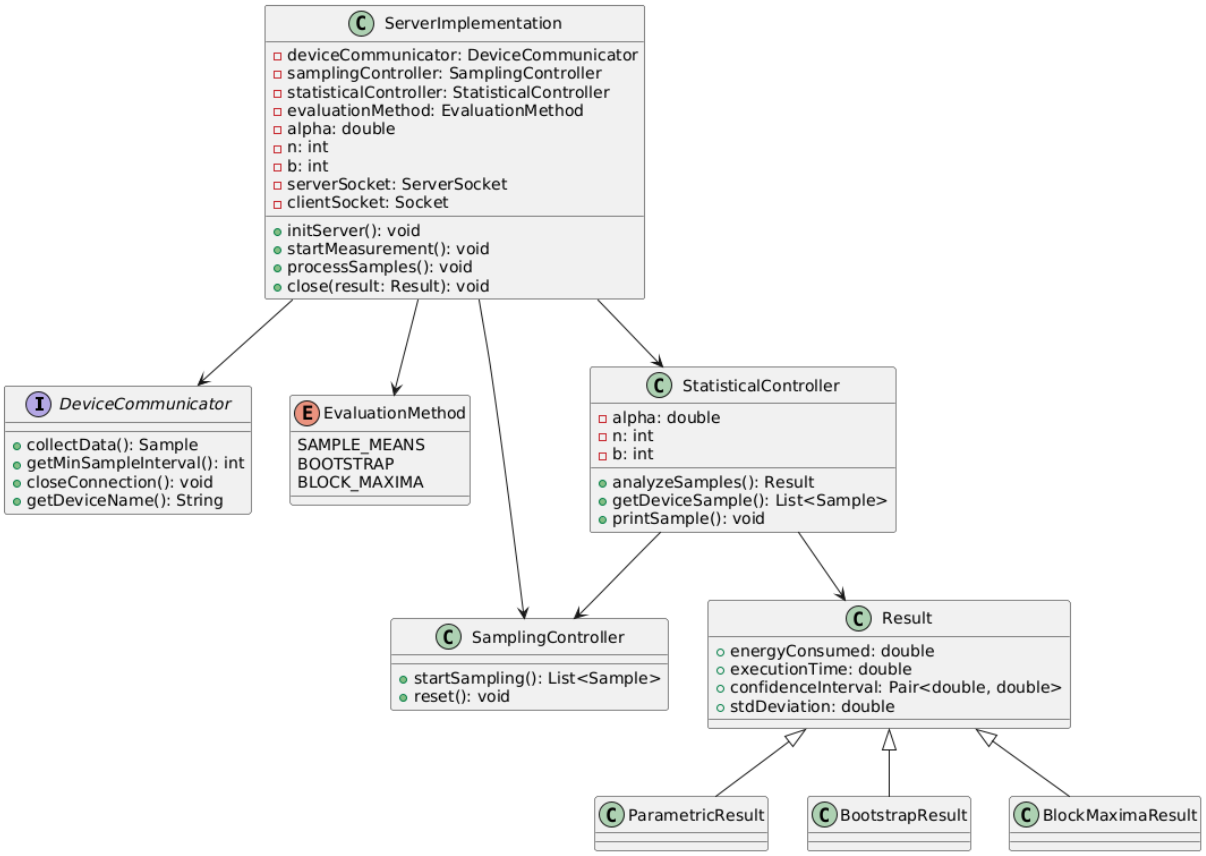


Figura 8: Diagrama UML do Server Implementation

Um dos principais parâmetros é *deviceCommunicator*, responsável por mediar a comunicação entre o servidor e o dispositivo de medição. O objeto é derivado da classe *DeviceCommunicatorFactory*, que retorna o comunicador de dispositivo adequado conforme o tipo de dispositivo solicitado. As estruturas são altamente escaláveis e não requerem alterações estruturais para adicionar novos tipos de dispositivos. Além disso, (n) e (b): quanto de dados coletar antes de aplicar técnicas estatísticas.

A técnica estatística utilizada para os dados coletados é definida pelo método de avaliação *evaluationMethod*. O servidor oferece vários métodos, tanto paramétricos quanto não paramétricos. Outros parâmetros também podem ser ajustados para tornar os resultados mais precisos, como nível de significância $\alpha$ , margem de erro *erro*, probabilidade para valores extremos, *probability*, configuração de verbosidade *verbose*.

No método *initServer*, observamos o processo de inicialização, que configura um servidor socket *ServerSocket* e emite um comando para escutar conexões de clientes entrantes na *serverPort* especificada. Após a conexão do cliente, é criado um canal de comunicação bidirecional, que possibilita fluxo contínuo de dados. Para isso, fluxos de

entrada e saída de dados são criados trabalhando com as classes *BufferedReader* e *BufferedWriter*, veja que, para enviar mensagens, entre cliente e servidor. Com este mecanismo, o Servidor de Medição consegue receber comandos de controle, enviar medições e lidar com solicitações ao mesmo tempo.

Assim que os dados são coletados, o servidor implementa uma técnica estatística conforme determinado pelo usuário. Os resultados fornecidos em um objeto *Result* incluem informações como:

- Energia consumida em Joule
- Tempo de execução em segundos
- Limites inferiores e superiores
- Desvio padrão da energia e do tempo (quando aplicável)
- Valores extremos, no caso de uma análise baseada em máxima probabilidade

Após o processamento ser concluído, podemos passar o resultado para o cliente e a conexão pode ser encerrada com segurança. Os detalhes de gerenciamento do encerramento da comunicação são geralmente realizados através do método *close(Result result)* que garante:

- Os resultados sejam impressos no console.
- Os dados sejam transmitidos ao cliente via socket.
- O dispositivo de medição seja desligado corretamente.
- O servidor aguarde 2 segundos para garantir que o cliente recebeu todas as informações antes da desconexão.

### 3.2.2 DeviceCommunicator

O Ambiente de medição se comunica com os dispositivos de medição utilizando conexões seriais utilizando a biblioteca *jSerialComm*, garantindo a precisão e a confiabilidade na coleta de dados elétricos coletados pelos sensores.

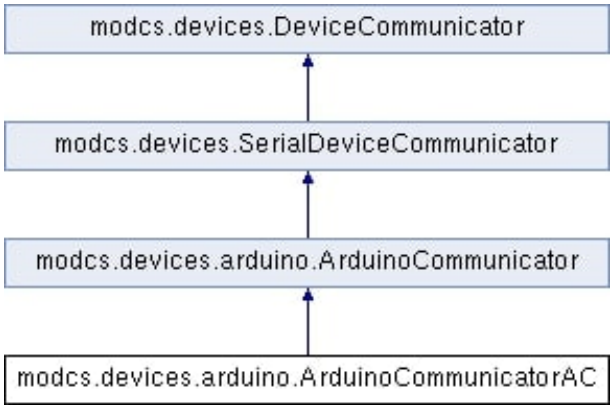


Figura 9: Hierarquia do Device Communicator

Arquiteturalmente, o sistema para a comunicação foi projetado utilizando interfaces e classes especializadas, para permitir a flexibilidade e a modularidade, desta forma, é possível adequar o sistema de medição a diferentes dispositivos, sem interferir na lógica principal do servidor. A comunicação entre o servidor e os dispositivos segue uma estrutura hierárquica conforme mostrado na Figura 9, onde uma interface genérica *DeviceCommunicator* define os métodos essenciais que incluem:

- Um método para coletar e estruturar os dados do dispositivo.
- Um método que retorna o tempo mínimo entre amostras.
- Um método para encerrar a conexão com o dispositivo.
- Um método que retorna o nome do dispositivo conectado.

A partir do *DeviceCommunicator*, foi implementado o *SerialDeviceCommunicator*, uma classe base que implementa procedimentos comuns, necessários para comunicar o ambiente de medição aos dispositivos que utilizam a porta serial, onde ela identifica automaticamente as portas disponíveis no sistema e configura a comunicação conforme as propriedades de cada dispositivo.

#### 3.2.2.1 Comunicação com Dispositivos Baseados em Arduino

Para dispositivos de medição baseados em Arduino, o protocolo de comunicação serial é simples, onde o servidor envia comandos para o Arduino, sincronizando a medição, solicitando os dados e recebendo as medições em resposta. O *ArduinoCommunicator* é a

classe implementada responsável por gerenciar essas comunicações, garantindo uma conexão corretamente estabelecida, definindo as configurações adequadas de taxa de transmissão, numero de bits e paridade e garante que os dados sejam recebidos sem erros. A partir dela foram instanciados classes que implementam um protocolo especializado para receber medições detalhadas, por exemplo, o *ArduinoCommunicatorAC* que recebe valores como:

- Potência real
- Fator de potência
- Potência aparente

#### 3.2.2.2 Comunicação com Dispositivos Comerciais

Como o ambiente de medição utilizou abordagens modulares, é possível a implementação de protocolos para novos dispositivos de medição. O ambiente de medição tem suporte a se comunicar ao Minipa ET-4091, um medidor de potência comercial amplamente utilizado em aplicações elétricas.

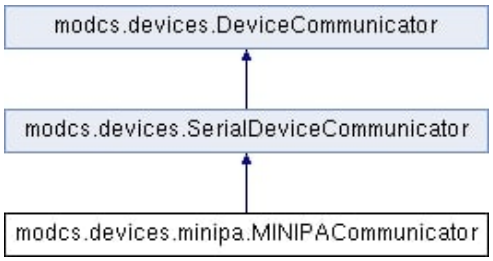


Figura 10: Hierarquia do Device Communicator

Para a comunicação com esse dispositivo, foi implementado uma nova classe, a *MinipaCommunicator*, que analogamente ao ArduinoCommunicator também tem relação de herança com *SerialDeviceCommunicator*, mostrado na Figura 11, mas tem um comportamento utilizado para interpretar corretamente os dados enviados pelo medidor. O Minipa ET-4091, diferente do Arduino, exige um tempo mínimo entre as coletas, pois não tem leituras instantâneas, esse comportamento é gerenciado pela classe de modo a evitar falhas na comunicação e garantir a precisão dos dados.

### 3.2.3 Statistical Controler

O *StatisticalController* é o componente estatístico principal do ambiente de medição, ela utiliza métodos de análise sobre os valores de consumo de energia e tempo de execução para garantir que as medições sejam interpretadas corretamente. Para isso, a classe é alimentada com amostras do *SamplingController*, processando as informações e retornando as estatísticas relevantes como média, intervalos de confiança e estimativas baseadas em distribuições probabilísticas. No momento da inicialização, *StatisticalController* recebe um conjunto de parâmetros que definem o comportamento das análises:

- alpha, o nível de confiança que define a precisão das estimativas geradas.
- n, especifica a quantidade de amostras a serem coletadas e analisadas.
- b, controla o tamanho dos blocos utilizados em técnicas que segmentam os dados antes do processamento.

Além disso, no mapeamento de configuração da classe, devemos incluir o *SamplingController* como a fonte de dados a ser analisada. Configurado, o *StatisticalController* serve como uma ponte entre os dados brutos e quaisquer métodos estatísticos disponíveis. A classe solicita acesso às amostras do *SamplingController* quando estas são recebidas para processamento, confirmando que os dados estão disponíveis para análise. Isso é feito através do método *getDeviceSample()*, que pega uma série temporal de medições para produzir uma representação comum dos valores de potência e tempo.

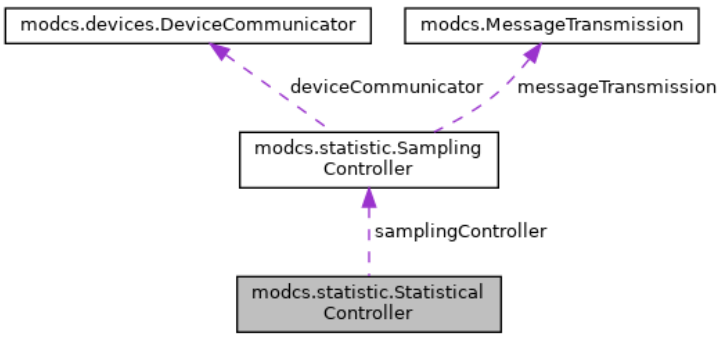


Figura 11: Hierarquia do Statistical Controller

Uma vez que as amostras foram extraídas, cálculos estatísticos podem ser realizados sobre elas, dependendo de qual método foi escolhido. Na implementação do *StatisticalController*, fornecemos essas abordagens da seguinte forma:

- Sample Means: Obtém a média das amostras aleatórias tiradas em um intervalo, juntamente com o intervalo de confiança usado para estar amostras observadas.
- Batch Means Technique: Divide as medições em blocos de tamanho fixo e depois realiza os cálculos estatísticos individualmente para cada bloco, mitigando assim o impacto das variações aleatórias.
- Valores extremos: ajusta distribuições estatísticas para encontrar padrões de pico nas mediações.
- bootstrap Technique: Permite a reamostragem para obter estimativas estatísticas sem precisar assumir uma distrivuição específica para os dados.

Independente do método utilizado, os resultados das análises são armazenados em objetos especializados, garantindo que as informações processadas possam ser recuperadas posteriormente. Os objetos para esses fins são:

- ParametricResult: armazena estatísticas como a média e o desvio padrão das mediações.
- BootstrapResult: mantém as estimativas das técnicas de reamostragem.
- BlockMaximaResult: Onde são armazenados os resultados relacionados aos extremos detectados.

Além de processar as medições, o *StatisticalController* contém funcionalidades auxiliares para facilitar a manipulação dos dados coletados. Um método simples *printSample()* é implementado para visualizar amostras antes de executar a análise, permitindo verificar se os valores obtidos são consistentes. Outro método pertinente aqui é o *convertListToDouble()*, que converte listas de valores em um formato apropriado para cálculos estatísticos. São essas funções complementares que tornam a classe mais adaptável, pois permitem que os dados sejam geridos e estruturados.

#### 3.2.3.1 Fluxos de Execução do StatisticalController

O fluxo de execução no *StatisticalController* é bem definido, no início, o Servidor de Medição faz uma solicitação de análise estatística, e a classe tem um *SamplingController* para solicitar amostras. Os dados são então analisados com o método estatístico

apropriado e os resultados são formatados e armazenados. Por fim, prepara-se os dados calculados para o Servidor de Medição, para poderem ser usados em análises futuras ou armazenados para uso posterior. Eles são então analisados separadamente, para que a coleta de dados e a análise estatística não interfiram entre si, proporcionando uma abordagem mais limpa a um sistema flexível.

Uma das forças da implementação do *StatisticalController* é que ele desacopla a coleta do processamento. Separar estatísticas da lógica também facilita adicionar mais métodos estatísticos, já que nenhuma outra parte do código precisa ser atualizada, além de modificar o próprio Servidor de Medição.

Além disso, escrevendo os resultados como objetos reutilizáveis, a classe mantém um histórico de todas as análises conduzidas, o que nos dá mais controle sobre os dados resultantes, os módulos são ajustados para fornecer variadas abordagens estatísticas, podendo atender a todas as necessidades estatísticas.

## 3.3 **Fluxo de Comunicação e Processamento de Dados**

O Fluxo de comunicação é incremental e envolve comunicação de ida e volta entre servidor e cliente para reunir e analisar o consumo de energia e estatísticas de tempo e execução. A comunicação é dividida em três fases principais:

1. O servidor sinaliza ao cliente que está pronto para receber os dados.
2. Os comandos para iniciar ou parar a coleta são emitidos pelo cliente.
3. O servidor recebe os valores de energia e tempo e os envia para o cliente, até que todas as amostras necessárias tenham sido coletadas.
4. O servidor encerra a comunicação quando todas as amostras foram processadas e envia os resultados para o cliente.

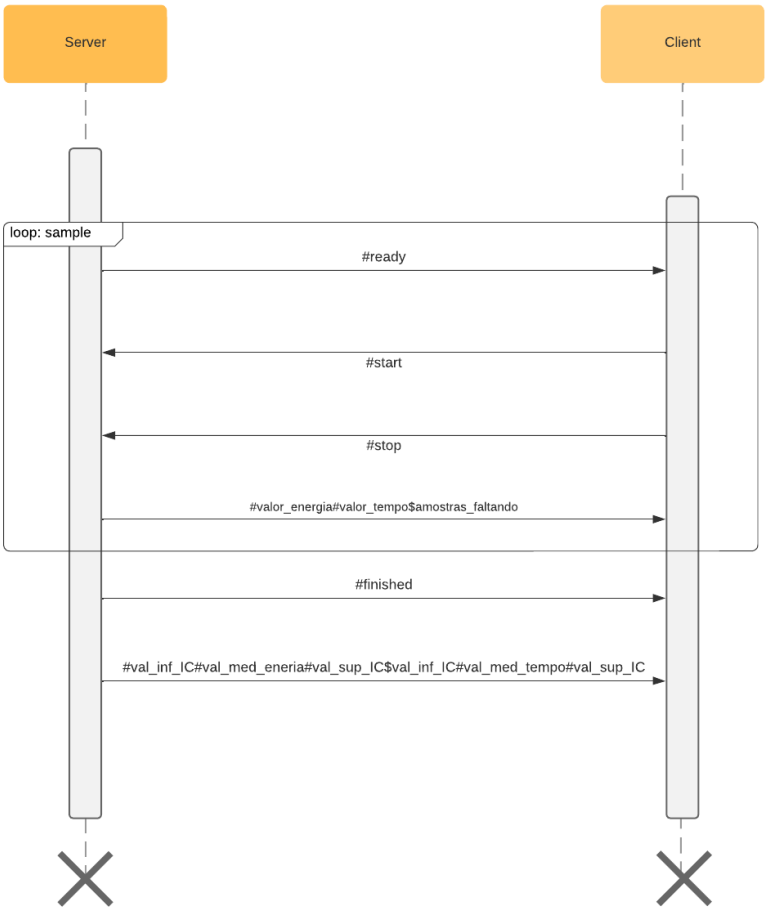


Figura 12: Diagrama de sequência

Conforme mostrado na Figura 12, o servidor de medição sincroniza o início e o fim de uma amostra (ou seja, replicação) com o cliente, enviando um comando de *#ready*, que indica que o servidor está pronto para fazer as medições. Em seguida, ao receber o comando de *#start* vindo do cliente, inicia a coleta, medindo o respectivo tempo de execução e obtém o consumo de energia com base nos dados da plataforma Arduino, que mede o consumo de energia das funções que o cliente envia para a máquina alvo.

Após o servidor receber o comando de *#stop*, finaliza a coleta de uma amostra e retorna o valor de energia mensurado, o tempo necessário para a coleta da amostra e quantas amostras ainda são necessárias. Caso tenha coletado amostras suficientes, envia o comando *#finished* para o cliente e em seguida fornece técnicas estatísticas baseadas em métodos paramétricos, técnicas não paramétricas e teoria de valores extremos para estimar valores extremos ou médios em relação ao tempo de execução e consumo de energia.

```
Evaluation technique: bootstrap
Sample size: 10
Significance level: 0.05
```

```
Batch or bootstrap sample size: 40
Device: arduino
------------
Server initiated using port 12345
Client connected: 10.0.0.163
Obtaining sample 1 - Energy(J): 19.09967  Time(s):  3.954999999999999
Obtaining sample 2 - Energy(J): 20.34978  Time(s):  4.189999999999997
Obtaining sample 3 - Energy(J): 20.07941999999992 Time(s):  4.0449999999999997
Obtaining sample 4 - Energy(J): 20.09633999999998 Time(s):  4.0639999999999997
Obtaining sample 5 - Energy(J): 20.38299  Time(s):  4.165999999999998
Obtaining sample 6 - Energy(J): 20.42420000000006 Time(s):  4.0269999999999975
Obtaining sample 7 - Energy(J): 19.91023999999998 Time(s):  4.059999999999997
Obtaining sample 8 - Energy(J): 20.06813999999996 Time(s):  4.099999999999998
Obtaining sample 9 - Energy(J): 19.9346     Time(s):  4.0929999999999997
Obtaining sample 10 - Energy(J):  20.57604  Time(s):  4.198999999999999
------------
Result energy (J):  20.067856499999998  [19.800689,20.335024]
Result time (s):  4.089049999999998 [4.039699999999998,4.138399999999997]
Server Closed
```

Código 3.1: Log de execução da carga de trabalho com obtenção de amostras no Cliente.

```
import socket
import time
import subprocess

HOST = "127.0.0.1"  # The server's hostname or IP address
PORT = 12345  # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    x = s.makefile("rb")
    delayStartStop = 10
    resp = x.readline()
    #print(s.recv(1024).decode('ascii'))
    print(resp.decode('ascii'))
    count = 0
    while True:
        s.sendall("#start\n".encode('ascii'))

        comand = "/YCSB/bin/ycsb.bat run arangodb -s -P workloads/
    workloada -p arangodb.ip=10.0.0.149 -p arangodb.port=8529 -p
    operationcount=1000 -p arangodb.protocol=HTTP_JSON >
    arangodbUpdate_v01_1000_"

        count = count + 1
        countPlus = str(count) + ".txt"
```

```
23          comand = comand + countPlus
24          process = subprocess.Popen(comand, shell=True, stdout=subprocess
    .PIPE)
25          process.wait()
26          print("Process return code " + str(process.returncode))
27          s.sendall("#stop\n".encode('ascii'))
28          #resp = s.recv(1024)
29          resp = x.readline()
30          print(resp.decode('ascii'), end ="")
31          resp = x.readline()
32          print(resp.decode('ascii'))
33
34          if resp.decode('ascii').find("#finished") > -1:
35              break
36      resp = x.readline()
37      print(resp.decode('ascii'))
38      s.close()
```

Código 3.2: Exemplo - Script de execução da *Workload*.

# 4 RESULTADOS EXPERIMENTAIS

## 4.1 Validação

A plataforma Arduino proposta foi validada utilizando o alicate amperímetro MINIPA ET-4091 [30], que também é compatível com o servidor de medição. A validação adotou um computador com CPU Core 2 Duo T5450 1.66GHz, 8GB de RAM, rodando Ubuntu 20.04.4 LTS (Linux) com EXT4 como arquivo de sistema. Todos os serviços do sistema operacional (SO) foram reduzidos ao mínimo para não afetar a coleta de dados, considerando 7 experimentos, nos quais o consumo de energia foi medido no ET-4091 e na plataforma Arduino: (i) uma reprodução de vídeo; (ii) a execução de 3 cargas de trabalho do sysbench (CPU, Arquivo, Memória); (iii) e uma carga de trabalho YCBS em MongoDB e ArangoDB.

Para cada dispositivo de medição, 100 amostras foram coletadas e bootstrap foi adotado para estimar os valores médios,Tabela 1 retratando-os. Realizamos um teste T pareado e a Tabela 2 fornece os resultados. Como 0 está contido no intervalo de confiança de 95%, não há evidência estatística para indicar que ambas as plataformas fornecem resultados diferentes. [31]

Tabela 1: Comparação entre ET-4091 e Arduino - Consumo de energia

| | ET-4091 ($J$) | Arduino ($J$) |
|---|---|---|
| Video | 1821.70 | 1822.85 |
| CPU Sysbench | 516.78 | 509.95 |
| Hibernando | 287.98 | 309.15 |
| Arquivos Sysbench | 398.49 | 402.65 |
| Memoria Sysbench | 495.70 | 483.58 |
| MongoDb | 1125.57 | 1015.73 |
| ArangoDb | 574.66 | 520.24 |

Tabela 2: "Emparelhamento T-test"

| 95% CI | p-valor |
|---|---|
| [-19.3;64.1] | 0.237 |

### 4.1.1 Configuração YCSB

Para conduzir este *benchmark*, é necessário um gerador de carga de trabalho que atenda a dois critérios principais: o primeiro é a definição do conjunto de dados que será inserido no banco de dados, e o segundo é a execução de operações sobre esse conjunto, permitindo a avaliação do desempenho, ou seja, o tempo de execução.

Nos experimentos, foram utilizados dez campos por registro, conforme padrão do YCSB, com cada campo possuindo um tamanho de *100 bytes*. Durante as leituras, todos os campos foram acessados, e a proporção das operações foi estabelecida conforme os comandos do *benchmark*. Por exemplo, para as leituras, 100% das operações realizadas foram de leitura; as escritas e atualizações seguiram uma proporção semelhante, com o número de operações variando conforme o cenário testado (1.000, 5.000 ou 10.000 operações).

A cada execução do *benchmark*, uma carga inicial foi realizada para preparar o ambiente. O parâmetro *-target* não foi especificado, o que permitiu ao YCSB operar sem limitar o número de operações por segundo, utilizando a máxima capacidade de vazão do ambiente.

Este estudo fez uso da distribuição *ipfian*, baseada no algoritmo *zipfian* customizado pelo Yahoo!, o qual assegura que os itens populares estejam distribuídos por todo o *keyspace* do banco de dados, conforme apresentado em [32].

```
1  # Copyright (c) 2010 Yahoo! Inc. All rights reserved.
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License"); you
4  # may not use this file except in compliance with the License. You
5  # may obtain a copy of the License at
6  #
7  # http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
12 # implied. See the License for the specific language governing
13 # permissions and limitations under the License. See accompanying
14 # LICENSE file.
15
```

```
16
17 # Yahoo! Cloud System Benchmark
18 # Workload A: Update heavy workload
19 #   Application example: Session store recording recent actions
20 #
21 #   Read/update ratio: 50/50
22 #   Default data size: 1 KB records (10 fields, 100 bytes each, plus key
   )
23 #   Request distribution: zipfian
24
25 recordcount=1000
26 operationcount=1000
27 workload=site.ycsb.workloads.CoreWorkload
28
29 readallfields=true
30
31 readproportion=0
32 updateproportion=0
33 scanproportion=0
34 insertproportion=0
35 writeproportion=0
36 readmodifywriteproportion=0
37
38 requestdistribution=zipfian
```

Código 4.1: Exemplo - Script Workload.

Dessa forma, é praticamente garantido que muitos registros diferentes sejam atualizados, embora haja uma boa chance de que certos itens sejam atualizados múltiplas vezes. Para cada execução, foi realizada a preparação do ambiente através da carga, garantindo, por exemplo, que ao executar uma leitura, os dados já estivessem presentes no banco de dados.

### 4.1.2 Metodologia

Este trabalho adota uma metodologia de avaliação baseada em planejamento de experimentos (DoE) [33], no qual um projeto fatorial $l^k$ com $r$ replicações é adotado. Dois fatores ($k = 2$) são considerados: (i) $DBMS$ – ArangoDB-DOC, ArangoDB-KEY,

OrientDB-DOC, OrientDB-KEY, MongoDB, redis; e (ii) *Command* (*Comm*) - inserir, ler, atualizar. Esses DBMSs foram escolhidos, pois são os mais populares em relação a NoSQL. Consequentemente, documento (DOC) e chave-valor (KEY) são os modelos de dados comuns, pois são adotados pelo MongoDB e Redis, respectivamente. Esses modelos de dados também foram considerados para ArangoDB e OrientDB, pois são DBMSs multimodelo e frequentemente utilizados para persistência poliglota. ArangoDB-DOC indica a utilização do modelo de documento para ArangoDB, e ArangoDB-KEY significa a utilização do modelo chave-valor. A mesma notação é utilizada para OrientDB. [31]

Os experimentos levam em conta 3 cargas de trabalho diferentes (1.000 operações, 5.000 operações, 10.000 operações) geradas pelo *benchmark YCSB*. As métricas de interesse são consumo de energia ($J$) e tempo de execução ($s$). O YCSB é um conjunto de benchmark de código aberto para avaliar aplicativos de computador, e este software é comumente adotado para avaliar o desempenho do NoSQL DBMS [32]. Além disso, 100 replicações ($r = 100$) são consideradas para obter valores médios (com distribuição normal aproximada) e reduzir o impacto de ruídos de medição. Neste trabalho, os resultados são analisados usando ANOVA [33].

## 4.2 **Resultados Experimentais**

A Tabela 3 e a Tabela 4 apresentam os resultados da análise ANOVA (nível de significância $\alpha = 0{,}05$) para tempo de execução e consumo de energia. As análises de tempo e energia mostradas nos gráficos consideram os valores médios e os respectivos intervalos de confiança de 95% (no topo de cada barra). [31]

Tabela 3: ANOVA - Tempo de execução

| Trabalho 1.000 | | | | Trabalho 5.000 | | | | Trabalho 10.000 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Fonte | Var. (%) | Estat. F | valor-p | Fonte | Var. (%) | Estat. F | valor-p | Fonte | Var. (%) | Estat. F | valor-p |
| DBMS | 84.78 | 11093.85 | <0.001 | DBMS | 93.63 | 57432.097 | <0.001 | DBMS | 72.23 | 76308.63 | <0.001 |
| Comm | 5.41 | 1770.82 | <0.001 | Comm | 3.76 | 5770.39 | <0.001 | Comm | 11.25 | 29722.74 | <0.001 |
| DBMS*Comm | 7.08 | 463.27 | <0.001 | DBMS*Comm | 2.03 | 622.25 | <0.001 | DBMS*Comm | 16.18 | 8544.13 | <0.001 |
| Erro | 2.72 | | | Erro | 0.58 | | | Erro | 0.34 | | |

A Tabela 3 mostra que todos os fatores e suas interações (*source*) impactam significativamente o tempo de execução (por exemplo, $p - value < 0,001$). Dependendo da carga de trabalho (work.), o fator pode ter uma influência distinta na métrica (var.%), pois alguns DBMSs são influenciados pela variação de seu desempenho para a quantidade de

Tabela 4: ANOVA - Consumo de energia

| Trabalho 1.000 | | | | Trabalho 5.000 | | | | Trabalho 10.000 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Fonte | Var. (%) | Estat. F | valor-p | Fonte | Var. (%) | Estat. F | valor-p | Fonte | Var. (%) | Estat. F | valor-p |
| DBMS | 94.16 | 43393.38 | <0.001 | DBMS | 78.24 | 28247.77 | <0.001 | DBMS | 69.36 | 28863.97 | <0.001 |
| Comm | 2.84 | 3275.64 | <0.001 | Comm | 14.38 | 12980.28 | <0.001 | Comm | 17.55 | 18254.09 | <0.001 |
| DBMS*Comm | 2.22 | 511.66 | <0.001 | DBMS*Comm | 6.39 | 1154.27 | <0.001 | DBMS*Comm | 12.23 | 2545.24 | <0.001 |
| Erro | 0.77 | | | Erro | 0.99 | | | Erro | 0.86 | | |

operações realizadas. *DBMS* tem o maior impacto nos resultados, e os ruídos de medição (*Error*) são muito pequenos, mas o comando (DBMS*Comm) tem baixa influência nos resultados. Em 10.000 de carga de trabalho, a interação influencia as mudanças devido ao comportamento do OrienteDB e resultados semelhantes são obtidos para o consumo de energia, no entanto, para uma carga de trabalho de 10.000, há uma pequena diferença devido a um aumento repentino no consumo de energia pelo OrienteDB.

A seguir, a explicação é baseada na avaliação obtida com o procedimento de Tukey [33] (um teste pós-ANOVA). Em geral, o modelo de dados não influencia significativamente o desempenho do ArangoDB e do OrientDB. Somente quando a diferença é representativa, a explicação declara explicitamente o impacto para evitar uma apresentação incômoda.

A Figura 13 mostra os tempos médios de execução para 1.000 operações. Para todos os comandos, a diferença entre todos os DBMSs é estatisticamente significativa, e o modelo de dados (DOC e KEY) não influencia o tempo de execução no ArangoDB. Para o OrientDB, a diferença entre os valores não é significativa e, portanto, não há estatísticas significativas para igualdade de leitura entre os tempos de execução. [31]

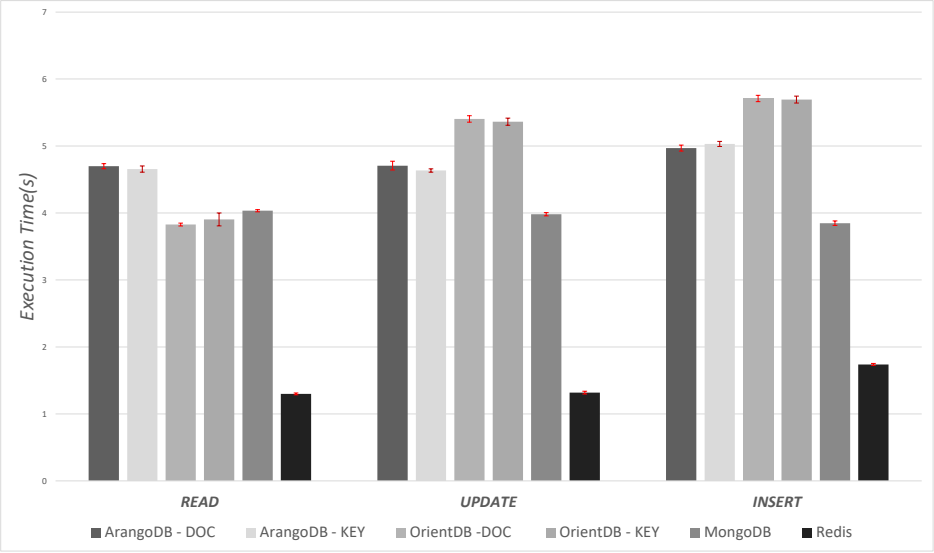


Figura 13: Média de tempo executando 1000 operações

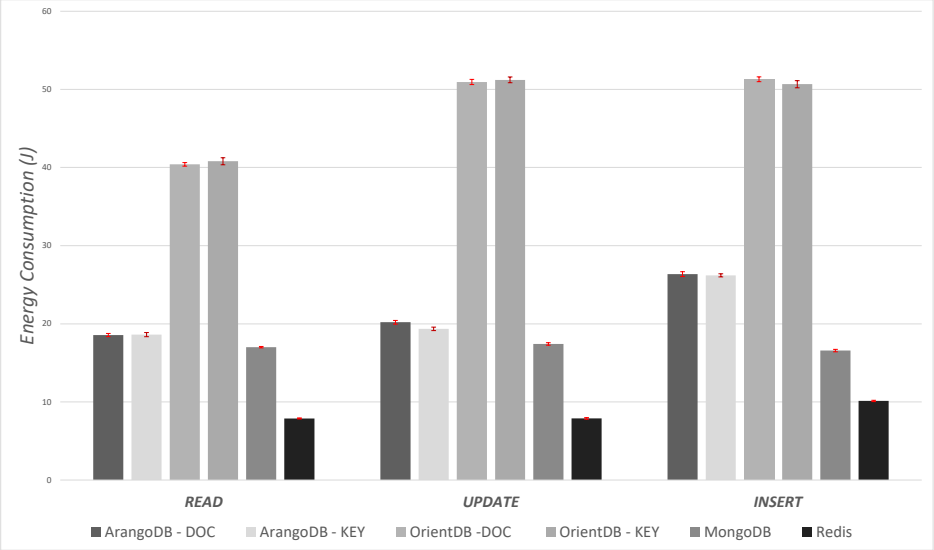


Figura 14: Média de consumo de Energia considerando 1000 operações

O Redis é o mais rápido, e seu tempo de execução é quase três vezes menor do que as outras contrapartes para operação de leitura, respectivamente, que fornecem o tempo de execução mais longo. Para esta carga de trabalho, o OrientDB é melhor do que o ArangoDB e o MongoDB apenas para leitura de dados. Em relação à atualização e inserção, o MongoDB está atrás apenas do Redis. que é 24,12% mais longo que o ArangoDB e 28,19% mais longo que o OrientDB para este comando. O modelo de documento também não influencia a operação no OrientDB e no ArangoDB. [31]

Para o comando *insert*, os resultados são semelhantes aos de update, no contexto do consumo de energia (Figura 14), o OrientDB tem o maior consumo para todos os comandos, mesmo para operações de leitura e o tempo de execução se correlaciona com o consumo em todos os comandos. O Redis ainda fornece o menor consumo, e pode ser sete vezes menor que o OrientDB. Por exemplo, respectivamente 51,40% e 83,10% mais lento que o ArangoDB e o Redis (que forneceram o menor tempo de execução). E o MongoDB também oferece o melhor valor para esta carga de trabalho em comparação com o OrientDB. [31]

Considerando 5.000 operações (Figura 15) o ArangoDB parece ser menos escalável para todos os comandos, e o modelo de dados não influencia estatisticamente o resultado para o ArangoDB. Por exemplo, em relação à leitura, o ArangoDB foi 500% mais lento que o Redis, que continua a superar todos os DBMSs. e 335% mais lento que o OrientDB. Portanto, o MongoDB é 50,11% mais rápido que o OrinteDB para o comando de leitura.

Comparando com 1.000 cargas de trabalho, o tempo de execução do ArangoDB aumentou em 283,51%, mas o Orient, o Mongo e o Redis aumentaram 3,67% e 37,26%, 51,68% respectivamente. O Redis continua a fornecer bom desempenho, sendo o DBMS mais rápido. [31]

O MongoDB foi o segundo mais lento para o comando de leitura, enquanto o OrientDB demonstrou boa escalabilidade para cargas de trabalho crescentes. Para operações de inserção, não há diferença estatisticamente significativa entre OrientDB e MongoDB, e este último foi apenas 3,88% mais rápido para atualização.

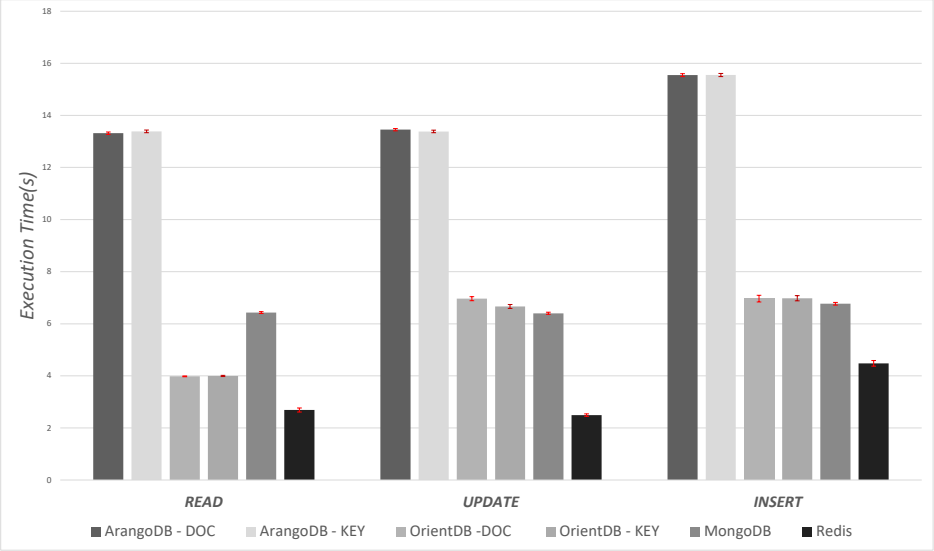


Figura 15: Média de tempo executando 5000 operações

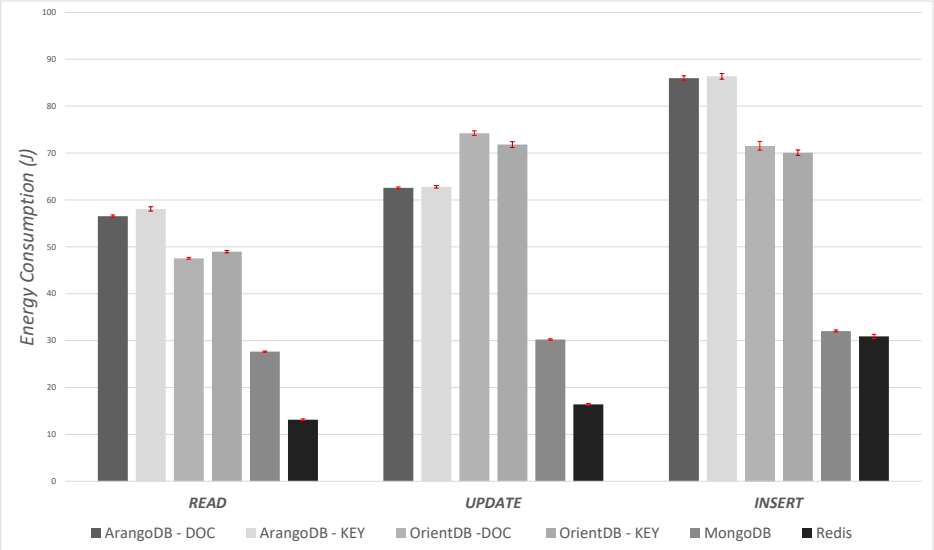


Figura 16: Média de consumo de Energia considerando 5000 operações

Comparando a escalabilidade com uma carga de 1.000, o OrientDB foi 54,24% mais rápido que o ArangoDB e 13,34% mais rápido que o MongoDB, indicando um melhor desempenho sob aumento de carga.

No que diz respeito à operação de insert, a diferença entre os tempos de execução de OrientDB e MongoDB não foi estatisticamente significativa, não havendo evidências suficientes para rejeitar a igualdade entre eles. Por outro lado, o Redis apresentou um desempenho consideravelmente superior ao OrientDB e ArangoDB para essa operação.

A Figura 16 descreve os valores de consumo de energia para tal carga de trabalho. Os valores indicam que a variação da carga de trabalho impacta o consumo de energia de forma diferente para cada DBMS. Embora o OrientDB tenha um tempo de inserção similar ao do MongoDB, o consumo de energia é 132,6% maior. Além disso, o OrientDB é mais rápido que o ArangoDB para atualização, mas consome 14% mais energia. [31]

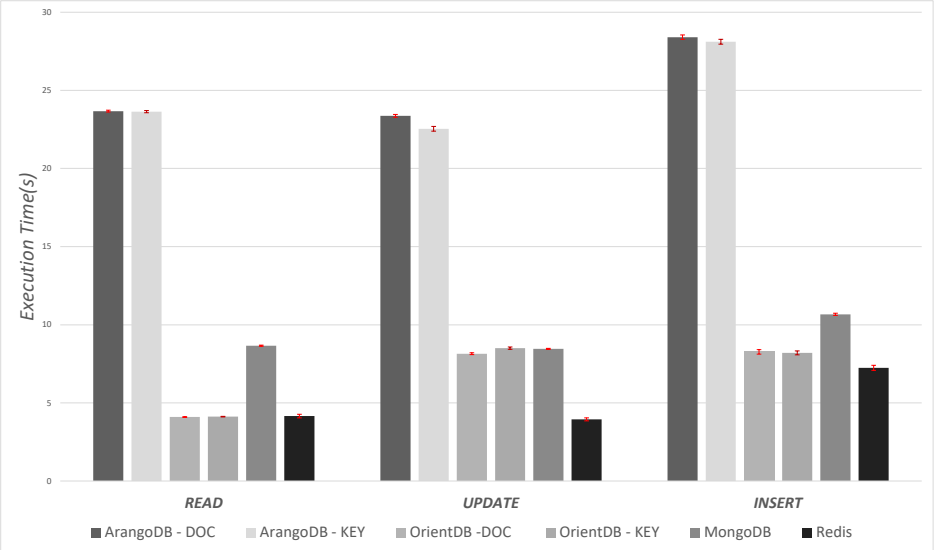


Figura 17: Média de tempo executando 10000 operações

O Redis fornece a melhor economia de energia para todos os comandos. Como exemplo, o tempo de execução do ArangoDB aumentou em 68,25% para o comando de leitura, mas o consumo de energia aumentou 62,65%. Para o OrientDB e o MongoDB, o Redis o incremento foi de 48,56%, 38,49% e 369,93%, respectivamente.

A Figura 17 mostra os tempos de execução para a carga de trabalho de 10.000 operações. O comportamento se assemelha à carga de trabalho de 5.000, no sentido de que o ArangoDB fornece o desempenho mais lento. Para leitura, o OrientDB é rápido como o Redis, e tal DBMS é apenas 13,17% mais lento que o Redis para inserção. Além

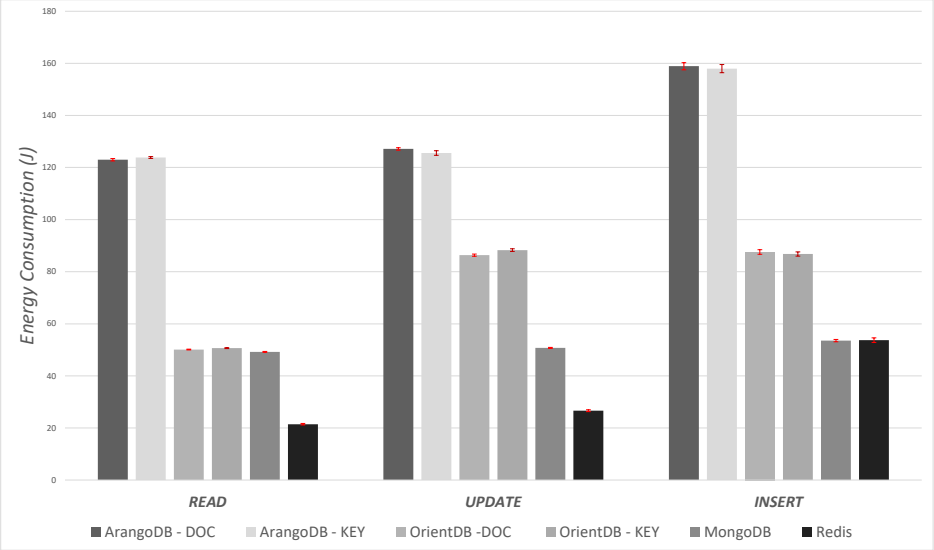


Figura 18: Média de consumo de Energia considerando 10000 operações

disso, o MongoDB é mais lento para todas as operações em comparação ao OrientDB, exceto para atualização e OrientDB-KEY. OrientDB-DOC e OrientDB-KEY têm uma pequena diferença $(0,35s)$, na qual o primeiro realiza a atualização mais rapidamente.

Em relação ao ArangoDB-DOC, ArangoDB-Key e atualização, há uma diferença representativa, que é de cerca de $1s$ para esta carga de trabalho. Em relação ao consumo de energia, a Figura 18 apresenta os valores médios. Embora o OrientDB tenha um tempo de execução notável para uma carga de trabalho maior, o consumo de energia é maior que o MongoDB e o Redis para atualização e inserção. [31]

### 4.2.1 Correlação

Para todos os SGBDs, existe uma forte correlação linear entre o tempo de execução e o consumo de energia, conforme indicado pelo coeficiente de determinação ($R^2$), dados por:

- **ArangoDB**
  - **ArangoDB-DOC**: $R^2 = 0,985$, com a equação de correlação $y = -8.474x - 5.744$ (Figura 19).
  - **ArangoDB-KEY**: $R^2 = 0,986$, com a equação de correlação $y = -8.778x - 5.785$ (Figura 20).
- **OrientDB**

  - **OrientDB-DOC**: $R^2 = 0,911$, com a equação de correlação $y = 5.262x - 9.621$ (Figura 21).
  - **OrientDB-KEY**: $R^2 = 0,902$, com a equação de correlação $y = 6.954x - 9.306$ (Figura 22).

- **MongoDB**
  - $R^2 = 0,943$, com a equação de correlação $y = -7.589x - 6.125$ (Figura 23).

- **Redis**
  - $R^2 = 0,927$, com a equação de correlação $y = -2.895x - 7.292$ (Figura 24).

As figuras de correlação também descrevem a equação linear para cada SGBD. A inclinação (ou seja, a primeira derivada) fornece uma informação interessante sobre a influência do benchmark no consumo de energia, mas esse valor não deve ser o único mecanismo de comparação com outros sistemas de banco de dados. O ArangoDB - DOC tem a menor taxa $J/s$ (5.744), mas esse sistema leva mais tempo para executar as cargas de trabalho. Portanto, o consumo de energia é consideravelmente alto, levando em consideração todas as operações em uma solicitação.

Conforme apresentado nos gráficos de tempo e consumo de energia, o Redis tem um desempenho proeminente com consumo de energia reduzido. Embora esse SGBD tenha uma taxa significativa (7.292), mostrado na Figura 24, os tempos de execução são os mais baixos em relação a todos os sistemas de banco de dados. Por exemplo, uma comparação direta apenas com as inclinações indicaria que o ArangoDB é mais eficiente em termos de energia do que o Redis, o que contradiz os resultados apresentados. Usando correlação linear, os tempos de execução não podem ser negligenciados.

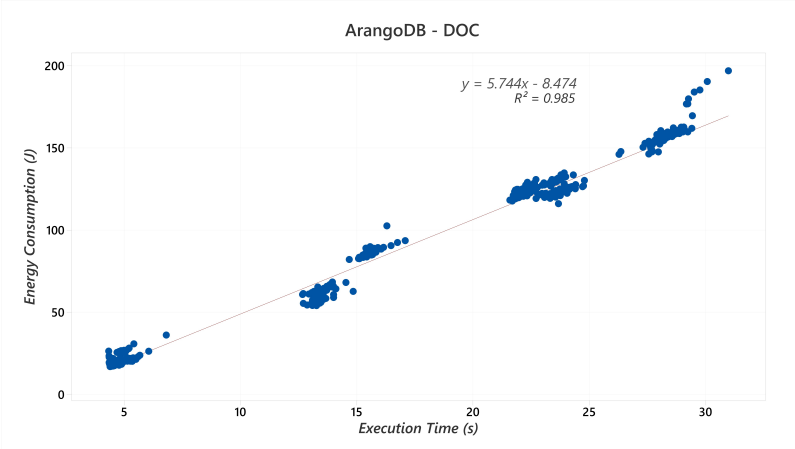


Figura 19: ArangoDB-DOC: Correlação Energia × Tempo Execução

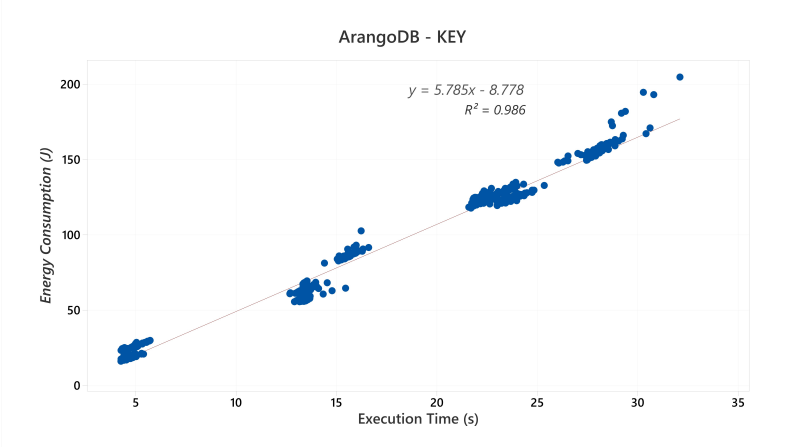


Figura 20: ArangoDB-KEY: Correlação Energia × Tempo Execução

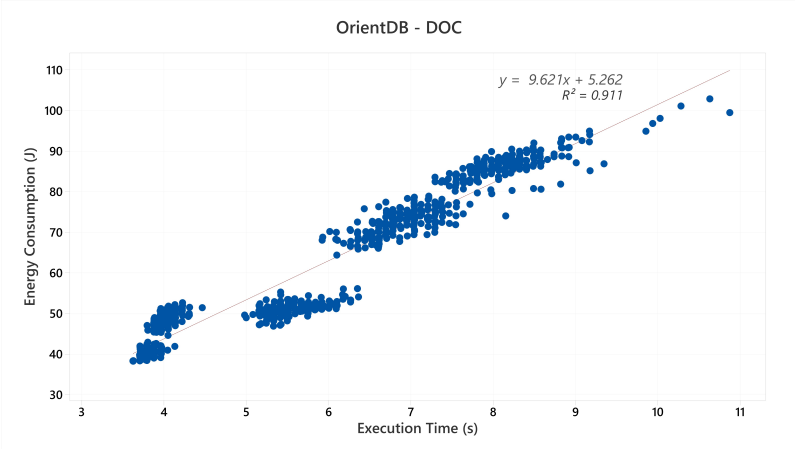


Figura 21: OrientDB-DOC: Correlação Energia × Tempo Execução

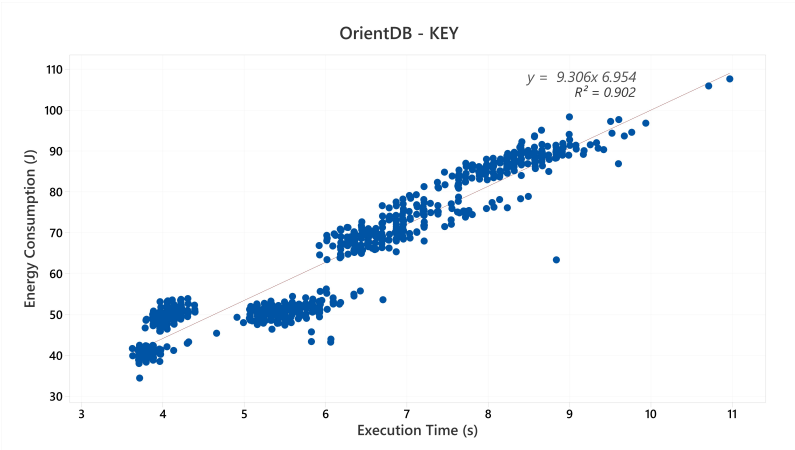


Figura 22: OrientDB-KEY: Correlação Energia × Tempo Execução

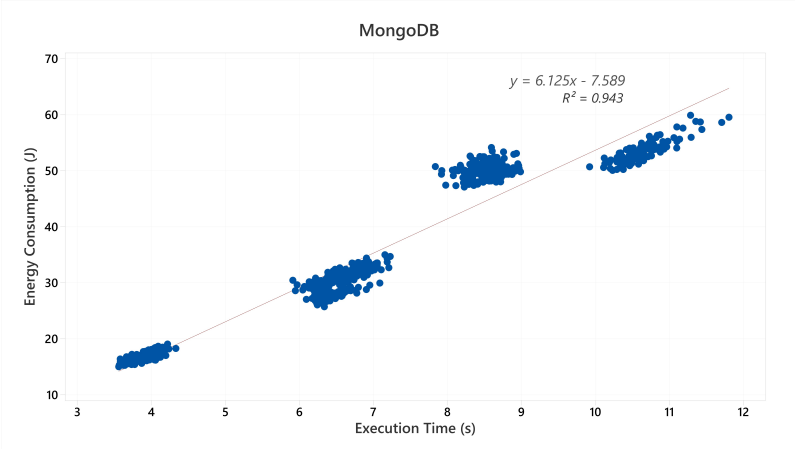


Figura 23: MongoDB: Correlação Energia × Tempo Execução

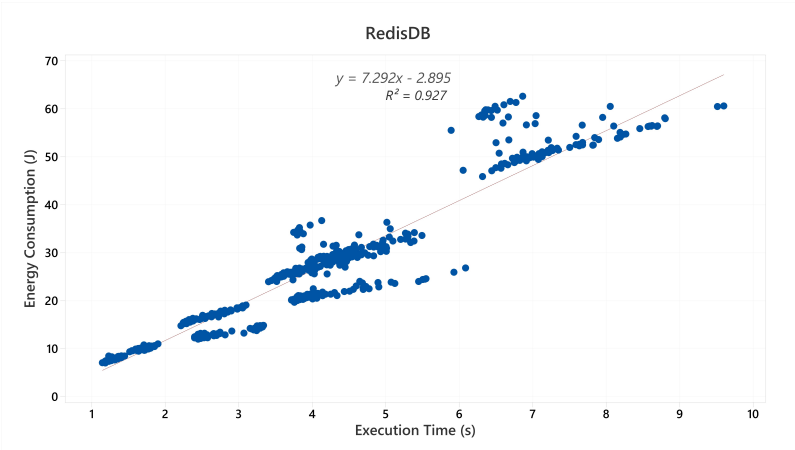


Figura 24: Redis: Correlação Energia × Tempo Execução

# 5 CONCLUSÃO

Nos anos recentes, o aumento exponencial do volume de dados processados por aplicações computacionais tem exigido soluções mais eficientes em desempenho e consumo energético. A crescente utilização de Sistemas de Gerenciamento de Bancos de Dados (SGBDs) NoSQL para lidar com cargas de trabalho intensivas levanta questões sobre o impacto dessas tecnologias no consumo energético, especialmente em infraestruturas distribuídas e data centers.

Diante desse cenário, este estudo teve como objetivo investigar como diferentes arquiteturas de bancos de dados influenciam o consumo de energia, possibilitando a otimização de recursos e a redução de custos operacionais. Para isso, foi desenvolvido e validado um ambiente para mensuração do consumo energético de SGBDs NoSQL, permitindo uma análise experimental da relação entre eficiência energética e operações de armazenamento e recuperação de dados.

Com o intuito de analisar essas diferenças de forma detalhada, foi desenvolvido um ambiente modular de medição de consumo energético, permitindo a interconexão entre o servidor de medição e os dispositivos de coleta de dados. Esse sistema possibilitou a obtenção precisa de informações sobre o consumo energético durante a execução de operações em diferentes bancos de dados.

Para validar a confiabilidade do ambiente de medição, um medidor comercial MINIPA ET-4091 foi utilizado como referência, possibilitando a comparação e calibração dos valores obtidos. Métodos estatísticos rigorosos, como Bootstrap e T-Student, foram empregados para assegurar a precisão das medições e garantir que os resultados refletissem corretamente o impacto energético das operações analisadas.

A relevância da pesquisa estende-se tanto para o meio acadêmico quanto para o setor industrial. Os achados desta pesquisa destacam a importância do consumo energético dos SGBDs NoSQL em data centers e computação em nuvem. Como esses bancos de dados são amplamente utilizados em infraestruturas distribuídas, a escolha de um sistema mais eficiente pode reduzir custos operacionais e impacto ambiental. Nesse contexto, foi projetado e implementado um servidor de medição baseado em Arduino, equipado com sensores especializados para monitoramento contínuo do consumo de energia dos bancos de dados.

Para a medição do consumo de energia, foram empregados sensores como o ACS712-20A, para monitoramento da corrente elétrica, e o ZMPT101B, para a medição da tensão. Esses sensores permitiram a quantificação do consumo real de energia com base nos princípios da potência elétrica. A estrutura modular do sistema garantiu a flexibilidade e escalabilidade da solução, possibilitando a replicação dos testes em diferentes cenários operacionais.

Além disso, os testes foram conduzidos utilizando o *Yahoo! Cloud Serving Benchmark (YCSB)*, um *framework* amplamente utilizado para avaliação de desempenho de bancos de dados NoSQL. O YCSB foi configurado para simular diferentes cargas de trabalho, abrangendo operações como inserção, leitura e atualização de dados. Isso permitiu a comparação direta entre os bancos de dados analisados, fornecendo uma visão clara sobre como cada sistema gerencia energia sob diferentes condições operacionais.

Os experimentos realizados permitiram a identificação de padrões distintos de consumo energético entre diferentes SGBDs NoSQL. Os resultados indicaram que bancos de dados chave-valor, como o Redis, apresentam menor consumo energético, especialmente em operações de leitura, devido à sua arquitetura otimizada para acessos rápidos. Em contrapartida, bancos orientados a documentos, como o MongoDB, consumiram mais energia, particularmente sob cargas intensas de inserção e atualização. Já bancos orientados a grafos e colunares apresentaram variações de consumo, dependendo da complexidade das consultas e do volume de dados manipulados.

Além disso, os resultados experimentais revelaram correlações diretas entre tempo de execução e consumo energético, evidenciando que a eficiência dos algoritmos de recuperação e armazenamento de dados impacta significativamente a economia de energia. Essas observações ressaltam a importância da otimização das consultas e da modelagem dos dados para reduzir o consumo energético, tornando esse fator essencial na escolha do banco de dados mais adequado para cada aplicação.

## 5.1 **Trabalhos Futuros**

Os resultados obtidos neste estudo abriram diversas possibilidades para investigações futuras sobre eficiência energética em SGBDs NoSQL. Embora este trabalho tenha fornecido uma análise detalhada do impacto energético de diferentes bancos de dados, algumas melhorias podem ser exploradas para aprofundar e expandir a compreensão

sobre esse tema.

Embora este estudo tenha apresentado descobertas relevantes, algumas limitações foram identificadas e podem ser abordadas em trabalhos futuros. A análise considerou um conjunto específico de SGBDs NoSQL, e, apesar da diversidade das arquiteturas estudadas, outras abordagens podem apresentar comportamentos energéticos distintos. Além disso, a carga de trabalho utilizada nos testes, baseada no *Yahoo! Cloud Serving Benchmark (YCSB)*, pode não representar integralmente padrões reais de uso. Dessa forma, a adoção de *benchmarks* alternativos e a realização de testes em aplicações reais podem fornecer uma visão ainda mais ampla do impacto energético.

A infraestrutura de hardware utilizada também pode ter influenciado os resultados. O servidor de medição baseado em Arduino, apesar de validado, possui limitações na frequência de coleta de dados. Sensores mais precisos e métodos de amostragem avançados podem aprimorar a confiabilidade das medições e reduzir possíveis variações nos resultados. Além disso, fatores externos, como variações de temperatura, consumo de memória e otimizações específicas de software, não foram analisados, mas podem impactar significativamente o consumo energético.

Outro aspecto relevante para trabalhos futuros é a implementação de um mecanismo dinâmico de otimização energética, que possa sugerir ajustes automáticos para reduzir o consumo de energia sem comprometer o desempenho. Estratégias baseadas em aprendizado de máquina ou heurísticas adaptativas podem tornar essa abordagem mais aplicável a cenários reais, permitindo recomendações automatizadas para maior eficiência energética.

Uma direção promissora para estudos futuros envolve a expansão do conjunto de bancos de dados analisados. A inclusão de outras tecnologias NoSQL, como Cassandra, CouchDB e Neo4j, pode oferecer uma visão mais ampla sobre o impacto energético em diferentes arquiteturas e modelos de armazenamento. Além disso, testes em ambientes distribuídos, onde múltiplos servidores operam simultaneamente, permitirão uma avaliação mais precisa dos impactos energéticos em infraestruturas escaláveis.

Outra possibilidade relevante está na diversificação dos cenários de carga de trabalho. Embora o YCSB tenha sido amplamente utilizado neste estudo, a adoção de benchmarks alternativos ou a coleta de dados em aplicações reais pode fornecer uma análise mais representativa. Testes envolvendo cargas variáveis, picos de acesso e padrões

de uso típicos de aplicações industriais podem aprimorar a capacidade de generalização dos resultados.

Além disso, é importante aprimorar as estratégias de otimização do consumo energético, incluindo *tuning* automático de parâmetros, gerenciamento dinâmico de recursos e uso de aprendizado de máquina para prever padrões de consumo e sugerir configurações mais eficientes.

Por fim, futuros estudos podem explorar fatores externos que influenciam o consumo energético, como temperatura, consumo de memória e otimizações de *hardware* e *software*, contribuindo para modelos mais precisos e soluções computacionais mais sustentáveis.

# REFERÊNCIAS

1 GSMA. *The Mobile Economy 2023*. 2023. ⟨https://www.gsma.com/mobileeconomy/⟩. Acesso em: 10 jan. 2025.

2 VIRMANI, S. Investigating the environmental sustainability of data centers. 2025.

3 LELLA, H. S. et al. Towards comprehending energy consumption of database management systems - a tool and empirical study. In: *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2024. (EASE '24), p. 272–281. ISBN 9798400717017. Available at: ⟨https://doi.org/10.1145/3661167.3661174⟩.

4 LELLA, H. S. et al. *DBJoules: An Energy Measurement Tool for Database Management Systems*. 2023. Available at: ⟨https://arxiv.org/abs/2311.08961⟩.

5 BARROSO, L. A.; HÖLZLE, U. The case for energy-proportional computing. *Computer*, IEEE, v. 40, n. 12, p. 33–37, 2007.

6 PROKHORENKO, V.; BABAR, M. A. Offloaded data processing energy efficiency evaluation. *Informatica*, Vilnius University Institute of Data Science and Digital Technologies, v. 35, n. 3, p. 649–669, 2024. ISSN 0868-4952.

7 IMRAN, S. et al. Big data analytics in healthcare  a systematic literature review and roadmap for practical implementation. *IEEE/CAA Journal of Automatica Sinica*, v. 8, n. 1, p. 1–22, 2021.

8 CHANG, F. et al. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, Association for Computing Machinery, New York, NY, USA, v. 26, n. 2, jun 2008. ISSN 0734-2071. Available at: ⟨https://doi.org/10.1145/1365815.1365816⟩.

9 DB-ENGINES. *Knowledge Base of Relational and NoSQL Database Management Systems*. 2023. Disponível em: ⟨https://db-engines.com/en/ranking⟩. Acessado em: 01/10/2022.

10 SANFILIPPO, S.; NOORDHUIS, P. *Redis*. 2009.

11 MEIER, A.; KAUFMANN, M. *SQL & NoSQL databases*. [S.l.]: Springer, 2019.

12 BRADSHAW, S.; BRAZIL, E.; CHODOROW, K. *MongoDB: the definitive guide: powerful and scalable data storage*. [S.l.]: O'Reilly Media, 2019.

13 ROBINSON, I.; WEBBER, J.; EIFREM, E. *Graph databases: new opportunities for connected data*. [S.l.]: "O'Reilly Media, Inc.", 2015.

14 HEWITT, E. *Cassandra: the definitive guide*. [S.l.]: "O'Reilly Media, Inc.", 2010.

15 FREIRE-GONZÁLEZ, J. *Eficiencia energética y efecto rebote. Conceptos, métodos y políticas*. [S.l.], 2016.

16 RONG, H. et al. Optimizing energy consumption for data centers. *Renewable and Sustainable Energy Reviews*, v. 58, p. 674–691, 2016. ISSN 1364-0321. Available at: ⟨https://www.sciencedirect.com/science/article/pii/S1364032115016664⟩.

17 2024 United States Data Center Energy Usage Report. Available at: ⟨https://eta-publications.lbl.gov/sites/default/files/2024-12/lbnl-2024-united-states-data-center-energy-usage-report.pdf⟩.

18 HENNESSY, J. L.; PATTERSON, D. *Arquitetura de computadores: uma abordagem quantitativa*. [S.l.]: Elsevier Brasil, 2014.

19 DANDAMUDI, S. P. *Fundamentals of computer organization and design*. [S.l.]: Springer, 2003.

20 CHAPMAN, S. J. *Fundamentos de máquinas elétricas*. [S.l.]: AMGH editora, 2013.

21 DAS, J. *Power system analysis: short-circuit load flow and harmonics*. [S.l.]: CRC press, 2002.

22 RIGGIO, R.; RASHEED, T. Toward enterprise virtual power consumption monitoring with joule. In: IEEE. *2014 IEEE Network Operations and Management Symposium (NOMS)*. [S.l.], 2014. p. 1–6.

23 COSTA, F. et al. Real-time power measurement using the maximal overlap discrete wavelet-packet transform. In: *2018 IEEE Power Energy Society General Meeting (PESGM)*. [S.l.: s.n.], 2018. p. 1–1.

24 EFRON, B. *Bootstrap methods: another look at the jackknife*. [S.l.]: Springer, 1992.

25 WILCOXON, F. Individual comparisons by ranking methods. *Biometrics Bulletin*, [International Biometric Society, Wiley], v. 1, n. 6, p. 80–83, 1945. ISSN 00994987. Available at: ⟨http://www.jstor.org/stable/3001968⟩.

26 MEASUREMENT Server Documentation. Available at: ⟨https://www.cin.ufpe.br/~eagt⟩.

27 ACS21720A Datasheet. Available at: ⟨https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf⟩.

28 ZMPT101B Datasheet. Available at: ⟨https://innovatorsguru.com/wp-content/uploads/2019/02/ZMPT101B.pdf⟩.

29 NILSSON, J.; RIEDEL, S. *Electric Circuits*. [S.l.]: Pearson Education, Incorporated, 2018. (Always Learning). ISBN 9781292261041.

30 MINIPA ET-4091 Datasheet. Available at: ⟨https://www.minipa.com.br/images/Manual/ET-4091-1102-BR.pdf⟩.

31 FALCãO, F. et al. Energy consumption and performance evaluation of multi-model nosql dbmss. *Revista de Informática Teórica e Aplicada*, v. 30, n. 2, p. 132–140, May 2024. Available at: ⟨https://seer.ufrgs.br/index.php/rita/article/view/136568⟩.

32 YCSB. *Yahoo! Cloud Serving Benchmark. COOPER, B. F.* 2022. Disponível em: ⟨https://github.com/brianfrankcooper/YCSB⟩. Acessado em: 01/10/2022.

33 MONTGOMERY, D. C.; RUNGER, G. C. *Applied statistics and probability for engineers*. [S.l.]: John wiley & sons, 2010.