UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE DE IMFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO CIÊNCIA DA COMPUTAÇÃO

FÚLVIO MYBBSON CARNEIRO FALCÃO

Avaliação de Consumo de Energia e de Desempenho de SGBDs NoSQL Multimodelos

Recife

2024

FÚLVIO MYBBSON CARNEIRO FALCÃO

Avaliação de Consumo de Energia e de Desempenho de SGBDs NoSQL Multimodelos

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Eduardo Antônio Guimarães Tavares

Coorientador: Prof. Dr. Carlos Gomes Araujo

Recife

2024

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Falcão, Fúlvio Mybbson Carneiro.
   Avaliação de consumo de energia e de desempenho de SGBDs NoSQL Multimodelos / Fúlvio Mybbson Carneiro Falcão. - Recife, 2024.
   93f.: il.

   Dissertação (Mestrado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciências da Computação, 2024.
   Orientação: Eduardo Antônio Guimarães Tavares.
   Coorientação: Carlos Gomes Araujo.
   Inclui referências e apêndices.

   1. Consumo de Energia; 2. Armazenamentos de Dados Múltiplos; 3. Comparação de Desempenho; 4. Sistema de Banco de Dados Multimodelo; 5. Persistência Poliglota; 6. NoSQL. I. Tavares, Eduardo Antônio Guimarães. II. Araujo, Carlos Gomes. III. Título.

UFPE-Biblioteca Central

Fúlvio Mybbson Carneiro Falcão

“AVALIAÇÃO DO CONSUMO DE ENERGIA E DESEMPENHO DE SGBDS NOSQL MULTIMODELOS”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Redes de Computadores.

Aprovado em: 02 de fevereiro de 2024.

BANCA EXAMINADORA

_____

Prof. Dr. Jamilson Ramalho Dantas
Centro de Informática / UFPE

_____

Prof. Dr. Dra. Erica Teixeira Gomes de Sousa
Departamento de Computação / UFRPE

_____

Prof. Dr. Eduardo Antonio Guimarães Tavares
Centro de Informática / UFPE
(**orientador**)

Dedico este trabalho a Deus e à minha família, a todos que de alguma forma se inspiram em mim e que me encorajam a enfrentar os desafios, que foram meu porto seguro perante as dificuldades durante este percurso. Essa conquista não é só minha, também é de vocês.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por todas as oportunidades de aprendizado intelectual e moral que tive ao longo desta pesquisa, e pela força para seguir em frente.

À minha família, minha querida esposa e filha, as quais privei um bom tempo de lazer nesta fase das nossas vidas. Obrigado pela paciência, agora colheremos alguns frutos.

Ao meu Pai, onde quer que esteja, a minha Mãe, pois foi a sua dedicação sem limites, o esforço incessante, o investimento em minha educação, e credibilidade cega em minha capacidade, além do apoio constante, que me permitiram chegar até aqui.

Aos meus familiares, irmãos, tios, primos, avós, sogros, cunhados, que acompanharam minha luta desde o início (mesmo que às vezes de longe), acreditaram em mim, e me incentivaram até o final.

Também quero lembrar dos colegas que fiz durante o curso de mestrado, onde vivenciamos momentos de aprendizado, e aos colegas do MoDCS pelo apoio e troca de conhecimento.

Gostaria de expressar minha profunda gratidão ao meu orientador, Professor Eduardo Antônio Guimarães Tavares, por seu apoio, orientação e contribuições inestimáveis ao longo deste projeto. Sua expertise e dedicação foram fundamentais para o sucesso desta pesquisa.

Ao Professor Carlos Araujo, que me ajudou nesta caminhada, obrigado pela coorientação, sou grato pelos seus ensinamentos, comentários e sugestões que foram muito importantes nas tomadas de decisões.

Aos professores Erica Teixeira Gomes de Sousa e Jamilson Ramalho Dantas, pela disponibilidade em participar da banca deste trabalho e pelas excelentes contribuições.

À Universidade Federal de Pernambuco (UFPE), pelo suporte acadêmico durante esses anos.

Por fim, agradeço aos demais colegas, amigos e parentes que contribuíram direta ou indiretamente para a realização deste objetivo.

"Você não consegue ligar os pontos olhando pra frente; você só consegue ligá-los olhando pra trás. Então você tem que confiar que os pontos se ligarão algum dia no futuro. Você tem que confiar em algo – seu instinto, destino, vida, carma, o que for. Esta abordagem nunca me desapontou, e fez toda diferença na minha vida. " (JOBS, 2005).

RESUMO

Os Sistema de Gerenciamento de Banco de Dados (SGBDs) NoSQL foram introduzidos recentemente como alternativas aos sistemas de gerenciamento de banco de dados relacionais tradicionais. Esses sistemas empregam modelos de dados notavelmente simples e altamente escaláveis que aumentam a eficiência e o desempenho de uma nova geração de sistemas com alta exigência de acesso e escalabilidade. NoSQL não substitui as abordagens de sistemas de gerenciamento de banco de dados relacionais, mas sim atende às restrições relacionadas à manipulação de dados em massa. Entretanto, novos tipos de aplicações que modelam seus dados usando dois ou mais modelos de dados NoSQL são conhecidos como aplicações com persistência poliglota. Normalmente, suas implementações são complexas porque elas devem gerenciar e armazenar seus dados utilizando simultaneamente vários sistemas de gerenciamento de bancos de dados. Recentemente, foi introduzida uma nova família de sistemas de gerenciamento chamados de multi-modelo que integram vários modelos de dados NoSQL em um único sistema. A importância de tal tecnologia motiva muitos trabalhos, principalmente em relação ao desempenho.

Existem poucos estudos e trabalhos que caracterizam e comparam o consumo de energia no contexto de SGBDs NoSQL e SGBDs multimodelo, apesar de sua importância. De fato, o consumo de energia não deve ser negligenciado devido ao aumento dos custos financeiros e ambientais.

A fim de avaliar essa questão, esta dissertação de mestrado analisa uma avaliação do desempenho dos gerenciadores de bancos de dados SGBDs tradicionais e SGBDs multimodelo quando utilizados em aplicações com persistência poliglota, mais especificamente MongoDB (orientado a documentos), Redis (chave-valor) e ArangoDB (multimodelo), OrientDB (multi-modelo). Para essa avaliação, foram aplicados um conjunto de testes (*benchmark*) simulando uma aplicação com persistência poliglota executando operações básicas em bancos de dados. A proposta baseia-se em *Design of Experiments*, de tal forma que as cargas de trabalho são geradas por Yahoo! *Cloud Serving Benchmark* (YCSB) produzindo leitura, escrita e atualização, por ciclos de 1.000, 5.000 e 10.000 operações.

Para a mensurar o consumo de energia foi implementado um framework de medição que pode ser usado para comparar o consumo de energia de diferentes aspectos. Onde no nosso contexto permite aos usuários possam tomar decisões ao selecionar o SGBDs mais adequado para suas necessidades, considerando não apenas o desempenho em termos de

consultas e transações, mas também o impacto ambiental associado. Incentivando práticas mais sustentáveis e responsáveis no desenvolvimento e operação de sistemas de banco de dados.

As métricas são tempo de execução e consumo de energia, assim como a evolução no incremento da carga de trabalho. Os resultados demonstram que o consumo de energia pode variar significativamente entre os SGBDs para comandos distintos (por exemplo, leitura) e cargas de trabalho.

Palavras-chaves: Consumo de Energia. Armazenamentos de Dados Múltiplos. Comparação de Desempenho. Sistema de Banco de Dados Multimodelo. Persistência Poliglota. NoSQL.

ABSTRACT

NoSQL database management systems have recently been introduced as alternatives to traditional relational database management systems. These systems implement simpler and more scalable data models that increase the efficiency and performance of a new generation of systems with high access and scalability requirements. NoSQL does not replace relational database management system approaches, but rather addresses the constraints related to mass data manipulation. However, new types of applications that model their data using two or more NoSQL data models are known as applications with polyglot persistence. Typically, their implementations are complex because they must manage and store their data using several database management systems simultaneously. Recently, a new family of management systems called multimodel has been introduced, which integrate several NoSQL data models into a single system. The importance of this technology motivates a lot of work, especially in relation to performance.

There are few studies and papers that characterize and compare energy consumption in the context of NoSQL Database Management Systems (DBMSs) and multimodel DBMSs, despite their importance. In fact, energy consumption should not be neglected due to the increase in financial and environmental costs.

In order to assess this issue, this master's thesis analyzes an evaluation of the performance of traditional NoSQL and multimodel NoSQL database managers when used in applications with polyglot persistence, more specifically MongoDB (document-oriented), Redis (key-value) and ArangoDB (multimodel), OrientDB (multimodel). For this evaluation, a set of tests (*benchmark*) were applied simulating an application with polyglot persistence performing basic database operations. The proposal is based on *Design of Experiments*, in such a way that the workloads are generated by *Yahoo! Cloud Serving Benchmark* (YCSB) producing reads, writes, and updates, for cycles of 1,000, 5,000, and 10,000 operations.

To measure energy consumption, a measurement framework was implemented that can be used to compare energy consumption from different aspects. Where in our context it allows users to make decisions when selecting the most suitable DBMS for their needs, considering not only performance in terms of queries and transactions, but also the associated environmental impact. Encouraging more sustainable and responsible practices in the development and operation of database systems.

The metrics are execution time and energy consumption, as well as the evolution of the workload increment. The results show that energy consumption can vary significantly between SGBDs for different commands (e.g. reading) and workloads.

Keywords: Energy consumption. Multiple Data Stores. Performance Comparison. Multimodel Database System. Polyglot Persistence. NoSQL.

LISTA DE FIGURAS

Figura 1 – Exemplo de objeto armazenado em um banco de dados orientado a objetos, utilizando JSON, adaptado de (MONGO-DB, 2022). 39
Figura 2 – Distribuição da popularidade dos bancos orientados a documentos. 39
Figura 3 – Armazenamento chave-valor em um banco de dados NoSQL, adaptado de (REDIS, 2022). 40
Figura 4 – Distribuição da popularidade dos bancos orientados a documentos. 41
Figura 5 – Persistência poliglota em banco de dados. 42
Figura 6 – (a) Persistência poliglota e (b) SGBDs MultiModelo, adaptado de (ORACLE, 2022). 43
Figura 7 – Modelo básico de um processo ou sistema, adaptado de (MONTGOMERY; RUNGER, 2020). 50
Figura 8 – Efeito principal dos fatores A e B. 51
Figura 9 – Efeito de interação dos fatores A e B. 52
Figura 10 – Arquitetura YCSB, adaptado de (YCSB, 2022). 56
Figura 11 – Distribuição zipfian YCSB. 57
Figura 12 – Metodologia proposta na dissertação. 60
Figura 13 – *Measurement Framework* 61
Figura 14 – Média Tempo de Execução considerando 1.000 Operações 67
Figura 15 – Média Tempo de Execução considerando 5.000 Operações 67
Figura 16 – Média Tempo de Execução considerando 10.000 Operações 68
Figura 17 – Média Consumo de Energia considerando 1.000 Operações 68
Figura 18 – Média Consumo de Energia considerando 5.000 Operações 69
Figura 19 – Média Consumo de Energia considerando 10.000 Operações 69
Figura 20 – Correlação Tempo de Execução x Consumo de Energia ArangoDB-DOC . . 76
Figura 21 – Correlação Tempo de Execução x Consumo de Energia ArangoDB-KEY . . 76
Figura 22 – Correlação Tempo de Execução x Consumo de Energia OrientDB-DOC . . 77
Figura 23 – Correlação Tempo de Execução x Consumo de Energia OrientDB-KEY . . . 77
Figura 24 – Correlação Tempo de Execução x Consumo de Energia MongoDB 78
Figura 25 – Correlação Tempo de Execução x Consumo de Energia Redis 78
Figura 26 – Arquitetura de Medição de Energia Elétrica. 86

Figura 27 – Diagrama de Sequencia - Medição de energia elétrica 87

LISTA DE CÓDIGOS

Código Fonte 1 – Exemplo de link do OrientDB, adaptado de (ORIENT-DB, 2022). . 44
Código Fonte 2 – Exemplo de *insert* de documentos em AQL, adaptado de (ARANGO-DB, 2022). 45
Código Fonte 3 – Exemplo de consulta em grafo no ArangoDB, adaptado de (ARANGO-DB, 2022). 46
Código Fonte 4 – Exemplo das operações *insert* e *update* utilizando Javascript, adaptado de (ORIENT-DB, 2022). 46
Código Fonte 5 – Exemplo de consulta em Javascript, adaptado de (ORIENT-DB, 2022). 46
Código Fonte 6 – Log de execução da carga de trabalho, servidor. 88
Código Fonte 7 – Log de execução da carga de trabalho com obtenção de amostras no Cliente. 88
Código Fonte 8 – Exemplo - Script de execução da *Workload*. 89
Código Fonte 9 – Exemplo - Script Workload. 90

LISTA DE TABELAS

Tabela 1 – Comparação entre este trabalho e trabalhos relacionados 34
Tabela 2 – Projeto de experimentos fatoriais completo com dois fatores com dois níveis e sua interação. 53
Tabela 3 – Projeto de experimentos fatoriais fracionado com 3 fatores e 2 níveis cada. 54
Tabela 4 – Consumo de Energia ET-4091 x Arduino 62
Tabela 5 – Paired T-test 62
Tabela 6 – Tempo de Execução e Consumo de energia para ARANGO DB - DOC . . . 64
Tabela 7 – Tempo de Execução e Consumo de energia para OrientDB - DOC 65
Tabela 8 – Tempo de Execução e Consumo de energia para ARANGO DB - KEY . . . 65
Tabela 9 – Tempo de Execução e Consumo de energia para OrientDB - KEY 65
Tabela 10 – Tempo de Execução e Consumo de energia para MongoDB 66
Tabela 11 – Tempo de Execução e Consumo de energia para Redis 66
Tabela 12 – ANOVA: Tempo de Execução 68
Tabela 13 – ANOVA: Consumo de Energia 69
Tabela 14 – Tempo de Execução Tukey 1.000 Operações (SDBD) 70
Tabela 15 – Tempo de Execução 1.000 Operações (SGBD X COMANDO) 70
Tabela 16 – Tempo de Execução Tukey 5000 Operações (SDBD) 71
Tabela 17 – Tempo de Execução 5.000 Operações (SGBD X COMANDO) 72
Tabela 18 – Consumo de Energia Tukey 5.000 Operações (SDBD) 72
Tabela 19 – Consumo de Energia Tukey 5.000 Operações (SGBD X COMANDO) . . . 73
Tabela 20 – Tempo de Execução Tukey 10000 Operações (SDBD) 73
Tabela 21 – Tempo de Execução 10.000 Operações (SGBD X COMANDO) 74
Tabela 22 – Consumo de Energia Tukey 10.000 Operações (SDBD) 74
Tabela 23 – Consumo de Energia Tukey 10.000 Operações (SGBD X COMANDO) . . . 75

LISTA DE ABREVIATURAS E SIGLAS

ACID *Atomicity, Consistency, Isolation, and Durability*

ANOVA *Analysis of Variance*

API *Application Programming Interface*

AQL *Acceptable Quality Level*

BASE *Basically Available, Soft state, and Eventually consistent*

CAP *Consistency, Availability, and Partition tolerance*

CPU *Central Processing Unit*

CRUD *Create, Read, Update, and Delete*

DB *Data Base*

DHT *Distributed Hash Table*

DOE *Design of Experimentse*

HTML *HyperText Markup Language*

HTTP *Hypertext Transfer Protocol*

IC *Confidence Interval*

JSON *JavaScript Object Notation*

MVCC *Multi-Version Concurrency Control*

MVP *Minimum Viable Product*

NoSQL *Not Only SQL*

REST *Representational State Transfer*

SGBDs Sistema de Gerenciamento de Banco de Dados

SQL *Structured Query Language*

XML *eXtensible Markup Language*

YCSB Yahoo! *Cloud Serving Benchmark*

SUMÁRIO

1 INTRODUÇÃO 19
1.1 CONTEXTO 19
1.2 MOTIVAÇÃO 22
1.3 OBJETIVOS 23
1.4 JUSTIFICATIVA 24
1.5 ESTRUTURA DA DISSERTAÇÃO 25
2 TRABALHOS RELACIONADOS 27
2.1 VISÃO GERAL 27
2.2 AVALIAÇÃO DE DESEMPENHO 28
2.3 CONSUMO DE ENERGIA 32
3 REFERENCIAL TEÓRICO 35
3.1 NOSQL SGBD E PERSISTÊNCIA POLIGLOTA 35
3.1.1 Banco de Dados NoSql 35
3.1.1.1 Características dos Bancos de Dados NoSQL 36
3.1.1.2 Tipos e Categoria dos Bancos de Dados NoSQL 38
3.1.2 Persistência Poliglota 41
3.1.2.1 SGBDs NoSQL MultiModelo 42
3.2 AVALIACAO DE DESEMPENHO 48
3.2.1 *Design of Experiments* (DoE) 49
3.2.1.1 Projeto de Experimentos Fatoriais Completos 52
3.2.1.2 Projeto de Experimentos Fatoriais Fracionado 53
3.3 *BENCHMARK* 55
3.3.0.1 Yahoo! Cloud Serving Benchmark (YCSB) 55
3.3.0.2 Distribuições 57
3.3.0.3 Análise de Variância (ANOVA) 58
4 METODOLOGIA E FERRAMENTAS 59
4.0.1 *Framework* de Medição 61
4.0.2 Configuração YCSB 62
5 RESULTADOS EXPERIMENTAIS 64
5.1 RESULTADO DOS TRATAMENTOS 67

5.1.1 Correlação 73
5.1.2 Discussão 75
6 COMENTÁRIOS FINAIS 80
6.1 LIMITAÇÕES 81
6.2 TRABALHOS FUTUROS 81
REFERÊNCIAS 82
APÊNDICE A – MEASUREMENT SERVER 86
APÊNDICE B – APRESENTAÇÃO DOS DADOS 92

1 INTRODUÇÃO

Este capítulo está organizado da seguinte maneira, na Seção 1.1 apresenta-se o contexto da pesquisa proposta neste trabalho. Em seguida, na Seção 1.2 discorre a respeito da motivação acerca do campo de estudo. Logo após, a Seção 1.3 exibe os objetivos, geral e específicos. Por fim, este capítulo denota a estrutura desta proposta de tese (Seção 1.5).

1.1 CONTEXTO

A presença e a importância dos aplicativos vêm crescendo constantemente na vida tanto de usuários quanto de empresas. Seja em busca de competitividade, interatividade ou entretenimento, as ferramentas móveis evidenciam uma busca por otimização, resultando em interações mais eficientes. Com o lançamento contínuo de novos produtos e sensores cada vez mais portáteis, essas soluções estão literalmente ao alcance das mãos dos usuários, presentes nos seus bolsos diariamente.

A tecnologia tem avançado rapidamente e as formas de integração do conteúdo em computadores pessoais, *notebooks*, *tabletes*, celulares, *wearables*, *SmartTV* devem evoluir na mesma velocidade, gerando uma sensação de ambientação e adaptação fácil em qualquer *gadget* que o usuário deseje interagir.

Para exibir resultados eficientes e utilizando menos recursos, tanto do servidor, quanto do *gadget* do usuário. A utilização de bancos de dados *Not Only SQL* NoSQL e *Structured Query Language* SQL relacionais integrados a linguagens de programação específica para cada serviço, otimiza as rotinas e melhora as entregas. A análise de desempenho dessas ferramentas é necessária e deve nos definir um trajeto otimizado para entrega e gerenciamento de conteúdo.

Um dos mais importantes desafios para a comunidade de pesquisadores na área de Banco de Dados têm sido o desenvolvimento da tecnologia para manipulação de expressivos volumes de dados heterogêneos, gerados por aplicações e pessoas(FLORES et al., 2018).

Atualmente os sistemas de gerenciadores de bancos de dados relacionais apresentam dificuldades na escalabilidade e distribuição de dados para manipulação de aplicações que trabalham com *Big Data*. Para suprir essa demanda, uma nova classe de sistemas gerenciadores de bancos de dados que não trabalham com o modelo relacional surgiram, sendo classificados sobre o termo guarda-chuva NoSQL.

Um recurso comum destes novos gerenciadores é que eles suportam modelos de dados especializados, eficientes na escalabilidade e distribuição de dados. Segundo (FLORES et al., 2018) os principais modelos NoSQL são documentos, pares chave-valor, orientado a colunas e orientado a grafos, mas estes modelos não são padronizados e formalizados como o modelo relacional.

Atualmente há mais de duzentas implementações de bancos de dados NoSQL, implementando diferentes tipos de modelos de dados (EDLICH, 2022). A questão de escolha sobre qual destes gerenciadores adotar é uma decisão complexa para arquitetos de software, principalmente porque cada um trabalha com um modelo de dados, linguagens de consulta que não são padronizadas, além de possuírem interfaces de acesso diferentes.

Um fator de decisão frequente é o desempenho dos gerenciadores existentes para cada modelo. Por outro lado, é comum que uma aplicação não se adapte totalmente a um único modelo de dados. Como consequência, essas aplicações são facilmente modeladas e obtêm melhor desempenho se utilizarem vários modelos de dados e diversos gerenciadores de bancos de dados simultaneamente. O termo persistência poliglota descreve as aplicações com este tipo de necessidades e requisitos (SADALAGE; FOWLER, 2012).

A utilização de persistência poliglota, o qual é o uso de diferentes linguagens de programação e diferentes bancos de dados, torna o caminho até a entrega mais rápido, intuitivo e favorável ao crescimento do mercado. Atualmente, com o uso de diferentes equipamentos para consultar, gerenciar ou apenas utilizar o conteúdo, cada um com um *background* de linguagem específica, seja *web*, com o simples e fácil PHP, aplicativos *mobile*, com o uso de java ou apenas *JavaScript*, *TypeScript* e *HyperText Markup Language* (HTML). A obtenção de conhecimento diversificado para programadores, se sentirem livres para trocar a linguagem quando necessário, mantendo suas linhas de projeto e excelência no produto final.

A principal vantagem da adoção da persistência poliglota é ter o melhor desempenho nas aplicações. Este melhor desempenho é o resultado de modelar diferentes componentes ou requisitos do sistema segundo o modelo de dados, mais apropriado para cada caso, ou seja, que executa mais eficientemente as requisições.

Neste caso, a aplicação precisa acessar diferentes gerenciadores de bancos de dados que implementam os modelos de dados utilizados. Os problemas ou desvantagens do uso de persistência poliglota estão relacionados fundamentalmente com a complexidade que introduz no processo de gerenciamento e desenvolvimento destas aplicações. Dentre os problemas podemos citar os seguintes: replicação de dados entre vários gerenciadores de bancos de dados, geren-

ciamento complexo de transações que utilizam vários sistemas, uso de diferentes interfaces de aplicação e ferramentas para cada sistema, dentre outros.

Uma recente alternativa que simplifica estes problemas tem sido a utilização de uma nova classe de sistemas gerenciadores de bancos de dados que integram vários modelos de dados. Estes sistemas são chamados de gerenciadores NoSQL multimodelo.

A introdução de gerenciadores multimodelo simplifica notavelmente o processo de gerenciamento e desenvolvimento de aplicações que incorporam persistência poliglota. Entretanto, surge a indagação sobre o impacto no desempenho dessas aplicações ao optar por esses gerenciadores como alternativa.

Avaliando experimentalmente o desempenho e o consumo de energia de aplicações poliglotas, com e sem o uso de gerenciadores multimodelo, esta dissertação visa contribuir para a pesquisa sobre o impacto desses aspectos.

Na condução desta pesquisa, adotou-se o *Yahoo! Cloud Serving Benchmark* (YCSB), um instrumento destinado a testar diversos gerenciadores de bancos de dados NoSQL, tanto multimodelo quanto monomodelo. Esse framework simula o comportamento de uma aplicação, avaliando os comandos de inserção, leitura e atualização para cada SGBDs utilizado. Além disso, apresenta-se um conjunto de ferramentas de medição, composto por componentes de hardware baseados em Arduino e uma aplicação de software, destinado a estimar tanto o consumo de energia quanto o tempo de execução.

A contribuição fundamental desta dissertação foi a avaliação de desempenho e do consumo de energia entre um SGBDs NoSQL baseado em um único modelo de dados e um SGBDs NoSQL multimodelo, em função de seu consumo de energia elétrica em situações de leitura, escrita e atualização bem como em cargas de trabalho diferentes em um mesmo ambiente de operação. Visto que não se conhece a relação entre o desempenho das operações de umSGBDs NoSQL baseado em um único modelo de dados e um SGBDs NoSQL multimodelo e seu consumo de energia elétrica.

Os resultados permitiram levantar hipóteses sobre quais os cenários em que cada uma das alternativas propostas consegue oferecer o melhor desempenho e consumo de energia. Adicionalmente, foi possível avaliar o impacto dos modelos de representação e implementação utilizados por estes gerenciadores no desempenho dos diferentes tipos de comandos.

1.2 MOTIVAÇÃO

Atualmente, uma enorme quantidade de dados está sendo gerada a partir de diversas fontes, tais como smartphones, computadores, sensores, câmeras, sistemas de posicionamento global, plataformas de redes sociais, transações comerciais, jogos, dispositivos da Internet das Coisas e tecnologia web em dispositivos móveis (BICEVSKA; ODITIS, 2017). Resultaram na explosão de dados estruturados, semiestruturados e não estruturados gerados por aplicativos em todo o mundo, indivíduos e organizações produzem grandes quantidades de dados em alta velocidade (DAVOUDIAN; CHEN; LIU, 2018).

Com a ajuda de ferramentas de análise, é possível extrair a partir desses dados inúmeras informações úteis e aplicá-las no dia a dia. Contudo, um dos desafios mais importantes para a comunidade de pesquisadores de *Data Base* (DB) nos últimos anos tem sido o desenvolvimento de tecnologias para gerenciar essa grande quantidade de dados heterogêneos gerados em uma alta taxa de velocidade por aplicativos e pessoas (FLORES et al., 2018).

Vale destacar que, a tecnologia de Banco de Dados Relacional não é adequada para lidar com essas grandes quantidades de dados altamente heterogêneos devido ao alto *overhead* com o controle da consistência dos dados. Nesse sentido as Comunidades de Banco de dados, buscam novas soluções de armazenamento para esses dados, uma dessas soluções são os Sistemas de Gerência de Banco de Dados (SGBDs) NoSQL (SADALAGE; FOWLER, 2012).

Os sistemas NoSQL frequentemente operam com armazenamento distribuído. A maioria dos bancos de dados NoSQL foi projetada desde o início para suportar ambientes distribuídos, o que significa que eles podem distribuir dados em vários servidores ou nós em um cluster. Essa arquitetura distribuída oferece várias vantagens, incluindo escalabilidade horizontal, alta disponibilidade e tolerância a falhas. Esses sistemas compõem uma família de modelos de dados, sendo os quatro principais: chave-valor, colunar, documento e grafo.

A variedade de dados é um dos maiores desafios para a investigação e a prática dos sistemas de gestão de dados. Os dados estão naturalmente organizados em diferentes formatos e modelos, incluindo dados estruturados, dados semi-estruturados e dados não estruturados. Em resposta direta a necessidade, Sistemas de Gerência de Banco de Dados (SGBDs) que suportam vários modelos surgiram para atender a necessidade de múltiplos formatos de dados, reduzindo a complexidade operacional e mantendo a consistência global dos dados, foram disponibilizados SGBDs que suportam vários modelos de dados simultaneamente (LU; HOLUBOVá, 2019).

O surgimento desses Sistemas de Gerência de Banco de Dados (SGBDs) tem como motivação atender às demandas de aplicações cujos requisitos não se enquadram exclusivamente em um único modelo de dados. Aplicações com essas características, conhecidas como "poliglotas", necessitam acessar dois ou mais (SGBDs) com modelos de dados distintos. Essa abordagem é adotada devido à natureza heterogênea dos dados manipulados, resultando na persistência de dados em diversos formatos (SADALAGE; FOWLER, 2012).

A união dos Sistemas de Gerência de Banco de Dados (SGBDs) NoSQL com a persistência poliglota é designada como SGBD NoSQL multimodelo. Esse tipo de solução é caracterizado por ser um único SGBD que realiza a persistência, indexação e consulta de dados mantidos em diversos modelos de dados NoSQL distintos. A introdução desse gerenciador multimodelo simplifica efetivamente o processo de gerenciamento e desenvolvimento de aplicativos que incorporam persistência poliglota. No entanto, surge a indagação sobre o impacto no desempenho dessas aplicações ao optar por esses gerenciadores como alternativa.

1.3 OBJETIVOS

O objetivo geral deste trabalho é apresentar uma avaliação do consumo de energia de SGBD NoSQL baseado em um único modelo de dados e em multimodelo, considerando comando de leitura, escrita e atualização, em cargas de trabalho diferentes em um mesmo ambiente de operação. Sendo assim, este trabalho tem como foco caracterizar:

- O consumo de energia elétrica de SGBDs NoSQL baseados em um único modelo de dados, Redis e MongoDB representantes respectivamente dos modelos Chave-Valor e orientado a documento os SGBDs NoSQL multimodelo, ArangoDB e OrienteDB
- O impacto dos comandos de leitura, escrita e atualização no consumo de energia elétrica dos SGBDs e
- A correlação do tempo de execução com o consumo de energia elétrica dos SGBDs.

Como objetivos específicos deste trabalho propõe-se:

- Validação do framework de análise de consumo de energia
- Avaliação de SGBDs NoSQL com modelo único e multimodelos
- Comparação o desempenho de NoSQL

- Indicação de melhor modelo de SGBD NoSQL, visto que atualmente é uma decisão complexa para arquitetos e pesquisadores definir qual o melhor gerenciador para determinada implementação e
- Relacionamento ds características dos SGBDs que influenciaram no consumo de energia elétrica

1.4 JUSTIFICATIVA

À medida que um número crescente de pessoas se conecta e busca atingir seus objetivos, seja realizando compras, buscando entretenimento ou explorando outros interesses, o estudo de desempenho de aplicações e a aplicação de melhores práticas, utilizando ferramentas disponíveis no mercado para armazenamento e acesso a esses dados, podem se tornar diferenciais significativos, tanto do ponto de vista econômico quanto em termos de visibilidade para os usuários.

No entanto, com o crescente volume de dados heterogêneos sendo gerados diariamente, intensificou-se a preocupação sobre como armazenar esses dados de maneira eficiente. Isso resultou no aumento do número de ferramentas disponíveis para manipular e armazenar esses dados, tornando essencial o domínio e conhecimento dessas ferramentas para uma gestão eficaz.

No entanto, as demandas para simplificar a interação entre aplicativos e SGBDs com interfaces simples de armazenamento e manipulação de dados nem sempre é possível usando apenas o modelo relacional (LIU et al., 2019).

Para suprir essa necessidade surgiram os SGBDs NoSQL multimodelo, que têm a capacidade de incorporar vários modelos de dados e permite os usuários manipular todos os modelos de dados declarativamente. Portanto, compreender as opções de SGBDs NoSQL multimodelo é essencial no processo de desenvolvimento de aplicações, principalmente para o engenheiro de software ou o projetista de Banco de Dados. Entender como esses sistemas funcionam é de alta importância para que se escolha o SGBD que mais se encaixa melhor às necessidades da aplicação que está sendo desenvolvida.

Com esse intuito, esta dissertação apresenta e analisa alguns SGBDs NoSQL multimodelo para auxiliar no entendimento e na seleção de qual tecnologia pode ajudar melhor a comunidade de pesquisadores de Banco de Dados sobre o consumo de energia em aplicações poliglotas é fundamental para entender como diferentes abordagens e tecnologias podem influenciar o

consumo de energia, permitindo o desenvolvimento de soluções mais sustentáveis, eficientes e economicamente.

1.5 ESTRUTURA DA DISSERTAÇÃO

No início deste capítulo, realiza-se uma contextualização da temática na qual o objeto de pesquisa está inserido. Em seguida, o tema é aprofundado com o intuito de elucidar a motivação em relação aos problemas identificados no campo de pesquisa em questão. Posteriormente, são definidos os objetivos, tanto o geral quanto os específicos, almejados por meio da abordagem proposta neste trabalho.

O Capítulo 2 apresenta os trabalhos relacionados a este estudo, visando expor as soluções existentes na literatura e identificar as lacunas que precisam ser preenchidas. A abordagem do capítulo está dividida em três seções principais: consumo energético, gerenciamento de dados e arquiteturas. Com o intuito de atingir os objetivos do trabalho, os conceitos abordados incluem Sistemas de Gerência de Banco de Dados (SGBDs) NoSQL, Avaliação de Desempenho e *Benchmark*.

O Capítulo 3 apresenta o referencial teórico essencial para a compreensão deste trabalho. São abordados os conceitos relacionados aos Sistemas de Gerência de Banco de Dados (SGBDs) NoSQL , especificamente aqueles baseados em um único modelo de dados, como Redis e MongoDB, respectivamente, dos modelos Chave-Valor e Orientado a Documento. Além disso, são explorados SGBDs NoSQL multimodelo, incluindo ArangoDB e OrienteDB, os quais são comparados neste estudo, juntamente com suas categorias. O Capítulo também discute conceitos fundamentais sobre consumo de energia, assim como os principais métodos de avaliação utilizados neste contexto.

Já o Capítulo 4 apresenta a metodologia adotada, para o desenvolvimento da solução proposta. O capítulo descreve os métodos, ferramentas e configurações utilizadas neste trabalho. Inicia demonstrando a utilização da metodologia de DOE neste trabalho, mostra o fluxograma e explica o fluxo do método utilizado, segue explicando a arquitetura básica da execução dos experimentos, a configuração do *Yahoo! Cloud Serving Benchmark* (YCSB) utilizada, detalha o *framework* de medição. Também são denotados os métodos e técnicas estatísticas para a análise dos resultados obtidos nos experimentos realizados. Por fim, as ferramentas e o ambiente para medição e coleta de dados são apresentados.

O Capítulo 5 apresenta os resultados obtidos dos experimentos realizados baseados em

DOE. O resultado de cada tratamento é apresentado, realiza-se a comparação dos resultados pertinentes e a evolução no acréscimo da carga de trabalho com fundamentos estatísticos; em seguida, apresenta-se a correlação para as métricas tempo de execução e consumo de energia; por fim, executa-se uma discussão dos resultados.

Finalmente, o Capítulo 6 apresenta observações acerca dos resultados já obtidos, bem como as contribuições alcançadas (publicações).

2 TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos existentes na literatura que propõem soluções para Sistemas de Gerenciamento de Bancos de Dados (SGBDs) Multimodelos, enfocando questões relacionadas ao desempenho e consumo energético. Os trabalhos discutidos são organizados em seções, considerando os principais aspectos analisados para aprimorar os sistemas de armazenamento de dados. As categorias abordadas incluem consumo energético, gerenciamento de dados e arquiteturas. Finalmente, o capítulo é encerrado com uma comparação entre as soluções apresentadas e a abordagem proposta neste trabalho.

2.1 VISÃO GERAL

Um dos importantes desafios para a comunidade de pesquisadores de bancos de dados tem sido o desenvolvimento da tecnologia para manipular expressivos volumes de dados heterogêneos, gerados por aplicações e pessoas.

Atualmente os gerenciadores de bancos de dados relacionais apresentam dificuldades na escalabilidade e distribuição de dados para manipular aplicações que trabalham com *Big Data*. Para suprir essa demanda, uma nova classe de gerenciadores de bancos de dados que não trabalham com o modelo relacional surgiram, sendo classificados sobre o termo guarda-chuva NoSQL, onde duzentas implementações de bancos de dados NoSQL, implementando diferentes tipos de modelos de dados. A questão de escolha sobre qual destes gerenciadores adotar é uma decisão complexa para arquitetos de *software*, principalmente porque cada um trabalha com um modelo de dados, possuem linguagens de consulta que não são padronizadas e também possuem interfaces de acesso diferentes. Um fator de decisão frequente é o desempenho dos gerenciadores existentes para cada modelo e seus respectivos consumos energéticos.

É comum que uma aplicação necessite simultaneamente de mais de um modelo de dados. Como resultado, essas aplicações operam com diversos modelos de dados e vários gerenciadores de bancos de dados ao mesmo tempo. O termo "persistência poliglota"foi introduzido para descrever esses requisitos. O principal objetivo ao adotar a persistência poliglota é otimizar o desenvolvimento e gerenciamento de aplicações, separando os requisitos da aplicação ao acessar diferentes gerenciadores de bancos de dados, utilizando o modelo mais adequado para cada caso específico. Alguns dos gerenciadores de bancos de dados NoSQL surgiram como

soluções para aplicações específicas, cujos requisitos não eram atendidos pelos bancos de dados relacionais.

Nos últimos anos, alguns trabalhos avaliaram o desempenho de SGBDs NoSQL, e poucas pesquisas consideraram sistemas de banco de dados multimodelo. Além disso, o consumo de energia geralmente não é considerado.

2.2 AVALIAÇÃO DE DESEMPENHO

Vários estudos de *benchmark* existentes concentram-se predominantemente no desempenho de diferentes bancos de dados NoSQL. Nesse contexto, o YCSB é usado como o sistema de *benchmarking* de fato. No entanto, o YCSB não é adequado para testar cargas de trabalho de dados de vários modelos simultaneamente.

Uma grande quantidade de trabalhos se concentra em avaliar a eficiência de desempenho da mesma categoria de bancos de dados (bancos de dados que usam o mesmo modelo de dados). No trabalho de (OSEMWEGIE et al., 2018), os autores usaram a estrutura *Yahoo! Cloud Serving Benchmark (YCSB)* para avaliar bancos de dados NoSQL de chave-valor (Redis e SSDB). Ambos os bancos de dados receberam cargas de trabalho variáveis para identificar a taxa de transferência de todas as operações fornecidas. Os resultados obtidos mostram que o SSDB oferece um melhor *throughput* para a maioria das operações para o desempenho do Redis. Já (PEREIRA; MORAIS; FREITAS, 2018) conduziram um extenso estudo de desempenho comparando três bancos de dados de documentos NoSQL populares: MongoDB, Couchbase e RethinkDB. Os experimentos foram realizados em dois cenários distintos: *single thread* e *multiple threads*. Os resultados experimentais no estudo revelaram que o Couchbase teve melhor desempenho na maioria das operações, exceto coletar uma grande quantidade de documentos e inserir documentos com muitas *threads*, o MongoDB obteve uma pontuação melhor.

A pesquisa de (LISSANDRINI; BRUGNARA; VELEGRAKIS, 2018) fornecem um estudo de bancos de dados de gráficos existentes e também introduzem uma estrutura de micro-benchmarking para fornecer informações sobre o desempenho de bancos de dados de gráficos que vão além do que os *macro-benchmarks* podem fornecer. A estrutura de micro-benchmarking inclui o conjunto mais abrangente de consultas e operadores já considerados no momento da publicação do artigo. O estudo usou um grande conjunto de dados para avaliar bancos de dados de gráficos em dados sintéticos e reais de diferentes domínios. As bases de dados avaliadas foram: ArangoDB, BlazeGraph, Titan, OrientDB, Neo4j, Sparksee e Sqlg. O estudo conclui

que o uso de *micro-benchmarks* oferece benefícios para desenvolvedores e pesquisadores e pode ajudá-los a entender melhor o design, desempenho e opções de recursos de bancos de dados gráficos

Em (JAKKULA, 2020) os autores avaliam o desempenho de Cassandra e HBase em um ambiente virtual implantado na plataforma Openstack. A partir da avaliação de desempenho realizada sobre HBase e Cassandra usando YCSB, diversas variações no comportamento do banco de dados foram observadas conforme o requisito e apresentadas nas avaliações. O foco está no tempo de execução. Em geral, o Cassandra tem uma latência flutuante mais alta durante as operações de leitura e esse SGBD também tem um tempo de execução maior do que o HBase. O trabalho não avalia o consumo de energia.

No trabalho de (MARTINS et al., 2021), os autores, apresentam o conceito de NoSQL, descrevem os sistemas de banco de dados Cassandra e MongoDB, apresentando um estudo comparativo (prático) de ambos os sistemas realizando os testes em diversas cargas de trabalho. O estudo envolveu testar as operações – leitura e escrita, por meio de aumentos progressivos em números de clientes para realizar as operações, de forma de comparar as duas soluções em termos de desempenho. Os resultados preliminares mostram que Cassandra geralmente tem melhor desempenho em todos os tipos de operações. Por outro lado, o MongoDB é capaz de obter melhores resultados em um cenário onde os recursos de *hardware* são menores, com níveis de desempenho ligeiramente superiores ao Cassandra.

Outros estudos, comparam diretamente o desempenho de diferentes modelos de dados NoSQL. Em (TANG; FAN, 2016), é apresentada uma comparação de desempenho de cinco bancos de dados NoSQL diferentes (Redis, MongoDB, Couchbase, Cassandra, HBase), excluindo bancos de dados gráficos. Esse estudo definiu três tipos de cargas de trabalho e testou o tempo de execução e a taxa de transferência dos cinco bancos de dados. Embora o estudo tenha envolvido SGBDs de características de provedores específicos, os autores concluíram que os bancos de dados de documentos, seguidos pelos bancos de dados da família de colunas, têm um bom desempenho médio, uma vez que possuem eficiência e escalabilidade.

Da mesma forma em, (ABRAMOVA; BERNARDINO; FURTADO, 2014), os autores avaliaram o desempenho do SGBD NoSQL (MongoDB, Cassandra e Redis), onde foram avaliados variados aspectos como o número de nós, o número de unidades de processamento e as cargas de trabalho utilizando o YCSB e a escalabilidade de um banco de dados NoSQL com contagem fixa de operações e tamanho fixo de registros. Onde na avaliação o MongoDB e Redis apresenta um bom desempenho na operação de leitura enquanto Cassandra apresenta melhor

desempenho em atualizações.

Em (ROY-HUBARA; SHOVAL; STURM, 2022), os autores abordaram este problema com sua própria abordagem, onde desenvolveram um método para selecionar o tipo de banco de dados mais adequado para fragmentos de um modelo de dados, com base nos requisitos da aplicação. Como base, eles usam um modelo de dados conceitual a partir do qual criam múltiplos fragmentos agrupando elementos em relação à estrutura do próprio modelo de dados, às consultas esperadas e aos múltiplos requisitos não funcionais. O tipo de banco de dados mais adequado para cada fragmento é selecionado comparando as propriedades do fragmento com as propriedades dos diferentes tipos de banco de dados. Onde eles tem como principal objetivo auxiliar os arquitetos a identificar combinações apropriadas de tecnologias de banco de dados.

A literatura apresenta vários estudos realizados para comparar bancos de dados NoSQL multimodelos com diferentes representantes de suas variantes de modelo único. A grande maioria dessas comparações envolve OrientDB em comparação com MongoDB ou Neo4j, ou uma combinação deles.

(MACAK et al., 2020), compararam o desempenho do OrientDB com suas variantes de modelo único MongoDB (para dados de documentos) e Neo4j (para dados de gráficos). Neste estudo, o OrientDB é comparado com o MongoDB com base em uma carga de trabalho centrada em documentos e o OrientDB com o Neo4j puramente com base em dados gráficos. No entanto, cargas de trabalho combinadas não são investigadas. Os autores concluem que o OrientDB supera o Neo4j para consultas que envolvem nível de profundidade de nó mais alto. No entanto, quando se trata de consultas em que a profundidade do nó foi significativamente menor, o Neo4j demonstrou superar o banco de dados multimodelo. Por fim, os resultados do *benchmark* mostram que, para consultas de documentos, o MongoDB geralmente é mais rápido que o OrientDB.

Um estudo conduzido por (OLIVEIRA; CURA, 2016) comparou o desempenho de bancos de dados NoSQL multimodelo, como OrientDB e ArangoDB, com um aplicativo que usa uma configuração de persistência poliglota. Os autores realizaram a implantação de três configurações distintas: duas delas empregam um banco de dados NoSQL multimodelo, enquanto a terceira configuração combina dois bancos de dados NoSQL de modelo único. Os resultados experimentais revelam que o desempenho das três configurações está diretamente relacionado ao nível de profundidade necessário nas consultas de grafo. Por exemplo, a combinação do MongoDB com o Neo4j demonstrou o melhor desempenho em aplicações de persistência

poliglota que exigiam níveis mais profundos de travessia de gráficos em conjuntos de dados maiores. Entretanto, o OrientDB apresentou desempenho comparável e, por vezes, superior em determinadas consultas em conjuntos de dados menores. Por fim, o ArangoDB exibiu o melhor desempenho ao executar consultas com até dois níveis de profundidade de travessia do grafo.

No artigo (MARTINS; ABBASI; SÁ, 2019), os autores dedicaram os estudos nas comparações de desempenho entre tecnologias NoSQL, na execução das operações de gravação/inserção e atualização/leitura em diferentes cargas de trabalho de *benchmark* para banco de dados NoSQL, com especial atenção para bases de dados não relacionais orientadas para documentos como MongoDB, CouchDB, CosmoDB e OrientDB, focadas no tempo de resposta de consultas, bem como no tempo de carregamento de dados, consumo de memória e impacto nos processadores e clusters das máquinas. Para avaliar as bases de dados mencionadas, são utilizadas cargas de trabalho representadas pelo YCSB.

Os autores (GUNAWAN; RAHMATULLOH; DARMAWAN, 2019) realizaram um estudo, com o objetivo, de avaliar o tempo de resposta de populares bancos de dados NoSQL orientados a documentos, como MongoDB e CouchDB (documento) com ArangoDB (multimodelo). Eles consideraram o tempo de resposta de cada consulta para operações *Create, Read, Update, and Delete* (CRUD). Os resultados mostram que o MongoDB superou os outros bancos de dados selecionados para operações de leitura, atualização e exclusão. Tal trabalho utilizou o *benchmark* YCSB, no entanto, ArangoDB tem o menor tempo de resposta quando se trata da operação de escrita. Em comparação com o trabalho descrito acima. O estudo apresentado nesta dissertação não considera bancos de dados do tipo grafo, mas consideram os bancos de dados baseados em modelos de dados chave-valor.

Já (SEGHIER; KAZAR, 2021) forneceram um estudo comparativo entre três soluções amplamente empregadas: Cassandra, Redis e MongoDB, testando o tempo de execução das operações de leitura, atualização, varredura e leitura-modificação-gravação. Em resumo, seus resultados mostram que, com operações de leitura e gravação, o Redis supera outros bancos de dados devido ao uso de memória volátil para armazenar e recuperar dados. Por outro lado, o MongoDB executa com maior eficiência nas operações de leitura do que o Cassandra devido à técnica de mapeamento de registradores do MongoDB, que ajuda a aumentar o desempenho de leitura.

2.3 CONSUMO DE ENERGIA

Conforme o relatório Intel Labs (MINAS; ELLISON, 2009), em um servidor, a *Central Processing Unit* (CPU) é o principal consumidor de energia seguido pela memória, indicando que os processos que requerem o uso intenso de CPU e de acesso à memória, como os utilizados por SGBD, levam à elevação do consumo de energia.

A literatura apresenta escassez de trabalhos que avaliem o desempenho e o consumo de energia de Sistemas de Gerenciamento de Bancos de Dados (SGBDs) NoSQL. Nas avaliações encontradas, os estudos mais comuns concentram-se em comparações relativas a parâmetros como tempo de execução e latência, frequentemente utilizando máquinas com alta capacidade de memória e processamento, além de elevada carga de trabalho. Ainda mais desafiador é encontrar na literatura trabalhos relacionados à comparação do consumo de energia elétrica em sistemas de gerenciamento de banco de dados NoSQL multimodelo.

No artigo, os autores (GOMES; TAVARES; JUNIOR, 2016) realizaram avaliações de desempenho e consumo de energia de três SGBDs NoSQL: MongoDB, Cassandra (coluna) e Redis (chave-valor). O experimento adotou a ferramenta YCSB, sendo consideradas as operações de leitura, escrita e atualização, assumindo um intervalo de 1.000 a 100.000 operações. Os resultados indicam que não um único SGBDs que supere outras contrapartes para todos os *benchmarks*.

Um estudo (BARROS; CALLOU; GONçALVES, 2018) conduziram uma avaliação de desempenho e consumo de energia do Cassandra SGBDs em um ambiente distribuído. O *benchmark* YCSB é adotado e o número de servidores (nós) também é um fator nos experimentos. Os resultados indicam que o número ideal de nós (equilíbrio entre utilização de energia e tempo de resposta) está relacionado ao tamanho dos dados.

Em (MAHAJAN; BLAKENEY; ZONG, 2019) é apresentado análise do requisito de compensação entre otimização de energia e otimização de consulta para o banco de dados NoSQL. Eles demonstraram algumas abordagens de otimização de consulta bem conhecidas em bancos de dados NoSQL (Cassandra e MongoDB) para analisar o impacto da otimização de consulta na otimização de energia e otimização de desempenho. Para isso, eles avaliaram *Powerup*, *SpeedUp* e *GreenUp*. Se todas essas três condições forem satisfeitas todas às vezes, a otimização de energia vem com a otimização de desempenho. Por outro lado, se as condições *GreenUp* e *SpeedUp* não forem satisfeitas para todos os cenários sem degradar o *PowerUp*, a otimização de desempenho e as otimizações de energia parecem dois objetivos diferentes. Eles mostraram

que a otimização de energia não é um subproduto nem um objetivo conflitante da otimização de desempenho em algumas situações e concluíram que a eficiência energética, nem sempre é dimensionada linearmente com o desempenho. Portanto, mostra o escopo da pesquisa para analisar em detalhes os *trade-offs*.

Por fim, (SHAH; KOTHARI; PATEL, 2022) fizeram uma análise de consumo de energia do NoSQL, onde usaram o índice de carga da Unidade Central de Processamento (CPU) para estimar o consumo de energia de todo o servidor do *Data Center*. Os autores estimaram o índice de carga da CPU para computação intensiva, bem como disco rígido e memória intensiva. Os resultados da pesquisa indicaram que o consumo de energia da CPU foi de 58% durante todo o processo de carregamento. Também forneceram uma breve descrição de pequenos esforços para reduzir o consumo de energia do NoSQL.

A Tabela 1 resume o trabalho relacionado e mostra as características de nossa abordagem. Embora todos os trabalhos tratem de NoSQL e avaliação de desempenho, poucas pesquisas adotam SGBDs multimodelo e avaliação de consumo de energia elétrica conforme o tempo de execução em diferentes SGBDs Multimodelo. Esta dissertação compara SGBDs NoSQL de modelo único e multimodelo, considerando um *benchmark* representativo e o consumo de energia.

Também é apresentada a medição do consumo de energia elétrica com um hardware específico de medição, um *framework* e *scripts* foram desenvolvidos especificamente para a execução e análises destes casos de testes. No qual o mais importante é que este trabalho inicia uma nova perspectiva para entendimento das características que mais influenciam no consumo de energia em SGBDs NoSQL multimodelo.

Tabela 1 – Comparação entre este trabalho e trabalhos relacionados

	NoSQL	NoSQL Multimodelo	Avaliação de Desempenho	Consumo Energia
Este trabalho	✔	✔	✔	✔
(ABRAMOVA; BERNARDINO; FURTADO, 2014)	✔		✔	
(OLIVEIRA; CURA, 2016)	✔	✔	✔	
(TANG; FAN, 2016)	✔		✔	
(GOMES; TAVARES; JUNIOR, 2016)	✔		✔	✔
(OSEMWEGIE et al., 2018)	✔		✔	
(LISSANDRINI; BRUGNARA; VELEGRAKIS, 2018)	✔	✔	✔	
(BARROS; CALLOU; GONçALVES, 2018)	✔		✔	✔
(PEREIRA; MORAIS; FREITAS, 2018)	✔		✔	
(MAHAJAN; BLAKENEY; ZONG, 2019)	✔		✔	✔
(MARTINS; ABBASI; SÁ, 2019)	✔	✔	✔	
(GUNAWAN; RAHMATULLOH; DARMAWAN, 2019)	✔	✔		
(MACAK et al., 2020)	✔	✔	✔	
(JAKKULA, 2020)	✔		✔	
(MARTINS et al., 2021)	✔		✔	
(SEGHIER; KAZAR, 2021)	✔	✔	✔	
(SHAH; KOTHARI; PATEL, 2022)	✔			✔
(ROY-HUBARA; SHOVAL; STURM, 2022)	✔		✔	

3 REFERENCIAL TEÓRICO

Este capítulo apresenta os conceitos fundamentais dos temas abordados nesta dissertação, dividindo-se em duas partes. Na primeira parte, são expostos os fundamentos dos Sistemas de Gerenciamento de Bancos de Dados (SGBDs) NoSQL e da Persistência Poliglota, destacando suas principais características, categorias e os SGBDs específicos abordados neste estudo. Na segunda parte, são apresentados os conceitos básicos da avaliação de desempenho de sistemas computacionais, abordando o *Design of Experimentse* (DOE), além de fornecer uma breve descrição da análise de variância *Analysis of Variance* (ANOVA).

3.1 NOSQL SGBD E PERSISTÊNCIA POLIGLOTA

Esta seção apresenta os conceitos fundamentais relacionados aos gerenciadores de banco de dados adotados nesta dissertação. Além disso, explora os aspectos da arquitetura de ambas as tecnologias, abordando o funcionamento de seus componentes de armazenamento. Adicionalmente, destacam-se as particularidades de desempenho, sendo identificadas suas vantagens e desvantagens associadas à adoção de cada uma das tecnologias.

3.1.1 Banco de Dados NoSql

A crescente popularização da internet gerou a necessidade de gerenciar volumes de dados cada vez maiores, os quais se tornaram mais complexos, menos estruturados e mais dinâmicos. Esses fatores, aliados à grande quantidade de usuários realizando operações simultâneas de leitura e escrita, tornaram desafiadora a adoção dos tradicionais bancos de dados relacionais. Isso se deve à impossibilidade de adequar os dados a um modelo rígido e às restrições na escalabilidade necessária para lidar com a enorme quantidade de informações (IMRAN et al., 2021).

Dessa maneira, novas ferramentas foram criadas para lidar com estes desafios, dentre elas os bancos de dados NoSQL. O termo NoSQL, refere-se aos mais populares Sistemas de Gerenciamento de Banco de Dados (SGBDs) não relacionais, onde não há a necessidade de um modelo de dados fixo e da exclusiva utilização da linguagem SQL para consulta e manipulação dos dados (MEIER; KAUFMANN, 2019).

Por volta do ano 2000, surgiram diversos novos projetos de SGBDs alternativos ao modelo relacional. Dentre diversos que surgiram nesta época, pode-se destacar o *BigTable* da Google em 2004, o Dynamo da Amazon, o início do MongoDB e o lançamento do Neo4j em 2007 e o começo do Cassandra pelo Facebook em 2008 (WANG et al., 2022).

Segundo (SADALAGE; FOWLER, 2012), SGBDs NoSQL são uma classe de SGBD que não atendem ao modelo relacional, são chamados de "Não Somente SQL", enfatizando que geralmente a SQL não é utilizada como única linguagem de consulta, como em Banco de Dados relacionais. Suas principais características são métodos de acesso simples, escaláveis para grandes volumes de dados que não exigem que um esquema seja definido a priori (*schemaless*).

3.1.1.1 Características dos Bancos de Dados NoSQL

Atualmente, não há uma definição amplamente aceita para o que são os NoSQL, porém há algumas características que podem ser identificadas na maioria destes bancos. São elas de acordo com (SADALAGE; FOWLER, 2012) e (MEIER; KAUFMANN, 2019):

- Escalabilidade horizontal e vertical;
- Processamento distribuído;
- O modelo de dados não possui um esquema definido, permitindo maior flexibilidade quanto ao tipo dos dados armazenados;
- Grande parte dos NoSQL são *open source*;
- As transações possuem maior flexibilidade em relação às propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), sacrifício necessário para que se possa obter um processamento rápido proveniente da distribuição do sistema;
- Suporta o armazenamento de uma abundância de dados e permite operações simultâneas de leitura e escrita;
- Suporte a outras linguagens de consultas.

Além dessas características, os bancos de dados NoSQL, em sua maioria, são projetados com base no teorema CAP, que diz que, para um sistema de banco de dados distribuídos, só é possível garantir duas das três principais propriedades independentes. As três propriedades são de acordo com (MEIER; KAUFMANN, 2019):

- Consistência *(Consistency)*: Sempre que novos dados forem escritos no banco, todos os usuários que realizarem operação de leitura neste banco irão receber a versão mais recente dos dados;

- Disponibilidade *(Availability)*: Significa que os usuários sempre poderão esperar que cada operação realizada resulte em uma resposta esperada. Alta disponibilidade é alcançada através de um grande número de servidores físicos agindo como um único banco de dados através de compartilhamento e replicação de dados;

- Tolerância à partição *(Partition tolerance)*: Significa que a base de dados pode ser lida ou pode receber escrita mesmo que algumas partes dela estejam indisponíveis. Isso é alcançado salvando os dados que eram destinados a uma das partes indisponíveis em uma parte disponível para que, posteriormente, haja a sincronização entre as duas partes.

Constantemente, diversas operações são realizadas em um Sistema de Gerenciamento de Banco de Dados (SGBDs), podendo, inclusive, ocorrer concorrentemente. Isso demanda a necessidade de um mecanismo que garanta a ausência de inconsistências no banco de dados. No contexto relacional, os sistemas gerenciadores de bancos de dados asseguram a utilização das propriedades ACID para manter a consistência das informações. Contudo, é importante observar que nem sempre é imprescindível alcançar uma consistência total dos dados, como evidenciado no teorema CAP. Em certos casos, parte dessa consistência pode ser sacrificada para dar lugar a uma consistência eventual, proporcionando maior disponibilidade e escalabilidade horizontal do banco de dados. Visando esse cenário, o modelo transacional BASE foi proposto. BASE é uma abreviação para (POUYANFAR et al., 2018):

- Basicamente disponível *(Basically available)*: Todos os dados são distribuídos, mesmo quando há uma falha o sistema continua em funcionamento;

- Estado leve *(Soft-state)*: Não há garantia de consistência;

- Eventualmente consistente *(Eventual consistency)*: O sistema garante que caso não ocorram novas mudanças nos dados, eventualmente haverá consistência dos dados.

Pode-se dizer que a propriedade BASE é mais flexível que o ACID, fornecendo consistência eventual e alto particionamento. Em resumo, é interessante a adoção do modelo BASE quando a aplicação necessita de um sistema distribuído com vários clusters, devendo-se privilegiar a

capacidade de escalar o sistema e a performance sob a consistência dos dados (SADALAGE; FOWLER, 2012; MEIER; KAUFMANN, 2019; POUYANFAR et al., 2018).

3.1.1.2 Tipos e Categoria dos Bancos de Dados NoSQL

De acordo com a classificação em (DB-ENGINES, 2023), atualmente, existem mais de 225 bancos de dados NoSQL, categorizados conforme o modelo e a estrutura de dados que empregam. Dentre essas categorias, quatro se destacam: modelagens orientadas a grafo, documentos, chave-valor e colunas. As três últimas categorias são as mais populares e compartilham uma característica comum em seus modelos de dados: a orientação agregada. Nesse contexto, os dados são manipulados como unidades, apresentando estruturas mais complexas do que um simples conjunto de tuplas (MEIER; KAUFMANN, 2019). Para esta pesquisa de dissertação detalharemos apenas dois Tipos e Categoria Bancos de Dados NoSQL. Que são eles MongoDB (documento) e RedisDB (chave-valor).

BANCOS DE DADOS ORIENTADOS A DOCUMENTOS:

Os bancos de dados orientados a documentos armazenam e organizam os dados em coleções de documentos. Esses documentos podem assumir a forma de arquivos *eXtensible Markup Language* (XML), *JavaScript Object Notation* (JSON) ou qualquer outro formato, sendo eles estruturados, semiestruturados ou não estruturados. A característica fundamental desse modelo é a flexibilidade, uma vez que não exige a definição de um esquema rígido, permitindo a adição de novos documentos sem conhecimento prévio de sua estrutura.

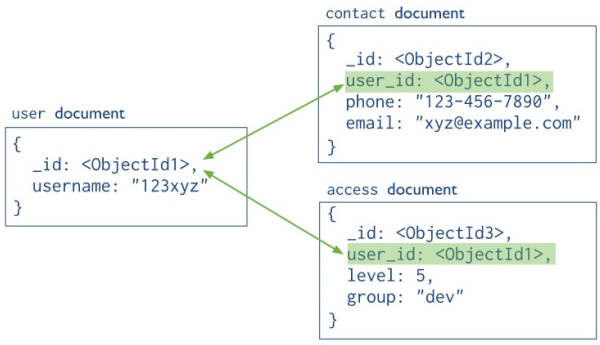
Essa flexibilidade torna o modelo orientado a documentos considerado o mais versátil entre os bancos de dados não relacionais. Além disso, a abordagem orientada a documentos permite a realização de consultas sobre qualquer valor ou chave presente no banco de dados. Isso difere dos bancos de dados orientados a chave-valor, nos quais as operações de busca estão limitadas às chaves (MEIER; KAUFMANN, 2019).

Traçando um paralelo com os bancos de dados relacionais, pode-se afirmar que uma coleção de documentos em um banco de dados orientado a documentos equivale a uma tabela, enquanto um documento nessa coleção corresponde a um registro da tabela. A relação entre documentos pode existir sem a necessidade de utilizar chaves estrangeiras e campos equivalentes, conforme modelo relacional. É crucial observar, no entanto, que, embora haja semelhanças

entre os dois modelos, os bancos de dados orientados a documentos não possuem a rigidez inerente ao modelo relacional. Isso permite que um documento apresente uma estrutura mais complexa do que um registro em uma tabela. Essa flexibilidade é uma das características distintivas dos bancos de dados orientados a documentos em comparação com seus equivalentes relacionais (MARTINS et al., 2021).

Quando armazenados, os documentos são codificados em um formato padrão de troca de dados, tal como XML ou JSON.

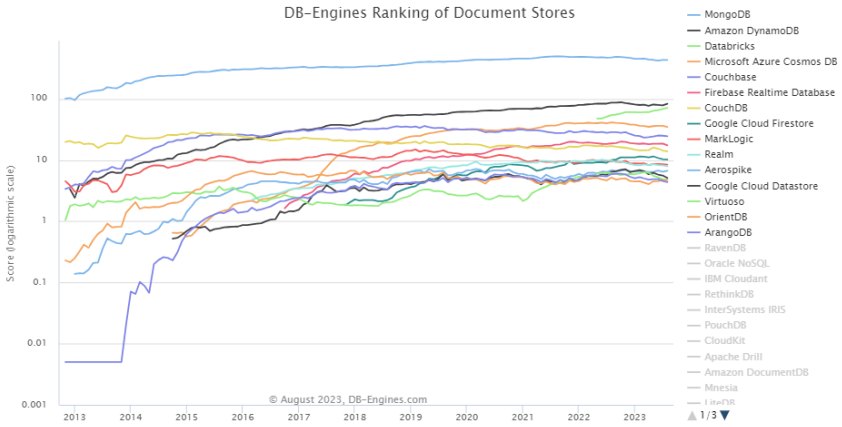
Figura 1 – Exemplo de objeto armazenado em um banco de dados orientado a objetos, utilizando JSON, adaptado de (MONGO-DB, 2022).



Fonte: MONGO-DB (2022)

De acordo com (DB-ENGINES, 2023), o banco de dados orientados a documento mais popular é atualmente o MongoDB. A Figura 2 mostra a popularidade dos principais bancos orientados a documento.

Figura 2 – Distribuição da popularidade dos bancos orientados a documentos.

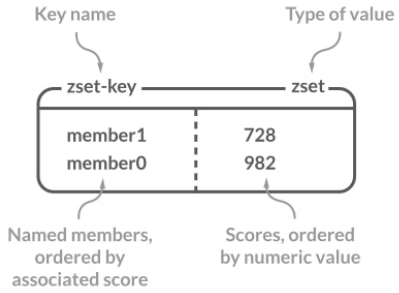


Fonte: DB-ENGINES (2023)

BANCOS DE DADOS ORIENTADOS A CHAVE-VALOR:

No modelo chave-valor, os registros são compostos por um identificador alfanumérico único, chamado de chave, que possui um valor associado em uma grande tabela, chamada de tabela *Hash*. Esses valores nada mais são que as informações contidas nos dados em um formato primitivo, como strings, inteiros e vetores, eliminando a necessidade de um modelo de dados rígido e fixo (MEIER; KAUFMANN, 2019). A Figura 3 exemplifica o armazenamento de um objeto em um banco de dados chave-valor.

Figura 3 – Armazenamento chave-valor em um banco de dados NoSQL, adaptado de (REDIS, 2022).



Fonte: REDIS (2022)

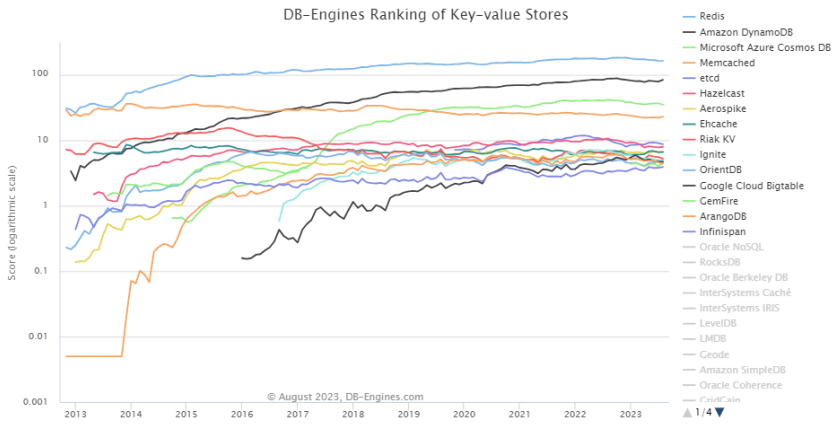
O modelo chave-valor utiliza uma tabela *hash* ou uma tabela com várias chaves, na qual cada chave referencia um valor. Um Sistema de Gerenciamento de Banco de Dados (SGBDs) que adota esse modelo de dados é o Redis. Este modelo é considerado o notavelmente simples, com a manipulação de dados realizada por meio de operações elementares de inserção, deleção, atualização e busca, utilizando o campo chave. Vale destacar que não há suporte para chaves secundárias, e os resultados estão limitados a combinações exatas (MEIER; KAUFMANN, 2019).

Devido a sua simplicidade, esta abordagem torna-se ideal para aplicações onde é necessária uma grande escalabilidade e velocidade na recuperação de valores no banco por uma tarefa, tal como o gerenciamento de perfis ou a recuperação de nomes de produtos.

Uma das vantagens do modelo chave-valor é a sua facilidade de implementação e de adição de novos dados em tempo de execução, uma vez que o valor do dado é visto como uma "caixa preta" pelo SGBD. No entanto, ele não suporta relacionamentos entre dados, definição de esquema e linguagem de consulta.

De acordo com (DB-ENGINES, 2023), o banco de dados orientados a chave-valor mais popular atualmente e o Redis. A Figura 4 mostra a popularidade dos principais bancos orientados a documento.

Figura 4 – Distribuição da popularidade dos bancos orientados a documentos.



Fonte: DB-ENGINES (2023)

3.1.2 Persistência Poliglota

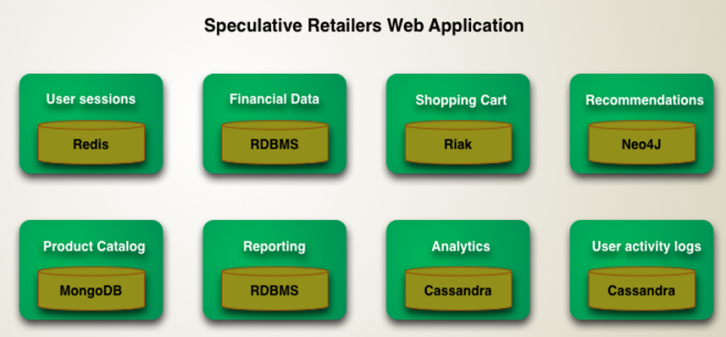
Desde o início da era da informática, o desenvolvimento de sistemas e o armazenamento de dados têm sido fundamentais. Com isso, surgiram inúmeras linguagens de programação e bancos de dados, cada um com o propósito de resolver diferentes problemas. No entanto, é comum nos depararmos atualmente com sistemas complexos e profissionais que optam por adotar e aplicar apenas uma linguagem. Essa abordagem visa otimizar o tempo de aprendizagem e adaptação dos profissionais, mesmo diante da diversidade de tecnologias disponíveis.

Porém (FORD, 2006) lançava um texto em seu blog, que mostrava o que vinha pela frente no mundo da programação, o aparecimento de diferentes dispositivos, novas tecnologias e problemas diferentes. Sugerindo que *"as aplicações do futuro aproveitarão a natureza poliglota do mundo das linguagens. Devemos abraçar essa ideia. Enquanto um fará algumas tarefas mais difíceis, outros executam as mais fáceis. É tudo questão de escolha da ferramenta certa para o trabalho e debugá-lo corretamente. Os testes ajudam com problemas de depuração. SQL, Ajax e XML são apenas o começo. Cada vez mais, começaremos a adicionar linguagens específicas do serviço. Os tempos de escrever um aplicativo em uma única linguagem de uso geral terminaram"* (FORD, 2006).

Em seguida, (SADALAGE; FOWLER, 2012) explica e exemplifica como a persistência poliglota em bancos de dados pode ser útil para grandes aplicações que desejam escalabilidade e performance, criando um mix de uso e justificando cada um deles. A ideia, de que as aplicações devem ser escritas em um mix de linguagens para aproveitar que diferentes linguagens são adequados para enfrentar diferentes problemas. Aplicações complexas combinam diferentes tipos

de problemas, então escolher o idioma certo para o trabalho pode ser mais produtivo do que tentar encaixar todos os aspectos em uma única linguagem (SADALAGE; FOWLER, 2012).

Figura 5 – Persistência poliglota em banco de dados.



Fonte: SADALAGE; FOWLER (2012)

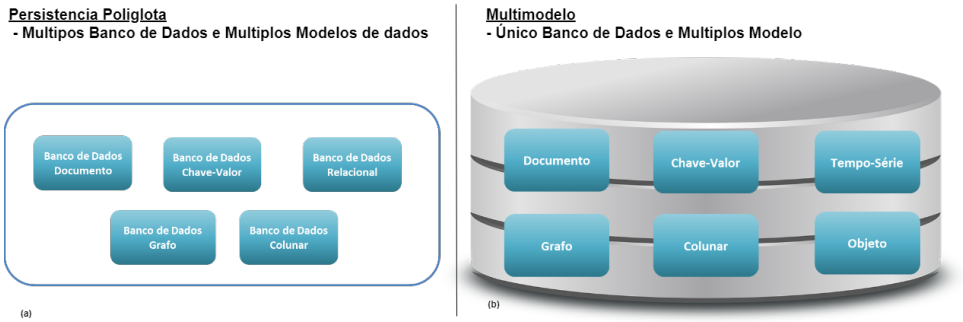
Esse fenômeno poliglota é aplicado mesmo dentro de uma única aplicação. Um aplicativo corporativo complexo utiliza diferentes tipos de dados e, frequentemente, integra informações de fontes diversas. Cada vez mais, observamos que esses aplicativos gerenciam seus próprios dados utilizando diferentes tecnologias, dependendo de como os dados são utilizados. Essa tendência complementa a prática de dividir o código da aplicação em componentes separados que se integram por meio de serviços web. Estabelecer limites claros entre esses componentes é uma abordagem eficaz para encapsular uma tecnologia específica de armazenamento, ditando como seus dados são manipulados (SADALAGE; FOWLER, 2012).

3.1.2.1 SGBDs NoSQL MultiModelo

Atualmente, diversas aplicações requerem simultaneamente o armazenamento de modelos de dados distintos, o que é usualmente referido como persistência poliglota. A Figura 6 descreve as abordagens comuns para implementar um armazenamento de dados poliglota. Um mecanismo (Figura 6 (a)) adota sistemas de gerenciamento de dados separados, de modo que cada SGBD implemente em um modelo de dados específico.

No entanto, a aplicação deve lidar com uma complexidade adicional, pois, por exemplo, cada SGBD pode ter diferentes interfaces de comunicação (por exemplo, linguagem de consulta). Por outro lado, (a Figura 6 (b), SGBDs multimodelos fornecem vários modelos de dados simultaneamente, em um único sistema de gerenciamento (persistência poliglota), permitindo grande flexibilidade. Além disso, os SGBDs multimodelos também evitam a duplicação de dados, e um usuário só precisa se preocupar com um fornecedor de SGBD.

Figura 6 – (a) Persistência poliglota e (b) SGBDs MultiModelo, adaptado de (ORACLE, 2022).



Fonte: ORACLE (2022)

SGBDs multimodelos comumente adotam uma camada de processamento de dados (SADALAGE; FOWLER, 2012), na qual é responsável por converter um modelo de dados em uma representação interna. Geralmente, um gerenciador de banco de dados multimodelo pode trabalhar com documentos, gráficos e chaves/valores oferecendo recursos de relacionamento entre modelos de dados. Uma das diferenças fundamentais entre os diferentes gerenciadores multimodelos são as diferentes estratégias para representar e implementar os grafos. A seguir, apresentaremos detalhes dos dois sistemas de gestão multimodelo que serão adotamos nesta dissertação. O OrientDB e o ArangoDB.

O OrientDB é um SGBD NoSQL multimodelo de código aberto, amplamente utilizado por mais de 500 empresas, entidades governamentais e startups que desenvolvem aplicações inovadoras. Ele une o modelo de grafo com os modelos de documento, chave-valor, orientados a objetos e geoespaciais em um único banco de dados operacional, escalável e de alto desempenho. O OrientDB apresenta duas edições: uma comunitária de código aberto e uma edição paga conhecida como *'Enterprise'*. A edição paga foi desenvolvida a partir da edição aberta e inclui recursos adicionais como backup e restauração contínuos, backups completos e incrementais agendados, gerenciador de consultas, configuração de *clustering* distribuído e gravação de métricas.

ORIENTDB:

Para suportar todos os modelos de dados, o OrientDB possui algumas características de como moldar esses dados. No modelo de documentos, os dados são guardados em documentos e suas relações entre documentos são feitas através de links. Esses links são utilizados para realizar junções. O Código 1 apresenta uma consulta com múltiplos *joins* utilizando links. Esses

links são especificados mediante uma notação de ponto (“.”) para navegar nos relacionamentos.

Código Fonte 1 – Exemplo de link do OrientDB, adaptado de (ORIENT-DB, 2022).

```
1    SELECT * FROM Employee WHERE City.coutry.name ='Brazil'
```

O modelo chave/valor utilizado pelo OrientDB é um pouco mais robusto que o modelo chave/valor clássico. Ele suporta elementos de documentos e grafos como valores que permitem um modelo mais rico. Já o modelo de objeto foi herdado pela programação Orientada a Objetos e suporta herança entre tipos (subtipos estendem os super tipos), esse modelo suporta também polimorfismo ao se referir a uma classe.

Uma das vantagens do OrientDB é que ele pode ser utilizado nas formas *schemaless*, *schema-full* e *schema* híbrido. O *schemaless* não obriga o usuário a definir uma estrutura pré-definida para os dados, ao contrário do *schema-full*. Já o schema híbrido é uma mistura dos dois anteriores, permitindo algumas partes dos dados tenham estrutura pré-definida e outras partes não.

O OrientDB oferece diversas interfaces para acesso aos dados. Uma delas é a linguagem SQL com extensões para manipulação de árvores e grafos, *Application Programming Interface* (API) Nativa e Gremlin. Em relação aos índices, o OrientDB suporta quatro tipos. Um deles é o *SB-Tree*, um índice padrão que é durável, transacional e suporta consultas de intervalo. Ele é baseado no índice *B-tree*, porém, adaptado com otimizações relacionadas à inserção de dados. Esse índice fornece um mix de recursos disponíveis em outros tipos de índices para uso geral, sendo eles: *Unique*, *NotUnique*, *FullText* e *Dictionary*.

O OrientDB também suporta o índice *Hash* que proporciona pesquisas rápidas, duráveis e transacionais, embora não ofereça suporte a consultas de intervalo. Esse índice opera semelhante a um *HashMap*, o que resulta em buscas pontuais mais rápidas e menor consumo de recursos em comparação com outros tipos de índices. Outro recurso suportado pelo OrientDB é o índice *Auto Sharing*, que representa uma implementação de uma Tabela de *Distributed Hash Table* (DHT), na qual as chaves são armazenadas em uma partição separada. Esse índice é durável e transacional, mas também não oferece suporte a consultas de intervalo.

Outra vantagem do OrientDB é a capacidade de garantir as propriedades *Atomicity, Consistency, Isolation, and Durability* (ACID) mesmo em modo distribuído. Em relação a sua estratégia de armazenamento, o OrientDB consegue armazenar os dados em disco ou em memória. No armazenamento em memória, todos os dados ficam em memória virtual Java. Nesta opção de armazenamento, o ‘D’ do ACID não é garantido, já o armazenamento em

disco garante todas as propriedades ACID .

O controle de transações é feito pela abordagem *Multi-Version Concurrency Control* (MVCC), ou seja, Controle de Concorrência de Múltiplas Versões. Nesta estratégia, as operações geram uma nova versão dos dados que fica disponível assim que a transição terminar com sucesso, porém enquanto a transação não termina a última versão do dado é disponível somente para leitura. (TAN et al., 2019).

ARANGODB:

ArangoDB é um SGBD de código aberto escrito em C++ é baseada em documentos (ARANGO-DB, 2022). É um SGBD nativo multimodelo, sendo concebido para armazenar seus dados como pares, chave/valor, grafos ou documentos, e acessar qualquer representação de dados usando uma única linguagem de consulta declarativa. Um dos benefícios do ArangoDB é utilizar um único *backend* para gerenciar seus diferentes modelos de dados. Além disso, ele possui suporte a transações ACID, oferecendo forte consistência em uma única instância e operações atômicas ao operar no modo de cluster.

O SGBD é classificado como *schemaless*, não obrigando o usuário a nenhuma estrutura pré-definida. Uma de suas características distintas é possuir uma linguagem unificada própria chamada *Acceptable Quality Level* (AQL), que permite consultar dados em diferentes modelos suportados pelo ArangoDB.É inspirada na linguagem SQL, no entanto, a AQL não suporta operações de definição de dados, como criar e excluir bancos de dados, coleções e índices. As declarações são enviadas via *Hypertext Transfer Protocol* (HTTP) a partir da aplicação ArangoDB, sendo retornada uma resposta no formato JSON. O Código 2 apresenta um exemplo de inserção de um documento em uma coleção utilizando a linguagem AQL. As aspas em torno das chaves de atributo são opcionais.

Código Fonte 2 – Exemplo de *insert* de documentos em AQL, adaptado de (ARANGO-DB, 2022).

```
1   INSERT {
        name : "Ned",
3       surname : "Stark",
        alive : TRUE,
5       age : 40
    } INTO Characters
```

No ArangoDB o armazenamento dos dados de grafo é feito no modelo de documentos, com documentos ligando outros documentos por meio de referências entre as coleções de documen-

tos. Esses documentos especiais que armazenam as ligações entre os documentos vértices são chamados de documentos *Edge* (Arestas). Os *Edges* possuem atributos denominados *_from* e *_to* com os identificadores dos vértices. Os algoritmos de grafos foram adaptados para usar essas estruturas, realizando acesso a índices apropriados para acelerar o busca nos documentos (WEINBERGER, 2022).

Código Fonte 3 – Exemplo de consulta em grafo no ArangoDB, adaptado de (ARANGO-DB, 2022).

```
FOR c IN Characters
    FILTER c.mane == "Ned"
    FOR v IN 1..1 IMBOUND c Childof
        RETURN v.name
```

O Código 3 apresenta uma consulta utilizando AQL para buscar os filhos (relações) de "Ned" no qual os dados foram modelados em grafos. A expressão *INBOUND* é utilizada para buscar as relações em direção do nó pai para os nós filhos, sendo "1..1" o número de passos (relações) entre um nó e outro. Caso seja necessário a busca dos nós filhos para os pais, a expressão a ser utilizada é *OUTBOUND*.

Embora a linguagem AQL seja a principal escolha para consultas, ela não é a única forma de acessar os dados. O ArangoDB possui outras duas formas, sendo elas uma interface *Representational State Transfer* (REST) que fornece medidas simples para criação, acesso e manipulação dos dados usando o identificador do documento ou consulta. A outra forma de acesso é o uso do JavaScript, que é incorporado ao ArangoDB permitindo que todas as partes da API sejam diretamente acessíveis a partir do JavaScript, sendo isso um grande diferencial. O JavaScript *Driver* permite a criação, manipulação e consulta de dados. Na Figura 4 é possível visualizar as operações de *insert* e *update* utilizando o Javascript. Já o Código 5 apresenta uma consulta que retorna usuários maiores de 12 anos utilizando a linguagem AQL em Javascript.

Código Fonte 4 – Exemplo das operações *insert* e *update* utilizando Javascript, adaptado de (ORIENT-DB, 2022).

```
const db = new Databese();
const collection = db.collection("some-collection");
const doc = { number: 1, hello: "word" };
const doc1 = await collection.save(doc);
const doc2 = await collection.update(doc1, { number:2 });
```

Código Fonte 5 – Exemplo de consulta em Javascript, adaptado de (ORIENT-DB, 2022).

```
const db = new Databese();
const action = String (function(params){
```

```
3         const db require("@arangodb").db;
          return db
5             ._query(
                  aql`
7                 FOR user IN _users
                  FILTER user.age > ${params.age}
9                 RETURN undefined.user
                  `
11            )
              .toArray();
13    });
      const result = await db.transaction({ read: "_users" }, action, { age:12 })
```

O ArangoDB apresenta uma ampla variedade de índices, incluindo *hash index*, *skiplist index*, *fulltext index* e *geo index*. O *hash index* é adequado para localizar rapidamente documentos com valores de atributos específicos, sendo indicado para pesquisas por igualdade, mas não para pesquisas por intervalo de valores, uma vez que não mantém os dados ordenados. O *skiplist index* é uma estrutura de dados ordenada, recomendada para pesquisas por intervalo e conjuntos de dados sujeitos a muitas atualizações. Ele proporciona eficiência sem a necessidade de balanceamento de estruturas de árvore, como exigido pelos índices B-tree ou AVL. O *fulltext index* pode ser empregado para encontrar palavras ou prefixos em documentos. Por fim, o *geo index* é utilizado para localizar rapidamente lugares específicos na superfície da Terra. Ele armazena coordenadas bidimensionais, sendo indicado para operações de localização de documentos com coordenadas próximas a uma determinada coordenada de comparação, bem como documentos com coordenadas dentro de um raio "X"em torno de uma coordenada de comparação.

Por padrão, a estrutura de dados disponível no ArangoDB é o JSON, e ele possui 2 estratégias de armazenamento: MMFiles e RocksDB. O MMFiles persiste os dados em arquivos na memória e, para garantir consistência quando uma operação de alteração nos dados é invocada, primeiro ela é escrita em um log e depois aplicada aos dados, garantindo uma recuperação após alguma falha. Outra característica importante é os arquivos serem sempre incrementados, ou seja, quando um documento sofre alguma alteração, o conteúdo é copiado para um novo arquivo e inserido no fim dos arquivos de dados. O arquivo antigo é marcado como lixo e é posteriormente descartado. Quando o volume de dados é maior que a memória, o método de armazenamento RocksDB é considerado uma opção melhor. Ele mantém um conjunto atualizado de dados na memória principal, mas carrega dados adicionais do disco, caso o dado não esteja no conjunto armazenado em *cache*. O método também grava todos

os valores de índice no disco. Portanto, não há necessidade de reconstruir índices ao acessar pela primeira vez uma coleção após uma reinicialização (ARANGO-DB, 2022). Isso torna a inicialização do BD muito rápida. Após a inicialização, o RocksDB preenche gradualmente seus *caches* durante as operações, aumentando o seu desempenho com o tempo.

3.2 AVALIACAO DE DESEMPENHO

A avaliação de desempenho é um critério fundamental que permite tratar problemas comumente encontrados em sistemas computacionais, tais quais identificação de gargalos, caracterização de cargas de trabalho, previsão de desempenho e planejamento de capacidade. O objetivo dos provedores de sistemas de informação é obter o melhor desempenho para determinado custo. Para atingir esse objetivo, estes profissionais necessitam, ao menos, conhecer a terminologia de avaliação de desempenho e suas técnicas devem ser capazes de indicar as exigências de seus sistemas de desempenho e comparar diferentes alternativas para encontrar aquela que melhor atenda às suas necessidades (BUKH, 1992).

As principais técnicas para avaliação de desempenho de sistemas computacionais são: medição de sistemas reais, modelagem analítica e simulação. A escolha da técnica correta depende do tempo e recursos disponíveis para solução do problema, além disso, deve ser levado em consideração o nível de detalhes desejados nos resultados.

Para a avaliação de desempenho, um conjunto de métricas deve ser escolhido. Uma maneira de preparar este conjunto é listar os serviços oferecidos pelo sistema. Para cada solicitação de serviço feita ao sistema, há vários resultados possíveis. Por exemplo, um banco de dados oferece o serviço de responder às consultas. Quando executada uma consulta, ele pode responder corretamente, ele pode responder incorretamente ou não responder (GOMES; TAVARES; JUNIOR, 2016). Se o sistema executa o serviço corretamente, o seu desempenho é medido pelo tempo necessário para realizar o serviço, a taxa na qual o serviço é executado, e os recursos consumidos durante a execução do serviço. (BUKH, 1992) atribui a estas três métricas o rótulo de capacidade de resposta. A maioria dos sistemas oferece mais de um serviço, portanto, o número de métricas cresce proporcionalmente. Para muitas métricas, o valor médio é o mais importante. No entanto, o efeito da variabilidade deve ser considerado, ao poder degradar significativamente a produtividade.

(LILJA, 2000) apresenta vários objetivos típicos para avaliação de desempenho:

- Comparar alternativas: Na escolha de determinado sistema podem existir várias alternativas. Além disso, em cada sistema podem existir diversas possibilidades de configuração que, de maneira geral, podem afetar tanto o custo quanto o desempenho da solução desejada. Comparar alternativas pode fornecer informações quantitativas a respeito de quais configurações são melhores em condições específicas;

- Determinar o impacto de uma característica: Na concepção ou alteração de sistemas, muitas vezes é necessário conhecer o impacto dessa nova característica adicionada;

- Ajuste do sistema: Neste caso, utilizamos para encontrar a combinação de parâmetros ideais do sistema para a obtenção de sua melhor performance na métrica desejada;

- Identificar o desempenho relativo: O desempenho de um sistema de computador, geralmente, apenas tem significado no contexto da comparação a algum outro sistema ou configuração de outro sistema. Outro objetivo pode ser quantificar o desempenho em relação a uma expectativa; e

- Definir expectativas: Através da análise de desempenho, podem ser inferidas as capacidades de uma próxima linha de sistemas de computadores;

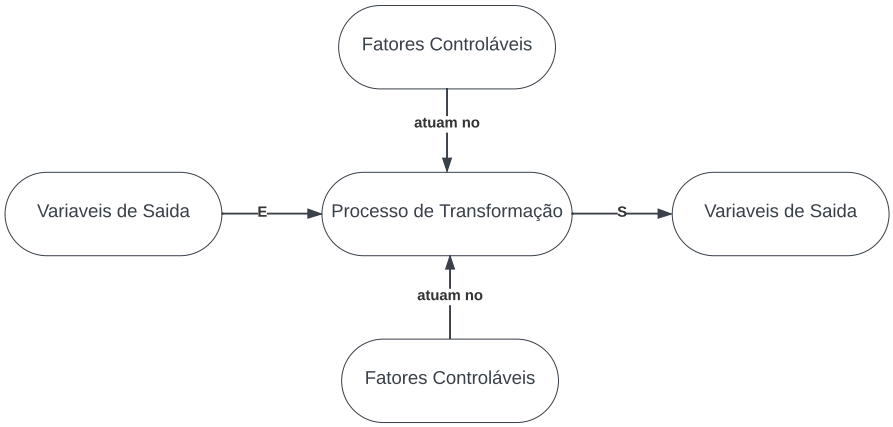
De acordo com (BUKH, 1992), a medição de desempenho consiste essencialmente na monitoração do sistema enquanto está sob ação de uma carga de trabalho. Para adquirir resultados representativos, a carga de trabalho deve ser cuidadosamente selecionada. Essa carga é utilizada nos estudos de desempenho, pode ser real ou sintética. Embora a carga de trabalho real seja uma boa escolha por representar fielmente o sistema, ocasionalmente esta opção não é a desejável. Isso acontece quando o tamanho da carga não é considerável, e também quando esses dados recebem muitas perturbações ou, até mesmo, por questões de facilidade de acesso destes. Devido a esses motivos, as cargas de trabalho sintéticas têm sido utilizadas como *benchmark* e programas.

3.2.1 *Design of Experiments* (DoE)

Projeto de experimentos, ou DOE (MONTGOMERY; RUNGER, 2020), é uma técnica utilizada para analisar o efeito da variação simultânea de variáveis (fatores) visando obter o máximo de informação com o menor número de experimentos. O projeto de experimentos se

inicia determinando os objetivos dos mesmos e selecionando os fatores que serão objeto de estudo. Um projeto de experimentos bem definido objetiva maximizar a quantidade de informação que se pode obter com uma quantidade determinada de esforço ou minimizar o esforço para uma quantidade de informação desejada. Um experimento é composto por uma ou mais saídas, um conjunto de entrada com fatores controláveis e outro conjunto com fatores que não se pode, ou não se deseja, controlar (Figura 7).

Figura 7 – Modelo básico de um processo ou sistema, adaptado de (MONTGOMERY; RUNGER, 2020).



Fonte: MONTGOMERY; RUNGER (2020)

O projeto de experimentos pode ser utilizado para atingir diversos objetivos, entre eles:

- *Screening Experiments* – Identificar, entre aos fatores candidatos, os que mais influenciam a saída;
- *Comparative experiments* – Concluir a respeito de um importante fator;
- *Modelagem de Superfície de Resposta* para:

  Maximizar ou minimizar a saída;

  Reduzir variação de saída, identificando regiões onde um processo é mais fácil de ser gerenciado;

  Tornar um processo robusto e;

- *Regression Modeling* – Estimar um modelo preciso, quantificando a dependência da saída em função das entradas, particularmente útil para sistemas com poucos fatores

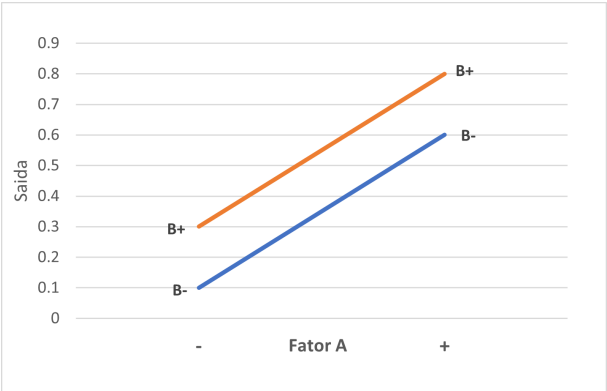
Para entender o projeto de experimentos, algumas definições básicas devem ser descritas, como apresentadas a seguir.

- *Fator* - Um fator é uma variável, controlada ou não, que exerce influência sobre a saída do sistema.

- *Nível* - Os níveis de um fator são os valores deste fator que estão sendo examinados. Como exemplo, seja um experimento onde se deseja observar a qualidade de um produto submetido a quatro diferentes condições de temperatura durante sua produção. Neste caso, as quatro condições são os níveis deste fator.

- *Efeito principal* - O efeito principal de um fator é aquele associado à sensibilidade do sistema a um fator isoladamente. Na Figura 8 têm-se dois fatores (A e B) com dois níveis cada, simbolizados pelos sinais "-" e "+". Assim, "A-" significa o fator A experimentado no nível "-", significando, por exemplo, uma temperatura baixa. Pode-se observar na figura o efeito da variação de nível de cada fator sobre a saída. Percebe-se que os fatores são independentes. Em outras palavras, a mudança do nível de um fator influencia na saída de forma independente dos demais. Ainda na Figura 8, pode-se observar que quando o fator A muda do nível "-" para "+", a variação na saída é sempre de 0,5, independente do fator B estar no nível "-" (0,1 – 0,6) ou "+" (0,3 – 0,8). A mesma análise pode ser realizada para o fator B.

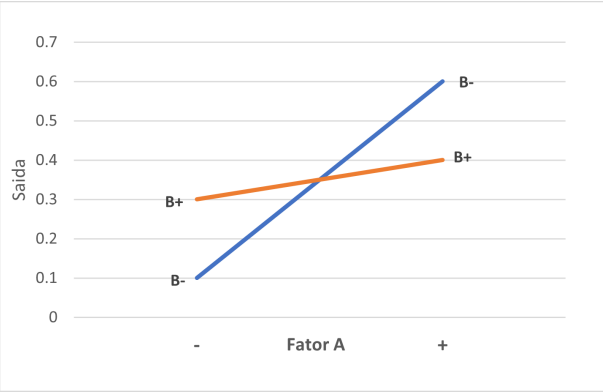
Figura 8 – Efeito principal dos fatores A e B.



- *Efeito de interação* - O efeito de interação é observado quando um fator, em conjunto com outro, influência na saída do sistema (Figura 3). Neste caso, fazendo uma análise semelhante à realizada anteriormente, pode-se observar que quando o fator A muda do nível "-" para "+", a variação na saída é de 0,5 quando o fator B está no nível "-" (0,1

– 0,6); já quando o fator B está no nível "+", a variação é de apenas 0,1 (0,3 – 0,4), indicando haver efeito resultante da interação entre os fatores A e B.

Figura 9 – Efeito de interação dos fatores A e B.



3.2.1.1 *Projeto de Experimentos Fatoriais Completos*

Em um projeto de experimentos fatoriais completos, todas as combinações dos fatores são analisadas. A quantidade de ensaios necessária para que se possa realizar todas as combinações de níveis dos fatores se dá pela equação:

$$q = \prod_{i=1}^{k} n_i$$

onde: q é a quantidade de ensaios ou execuções, k é a quantidade de fatores e n é a quantidade de níveis de cada fator.

De fato, um projeto de experimentos fatoriais completos é considerado viável quando apenas alguns fatores precisam ser investigados. No entanto, torna-se impraticável quando há muitos fatores a serem considerados. Isso ocorre porque esse tipo de experimento considera todas as combinações possíveis dos fatores de controle. Dessa maneira, ao realizar um projeto de experimentos fatoriais completos, é possível estimar tanto os efeitos principais quanto aqueles provenientes das interações entre os fatores. Na Tabela 2 tem-se um projeto de experimentos fatoriais completos com dois fatores (A e B) e dois níveis cada, simbolizados pelos sinais de "+" e "-", e sua interação. Os sinais da coluna de interação são obtidos multiplicando os sinais das colunas dos respectivos efeitos principais. Os sinais referentes aos níveis dos fatores poderiam simbolizar alto e baixo ou "operador João" e "operador José".

Tabela 2 – Projeto de experimentos fatoriais completo com dois fatores com dois níveis e sua interação.

Ensaio	Fator (A)	Fator (B)	Interação (AB)
1	-	-	+
2	-	+	-
3	+	-	-
4	+	+	+

3.2.1.2 Projeto de Experimentos Fatoriais Fracionado

Diferentemente do caso anterior, em um projeto de experimentos fatoriais fracionados, apenas uma fração adequadamente escolhida das combinações necessárias de um projeto de experimentos fatoriais completos é selecionada para ser executada. A necessidade de um projeto de experimentos fatoriais fracionado surge principalmente devido ao aumento na quantidade de ensaios (execuções) necessários em um projeto de experimentos fatoriais completos quando novos fatores são adicionados. Para ilustrar, considerando apenas seis fatores com dois níveis cada, seriam necessários 64 experimentos. Se o número de fatores aumentar para oito, serão necessários 256 experimentos. Dessa forma, um projeto fracionado permite explorar uma porção representativa das combinações sem a necessidade de realizar todos os ensaios, reduzindo significativamente o número total de experimentos a serem conduzidos.

Diante do, normalmente alto, custo de experimentação e de observações onde se têm muitos fatores controlados, uma estratégia que reduza a quantidade de experimentos se faz necessária. A solução se dá pelo uso de uma fração dos experimentos determinados pelo projeto de experimentos fatoriais completos. O interesse é determinar quais ensaios devem ser realizados e quais não devem. Existem diversas estratégias para garantir uma escolha apropriada dos ensaios a realizar. Entendem-se como apropriada, neste caso, a garantia das propriedades de balanceamento (caso em que a combinação dos níveis tem a mesma quantidade de observações) e ortogonalidade entre fatores (MONTGOMERY; RUNGER, 2020).

A técnica utilizada para se obter essa fração é baseada no confundimento (*confounding* ou *alises*) e se constitui em uma forma de arranjar as combinações de um ensaio fatorial completo em blocos com pequenos números de combinações, aproveitando-se das interações não significativas para tal fim. Neste caso, substituem-se as interações entre fatores considerados não influentes por outros fatores. O custo associado ao uso das interações de alta ordem é que se impossibilita distinguir o efeito resultante destas interações e o efeito resultante do fator que está sendo substituído (confundindo). De fato, o efeito será resultante da soma dos efeitos

Tabela 3 – Projeto de experimentos fatoriais fracionado com 3 fatores e 2 níveis cada.

Ensaio	Fator (A)	Fator (B)	Fator (C)
1	-	-	$+$
2	-	$+$	-
3	$+$	-	-
4	$+$	$+$	$+$

da interação e do fator confundido. Isso não se torna um problema, desde que seja possível assumir a hipótese de que as interações de alta ordem são insignificantes e podem ser negligenciadas. Essa técnica é uma estratégia eficaz para reduzir a quantidade total de experimentos necessários, mantendo uma representação válida das combinações de fatores. Tomando como referência a Tabela 2, na Tabela 3 pode-se observar a substituição da interação entre os fatores A e B pelo fator C.

Um projeto de experimentos fatoriais fracionados do tipo 2^{k-p} contém 2^{k-p} experimentos (ou combinações) e são chamados como $\frac{1}{2^2}$ frações de 2^k , ou simplesmente 2^{k-p} delineamento fatorial fracionário.

O principal uso de experimentos fatoriais fracionados é na identificação dos fatores influentes *(screening expriments)* (MONTGOMERY; RUNGER, 2020). O grau em que os efeitos se associam é denominado resolução. Um projeto de experimentos é de uma resolução *R* se nenhum *"p"* efeito de fator é confundido com outro efeito menor que *"R-p"* fatores. Os mais utilizados são os de resolução III, IV e V:

Resolução III: Neste tipo de projeto, nenhum efeito principal se confunde com outro efeito principal, porém podem se confundir com efeitos de interações de dois fatores (primeira ordem);

Resolução IV: Neste tipo de projeto, nenhum efeito principal se confunde com outro efeito principal ou com interações de primeira ordem, porém interações de primeira ordem podem se confundir entre si;

Resolução V: Não há confundimento entre efeitos principais ou o efeito de interação de dois fatores é confundido com efeitos principais, ou de interações de dois fatores. Apenas efeitos de interação de dois fatores se confundem com efeitos de interação de três fatores.

Desta forma, sempre que se desejar diferenciar os efeitos resultantes dos fatores principais e os efeitos resultantes da interação entre fatores, se desejará, por consequência, a maior resolução que seja possível para o grau de fracionamento necessário. Porém, no limite, a técnica apresentada permite que k fatores possam ter seus efeitos analisados com apenas *k*

+ 1 observações (MONTGOMERY; RUNGER, 2020), lidando com a perda de possibilidade da distinção entre os efeitos determinados pelos fatores principais e os oriundos de interação de fatores.

3.3 *BENCHMARK*

O *benchmarking* é amplamente utilizado para avaliação de sistemas computacionais e existe para uma grande variedade de níveis de abstração. O emergente número de sistemas, junto à falta de padrões de comparação de desempenho, dificulta o entendimento das relações entre os sistemas e sua carga de trabalho.

(BORAL; DEWITT, 1984) Explica-se que o *benchmark* pode ser aplicado para comparar com precisão a eficiência de diferentes produtos, uma vez que é padronizado e seus testes são invariáveis e bem definidos. Onde um *benchmark* deve conter as seguintes características: ser relevante para a aplicação-alvo a qual representa; ser portátil entre diferentes arquiteturas; ser escalável, podendo ser executado em diferentes sistemas computacionais e ser, dentro do possível, simples de entender para manter a sua credibilidade.

3.3.0.1 Yahoo! Cloud Serving Benchmark (YCSB)

O YCSB fornece um conjunto padronizado de cargas de trabalho e métricas de desempenho, o que facilita a comparação de diferentes sistemas de armazenamento de dados. Isso permite aos desenvolvedores e pesquisadores avaliar e comparar o desempenho de diferentes sistemas de forma consistente e objetiva. Trata-se de uma ferramenta valiosa para desenvolvedores, engenheiros e pesquisadores que desejam avaliar e comparar o desempenho de sistemas de armazenamento de dados em diferentes contextos e cenários de uso.

O *Benchmark* foi desenvolvido pela empresa Yahoo com o propósito de testar vários gerenciadores de bancos de dados NoSQL. É um projeto open source e extensível. Esse framework é um dos *benchmarks* mais utilizados para comparação de SGBDs não relacionais (YCSB, 2022) e o YCSB pode ser utilizado para testar novos SGBDs apenas implementando uma nova classe que executa os métodos: *read, insert, update, delete* e *scan*. Além disso, este framework, possui implementações para diversos outros SGBDs, apresentando código open *source* e possibilidade de parametrização dos testes.

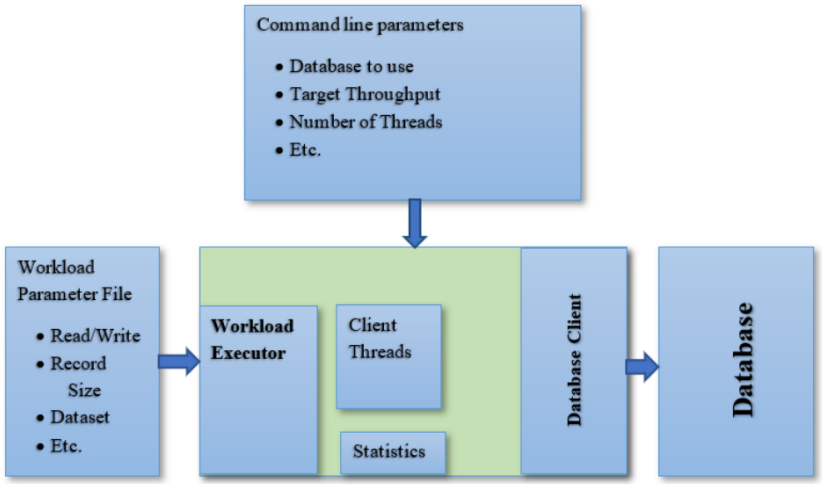
Nesta ferramenta, é possível definir o número de *threads* que serão executadas visando

aumentar a carga computacional de um sitemap de gerenciamento para medir o tempo de resposta das requisições. Além disso, a ferramenta permite adicionar novos SGBDs, implementando uma interface específica com métodos de leitura, busca, inserção e alteração.

No entanto, vale notar que a ferramenta pode apresentar limitações quando se trata da comparação de aplicações de persistência poliglota, uma vez que ela só permite testar um gerenciador de banco de dados por vez. Para avaliações abrangentes que envolvam múltiplos SGBDs simultaneamente, pode ser necessário considerar outras ferramentas ou abordagens que suportem testes comparativos entre sistemas de persistência poliglota.

Conforme ilustrado na Figura 10, a estrutura consiste em um cliente gerador de carga de trabalho que cobre aspectos como trabalhos de leituras/escritas. Para desenvolver o núcleo do pacote do YCSB, foi examinada uma variedade de sistemas e aplicações com o fim de identificar os tipos fundamentais de carga de trabalho. Alguns sistemas podem ser altamente otimizados para leitura, mas não para escrita, outros podem ser para escrita, mas não para atualizações. Essas cargas são uma variação do mesmo tipo básico de aplicativo, nesta aplicação existe uma tabela básica de registros, com determinados campos, cada registro é identificado por uma chave primaria. Os valores de cada campo são uma sequência aleatória de caracteres ASCII de comprimento determinado (YCSB, 2022; ARAÚJO, 2016).

Figura 10 – Arquitetura YCSB, adaptado de (YCSB, 2022).



Fonte: YCSB (2022)

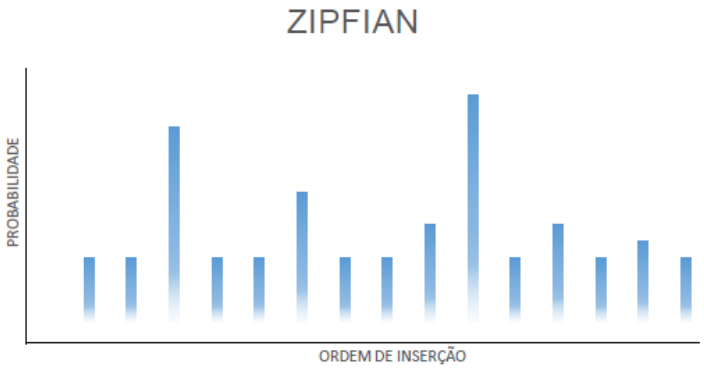
3.3.0.2 Distribuições

O gerador de carga de trabalho realiza muitas escolhas aleatórias, ao gerar os dados a serem carregados no *benchmark* (escrita, atualização, leitura). Essas decisões são regidas por distribuições que utilizam diferentes maneiras de criar esta aleatoriedade dada uma probabilidade do dado.

Onde foi utilizada a distribuição *Zipfian* a estatística para descrever a frequência de ocorrência de elementos em um conjunto de dados, onde poucos elementos são muito comuns e a maioria dos elementos é rara. Ela segue a chamada "lei de Zipf". A distribuição *Zipfian* é comumente usada para modelar a distribuição de frequência popularidade, ou fenômenos onde alguns elementos são muito mais frequentes do que outros. A principal característica de uma distribuição *Zipfian* é a relação inversa entre a frequência de ocorrência de um elemento e a sua posição no ranking de frequência. O elemento mais comum é geralmente designado como o primeiro na lista, o segundo mais comum é o segundo na lista, e assim por diante.

Onde e possível simular um comportamento real das características de acesso a banco de dados. Por exemplo, ao escolher um registro, alguns serão mais populares enquanto a maioria dos registros será impopular.

Figura 11 – Distribuição zipfian YCSB.



Tendo em vista que, em se tratando de probabilidade, os resultados podem variar, a Figura 11 apenas exemplifica visualmente a distribuição *zipfian*. Os eixos horizontais na figura representam os itens que podem ser escolhidos na ordem de inserção, enquanto as barras verticais representam a probabilidade de o conjunto de caráter ser escolhido.

3.3.0.3 Análise de Variância (ANOVA)

A ANOVA é uma técnica estatística utilizada para comparar as médias de três ou mais grupos, ou populações, para determinar se há diferenças estatisticamente significativas entre eles. Ela é amplamente aplicada em pesquisas e experimentos, especialmente quando se deseja avaliar a influência de diferentes fatores ou variáveis independentes em uma variável dependente.

O *F-test* é uma medida estatística usada em Análise de Variância ANOVA para determinar se há diferenças significativas entre as médias de três ou mais grupos. Na ANOVA, comparamos a variabilidade entre os grupos (variância entre grupos) com a variabilidade dentro dos grupos (variância dentro dos grupos). O teste F é calculado dividindo a variância entre os grupos pela variância dentro dos grupos.

A hipótese nula (H0) do teste F é que não há diferenças significativas pré-definidas (geralmente 0,05), entre as médias dos grupos, enquanto a hipótese alternativa (H1) é que pelo menos uma das médias é significativamente diferente das outras.

(TRIOLA, 2013) define ainda a ANOVA como uma decomposição da variância total em um conjunto em variâncias parciais, correspondentes a fontes de variação diferentes e determinadas. Posteriormente, as variâncias serão comparadas entre si (pelo teste F) que assim gerará uma significância estatística para diferenças entre e com os grupos, estabelecendo-se probabilidades entre 0,05, 0,001 ou menos.

Em suma, a análise compara a magnitude das variações de mais de duas amostras, decompondo a variância total em duas partes:

- Entre as amostras, constituindo o chamado quadrado médio dos tratamentos;
- Dentre cada tratamento, compondo o denominado quadrado médio do erro experimental.

O procedimento básico da ANOVA envolve calcular a variância dentro de cada grupo (variância intragrupo) e a variância entre os grupos (variância intergrupo). Em seguida, é calculada a estatística F, sendo a razão entre a variância intergrupo e a variância intragrupo. Se o valor de F for estatisticamente significativo, isso indica que pelo menos um dos grupos é estatisticamente diferente dos outros.

4 METODOLOGIA E FERRAMENTAS

A metodologia utilizada baseia-se no *Design of Experiments* - DOE (MONTGOMERY; RUNGER, 2020), no qual é adotado um planejamento fatorial l^k com r repetições. Foram considerados dois fatores *(k = 2)* com 3 níveis *(l = 3)*: (i) SGBDs – ArangoDB, OreinteDB, MongoDB, Redis; e (ii) comando - inserir, ler, atualizar. Além disso, são consideradas 3 cargas de trabalho diferentes (1000 operações, 5000 operações, 10000 operações) geradas pelo *benchmark* YCSB, e as métricas de interesse são consumo de energia em *joule* (J) e tempo de execução em (ms) para cada carga de trabalho (por exemplo, 1000 operações).

O Yahoo! *Cloud Serving Benchmarks* YCSB é um conjunto de *benchmark* de código aberto para avaliar aplicativos de computador e é frequentemente adotado para comparar o desempenho de SGBD NoSQL. O *benchmark* YCSB tem sido o padrão para avaliar o desempenho geral do SGBD. Além disso, 100 repetições (r = 100) são consideradas para obter valores médios (com distribuição normal aproximada) e reduzir o impacto do ruído de medição. A replicação é uma execução de carga de trabalho (por exemplo, uma única execução de 1.000 operações).

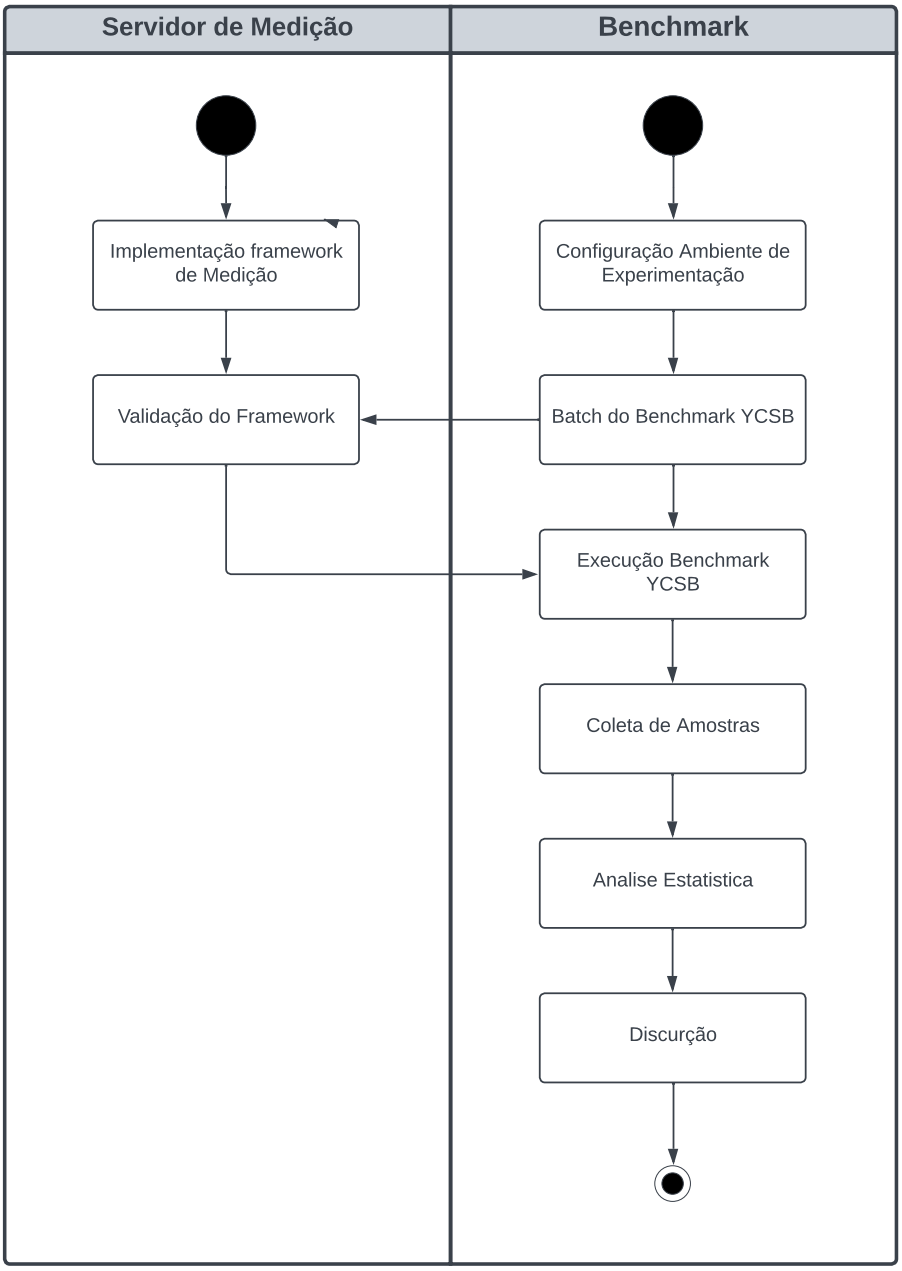
O fluxograma apresentado na Figure 12 denota a sequência das atividades realizadas. Diante da quantidade de SGBDs NoSQL Multimodelo e SGBDs NoSQL nomo disponíveis se fez necessário entender as categorias destes SGBDs mais populares e selecionar um SGBD de representatividade de cada uma destas categorias. Os SGBDs selecionados foram o ArangoDB 3.4.11 , OreinteDB 3.2.8 (multimodelo), MongoDB 7.0 (orientado a documentos) e Redis 7.2 (chave-valor).

Para atender ao objeto de estudo deste trabalho de pesquisa, foi necessário buscar um gerador de carga de trabalho compatível com SGBDs, comandos e cargas de trabalho selecionados, além de possuir boa reputação no meio acadêmico. Nesse sentido, foi selecionado o YCSB, que atendeu a todos os requisitos do estudo e proporcionou um bom suporte técnico em sua utilização.

O sistema operacional selecionado foi o Ubuntu 20.04.4 LTS (Linux) com EXT4 como arquivo de sistema, onde foi utilizado para configurar o framework de medição, esta escolha foi motivada pela popularidade deste sistema operacional.

Após a seleção do ambiente, realizou-se a montagem de *Minimum Viable Product* (MVP) em ambiente virtual para validar a arquitetura dos experimentos e em seguida executados os

Figura 12 – Metodologia proposta na dissertação.



primeiros testes de funcionamento.

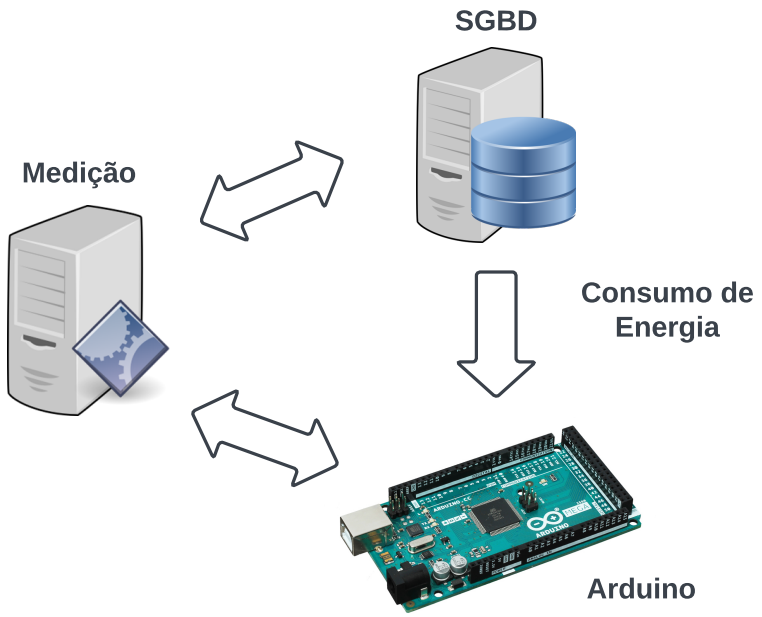
Após a execução dos primeiros testes, os resultados foram avaliados para validar se estavam conforme as expectativas. Após a obtenção de resultados válidos, ou seja, resultados que não apresentaram falhas em sua execução nos distintos tratamentos, foi constatada, devido aos repetidos passos em diferentes espaços de tempo, a necessidade de automatizar o processo do *benchmark*. Para atender a essa necessidade, foi desenvolvido um *script* em Python para a automação desse processo.

Paralelamente a essas etapas, foi criado um framework denominado *Measurement Server*

para a aferição do consumo de energia elétrica. Detalhes sobre esse framework serão explicados posteriormente. A automação da execução do processo possibilitou uma grande quantidade de repetições de cada cenário, com o intuito da obtenção de resultados que representassem uma distribuição normal. Após a coleta dos dados foi realizada a análise estatística e com base nos resultados desta análise para os SGBDs estudados. Neste trabalho, os resultados são analisados por ANOVA (MONTGOMERY; RUNGER, 2020).

4.0.1 *Framework* de Medição

Figura 13 – *Measurement Framework*



O *framework* de medição projetado (Figura 13), e um servidor de medição, que coleta dados relacionados ao consumo de energia e tempo de execução de um sistema computacional (MEASUREMENT, 2022). O *framework* é baseado na plataforma Arduino (MEGA-2560-CH340, 2022), que adota os sensores (ACS217-20A, 2022) e (ZMPT101B, 2022) para coleta, respectivamente, dos valores de corrente e tensão. Pode-se, então, determinar a potência real de um artefato a ser mensurado, de acordo com (NILSSON; RIEDEL, 2018) e, usando o tempo de execução, estimar o consumo de energia. O servidor de medição sincroniza o início e o fim de uma amostra (ou seja, replicação), coleta o respectivo tempo de execução e obtém o consumo de energia com base nos dados da plataforma Arduino.

Tabela 4 – Consumo de Energia ET-4091 x Arduino

	ET-4091 (J)	Arduino (J)
Vídeo	1821.70	1822.85
Sysbench CPU	516.78	509.95
Hibernando	287.98	309.15
Sysbench Arquivo	398.49	402.65
Sysbench Memoria	495.70	483.58
MongoDB	1125.57	1015.73
ArangoDB	574.66	520.24

Tabela 5 – Paired T-test

95% CI	p-value
[-19.3;64.1]	0.237

O *framework* proposto também 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. A plataforma proposta foi validada utilizando o alicate amperímetro (MINIPA-ET-409, 2022), 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 foram setados como default para não interferir na coleta de dados.

A validação também considerou 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 *sysbench* (CPU, Arquivo, Memória); (iii) e uma carga de trabalho YCSB em MongoDB e ArangoDB. Para cada aparelho de medida, foram coletadas 100 amostras e adotado *bootstrap* para estimar os valores médios, a Tabela 4 apresenta-os do *T-test* pareado realizado (MONTGOMERY; RUNGER, 2020). A Tabela 5 fornece os resultados.Uma vez que o valor 0 está dentro do intervalo de confiança de 95%, não há base estatística para afirmar que as duas plataformas oferecem resultados distintos..

4.0.2 Configuração YCSB

Para realizar este *benchmark*, necessita-se de um gerador de carga de trabalho que responda a dois requisitos: definir o conjunto de dados carregando no banco de dados e executar ope-

rações contra o conjunto de dados aferindo o desempenho (tempo de execução). O consumo de energia é aferido por outro *framework* que será explicado nas próximas seções.

Nos experimentos foram utilizados dez campos por registro, definido por padrão pelo SGBDs, sendo o tamanho de cada campo 100 *bytes*. Na leitura todos os campos eram acessados e a proporção das operações foi definida conforme os comandos do *benchmark*, por exemplo para leitura foram executados 100% de leitura; a escrita e a atualização seguiram com a mesma proporção, o número de operações seguiu de acordo com o cenário (1.000 operações, 5.000 operações ou 10.000 operações). Para cada execução do *benchmark* foi realizado um *load* que realiza a preparação do ambiente, o parâmetro -*target* não foi definido, desta forma o SGBDs não limitou a quantidade de operações por segundo utilizando a máxima capacidade de vazão do ambiente.

Este trabalho utilizou a distribuição *zipfian* baseada no algorítimo *zipfian* sendo customizada pelo Yahoo! para os itens populares serem espalhados por toda a *keyspace* do banco de dados conforme visto em (YCSB, 2022).

Dessa forma, é quase certo que são atualizados muitos registros diferentes, mas há uma boa chance de que certos itens sejam atualizados várias vezes. Para cada execução *run* foi realizada uma preparação do ambiente *load*, isto garante, por exemplo, que ao executar a leitura o dado esteja no banco de dados.

5 RESULTADOS EXPERIMENTAIS

Este capítulo apresenta os resultados obtidos dos experimentos realizados conforme o *Design of Experiments* (DOE) apresentado no Capítulo 3. Como explicado anteriormente, o *benchmark* YCSB é utilizado considerando cargas de trabalho de 1.000, 5.000 e 10.000 operações. O número de operações corresponde à quantidade de um mesmo procedimento, que é lançado contra o Sistema de Gerenciamento de Banco de Dados (SGBD) por execução. A seleção de três cargas de trabalho obedece a um incremento de 400% e 100%, o que é importante para observar a escalabilidade do SGBD.

Para cada SGBD, foi adotada a configuração padrão, e as métricas de interesse são o tempo de execução e o consumo de energia. Além disso, cada registro inserido no banco de dados possui 1 KB (valor-padrão definido pelo YCSB). O *Measurement Server* realizou a leitura do consumo. Os resultados dos experimentos consideraram apenas o consumo de energia no intervalo de tempo de execução do YCSB.

Conforme Tabela 6, Tabela 7 , Tabela 8, Tabela 9, Tabela 10 e Tabela 11, são apresentados os resultados gerais para as amostras consideradas em cada tratamento (N) o *Confidence Interval* (IC). O resultado são apresentados para cada carga de trabalho usando a análise ANOVA (MONTGOMERY; RUNGER, 2020). Em seguida, a correlação é discutida para ambas as métricas seguida pelas observações gerais.

Tabela 6 – Tempo de Execução e Consumo de energia para ARANGO DB - DOC

		tempo de execução (ms)		**consumo de energia (J)**	
comando	**carga**	**média**	**IC 95%**	**média**	**IC 95%**
escrita	1000	4.969	4.925;5.013	26.356	26.052;26.661
	5000	15.546	15.49;15.603	86.001	85.500;86.502
	10000	28.402	28.262;28.541	158.881	157.501;160.262
leitura	1000	4.699	4.661;4.738	18.558	18.360;18.756
	5000	13.317	13.21;13.364	56.546	56.294;56.797
	10000	23.670	23.612;23.729	122.977	122.500;123.455
atualização	1000	4.708	4.642;4.773	20.194	19.957;20.432
	5000	13.451	13.47;13.494	62.585	62.383;62.787
	10000	23.369	23.278;23.460	127.134	126.664;127.603

Tabela 7 – Tempo de Execução e Consumo de energia para OrientDB - DOC

		tempo de execução (ms)		consumo de energia (J)	
comando	**carga**	**média**	**IC 95%**	**média**	**IC 95%**
escrita	1000	5.710	5.663;5.757	51.288	50.978;51.598
	5000	6.966	6.838;7.094	71.555	70.613;72.497
	10000	8.263	8.118;8.408	87.553	86.671;88.435
leitura	1000	3.828	3.806;3.850	40.406	40.181;40.632
	5000	3.980	3.964;3.997	47.562	47.370;47.755
	10000	4.094	4.072;4.117	50.115	49.949;50.280
atualização	1000	5.405	5.356;5.453	50.949	50.623 51.274
	5000	6.963	6.886;7.039	74.246	73.753;74.739
	10000	8.147	8.087;8.207	86.362	85.940;86.784

Tabela 8 – Tempo de Execução e Consumo de energia para ARANGO DB - KEY

		tempo de execução (ms)		consumo de energia (J)	
comando	**carga**	**média**	**IC 95%**	**média**	**IC 95%**
escrita	1000	5.031	4.993;5.069	26.200	26.000;26.401
	5000	15.548	15.48;15.609	86.361	85.753;86.968
	10000	28.111	27.958;28.264	157.965	156.387;159.542
leitura	1000	4.656	4.610;4.702	18.617	18.349;18.885
	5000	13.385	13.36;13.435	58.079	57.616;58.542
	10000	23.636	23.567;23.705	123.842	123.478;124.206
atualização	1000	4.635	4.610;4.660	19.351	19.134;19.568
	5000	13.383	13.35;13.431	62.802	62.535;63.069
	10000	22.536	22.385;22.688	125.567	124.697;126.437

Tabela 9 – Tempo de Execução e Consumo de energia para OrientDB - KEY

		tempo de execução (ms)		consumo de energia (J)	
comando	**carga**	**média**	**IC 95%**	**média**	**IC 95%**
escrita	1000	5.693	5.642;5.744	50.662	50.200;51.124
	5000	6.981	6.882;7.080	70.092	69.503;79.798
	10000	8.200	8.074;8.326	86.787	85.986;87.588
leitura	1000	3.904	3.808;4.001	40.802	40.348;41.257
	5000	3.995	3.975;4.016	48.964	48.708;49.219
	10000	4.119	4.099;4.139	50.664	50.479;50.849
atualização	1000	5.363	5.310;5.416	51.217	50.848;51.587
	5000	6.663	6.587;6.738	71.802	71.192;70.681
	10000	8.504	8.430;8.577	88.282	87.781;88.783

Tabela 10 – Tempo de Execução e Consumo de energia para MongoDB

		tempo de execução (ms)		consumo de energia (J)	
comando	**carga**	**média**	**IC 95%**	**média**	**IC 95%**
escrita	1000	3.849	3.815;3.882	16.570	16.416;16.724
	5000	6.768	6.719;6.818	32.053	31.832;32.274
	10000	10.667	10.599;10.736	53.556	53.106;54.005
leitura	1000	4.034	4.017;4.050	17.002	16.913;17.091
	5000	6.430	6.395;6.464	27.643	27.502;27.784
	10000	8.659	8.622;8.697	49.206	49.032;49.380
atualização	1000	3.981	3.957;4.006	17.421	17.257;17.584
	5000	6.401	6.361;6.442	30.245	30.097;30.393
	10000	8.454	8.422;8.487	50.734	50.518;50.951

Tabela 11 – Tempo de Execução e Consumo de energia para Redis

		tempo de execução (ms)		consumo de energia (J)	
comando	**carga**	**média**	**IC 95%**	**média**	**IC 95%**
escrita	1000	1.739	1.726;1.753	10.130	10.052;10.207
	5000	4.480	4.374;4.587	30.891	30.442;31.341
	10000	7.240	7.078;7.402	53.708	52.828;54.588
leitura	1000	1.300	1.285;1.314	7.881	7.817;7.945
	5000	2.689	2.613;2.766	13.119	12.947;13.292
	10000	4.159	4.043;4.274	21.428	21.135;21.720
atualização	1000	1.318	1.298;1.338	7.888	7.784;7.992
	5000	2.493	2.447;2.540	16.410	16.223;16.597
	10000	3.946	3.852;4.040	26.657	26.303;27.011

5.1 RESULTADO DOS TRATAMENTOS

A Tabela 12 apresenta os resultados da análise ANOVA (nível de significância $\alpha = 0{,}05$) para tempo de execução e consumo de energia, com todos os fatores e consecutivas interações $(source)$ que impactam significativamente no tempo de execução (estatística *F -Fstat:- e p-value*). Dependendo da carga de trabalho, o fator pode ter um impacto significativamente diferente na métrica (var:%), pois alguns SGBDs tem o seu comportamento influenciado pelo quantidade de operações realizadas. $SGBD$ tem o maior impacto nos resultados, e os ruídos de medição ($Error$) são baixos.

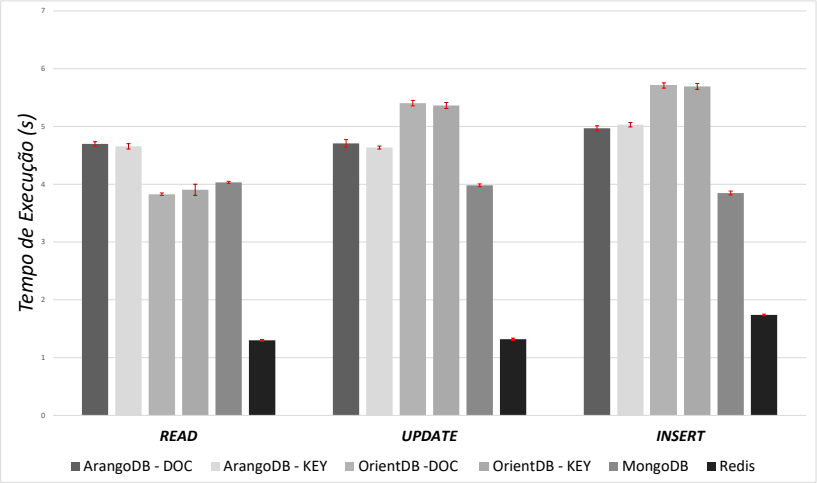


Figura 14 – Média Tempo de Execução considerando 1.000 Operações

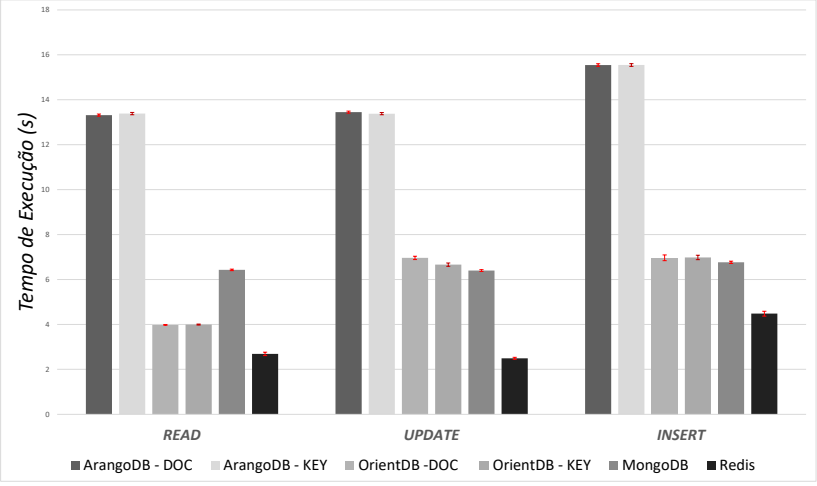


Figura 15 – Média Tempo de Execução considerando 5.000 Operações

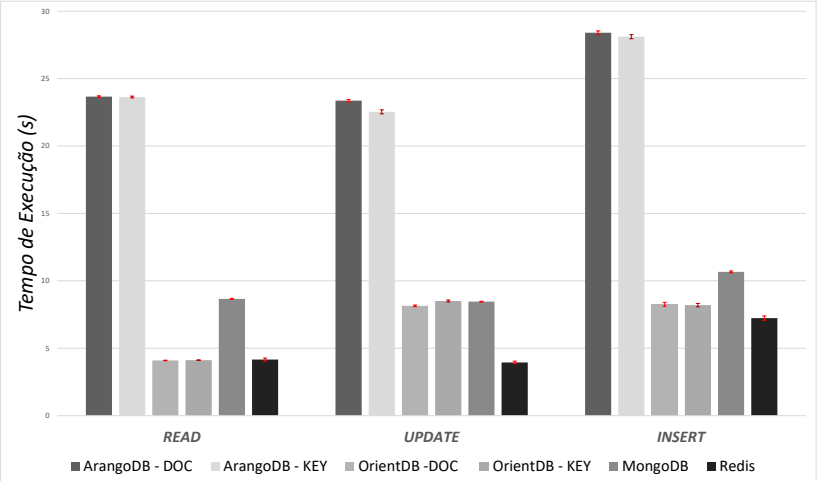


Figura 16 – Média Tempo de Execução considerando 10.000 Operações

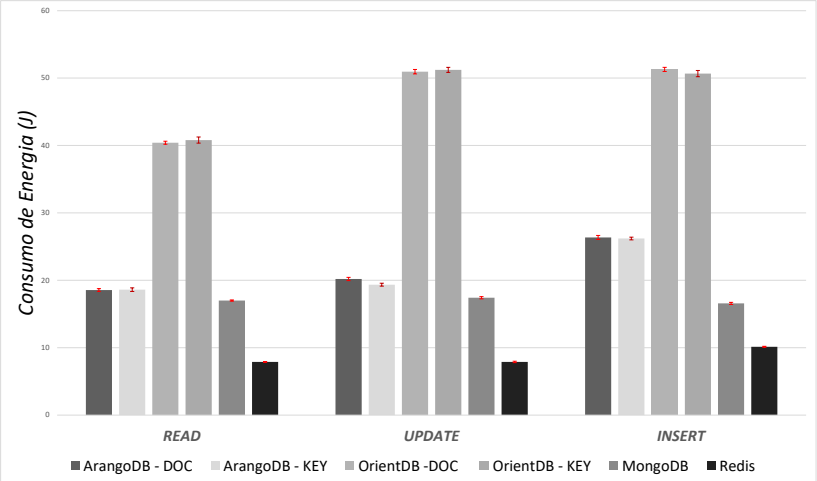


Figura 17 – Média Consumo de Energia considerando 1.000 Operações

Tabela 12 – ANOVA: Tempo de Execução

work. 1,000				work. 5,000				work. 10,000			
source	var.%	F stat.	p-value	source	var.%	F stat.	p-value	source	var.%	F stat.	p-value
SGBD	84.78	11093.85	<0.001	SGBD	93.63	57432.097	<0.001	SGBD	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
SGBD*Comm	7.08	463.27	<0.001	SGBD*Comm	2.03	622.25	<0.001	SGBD*Comm	16.18	8544.13	<0.001
Error	2.72			Error	0.58			Error	0.34		

A Figura 14 a Figura 19 representam os resultados considerando os valores médios e os respectivos intervalos de confiança de 95% (no topo de cada barra).

A explicação é fundamentada na avaliação realizada por meio do procedimento de Tukey (MONTGOMERY; RUNGER, 2020) , um teste pós-ANOVA. Em geral, o modelo de dados não demonstra uma influência significativa sobre o desempenho do ArangoDB e do OrientDB. Somente quando a discrepância é substancial, a explicação menciona explicitamente o impacto

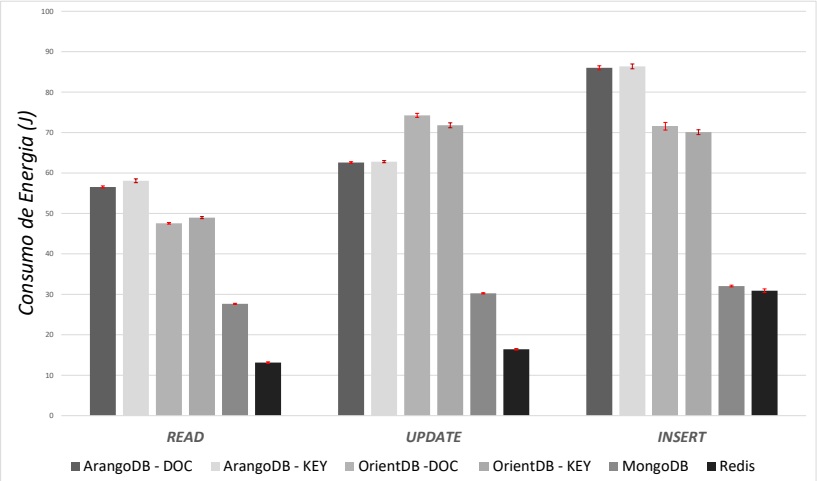


Figura 18 – Média Consumo de Energia considerando 5.000 Operações

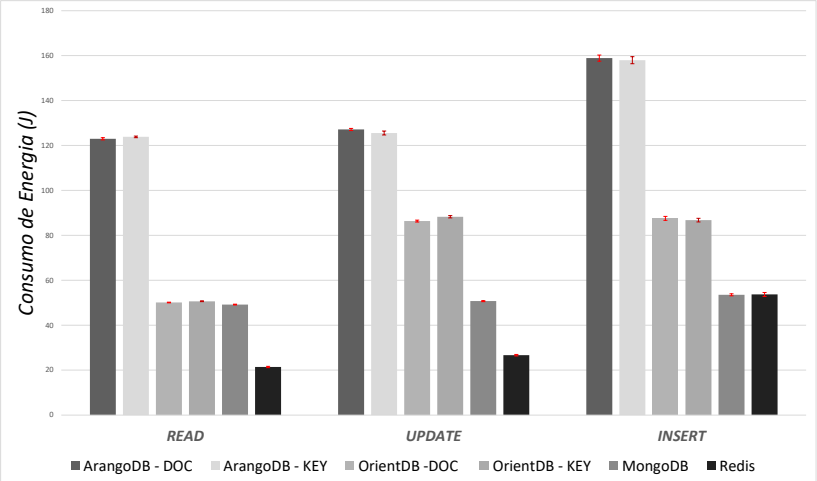


Figura 19 – Média Consumo de Energia considerando 10.000 Operações

Tabela 13 – ANOVA: Consumo de Energia

work. 1.000				work. 5.000				work. 10.000			
source	var.%	F stat.	p-value	source	var.%	F stat.	p-value	source	var.%	F stat.	p-value
SGBD	94.16	43393.38	<0.001	SGBD	78.24	28247.77	<0.001	SGBD	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
SGBD*Comm	2.22	511.66	<0.001	SGBD*Comm	6.39	1154.27	<0.001	SGBD*Comm	12.23	2545.24	<0.001
Error	0.77			Error	0.99			Error	0.86		

para evitar uma exposição excessivamente complexa.

As Tabelas 14 e 23, comparações são apresentadas de acordo com o teste de Tukey. Na coluna (Grupo) dessas tabelas, são agrupados todos os resultados que não diferem significativamente de acordo com este procedimento. Resultados que apresentam diferenças são identificados por letras distintas.

Avaliando os resultados contidos na Tabela 14 correspondente ao teste de Tukey, há evidên-

cias de que os valores médios por SGBD são estatisticamente diferentes, Redis obteve melhor desempenho, seguido por MongoDB ArangoDB e OrientDB. Os comandos de leitura e atualização influenciaram o melhor desempenho do Redis; por outro lado, o menor desempenho do Oriente que foi fortemente influenciado pelo comando de escrita. Os comandos leitura e atualização também possuem valores médios diferentes, escrita foi quem mais demandou tempo para sua conclusão, enquanto a atualização foi o comando com menor tempo de execução.

Tabela 14 – Tempo de Execução Tukey 1.000 Operações (SDBD)

SGBD	N	Média	Grupo
ORIENT-DOC	300	4.9754	A
ORIENT-KEY	300	4.9739	A
ARANGO-DOC	300	4.7823	B
ARANGO-KEY	300	4.7656	B
MONGO	300	3.94926	C
REDIS	300	1.4479	D

Tabela 15 – Tempo de Execução 1.000 Operações (SGBD X COMANDO)

SGBD	N	Média	Grupo
ORIENT-DOC-INSERT	100	5.7021	A
ORIENT-KEY-INSERT	100	5.6917	A
ORIENT-DOC-UPDATE	100	5.4034	B
ORIENT-KEY-UPDATE	100	5.3513	B
ARANGO-KEY-INSERT	100	5.0149	C
ARANGO-DOC-INSERT	100	4.9579	C
ARANGO-DOC-READ	100	4.6949	D
ARANGO-DOC-UPDATE	100	4.6941	D
ARANGO-KEY-READ	100	4.6415	D
ARANGO-KEY-UPDATE	100	4.6406	D
MONGO-READ	100	4.0283	E
MONGO-UPDATE	100	3.9813	E F
ORIENT-KEY-READ	100	3.8786	F G
MONGO-INSERT	100	3.8381	G
ORIENT-DOC-READ	100	3.82091	G
REDIS-INSERT	100	1.74295	H
REDIS-UPDATE	100	1.30916	I
REDIS-READ	100	1.29171	I

Considerando 5.000 operações (Figura 15) ArangoDB parece ser menos escalável para todos os comandos. Por exemplo, o ArangoDB foi 500% apresenta maior tempo de leitura o que o Redis, o qual continua superando todos os SGBDs.

O MongoDB foi o segundo banco de dados mais lento em termos de tempo necessário para executar operações de leitura. OrientDB demonstrou poder lidar bem com um aumento na

carga de trabalho. Quando a demanda por acesso aos dados aumenta, o OrientDB pode dimensionar de forma eficaz para atender a essas demandas adicionais sem comprometer significativamente o desempenho.

Além disso, não há diferença estatística entre OrientDB e MongoDB para inserção, o MongoDB é apenas 3,88% mais rápido para atualização.

Avaliando os resultados contidos na Tabela 16 correspondente ao teste de Tukey, há evidências de que os valores médios por SGBD são estatisticamente diferentes, Redis também obteve melhor desempenho, seguido por OrintetDB. Os comandos de leitura e atualização influenciaram o melhor desempenho do Redis; por outro lado, o menor desempenho do ArangoDB que foi fortemente influenciado pelo comando de escrita. Os comandos também possuem valores médios diferentes, escrita foi quem mais demandou tempo para sua conclusão, enquanto a leitura foi o comando mais ágil.

Tabela 16 – Tempo de Execução Tukey 5000 Operações (SDBD)

SGBD	N	Média	Grupo
ARANGO-KEY	300	14.1017	A
ARANGO-DOC	300	14.0799	A
MONGO	300	6.5289	B
ORIENT-DOC	300	5.9541	C
ORIENT-KEY	300	5.8681	C
REDIS	300	3.2056	D

A Figura 18 representa 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 SGBD. Embora o OrientDB tenha tempo de inserção semelhante ao 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. O Redis oferece a melhor economia de energia para todos os comandos.

Avaliando os resultados contidos na Tabela 18, os valores médios por SGBD são estatisticamente diferentes, dessa forma, houve uma mudança na classificação do desempenho geral em relação à carga de trabalho de 1.000 operações, agora, em comparação com 1.000 operações, tanto Redis quanto MongoDB mostraram melhor desempenho em 10.000 operações. Isso ocorreu porque Redis e MongoDB melhoraram significativamente seu desempenho, enquanto o desempenho do ArangoDB permaneceu relativamente o mesmo. Os comandos permanecem com valores médios diferentes, onde o comando de escrita foi quem mais demandou tempo para sua conclusão, enquanto a leitura contínua sendo o comando mais com menor tempo de

Tabela 17 – Tempo de Execução 5.000 Operações (SGBD X COMANDO)

SGBD	N	Média	Grupo
ARANGO-DOC-INSERT	100	15.5496	A
ARANGO-KEY-INSERT	100	15.5376	A
ARANGO-KEY-UPDATE	100	13.3929	B
ARANGO-DOC-UPDATE	100	13.3929	B
ARANGO-KEY-READ	100	13.3745	B
ARANGO-DOC-READ	100	13.2970	B
ORIENT-KEY-INSERT	100	6.9599	C
ORIENT-DOC-UPDATE	100	6.9575	C
ORIENT-DOC-INSERT	100	6.9287	C
MONGO-INSERT	100	6.7610	D
ORIENT-KEY-UPDATE	100	6.6476	D
MONGO-READ	100	6.4263	E
MONGO-UPDATE	100	6.3994	E
REDIS-INSERT	100	4.4583	F
ORIENT-KEY-READ	100	3.9967	G
ORIENT-DOC-READ	100	3.97596	G
REDIS-READ	100	2.6751	H
REDIS-UPDATE	100	2.4834	I

duração. Destacamos que o menor desempenho da escrita neste contexto sofre forte influência do Arango.

Tabela 18 – Consumo de Energia Tukey 5.000 Operações (SDBD)

SGBD	N	Média	Grupo
ARANGO-KEY	300	68.996	A
ARANGO-DOC	300	68.417	A
ORIENT-DOC	300	64.331	B
ORIENT-KEY	300	63.539	B
MONGO	300	29.979	C
REDIS	300	20.070	D

A Figura 16 mostra os tempos de execução para a carga de trabalho de 10.000 operações. O ArangoDB fornece maior tempo de execução para a atividade. Para leitura, OrientDB é rápido como Redis, e tal SGBD é apenas 13,17% mais lento que Redis para inserção. Além disso, MongoDB é mais lento para todas as operações em comparação com OrientDB, exceto para atualização e OrientDB-KEY. OrientDB-DOC e OrientDB-KEY possuem uma pequena diferença ($0.35s$) de tempo de execução, em que o primeiro realiza a atualização mais rapidamente. Em relação ao ArangoDB-DOC, ArangoDB-Key no comando de atualização, há uma diferença representativa, que é de cerca de $1s$ para esta carga de trabalho. Avaliando em 100.000 operações os comandos por SGBD na Tabela 20, o ArangoDB, continua utilizando o

Tabela 19 – Consumo de Energia Tukey 5.000 Operações (SGBD X COMANDO)

SGBD	N	Média	Grupo
ARANGO-KEY INSERT	100	86.179	A
ARANGO-DOC INSERT	100	85.894	A
ORIENT-DOC UPDATE	100	74.209	B
ORIENT-KEY UPDATE	100	71.647	C
ORIENT-DOC INSERT	100	71.275	C
ORIENT-KEY INSERT	100	70.011	D
ARANGO-KEY UPDATE	100	62.840	E
ARANGO-DOC UPDATE	100	62.840	E
ARANGO-KEY READ	100	57.968	F
ARANGO-DOC READ	100	56.516	G
ORIENT-KEY READ	100	48.958	H
ORIENT-DOC READ	100	47.509	I
MONGO INSERT	100	32.040	J
REDIS INSERT	100	30.772	K
MONGO UPDATE	100	30.2512	K
MONGO READ	100	27.6442	L
REDIS UPDATE	100	16.364	M
REDIS READ	100	13.0733	N

maior tempo de execução para o comando de escrita;

Em relação ao consumo de energia, a Figura 19 apresenta os valores médios. Embora OrientDB tenha um tempo de execução notável para uma carga de trabalho maior, o consumo de energia é maior que MongoDB e Redis para atualização e inserção.

Tabela 20 – Tempo de Execução Tukey 10000 Operações (SDBD)

SGBD	N	Média	Grupo
ARANGO-DOC	300	25.135	A
ARANGO-KEY	300	18.252	B
MONGO	300	9.2521	C
ORIENT-KEY	300	6.918	D
ORIENT-DOC	300	6.798	D
REDIS	300	5.1136	E

5.1.1 Correlação

Nesta seção é apresentada de maneira geral a correlação da variação dos escores nos eixos tempo de execução e consumo de energia.

As Figuras 20, 21, 22, 23, 24 e 25 mostram a correlação entre o consumo de energia e o tempo de execução para cada comando (escrita, leitura e atualização) contemplando

Tabela 21 – Tempo de Execução 10.000 Operações (SGBD X COMANDO)

SGBD	N	Média	Grupo
ARANGO-DOC-INSERT	100	28.3851	A
ARANGO-KEY-INSERT	100	28.1281	B
ARANGO-DOC-READ	100	23.6686	C
ARANGO-DOC-UPDATE	100	23.3524	D
ARANGO-KEY-UPDATE	100	22.5116	E
MONGO-INSERT	100	10.6559	F
MONGO-READ	100	8.6603	G
ORIENT-KEY-UPDATE	100	8.4829	G
MONGO-UPDATE	100	8.4400	G
ORIENT-DOC-INSERT	100	8.1700	H
ORIENT-KEY-INSERT	100	8.1537	H
ORIENT-DOC-UPDATE	100	8.1332	H
REDIS-INSERT	100	7.2322	I
REDIS-READ	100	4.1559	J
ORIENT-KEY-READ	100	4.1172	J
ARANGO-KEY-READ	100	4.1172	J
ORIENT-DOC-READ	100	4.08932	J
REDIS-UPDATE	100	3.9529	J

Tabela 22 – Consumo de Energia Tukey 10.000 Operações (SDBD)

SGBD	N	Média	Grupo
ARANGO-DOC	300	136.206	A
ARANGO-KEY	300	111.26	A
ORIENT-KEY	300	75.13	B
ORIENT-DOC	300	74.46	B
MONGO	300	51.129	C
REDIS	300	33.872	D

nos SGBDs. Para todos os SGBDs, existe uma forte correlação linear positiva indicada pelo coeficiente de determinação (R^2). Os valores são 0,985, 0,986, 0,911, 0,902, 0,943 e 0,927 para ArangoDB-DOC (Figura 20), ArangoDB-KEY (Figura 21), OrientDB-DOC (Figura 22), OrientDB-KEY (Figura 23), MongoDB (Figura 24) e Redis (Figura 25), respectivamente.

A Figura 20 representa os valores para ArangoDB-DOC, em que a equação y = 5.744x - 8.474 representa esta correlação. A equação y = 5.785x - 8.778 representa a correlação para ArangoDB-key (Figura 21), já o OrientDB -DOC (Figura 22) tem a equação y = 9.621x - 5.262 e o OrientDB-KEY (Figura 23) tem a equação y = 9.306x - 6.954 para representar esta correlação. E a Figura 24 tem a equação y = 6.125x - 7.589 que representa a correlação para MongoDB e por fim e a correlação do Redis (Figura 25) é apresentada pela equação y = 7.292x - 2.895.

As Figuras 20 e 25 também descrevem a equação linear para cada SGBD. A inclinação (ou

Tabela 23 – Consumo de Energia Tukey 10.000 Operações (SGBD X COMANDO)

SGBD	N	Média	Grupo
ARANGO-DOC-INSERT	100	158.644	A
ARANGO-KEY-INSERT	100	157.915	A
ARANGO-DOC-UPDATE	100	126.996	B
ARANGO-KEY-UPDATE	100	125.234	B
ARANGO-DOC-READ	100	122.979	C
ORIENT-KEY-UPDATE	100	88.203	D
ORIENT-DOC-INSERT	100	87.063	D E
ORIENT-KEY-INSERT	100	86.557	D E
ORIENT-DOC-UPDATE	100	86.222	E
REDIS-INSERT	100	53.601	F
MONGO-INSERT	100	53.523	F
MONGO-UPDATE	100	50.665	G
ORIENT-KEY-READ	100	50.632	G
ARANGO-KEY-READ	100	50.632	G
ORIENT-DOC-READ	100	50.088	G
MONGO-READ	100	49.1997	G
REDIS-UPDATE	100	26.611	H
REDIS-READ	100	21.403	I

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. Com essas informações constatamos que o ArangoDB-DOC tem a menor taxa J/s (5.744), mas tal sistema leva mais tempo para realizar as cargas de trabalho. Assim, o consumo de energia é consideravelmente elevado tendo em conta todas as operações de uma requisição.

Conforme apresentado na Figura 16 e 19, o Redis tem um desempenho proeminente com consumo de energia reduzido. Embora este SGBD tenha uma taxa significativa (7,292), os tempos de execução são os menores de todos os sistemas de banco de dados. Por exemplo, uma comparação direta apenas com as inclinações indicaria que o ArangoDB é mais eficiente energeticamente que o Redis, contradizendo os resultados apresentados na correlação linear.

5.1.2 Discussão

É importante observar que em todos os experimentos foram utilizadas as configurações padrão de instalação para os Sistemas de Gerenciamento de Bancos de Dados (SGBDs). Dessa forma, quaisquer otimizações específicas de configuração desses SGBDs estão fora do escopo deste trabalho. Da mesma forma, a mudança do ambiente de experimentação não foi consi-

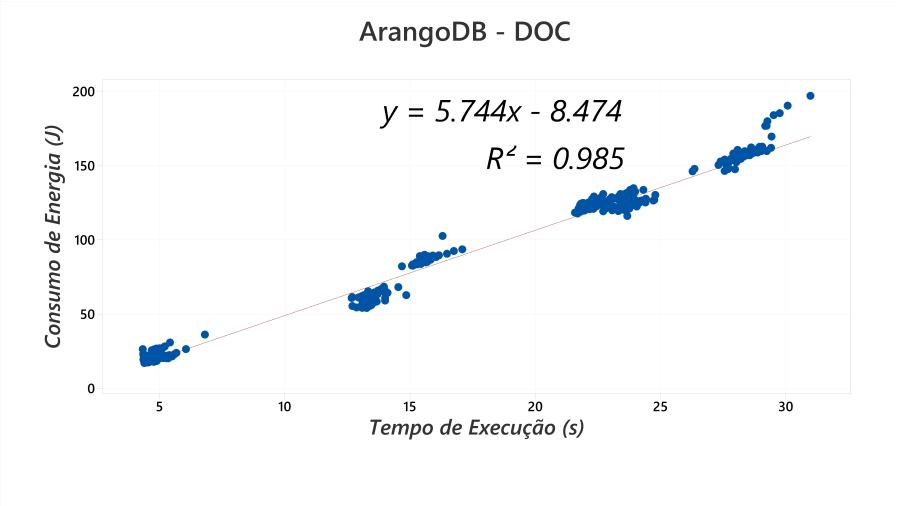


Figura 20 – Correlação Tempo de Execução x Consumo de Energia ArangoDB-DOC

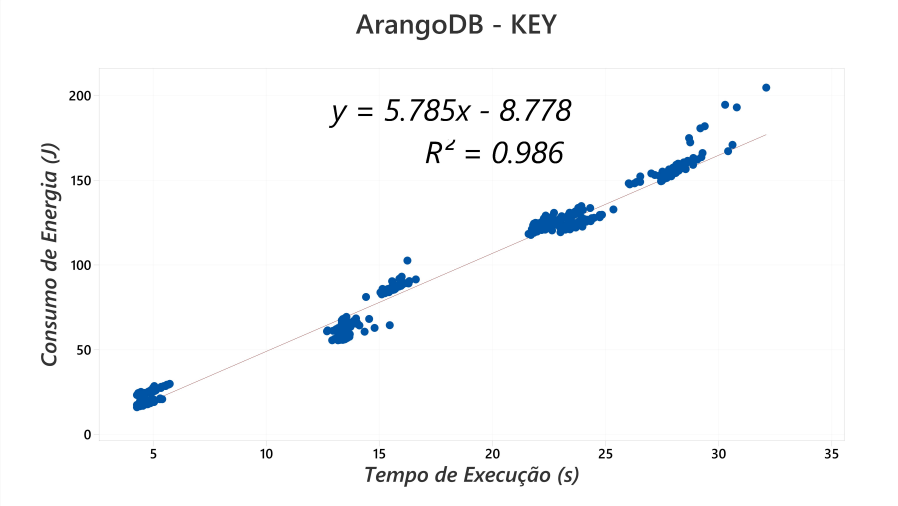


Figura 21 – Correlação Tempo de Execução x Consumo de Energia ArangoDB-KEY

derada, pois esses fatores podem gerar resultados diferentes dos obtidos nesta pesquisa. Além disso, para reduzir possíveis interferências relacionadas a questões associadas a um ambiente, adotou-se apenas um único computador para a realização dos experimentos. Essa escolha visa fornecer resultados mais controlados e específicos para a configuração utilizada. Antes de cada execução do *benchmark* foi realizado o procedimento de *load* para aquele*benchmark*, de forma que cada execução foi realizada com base em um novo conjunto de dados gerados com base

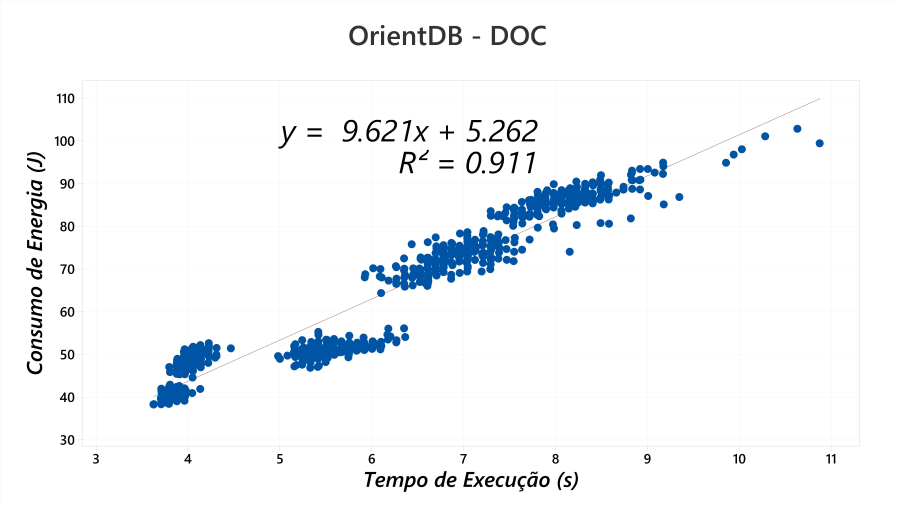


Figura 22 – Correlação Tempo de Execução x Consumo de Energia OrientDB-DOC

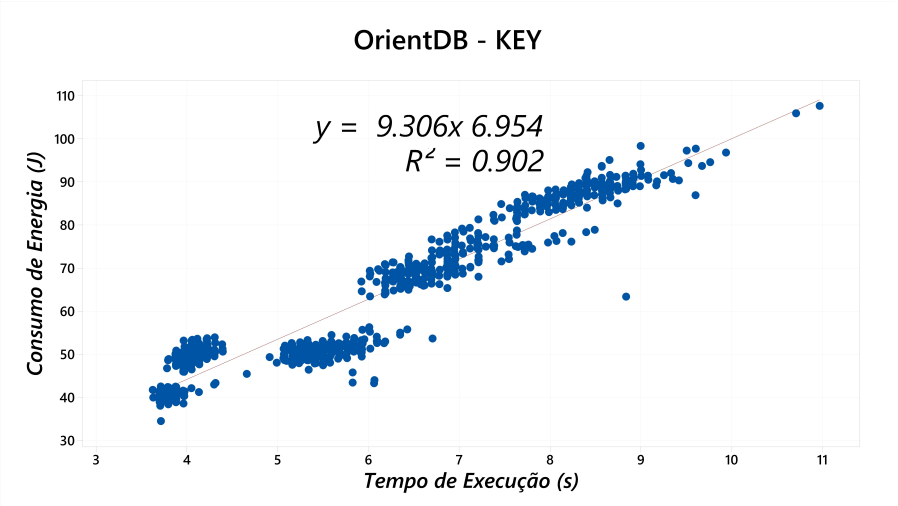


Figura 23 – Correlação Tempo de Execução x Consumo de Energia OrientDB-KEY

no algoritmo *zipfian* do YCSB.

Redis é baseado *in-memory*, mas com persistência em disco, essa característica influenciou na carga de trabalho das operações para o melhor rendimento nas métricas, tempo de execução e consumo de energia entre os SGBD comparados (REDIS, 2022).

O Redis teve um desempenho notável nos experimentos, uma vez que tal SGBD forneceu os menores tempos de execução e consumo de energia, o desempenho do SGBD pode estar

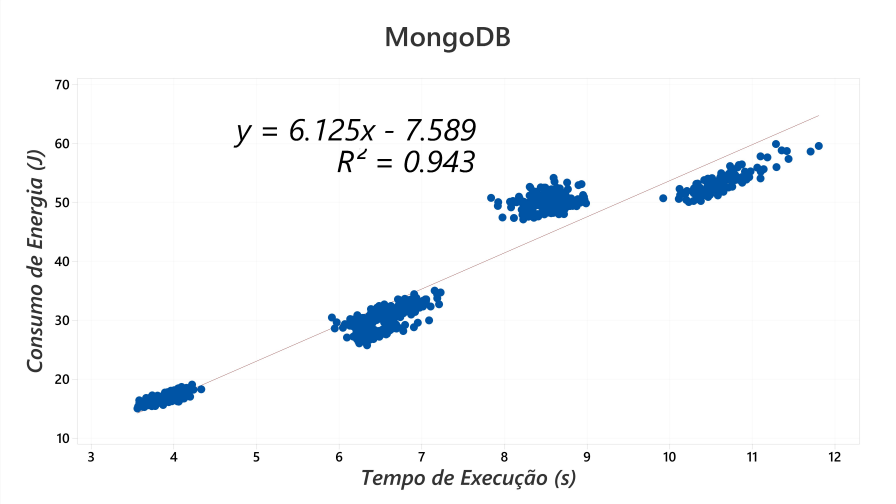


Figura 24 – Correlação Tempo de Execução x Consumo de Energia MongoDB

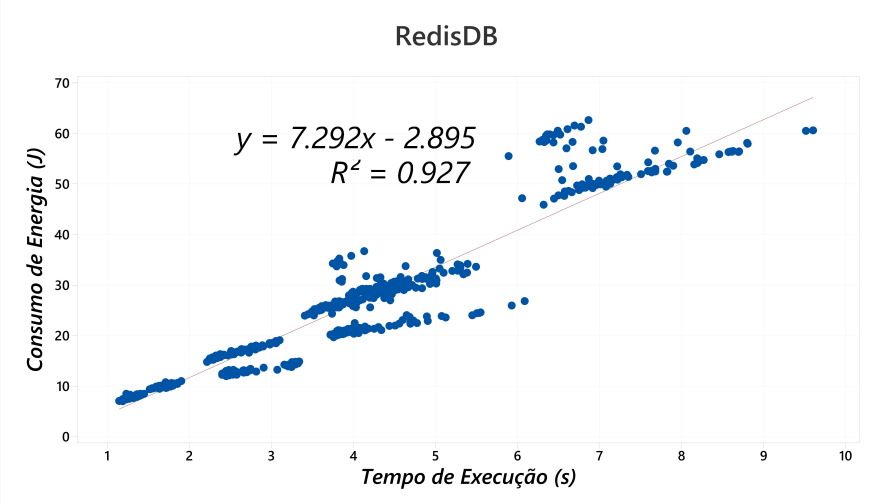


Figura 25 – Correlação Tempo de Execução x Consumo de Energia Redis

relacionado ao *snapshot* que é um procedimento de persistência em disco (acesso mais lento que a memória RAM) do Redis. Mesmo executando 10.000 operações, há indícios de que comandos não influenciaram no desempenho do Redis, uma vez que a escrita obteve a maior degradação de desempenho e maior consumo de energia. No entanto, os resultados indicam que, com solicitações maiores, OrientDB e MongoDB são semelhantes.

Com relação ao SGBD multimodelo, em geral, o ArangoDB possui os maiores tempos de

execução para o *benchmark* adotado, mas o consumo de energia não é alto para pequenas cargas de trabalho. Por outro lado, OrientDB tem o maior consumo de energia quando poucas operações são executadas, mas, para um conjunto maior de operações, a utilização de energia não aumenta consideravelmente.

O OrientDB utiliza mais memória do sistema do que outros SGBDs para cargas de trabalho menores, o que pode explicar o aumento no consumo de energia nesse cenário (ORIENT-DB, 2022) . A inicialização de estruturas de dados internas pode impactar o consumo de algumas operações, mas tal abordagem compensa a escalabilidade para lidar com mais solicitações (por exemplo, 10.000 cargas de trabalho). No caso do ArangoDB, a camada de processamento de dados pode ser responsável pelo aumento do consumo de energia e dos tempos de execução em função da variação da carga de trabalho.

O desempenho do MongoDB manteve um comportamento estável com relação ao consumo de energia com modificação das diferentes cargas de trabalho 1.000, 5.000 e 10.000 operações. Apesar de manter esse comportamento estável, o MongoDB também demonstrou uma melhora sutil em sua eficiência à medida de consumo de energia, que a carga de trabalho aumentava. O menor desempenho do MongoDB para escrita dentre seus próprios comandos pode estar relacionado ao recurso de *journaling* para prover a durabilidade dos dados em caso de falha do sistema. De maneira geral, MongoDB obteve o menor consumo de energia nas maiores cargas de trabalho.

Com relação aos comandos adotados, *read* teve o menor consumo de energia (como esperado), por exemplo, devido ao mecanismo de cache interno geralmente adotado por sistemas de gerenciamento de banco de dados. Já o *Insert* tem o maior consumo, pois o banco de dados precisa ser modificado. Particularmente, o ArangoDB consome quase duas vezes mais energia do que o OrientDB para a carga de trabalho de 10.000 e inserir comando.

É importante ressaltar que nenhum SGBD proporcionou um comportamento dominante (tempo de execução e consumo de energia) em todas as cargas de trabalho. De maneira geral, o SGBD mais rápido não foi o que obteve o menor consumo de energia e os resultados variaram em função dos SGBDs, comandos e carga de trabalho. A análise de variância demonstra que a iteração (SGBD * Comando) é a maior fonte de variação do consumo de energia, ou seja, a escolha de um SGBD NoSQL tem importante relação na eficiência do consumo em função do procedimento realizado.

6 COMENTÁRIOS FINAIS

Este trabalho teve como foco avaliar o desempenho e consumo de energia de gerenciadores de bancos de dados multimodelo populares, comparando os mesmos com gerenciadores de bancos de dados nativos. Ele apresenta um diferencial em relação aos demais trabalhos por analisar apenas SGBDs multimodelo e ao utilizar um conjunto distinto de critérios de comparação. O objetivo foi verificar a viabilidade na adoção desses gerenciadores, analisando também a diferença de implementação dos diferentes modelos. Como contribuição deste trabalho, vale ressaltar que foi proposta uma *benchmark*, juntamente com o desenvolvimento de uma ferramenta específica para este propósito.

O consumo de energia é uma preocupação importante, e os subsistemas de armazenamento têm uma contribuição notável na utilização de energia. Ao longo dos anos, essa questão ganhou atenção considerável de pesquisadores e profissionais para desenvolver plataformas e sistemas energeticamente eficientes. No entanto, poucos trabalhos exploraram a influência dos SGBDs NoSQL em relação ao consumo de energia, e os SGBDs multimodelos são geralmente negligenciados.

Esta pesquisa realizou a avaliação de desempenho e consumo de energia de SGBDs NoSQL multimodelo e monomodelo representativos. Os experimentos adotaram diferentes cargas de trabalho para leitura, escrita e atualização de dados usando o *benchmark* YCSB. O Redis apresentou os melhores resultados, mas, considerando uma carga de trabalho maior, OrientDB e MongoDB obtiveram valores próximos em relação ao tempo de execução e utilização de energia.

O desempenho médio das execuções entre os cenários relevantes entre SGBDs e a evolução para o incremento da carga de trabalho foi avaliado. Investigou-se também, por meio de análise estatística, mais precisamente a análise de variância (ANOVA), o impacto das fontes de variação nas métricas, tempo de execução e consumo de energia dos cenários analisados.

Avaliou-se também a correlação entre o tempo de execução e o consumo de energia em cada comando por SGBD, considerando os distintos cenários de utilização verifica-se que há diferença significativa entre eles. A interação (SGBD * Comando) é a maior fonte de variação no consumo de energia em todas as cargas de trabalho (1.000, 5.000 e 10.000 operações), este fato não se repetiu quando a métrica foi o tempo de execução onde na carga de trabalho de 10.000 operações onde o SGBD a maior fonte de variação. As diferenças de consumo de

energia encontradas justificam a continuidade de estudos em ambiente de *cluster* e outros espaços não avaliados neste trabalho.

6.1 LIMITAÇÕES

A avaliação de consumo de energia e desempenho de SGBDs NoSQL multimodelos enfrenta diversas limitações. Uma delas é a heterogeneidade dos modelos de dados suportados, que exige abordagens específicas para cada modelo, aumentando a complexidade da avaliação. Além disso, a variabilidade das configurações de hardware e software, juntamente com a sensibilidade às cargas de trabalho, dificulta a comparação entre sistemas e pode resultar em conclusões imprecisas sobre o desempenho e eficiência energética dos SGBDs avaliados. Superar essas limitações requer o uso de métricas adequadas, planejamento cuidadoso das avaliações e transparência na divulgação dos resultados, além do desenvolvimento de ferramentas especializadas e padrões de avaliação claros.

6.2 TRABALHOS FUTUROS

Este trabalho apresenta resultados que motivam a realização de estudos que ofereçam continuidade a esta pesquisa para contribuição no cenário de eficiência no consumo de energia. Os seguintes itens são considerados para a avaliação em trabalhos futuros.

- Replicar estes experimentos em ambiente de *cluster*, de modo a comparar os resultados em *cluster* com os resultados em ambiente de um único nó de banco de dados;
- Avaliar a influência dos componentes de *hardware* (processador, memória, disco rígido) no tempo de execução e o consumo de energia de SGBDs NoSQL;
- Criar modelos de simulação do tempo de execução e consumo de energia com base no ambiente de execução do SGBD NoSQL.

REFERÊNCIAS

ABRAMOVA, V.; BERNARDINO, J.; FURTADO, P. Experimental evaluation of nosql databases. *International Journal of Database Management Systems*, Academy & Industry Research Collaboration Center (AIRCC), v. 6, n. 3, p. 1, 2014.

ACS217-20A. *Documentação Técnica*. 2022. Disponível em: <https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf>. Acessado em: 01/10/2022.

ARANGO-DB. *Site Oficial, v3.4.11*. 2022. Disponível em: <https://www.arangodb.com/docs/3.4/index.html>. Acessado em: 01/10/2022.

ARAÚJO, C. G. *Avaliação do consumo de energia em sistemas de gerenciamento de banco de dados NoSQL*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2016.

BARROS, J.; CALLOU, G.; GONçALVES, G. Análise integrada de desempenho e consumo de energia em sistemas de armazenamento de dados distribuídos. In: *Proceedings of the 15th Clouds and Applications Workshop*. Porto Alegre, RS, Brasil: SBC, 2018. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/wcga/article/view/2551>.

BICEVSKA, Z.; ODITIS, I. Towards nosql-based data warehouse solutions. *Procedia Computer Science*, v. 104, p. 104–111, 2017. ISSN 1877-0509. ICTE 2016, Riga Technical University, Latvia. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050917300819>.

BORAL, H.; DEWITT, D. J. A methodology for database system performance evaluation. In: *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 1984. (SIGMOD '84), p. 176–185. ISBN 0897911288. Disponível em: <https://doi.org/10.1145/602259.602283>.

BUKH, P. N. D. *The art of computer systems performance analysis, techniques for experimental design, measurement, simulation and modeling*. [S.l.]: JSTOR, 1992.

DAVOUDIAN, A.; CHEN, L.; LIU, M. A survey on nosql stores. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 2, apr 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3158661>.

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.

EDLICH, S. *List of NoSQL Databases*. 2022. Disponível em: <http://nosql-database.org>. Acessado em: 01/10/2022.

FLORES, A.; RAMíREZ, S.; TOASA, R.; VARGAS, J.; BARRIONUEVO, R. U.; LAVIN, J. M. Performance evaluation of nosql and sql queries in response time for the e-government. In: *2018 International Conference on eDemocracy eGovernment (ICEDEG)*. [S.l.: s.n.], 2018. p. 257–262.

FORD, N. *Polyglot Programming. Meme agora*. 2006. Disponível em: <http://memeagora.blogspot.com.br/2006/12/polyglot-programming.html>. Acessado em: 01/10/2022.

GOMES, C.; TAVARES, E.; JUNIOR, M. de O. Energy consumption evaluation of nosql dbmss. In: *Anais do XV Workshop em Desempenho de Sistemas Computacionais e de Comunicação*. Porto Alegre, RS, Brasil: SBC, 2016. p. 2828–2838. ISSN 2595-6167. Disponível em: <https://sol.sbc.org.br/index.php/wperformance/article/view/9729>.

GUNAWAN, R.; RAHMATULLOH, A.; DARMAWAN, I. Performance evaluation of query response time in the document stored nosql database. In: *2019 16th International Conference on Quality in Research (QIR): International Symposium on Electrical and Computer Engineering*. [S.l.: s.n.], 2019. p. 1–6.

IMRAN, S.; MAHMOOD, T.; MORSHED, A.; SELLIS, T. 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.

JAKKULA, P. Hbase or cassandra? a comparative study of nosql database performance. *International Journal of Scientific and Research Publications (IJSRP)*, v. 10, p. p9999, 03 2020.

JOBS, S. *Discurso na Universidade de Stanford, 2005*. 2005.

LILJA, D. J. *Measuring Computer Performance: A Practitioner's Guide*. USA: Cambridge University Press, 2000. ISBN 0521641055.

LISSANDRINI, M.; BRUGNARA, M.; VELEGRAKIS, Y. Beyond macrobenchmarks: Microbenchmark-based graph database evaluation. *Proc. VLDB Endow.*, VLDB Endowment, v. 12, n. 4, p. 390–403, dec 2018. ISSN 2150-8097. Disponível em: <https://doi.org/10.14778/3297753.3297759>.

LIU, Z. H.; LU, J.; GAWLICK, D.; HELSKYAHO, H.; POGOSSIANTS, G.; WU, Z. Multi-model database management systems - a look forward. In: GADEPALLY, V.; MATTSON, T.; STONEBRAKER, M.; WANG, F.; LUO, G.; TEODORO, G. (Ed.). *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Cham: Springer International Publishing, 2019. p. 16–29. ISBN 978-3-030-14177-6.

LU, J.; HOLUBOVá, I. Multi-model databases: A new journey to handle the variety of data. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 52, n. 3, jun 2019. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3323214>.

MACAK, M.; STOVCIK, M.; BUHNOVA, B.; MERJAVY, M. How well a multi-model database performs against its single-model variants: Benchmarking orientdb with neo4j and mongodb. In: *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. [S.l.: s.n.], 2020. p. 463–470.

MAHAJAN, D.; BLAKENEY, C.; ZONG, Z. Improving the energy efficiency of relational and nosql databases via query optimizations. *Sustainable Computing: Informatics and Systems*, v. 22, p. 120–133, 2019. ISSN 2210-5379. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2210537918301112>.

MARTINS, P.; ABBASI, M.; SÁ, F. A study over nosql performance. In: ROCHA, Á.; ADELI, H.; REIS, L. P.; COSTANZO, S. (Ed.). *New Knowledge in Information Systems and Technologies*. Cham: Springer International Publishing, 2019. p. 603–611. ISBN 978-3-030-16181-1.

MARTINS, P.; TOMÉ, P.; WANZELLER, C.; SÁ, F.; ABBASI, M. Nosql comparative performance study. In: ROCHA, Á.; ADELI, H.; DZEMYDA, G.; MOREIRA, F.; CORREIA, A. M. R. (Ed.). *Trends and Applications in Information Systems and Technologies*. Cham: Springer International Publishing, 2021. p. 428–438. ISBN 978-3-030-72651-5.

MEASUREMENT. *Documentação Técnica*. 2022. Disponível em: <https://www.cin.ufpe.br/~eagt>. Acessado em: 01/10/2022.

MEGA-2560-CH340. *Documentação Técnica*. 2022. Disponível em: <https://docs.arduino.cc/hardware/mega-2560>. Acessado em: 01/10/2022.

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

MINAS, L.; ELLISON, B. *Energy efficiency for information technology: How to reduce power consumption in servers and data centers*. [S.l.]: Intel press, 2009.

MINIPA-ET-409. *Documentação Técnica*. 2022. Disponível em: <https://www.minipa.com.br/images/Manual/ET-4091-1102-BR.pdf>. Acessado em: 01/10/2022.

MONGO-DB. *Site Oficial*. 2022. Disponível em: <https://www.mongodb.com/docs/manual/core/document>. Acessado em: 01/10/2022.

MONTGOMERY, D. C.; RUNGER, G. C. *Applied statistics and probability for engineers*. 7. ed. Nashville, TN: John Wiley & Sons, 2020.

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

OLIVEIRA, F. R.; CURA, L. del V. Performance evaluation of nosql multi-model data stores in polyglot persistence applications. In: *Proceedings of the 20th International Database Engineering & Applications Symposium*. New York, NY, USA: Association for Computing Machinery, 2016. (IDEAS '16), p. 230–235. ISBN 9781450341189. Disponível em: <https://doi.org/10.1145/2938503.2938518>.

ORACLE. *Integração de aplicações e integração de dados*. 2022. Disponível em: <https://www.oracle.com/br/integration>. Acessado em: 01/10/2022.

ORIENT-DB. *Site Oficial, v3.2.8*. 2022. Disponível em: <http://orientdb.com/docs/3.0.x>. Acessado em: 01/10/2022.

OSEMWEGIE, O.; OKOKPUJIE, K.; NKORDEH, N.; NDUJIUBA, C.; JOHN, S.; STANLEY, U. Performance benchmarking of key-value store nosql databases. *International Journal of Electrical and Computer Engineering (IJECE)*, v. 8, n. 6, p. 5333–5341, 2018.

PEREIRA, D. A.; MORAIS, W. O. de; FREITAS, E. P. de. Nosql real-time database performance comparison. *International Journal of Parallel, Emergent and Distributed Systems*, Taylor Francis, v. 33, n. 2, p. 144–156, 2018.

POUYANFAR, S.; YANG, Y.; CHEN, S.-C.; SHYU, M.-L.; IYENGAR, S. S. Multimedia big data analytics: A survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 1, jan 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3150226>.

REDIS. *Site Oficial*. 2022. Disponível em: <https://redis.com>. Acessado em: 01/10/2022.

ROY-HUBARA, N.; SHOVAL, P.; STURM, A. Selecting databases for polyglot persistence applications. *Data Knowledge Engineering*, v. 137, p. 101950, 2022. ISSN 0169-023X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0169023X21000744>.

SADALAGE, P. J.; FOWLER, M. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. 1st. ed. [S.l.]: Addison-Wesley Professional, 2012. ISBN 0321826620.

SEGHIER, N. B.; KAZAR, O. Performance benchmarking and comparison of nosql databases: Redis vs mongodb vs cassandra using ycsb tool. In: *2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI)*. [S.l.: s.n.], 2021. p. 1–6.

SHAH, M.; KOTHARI, A.; PATEL, S. A comprehensive survey on energy consumption analysis for nosql. *Scalable Computing: Practice and Experience*, v. 23, n. 1, p. 35–50, 2022.

TAN, J.; GHANEM, T.; PERRON, M.; YU, X.; STONEBRAKER, M.; DEWITT, D.; SERAFINI, M.; ABOULNAGA, A.; KRASKA, T. Choosing a cloud dbms: Architectures and tradeoffs. *Proc. VLDB Endow.*, VLDB Endowment, v. 12, n. 12, p. 2170–2182, aug 2019. ISSN 2150-8097. Disponível em: <https://doi.org/10.14778/3352063.3352133>.

TANG, E.; FAN, Y. Performance comparison between five nosql databases. In: *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. [S.l.: s.n.], 2016. p. 105–109.

TRIOLA, M. F. *Introdução à estatística*. [S.l.]: Livros Técnicos e Científicos, 2013. ISBN 9788521611547.

WANG, J.; XU, C.; ZHANG, J.; ZHONG, R. Big data analytics for intelligent manufacturing systems: A review. *Journal of Manufacturing Systems*, v. 62, p. 738–752, 2022. ISSN 0278-6125. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0278612521000601>.

WEINBERGER, C. *Index Free Adjacency or Hybrid Indexes for Graph Databases*. 2022. Disponível em: <https://www.arangodb.com/2016/04/index-free-adjacency-hybrid-indexes-graph-databases>. Acessado em: 01/01/2023.

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

ZMPT101B. *Documentação Técnica*. 2022. Disponível em: <https://innovatorsguru.com/wp-content/uploads/2019/02/ZMPT101B.pdf>. Acessado em: 01/10/2022.

APÊNDICE A – MEASUREMENT SERVER

O *Measurement Server* foi composto pelas etapas de aquisição dos sensores e ferramentas de prototipação, montagem do circuito embarcado, calibração dos sensores para ajuste de precisão e monitoramento para apuração de dados para análise de consumo. Como base para a prototipação, foi selecionado o microcontrolador Arduíno, o dispositivo funciona como um microcomputador convencional, pois possui um microprocessador, memórias RAM e flash, temporizadores e entre outros, para que se possa realizar diferentes tarefas ou ações com o mesmo, também programando suas funcionalidade e tarefas que são compiladas internamente no dispositivo.

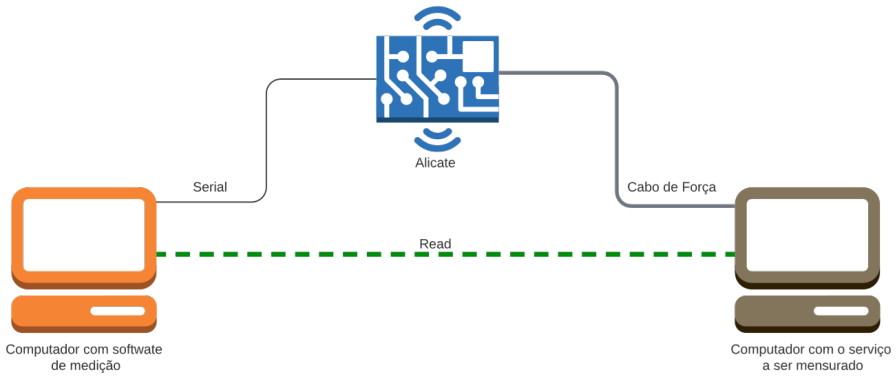


Figura 26 – Arquitetura de Medição de Energia Elétrica.

O microcontrolador pode ser integrado com muitos tipos de hardware, sendo eles sensores, outros computadores, motores entre outros e também pode comunicar-se com softwares, como um banco de dados ou até mesmo um servidor web, sendo trocar informações e registrar dados coletados pelo mesmo. O modelo utilizado foi o *Ethernet board*, pois além de ser programável como o Arduíno convencional, já possui as conexões necessárias para comunicação com a rede cabeada.

O sensor avaliado para capturar os dados da corrente elétrica dos aparelhos monitorados foi o sensor de corrente ACS712 AC/DC é um circuito de um sensor linear baseado em efeito Hall totalmente integrado. Este CI possui isolamento de tensão de 2,1kV RMS juntamente com um condutor de corrente de baixa resistência, podendo medir corrente contínua e corrente alternada. Ele Possui recursos de cancelamento de ruído e tempo de resposta muito alto. É

simplesmente um sensor de corrente que utiliza seu condutor para calcular e medir a quantidade de corrente aplicada, alcançando excelentes resultados.

Para a tensão, foi utilizado o sensor do tipo voltímetro transformador ZMPT101B, que suporta até 250 *Volts* (V), possui uma precisão de leitura de +-1% e também pode ser ajustado e calibrado diretamente com o microcontrolador Arduíno e seu potenciômetro acoplado na própria placa para ajustar a precisão da tensão de saída de uma tomada ou caso necessário de um dispositivo que está produzindo a energia conforme o *datasheet*.

A.1 DIAGRAMA DE SEQUÊNCIA

No diagrama de sequência (UML) é ilustrada a sequência das mensagens entre objetos em uma interação, onde e apresentado a sequência de mensagens transmitidas entre objetos. Exemplificando a linha de vida para um cenário de medição de um *benchmark*, representado pelo cliente, e um servidor.

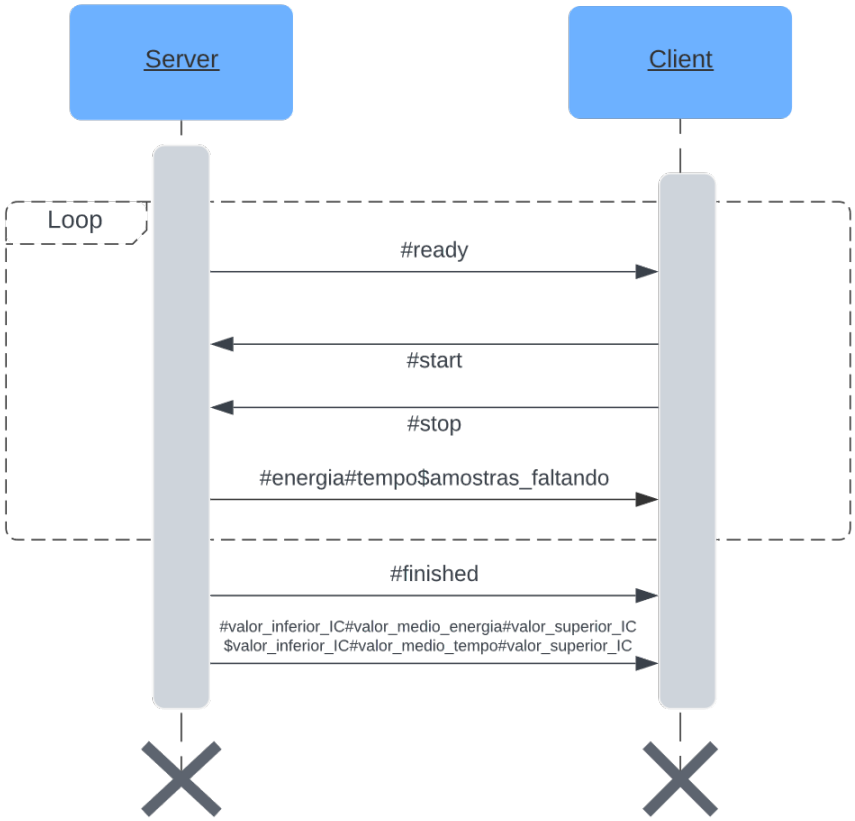


Figura 27 – Diagrama de Sequencia - Medição de energia elétrica

O diagrama de sequência 27 mostra os objetos e as mensagens entre os objetos, onde

fragmento de loop é usado para representar uma sequência repetitiva das interações.

Código Fonte 6 – Log de execução da carga de trabalho, servidor.

```
Process return code 0
#19.09967#3.9549999999999999$9

Process return code 0
#20.34978#4.1899999999999997$8
#ready

Process return code 0
#20.07941999999992#4.0449999999999997$7
#ready

Process return code 0
#20.09633999999998#4.0639999999999997$6

#ready
Process return code 0
#20.38299#4.1659999999999998$5

#ready
Process return code 0
#20.42420000000006#4.0269999999999975$4

#ready
Process return code 0
#19.91023999999998#4.0599999999999997$3

#ready
Process return code 0
#20.06813999999996#4.0999999999999998$2
#ready

Process return code 0
#19.9346#4.09299999999999997$1
#ready

Process return code 0
#20.57604#4.1989999999999999$0
#finished

19.800689,20.335024#20.067856499999998#20.067856499999998$
    [4.039699999999998#4.039699999999998#4.138399999999997
```

Código Fonte 7 – Log de execução da carga de trabalho com obtenção de amostras no Cliente.

```
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
```

A.2 CÓDIGO *BENCHMARK* PYTHON

A utilização do micro controlador foi uma ferramenta importante para realizar a coleta e o processamento de dados. Com este controlador foi possível programar na tecnologia Java por meio da comunicação USB. A elaboração da lógica de programação consistiu no emprego das linguagens de programação Python, e os dados adquiridos pelos sensores foram coletados pela linguagem Java e envidados para a aplicação em Python que abstraiu as informações e gerou informações de interesse da pesquisa.

Código Fonte 8 – Exemplo - Script de execução da *Workload*.

```
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"
        comand = comand + countPlus
        process = subprocess.Popen(comand, shell=True, stdout=subprocess.PIPE)
        process.wait()
        print("Process return code " + str(process.returncode))
        s.sendall("#stop\n".encode('ascii'))
        #resp = s.recv(1024)
        resp = x.readline()
        print(resp.decode('ascii'), end ="")
        resp = x.readline()
        print(resp.decode('ascii'))

        if resp.decode('ascii').find("#finished") > -1:
            break
    resp = x.readline()
    print(resp.decode('ascii'))
    s.close()
```

Código Fonte 9 – Exemplo - Script Workload.

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


# Yahoo! Cloud System Benchmark
# Workload A: Update heavy workload
#   Application example: Session store recording recent actions
```

```
#
#   Read/update ratio: 50/50
#   Default data size: 1 KB records (10 fields, 100 bytes each, plus key)
#   Request distribution: zipfian

recordcount=1000
operationcount=1000
workload=site.ycsb.workloads.CoreWorkload

readallfields=true

readproportion=0
updateproportion=0
scanproportion=0
insertproportion=0
writeproportion=0
readmodifywriteproportion=0

requestdistribution=zipfian
```

Através dos comandos no Código 8, conforme informado anteriormente, a ferramenta YCSB é executada apenas com os sistemas de interesse, com exemplo temos o ArangoDB. No qual o *Script* segue o diagrama de sequência 27.

APÊNDICE B – APRESENTAÇÃO DOS DADOS

Apresentação das amostras do resultado da execução de cada experimento apresentando neste trabalho, esta disponível em https://cin.ufpe.br/fmcf3/pesquisa/2023/amostras.