UNIVERSIDADE PAULISTA - UNIP

Cauê Fernando Viel

MONGO VS. CASSANDRA:
comparando o desempenho de sistemas gerenciadores de banco de dados NoSQL em operações CRUD

Limeira

2023

Cauê Fernando Viel

MONGO VS. CASSANDRA:
comparando o desempenho de sistemas gerenciadores de banco de dados NoSQL em operações CRUD

Trabalho de Conclusão de Curso apresentado à banca examinadora da Faculdade UNIP, como requisito parcial à obtenção do Bacharelado em Ciência da Computação sob a orientação do professor Me. Sergio Eduardo Nunes.

Limeira

2023

Cauê Fernando Viel

MONGO VS. CASSANDRA:
comparando o desempenho de sistemas gerenciadores de banco de dados NoSQL em operações CRUD

Trabalho de Conclusão de Curso apresentado à banca examinadora da Faculdade UNIP, como requisito parcial à obtenção do Bacharelado em Ciência da Computação sob a orientação do professor Me. Sergio Eduardo Nunes.

Aprovada em ___ de _____ de 2023.

BANCA EXAMINADORA

_____
Prof. Me. Sergio Eduardo Nunes (Orientador)

_____
Prof.

_____
Prof.

Dedico este trabalho a minha esposa, meus pais e meu irmão, pois, sem o apoio deles nunca chegaria aonde estou.

“*Você precisa confiar em algo – no seu instinto, destino, vida, karma, qualquer coisa. Essa ideia nunca me deixou na mão, e fez toda a diferença na minha vida*”.

Steve Jobs

RESUMO

Bancos de dados NoSQL vêm se tornando bastante populares no desenvolvimento de aplicações web e móveis devido principalmente à flexibilidade na modelagem e manipulação de dados. MongoDB e Cassandra são dois dos mais utilizados Sistemas de Gerenciamento de Banco de Dados (SGDB) NoSQL. Embora ambos compartilhem características, divergem em suas abordagens específicas e implementação — o MongoDB opta por um modelo orientado a documentos, enquanto o Cassandra adota uma estrutura orientada a colunas. À medida que a demanda por escalabilidade horizontal continua a crescer no contexto das aplicações web e móveis, torna-se cada vez mais crucial compreender a complexidade e desempenho de SGDB a fim de se selecionar o melhor sistema para um determinado cenário. Este trabalho apresenta um estudo comparativo entre MongoDB e Cassandra, focando nas diferenças de desempenho em operações CRUD. Com base em 11 experimentos controlados de desempenho com dados sintéticos, mostrou-se que o MongoDB é mais eficiente nas operações de inserção e recuperação, entretanto o Cassandra supera-o em operações de remoção e atualização. Portanto, ao selecionar entre os dois sistemas, é crucial considerar as necessidades específicas de cada aplicação.

Palavra-chave: NoSQL; banco de dados; Cassandra; MongoDB; CRUD.

ABSTRACT

NoSQL databases have become increasingly popular in web and mobile application development, primarily due to their flexibility in data modeling and manipulation. MongoDB and Cassandra are two of the most widely used NoSQL Database Management Systems (DBMS). Although both DBMS share some characteristics, they differ in their specific approaches and implementations — MongoDB adopts a document-oriented model, while Cassandra embraces a column-oriented structure. As the demand for horizontal scalability continues to grow in the context of web and mobile applications, it becomes increasingly crucial to understand the complexity and performance of DBMS in order to select the best system for a given scenario. This Graduation Project presents a comparative study between MongoDB and Cassandra, focusing on differences in CRUD (Create, Read, Update, Delete) operations' performance. Based on 11 controlled performance experiments with synthetic data, it was revealed that while MongoDB stands out in create and retrieve operations, Cassandra outperforms it in update and delete operations. Therefore, when choosing between these two systems, it is crucial to consider the specific needs of each application.

Key Words: NoSQL; database; Cassandra; MongoDB; CRUD.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CRUD	*Create, Read, Update and Delete*
NoSQL	*Not Only SQL*
SGDB	Sistemas de Gerenciamento de Banco de Dados
WSL	*Windows Subsystem for Linux*

SUMÁRIO

1 INTRODUÇÃO

O armazenamento e a gestão eficiente e dados são elementos cruciais para o sucesso de aplicações modernas. Com o crescimento exponencial do volume de informações geradas e a necessidade de acesso rápido e escalabilidade, surgiram novas tecnologias de banco de dados que visam atender a essas demandas de maneira eficaz. Entre essas tecnologias estão os dados NoSQL (Sadalage; Fowler, 2012).

MongoDB e Cassandra são exemplos proeminentes de bancos de dados NoSQL. MongoDB é o SBGB NoSQL mais utilizado, com um *market share* de 45% (6 Sense, 2023). Cassandra é um projeto *open source* mantido pela Apache Foundation baseado no proprietário Amazon DynamoDB, que é o terceiro SBGB NoSQL mais utilizado. Somados, o *mark share* do DynamoDB e do Cassandra é de 15%.

Apesar de tanto MongoDB e Cassandra apresentarem um modelo de dados flexíveis e terem sido desenvolvidas para lidar com a natureza não-estruturada e massiva dos dados gerados atualmente, promovendo alta disponibilidade, escalabilidade e desempenho para aplicações web e móveis, eles são intrinsicamente diferentes na representação interna dos dados armazenados. MongoDB é baseado em um modelo de documentos, enquanto Cassandra utiliza um modelo baseado em colunas. Tal diferença pode impactar diretamente no desempenho dos bancos de dados ao realizar operação de criação, leitura, atualização e remoção de dados (do inglês: *Create, Read, Update and Delete* - CRUD).

1.1 Objetivo

Como a seleção do banco de dados é uma das mais importantes decisões no desenvolvimento de uma aplicação, o objetivo principal desse trabalho é comparar o desempenho do MongoDB e do Cassandra a fim de determinar qual dos dois Sistemas de Gerenciamento de Banco de Dados (SGDBs) apresenta uma melhor capacidade em lidar com diferentes cargas de trabalho, volume de dados para cada uma das operações CRUD.

Com base nos resultados das comparações, pretende-se elencar as principais vantagens e desvantagens de cada sistema em relação a desempenho e escalabilidade. Espera-se que os resultados deste trabalho possam orientar

desenvolvedores, engenheiros de dados e empresas na tomada de decisões estratégicas em relação a solução de banco dados mais adequado em diferentes cenários e aplicações adequados às suas necessidades especificas.

1.2 Justificativa

Bancos de dados NoSQL estão se tornando bastante populares, especialmente em aplicações web e móveis, representando 25% de todo *mark share* de banco de dados no geral e demostrando crescimento contínuo (Whitfield, 2020). O MongoDB e o Cassandra são duas soluções NoSQL amplamente utilizados, mas que diferem suas arquiteturas e recursos, o que pode levar diferentes desempenhos em diferentes situações.

1.3 Metodologia

A primeira etapa deste trabalho foi estabelecer os objetivos específicos da comparação entre os dois bancos de dados comparados, incluindo os cenários de uso, carga de trabalho e tipos de operações que serão realizados.

A segunda etapa foi preparar um ambiente controlado e homogêneo para execução os testes, incluindo a definição dos requisitos mínimos de software e de hardware e posterior instalação e configuração dos bancos de dados MongoDB e Cassandra em um computador com configurações adequadas.

A terceira etapa foi definir um plano de teste detalhado, que incluía as etapas e serem executadas para permitir a comparação de operações CRUD de ambos os SGDB. Para isso foram desenvolvidos *scripts* para a realização dos testes nos cenários de maneira semiautomática.

A quarta etapa compreende a execução dos testes e coleta dos dados sobre o desempenho do sistema.

Na quinta etapa foram analisados os dados coletados durante os testes a fim de se comparar o desempenho de MongoDB e Cassandra nos diferentes cenários de testes. Para isso utilizou-se estatística descritiva e modelos numéricos para inferência de curvas de tendencia.

Na sexta etapa identificou-se as vantagens e desvantagens de cada SGBD com base na análise de cada sistema em relação a desempenho e escalabilidade.

Na última etapa será apresentada uma conclusão sobre o desempenho do MongoDB e Cassandra, destacando as principais descobertas e recomendações para desenvolvedores e empresas que desejam implementar um banco de dados NoSQL.

2 CONCEITOS TEORICOS E TECNOLOGIAS

Banco de dados NoSQL (do inglês: *Not Only SQL*) é um tipo de banco de dados que difere dos bancos de dados relacionais tradicionais em termos de estrutura, esquema e operações de consulta. Segundo Sadalage e Fowler (2012), em seu livro "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence", ao contrário dos bancos de dados relacionais, que utilizam tabelas e esquemas rígidos para armazenar e recuperar dados, os bancos de dados NoSQL utilizam diferentes estruturas de dados para lidar com dados não estruturados, sem um esquema pré-definido e com alta escalabilidade horizontal.

De acordo com Hueske, Lehmann e Rothermel (2012 *apud* Catarino, 2023), os bancos de dados NoSQL podem ser classificados em quatro categorias principais: bancos de dados de chave-valor, bancos de dados de documentos, bancos de dados de colunas e bancos de dados de grafos.

Conforme Sadalage e Fowler (2012) discutem em "NoSQL Distilled," os bancos de dados de chave-valor são os mais simples dos bancos de dados NoSQL. Eles armazenam os dados em pares chave-valor, em que cada valor é associado a uma chave única. Esses sistemas são usados principalmente para armazenar dados simples e de alta velocidade de acesso, como caches, sessões de usuários e catálogos de produtos.

No que tange aos bancos de dados de documentos, esses sistemas, conforme mencionado por Hueske, Lehmann e Rothermel (2012 *apud* Catarino, 2023), são utilizados para armazenar dados semiestruturados e não estruturados. Cada documento é uma unidade independente de dados que pode ter sua própria estrutura. Tais bancos de dados são adequados para o armazenamento de dados que mudam frequentemente, como registros de log, conteúdo da web e dados de jogos.

Sadalage e Fowler (2012), no livro "NoSQL Distilled," também explicam que os bancos de dados de colunas são projetados para armazenar grandes conjuntos de dados e são utilizados principalmente para análise de dados. Ao contrário dos bancos de dados relacionais, esses sistemas armazenam dados de forma orientada a colunas, agrupando colunas com valores semelhantes juntas. Eles são adequados para armazenar dados com estruturas complexas, como registros de transações, registros de cliques em anúncios e dados de sensores.

Quanto aos bancos de dados de grafos, Hueske, Lehmann e Rothermel (2012 *apud* Catarino, 2023) destacam que eles são utilizados para armazenar dados na forma de grafos, que consistem em nós e arestas que conectam esses nós. Esses tipos de bancos de dados são apropriados para modelar dados altamente relacionais, como redes sociais, sistemas de recomendação e sistemas de detecção de fraudes.

Como afirmam Sadalage e Fowler (2012) em "NoSQL Distilled," os bancos de dados NoSQL são projetados para serem escaláveis, tolerantes a falhas e de alta disponibilidade, sendo amplamente empregados em aplicativos que demandam grande escalabilidade, como serviços de redes sociais, sistemas de gerenciamento de conteúdo, jogos online e análise de big data.

2.1 MongoDB

O MongoDB é um sistema de gerenciamento de banco de dados NoSQL (Not Only SQL) orientado a documentos. De acordo com Chodorow e Dirolf (2013), ao contrário dos bancos de dados relacionais tradicionais, o MongoDB não usa tabelas para armazenar dados, mas sim um formato de documentos flexíveis, conhecido como BSON (Binary JSON).

Conforme mencionado por Chodorow e Dirolf (2013), cada documento no MongoDB é uma unidade independente de dados que pode ter uma estrutura diferente. Isso permite que os desenvolvedores armazenem dados não estruturados e sem esquema definido. A estrutura de documentos dinâmica oferece maior flexibilidade e agilidade no desenvolvimento de aplicativos, permitindo que as aplicações evoluam rapidamente, sem a necessidade de esquematizar os dados previamente.

O MongoDB é amplamente utilizado em aplicativos da Web, móveis e outros aplicativos que requerem alta escalabilidade e desempenho. Segundo Merriman, Horowitz e Ryan (2010 *apud* Matiola, 2023), o banco de dados possui recursos avançados de indexação, consultas, agregação, escalabilidade horizontal e redundância, garantindo alta disponibilidade e tolerância a falhas. Além disso, o MongoDB é compatível com várias linguagens de programação e possui uma comunidade ativa e um ecossistema de ferramentas e serviços robustos.

2.2 Conceito do banco Cassandra

O Apache Cassandra é descrito como "um sistema de gerenciamento de banco de dados NoSQL distribuído, de alta disponibilidade e escalabilidade linear" (Lakshman; Malik, 2010, p. 35). Desenvolvido inicialmente pelo Facebook em 2008, o sistema agora é mantido pela Apache Software Foundation.

De acordo com Lakshman e Malik (2010, p. 36), o Cassandra foi concebido para suprir as demandas de aplicações que necessitam de "alta disponibilidade, tolerância a falhas e escalabilidade linear", como é o caso de aplicações de comércio eletrônico, serviços financeiros e redes sociais. O sistema distribui dados entre diversos servidores, possibilitando um crescimento proporcional à demanda do aplicativo.

A abordagem do Cassandra é baseada em colunas, diferenciando-se dos bancos de dados relacionais por armazenar dados nesse formato. Como mencionado por Lakshman e Malik (2010), o Cassandra também adota um modelo de replicação de dados, onde a replicação ocorre em múltiplos servidores para garantir alta disponibilidade e tolerância a falhas. Adicionalmente, o sistema oferece suporte a consultas com semelhanças ao SQL, permitindo que os usuários executem consultas familiares para recuperar dados.

Uma característica notável do Cassandra é sua flexibilidade em relação à consistência dos dados. De acordo com Lakshman e Malik (2010), os usuários têm a possibilidade de configurar níveis personalizados de consistência, alcançando um equilíbrio entre disponibilidade e consistência dos dados.

O Cassandra tem uma ampla gama de aplicações, incluindo serviços de mensagens instantâneas, análise de dados e gerenciamento de conteúdo (Lakshman; Malik, 2010). Sua capacidade de lidar com volumes substanciais de dados e proporcionar alta disponibilidade o torna uma escolha popular em cenários de alta escala.

3 AMBIENTE DE TESTES

Para a realização dos testes foi utilizada um computador com processador Inter® Core™ I7- 4510OU com 4 threads 200Ghz, com 8 Gigabytes de memória RAM e uma SSD de 250GB. O sistema operacional utilizado foi o Windows 64 bits na edição Windows 10 Home na versão 22H2, com *Windows Subsystem for Linux* (WSL) para Ubuntu na versão 22.04.

Optou-se por este ambiente porque a maior parte do *backends* de aplicações moveis e web atualmente está localizado em plataformas Cloud, como a Amazon AWS, onde normalmente encontram-se configurações modestas de processamento e memória. Além disso, faz-se necessário utilizar um processador com múltiplos núcleos para validar o paralelismo e o SSD devido às intensivas operações escrita e leitura em disco nos testes a serem realizados.

Também foi utilizado Docker na versão 24.0.6 para facilitar a replicação e reinicialização do ambiente graças ao conceito de contêiners. Foi instalado e configurado MongoDB em sua versão 7.0,1 e Apache Cassandra em sua versão 4.1.3. Também foi utilizado a linguagem de programação Python com sua versão 3.11.1 para escrita e execução dos *scripts* de teste.

3.1 Estrutura dos Testes

Foi necessário criar *scripts* de testes diferentes para cada um dos bancos de dados devido às suas características e as bibliotecas disponíveis para a linguagem Python. Porém ambos os *scripts* compartilham a mesma estrutura geral, descrita em pseudocódigo como mostra a Figura 1:

Figura 1 – Pseudocódigo

```
// Importar bibliotecas necessárias
importar random
importar string
importar time
importar json
// Criar função para gerar uma string aleatória
função gerar_string_aleatória(comprimento):
    retorna juntar_caracteres(random.escolher(string.ascii_lowercase) para cada _ no intervalo(comprimento))
// Criar função para realizar operações CRUD
função operacoes_crud(num_registros):
    tempos = {}
    // Inserção de dados
    tempo_insercao = medir_tempo_execução(lambda: inserir_dados_amostra(num_registros))
    tempos["insercao"] = tempo_insercao
    // Recuperação de todos os registros
    tempo_recuperacao = medir_tempo_execução(recuperar_todos_registros)
    tempos["recuperacao"] = tempo_recuperacao
    // Atualização de um registro
    tempo_atualizacao = medir_tempo_execução(atualizar_registro)
    tempos["atualizacao"] = tempo_atualizacao
    // Exclusão de um registro
    tempo_exclusao = medir_tempo_execução(excluir_registro)
    tempos["exclusao"] = tempo_exclusao
    retornar tempos
// Função para medir o tempo de execução
função medir_tempo_execução(funcao):
    tempo_inicial = tempo_atual()
    funcao()
    tempo_final = tempo_atual()
    retorna tempo_final - tempo_inicial
// Função para inserir dados de amostra
função inserir_dados_amostra(num_registros):
    para cada _ no intervalo(num_registros):
        dados = {
            "nome": gerar_string_aleatória(10),
            "idade": random.inteiro_entre(18, 60),
            "email": concatenar(gerar_string_aleatória(8), "@example.com")
        } // Aqui seria a chamada para inserir os dados no banco de dados
// Função para recuperar todos os registros
função recuperar_todos_registros():     // Aqui seria a chamada para recuperar todos os registros do banco de dados
// Função para atualizar um registro
função atualizar_registro():     // Aqui seria a chamada para atualizar um registro no banco de dados
// Função para excluir um registro
função excluir_registro():     // Aqui seria a chamada para excluir um registro do banco de dados

se __nome__ == "__principal__":
    num_registros = 1000  // Número de registros de amostra a serem inseridos
    // Realizar operações CRUD e obter tempos de execução
    tempos_execucao = operacoes_crud(num_registros)
    // Imprimir tempos de execução
    para cada operacao, tempo no tempos_execucao:
        imprimir("{operacao.capitalize()} levou {tempo:.4f} segundos.")
    // Armazenar tempos de execução em um JSON
    com abrir('tempos_execucao.json', 'w') como arquivo_json:
        json.dump(tempos_execucao, arquivo_json)
```

Fonte: o autor.

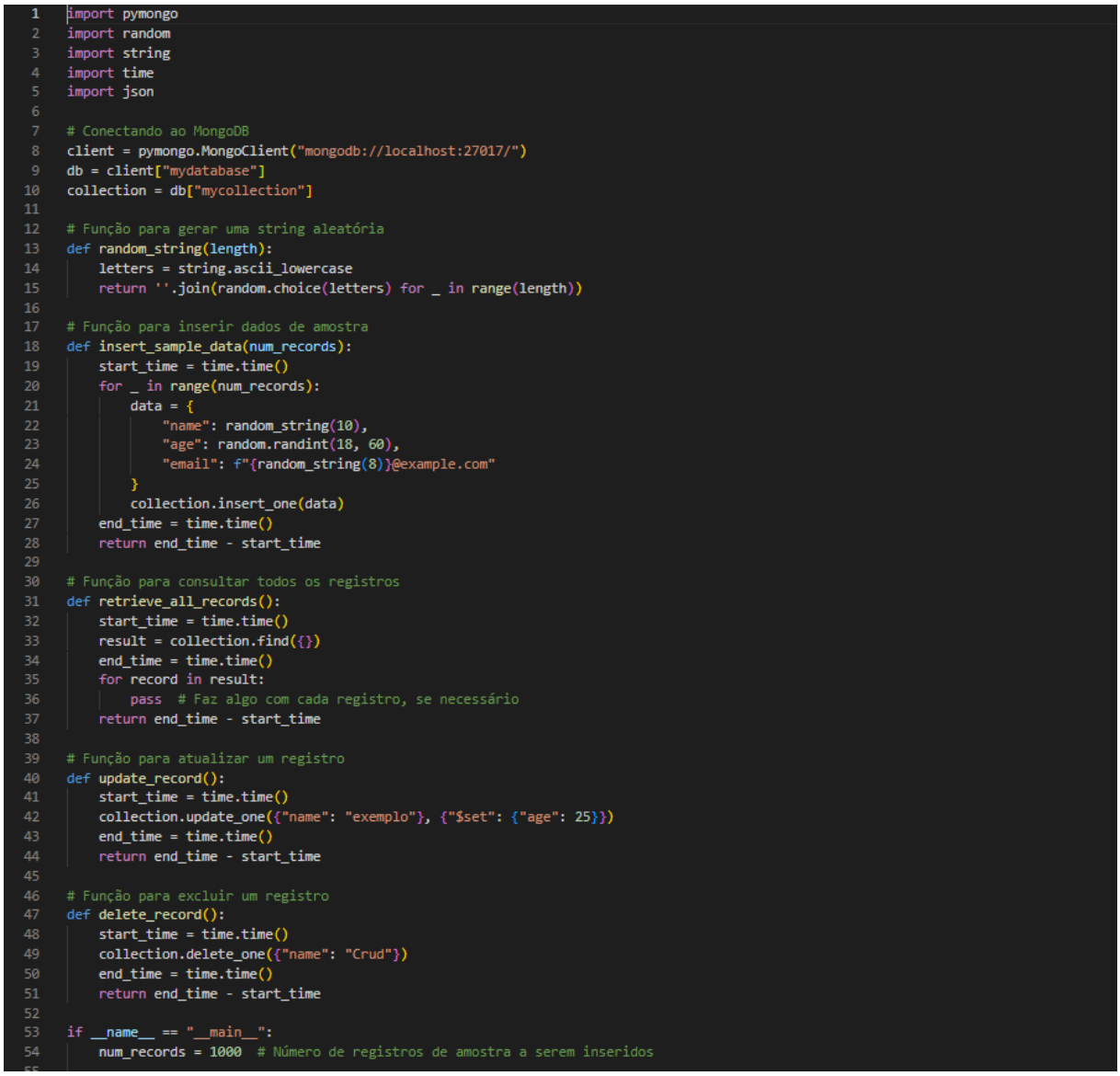
- **Função gerar_string_aleatória**: Essa função gera uma string aleatória de um determinado comprimento usando letras minúsculas do alfabeto;
- **Função operacoes_crud**: Essa função realiza operações CRUD (Create, Read, Update, Delete) em um banco de dados simulado. Chama funções para inserir dados, recuperar todos os registros, atualizar um registro e excluir um registro. Também mede o tempo de execução de cada operação e armazena os tempos em um dicionário;

- **Função time_execucao**: Uma função utilitária que mede o tempo de execução de uma função passada como argumento;
- **Função inserir_dados_amostra**: Gera dados de amostra (nome, idade, e-mail) e simula a inserção desses dados no banco de dados. No entanto, a implementação real da inserção está ausente no código;
- **Função recuperar_todos_registros**: Simula a recuperação de todos os registros do banco de dados. A implementação real da recuperação também está ausente;
- **Função atualizar_registro**: Simula a atualização de um registro no banco de dados. A implementação real da atualização está ausente;
- **Função excluir_registro**: Simula a exclusão de um registro do banco de dados. A implementação real da exclusão está ausente;
- **Bloco Principal (__name__ == "__main__")**: Define o número de registros de amostra a serem inseridos (num_registros = 1000); Realiza operações CRUD chamando a função operacoes_crud com o número de registros de amostra; Imprime os tempos de execução para cada operação; Armazena os tempos de execução em um arquivo JSON chamado tempos_execucao.json.

3.2 Script MongoDB

Este script, desenvolvido em Python, tem como objetivo executar operações de CRUD no MongoDB.

Figura 2 – Primeira parte do Código em Python fazer o CRUD em MongoDB



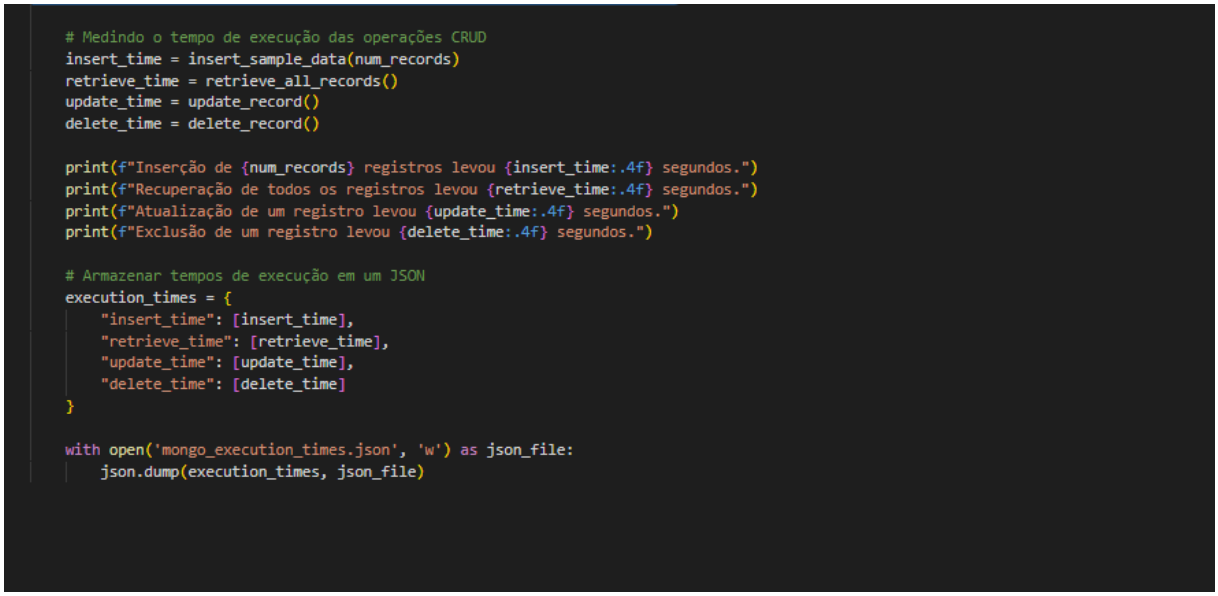
```python
import pymongo
import random
import string
import time
import json

# Conectando ao MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["mydatabase"]
collection = db["mycollection"]

# Função para gerar uma string aleatória
def random_string(length):
    letters = string.ascii_lowercase
    return ''.join(random.choice(letters) for _ in range(length))

# Função para inserir dados de amostra
def insert_sample_data(num_records):
    start_time = time.time()
    for _ in range(num_records):
        data = {
            "name": random_string(10),
            "age": random.randint(18, 60),
            "email": f"{random_string(8)}@example.com"
        }
        collection.insert_one(data)
    end_time = time.time()
    return end_time - start_time

# Função para consultar todos os registros
def retrieve_all_records():
    start_time = time.time()
    result = collection.find({})
    end_time = time.time()
    for record in result:
        pass  # Faz algo com cada registro, se necessário
    return end_time - start_time

# Função para atualizar um registro
def update_record():
    start_time = time.time()
    collection.update_one({"name": "exemplo"}, {"$set": {"age": 25}})
    end_time = time.time()
    return end_time - start_time

# Função para excluir um registro
def delete_record():
    start_time = time.time()
    collection.delete_one({"name": "Crud"})
    end_time = time.time()
    return end_time - start_time

if __name__ == "__main__":
    num_records = 1000  # Número de registros de amostra a serem inseridos
```

Fonte: o autor.

- **Conexão ao MongoDB:** Estabelece conexão com o MongoDB local, utilizando o banco de dados "mydatabase" e a coleção "mycollection";
- **Função de Geração de String Aleatória:** Define uma função random_string para gerar strings aleatórias de um determinado comprimento;
- **Função de Inserção de Dados de Amostra:** A função insert_sample_data insere um número especificado de registros de amostra na coleção, cada um contendo um nome, idade e email aleatórios;
- **Função de Consulta de Todos os Registros:** A função retrieve_all_records consulta todos os registros na coleção e mede o tempo de execução;

- **Função de Atualização de um Registro:** A função update_record atualiza um registro na coleção, alterando a idade para 25;
- **Função de Exclusão de um Registro:** A função delete_record exclui um registro da coleção com base no nome;
- **Execução Principal:** O código principal, dentro do bloco if __name__ == "__main__":, define o número de registros de amostra a serem inseridos (1000, no exemplo).

Figura 3 – Segunda parte do Código em Python fazer o CRUD em MongoDB



```python
# Medindo o tempo de execução das operações CRUD
insert_time = insert_sample_data(num_records)
retrieve_time = retrieve_all_records()
update_time = update_record()
delete_time = delete_record()

print(f"Inserção de {num_records} registros levou {insert_time:.4f} segundos.")
print(f"Recuperação de todos os registros levou {retrieve_time:.4f} segundos.")
print(f"Atualização de um registro levou {update_time:.4f} segundos.")
print(f"Exclusão de um registro levou {delete_time:.4f} segundos.")

# Armazenar tempos de execução em um JSON
execution_times = {
    "insert_time": [insert_time],
    "retrieve_time": [retrieve_time],
    "update_time": [update_time],
    "delete_time": [delete_time]
}

with open('mongo_execution_times.json', 'w') as json_file:
    json.dump(execution_times, json_file)
```

Fonte: o autor.

- **Medição de Tempos de Execução:** Chama as funções para inserir, recuperar, atualizar e excluir registros no MongoDB, medindo o tempo de execução de cada operação;
- **Impressão dos Tempos de Execução:** Imprime os tempos de execução formatados para cada operação no console;
- **Armazenamento dos Tempos em um Arquivo JSON:** Cria um dicionário com os tempos de execução de cada operação. Armazena esses tempos em um arquivo JSON chamado 'mongo_execution_times.json' para análise posterior.

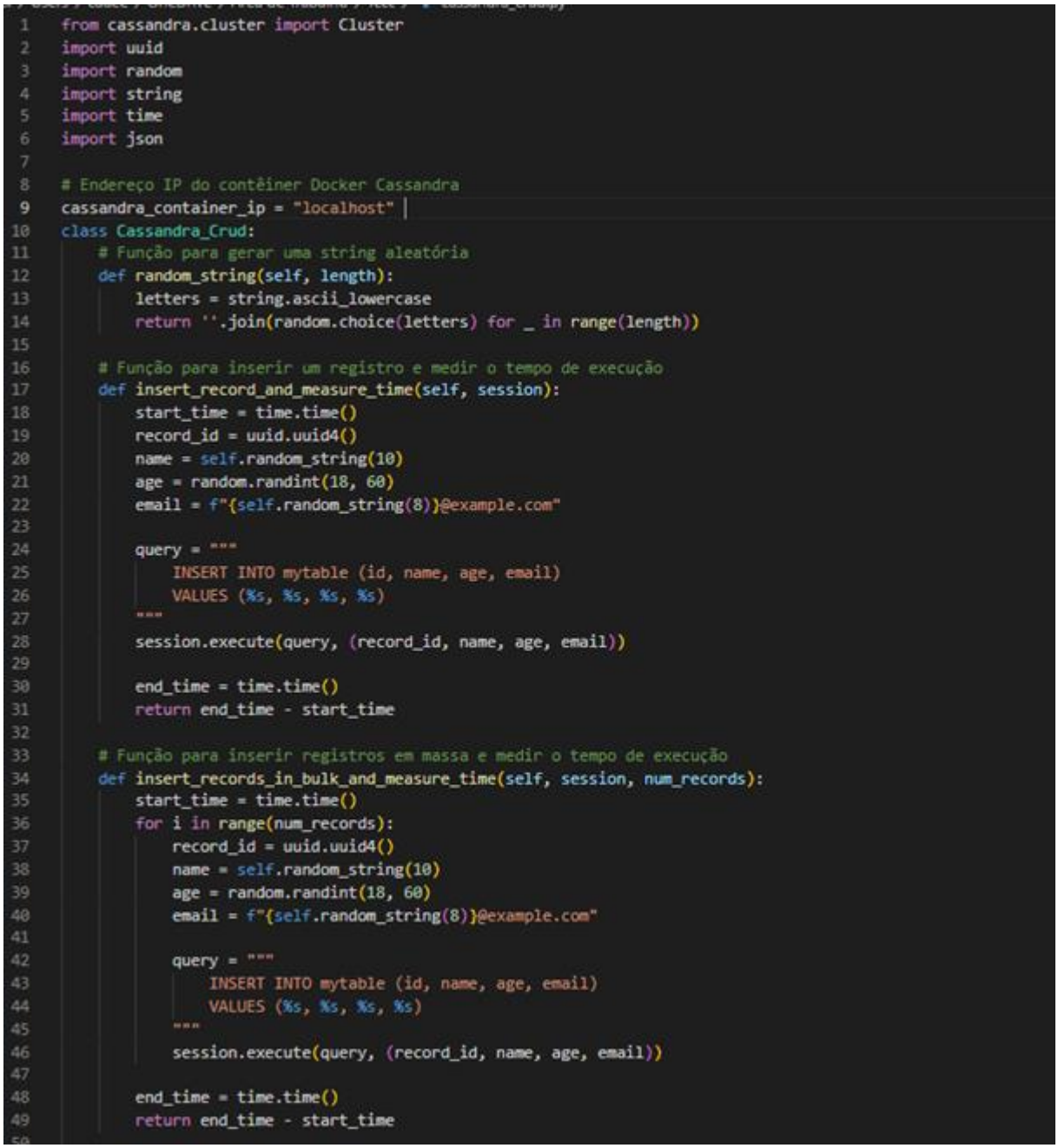
Em resumo: Este código em Python utiliza o driver pymongo para interagir com um banco de dados MongoDB, realizando operações CRUD. Ele mede o tempo de execução de inserção, consulta, atualização e exclusão de registros, imprime esses

tempos no console e os armazena em um arquivo JSON chamado 'mongo_execution_times.json' para análise posterior.

3.3 Script Cassandra

Este script, desenvolvido em Python, tem como objetivo executar operações de CRUD no Cassandra, como pode ser observado na Figura 4.

Figura 4 – Primeira parte do Código em Python fazer o CRUD em Cassandra



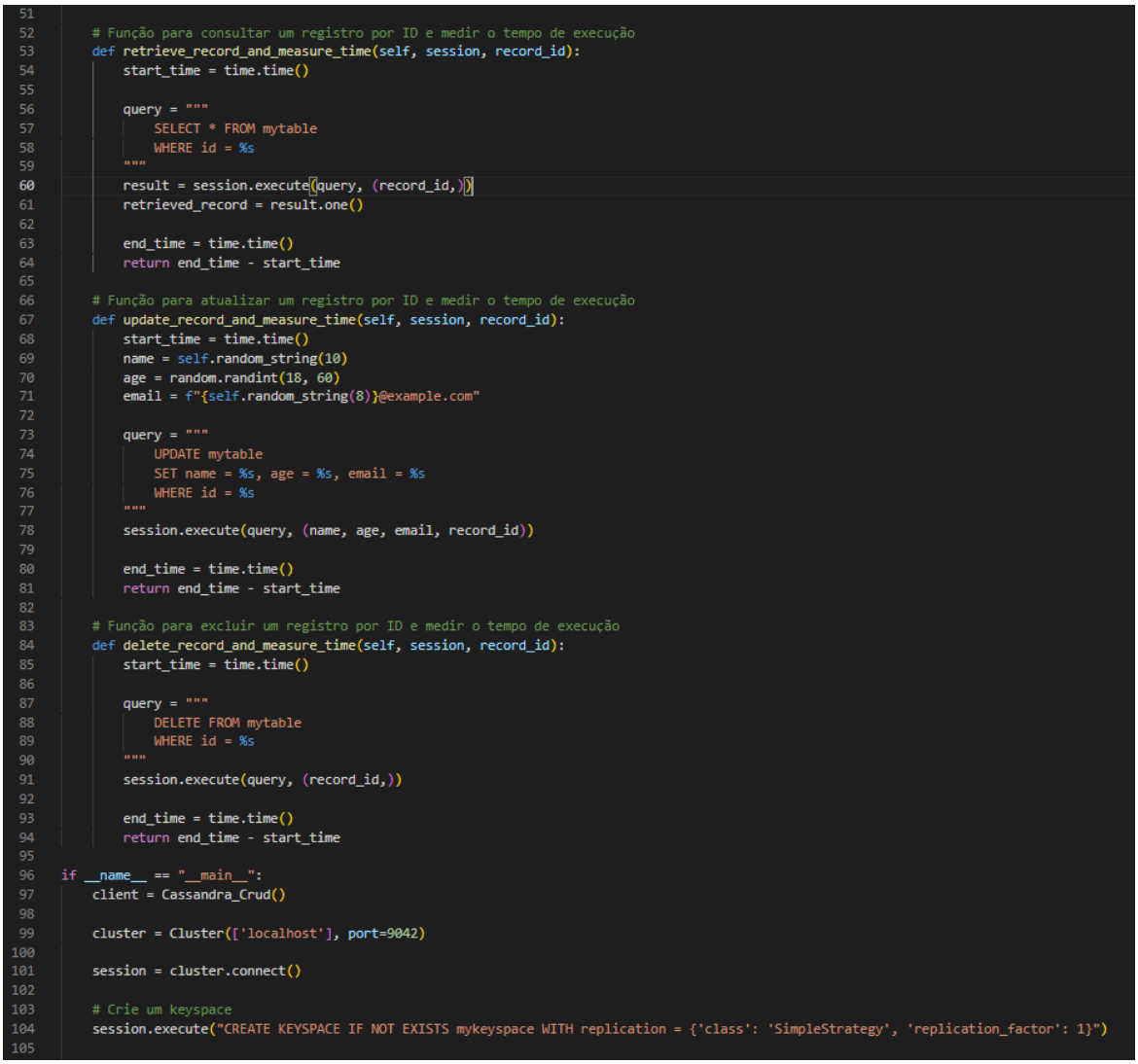
```python
from cassandra.cluster import Cluster
import uuid
import random
import string
import time
import json

# Endereço IP do contêiner Docker Cassandra
cassandra_container_ip = "localhost"
class Cassandra_Crud:
    # Função para gerar uma string aleatória
    def random_string(self, length):
        letters = string.ascii_lowercase
        return ''.join(random.choice(letters) for _ in range(length))

    # Função para inserir um registro e medir o tempo de execução
    def insert_record_and_measure_time(self, session):
        start_time = time.time()
        record_id = uuid.uuid4()
        name = self.random_string(10)
        age = random.randint(18, 60)
        email = f"{self.random_string(8)}@example.com"

        query = """
            INSERT INTO mytable (id, name, age, email)
            VALUES (%s, %s, %s, %s)
        """
        session.execute(query, (record_id, name, age, email))

        end_time = time.time()
        return end_time - start_time

    # Função para inserir registros em massa e medir o tempo de execução
    def insert_records_in_bulk_and_measure_time(self, session, num_records):
        start_time = time.time()
        for i in range(num_records):
            record_id = uuid.uuid4()
            name = self.random_string(10)
            age = random.randint(18, 60)
            email = f"{self.random_string(8)}@example.com"

            query = """
                INSERT INTO mytable (id, name, age, email)
                VALUES (%s, %s, %s, %s)
            """
            session.execute(query, (record_id, name, age, email))

        end_time = time.time()
        return end_time - start_time
```

Fonte: o autor.

- **Importações:** from cassandra.cluster import Cluster: Importa a classe Cluster do módulo cassandra.cluster, que é necessária para conectar-se ao banco de dados Cassandra. Import uuid, random, string, time, json: Importa módulos Python para lidar com UUIDs, gerar strings aleatórias, manipular tempo, e trabalhar com JSON;

- **Configuração do Endereço IP do Contêiner Cassandra:** cassandra_container_ip = "localhost": Define o endereço IP do contêiner Docker Cassandra como "localhost";
- **Classe Cassandra_Crud:** Esta classe contém métodos para realizar operações CRUD (Create, Read, Update, Delete) no banco de dados Cassandra;
- **Método random_string:** Gera uma string aleatória de um comprimento especificado usando letras minúsculas;
- **Método insert_record_and_measure_time:** Insere um único registro na tabela chamada "mytable". Gera um UUID para o ID, uma string aleatória para o nome, uma idade aleatória e um endereço de e-mail. Mede o tempo de execução da operação de inserção;
- **Método insert_records_in_bulk_and_measure_time:** Insere vários registros na tabela "mytable" em um loop. Gera um UUID, uma string aleatória para o nome, uma idade aleatória e um endereço de e-mail para cada registro. Mede o tempo de execução total para a inserção em massa.

Figura 5 – Segunda parte do Código em Python fazer o CRUD em Cassandra



```python

    # Função para consultar um registro por ID e medir o tempo de execução
    def retrieve_record_and_measure_time(self, session, record_id):
        start_time = time.time()

        query = """
            SELECT * FROM mytable
            WHERE id = %s
        """
        result = session.execute(query, (record_id,))
        retrieved_record = result.one()

        end_time = time.time()
        return end_time - start_time

    # Função para atualizar um registro por ID e medir o tempo de execução
    def update_record_and_measure_time(self, session, record_id):
        start_time = time.time()
        name = self.random_string(10)
        age = random.randint(18, 60)
        email = f"{self.random_string(8)}@example.com"

        query = """
            UPDATE mytable
            SET name = %s, age = %s, email = %s
            WHERE id = %s
        """
        session.execute(query, (name, age, email, record_id))

        end_time = time.time()
        return end_time - start_time

    # Função para excluir um registro por ID e medir o tempo de execução
    def delete_record_and_measure_time(self, session, record_id):
        start_time = time.time()

        query = """
            DELETE FROM mytable
            WHERE id = %s
        """
        session.execute(query, (record_id,))

        end_time = time.time()
        return end_time - start_time

if __name__ == "__main__":
    client = Cassandra_Crud()

    cluster = Cluster(['localhost'], port=9042)

    session = cluster.connect()

    # Crie um keyspace
    session.execute("CREATE KEYSPACE IF NOT EXISTS mykeyspace WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1}")

```

Fonte: o autor.

- **Retrieve_record_and_measure_time:** Consulta um registro na tabela "mytable" com base no ID fornecido, medindo o tempo de execução da operação de consulta;
- **Update_record_and_measure_time:** Atualiza um registro na tabela "mytable" com um novo nome, idade e endereço de e-mail gerados aleatoriamente, medindo o tempo de execução da operação de atualização;
- **Delete_record_and_measure_time:** Exclui um registro da tabela "mytable" com base no ID fornecido, medindo o tempo de execução da operação de exclusão;

- **O bloco if __name__ == "__main__":** Cria uma instância da classe Cassandra_Crud, conecta-se a um cluster Cassandra local e cria um keyspace chamado "mykeyspace" se ainda não existir.

Figura 6 – Terceira parte do Código em Python fazer o CRUD em Cassandra

```python
# Use o keyspace criado
session.set_keyspace('mykeyspace')

# Crie uma tabela
session.execute("CREATE TABLE IF NOT EXISTS mytable (id UUID PRIMARY KEY, name text, age int, email text)")

session = cluster.connect('mykeyspace')

num_records = 1000
individual_insert_times = []
bulk_insert_times = []
retrieve_times = []
update_times = []
delete_times = []

for _ in range(num_records):
    # Medir o tempo de execução para inserção individual
    individual_insert_time = client.insert_record_and_measure_time(session)
    individual_insert_times.append(individual_insert_time)

# Medir o tempo de execução para inserção em massa
bulk_insert_time = client.insert_records_in_bulk_and_measure_time(session, num_records)
bulk_insert_times.append(bulk_insert_time)

# Medir o tempo de execução para recuperação de um registro
record_id = uuid.uuid4()  # Substitua pelo ID do registro que deseja recuperar
retrieve_time = client.retrieve_record_and_measure_time(session, record_id)
retrieve_times.append(retrieve_time)

# Medir o tempo de execução para atualização de um registro
record_id = uuid.uuid4()  # Substitua pelo ID do registro que deseja atualizar
update_time = client.update_record_and_measure_time(session, record_id)
update_times.append(update_time)

# Medir o tempo de execução para exclusão de um registro
record_id = uuid.uuid4()  # Substitua pelo ID do registro que deseja excluir
delete_time = client.delete_record_and_measure_time(session, record_id)
delete_times.append(delete_time)

# Resto do código para outras operações CRUD...

session.shutdown()
cluster.shutdown()

# Armazenar tempos de execução em um JSON
execution_times = {
    # "individual_insert_times": individual_insert_times,
    "insert_times": bulk_insert_times,
    "retrieve_times": retrieve_times,
    "update_times": update_times,
    "delete_times": delete_times
}

with open('cassandra_execution_times.json', 'w') as json_file:
    json.dump(execution_times, json_file)

print("Tempos de execução salvos em 'execution_times.json'.")
```

Fonte: o autor.

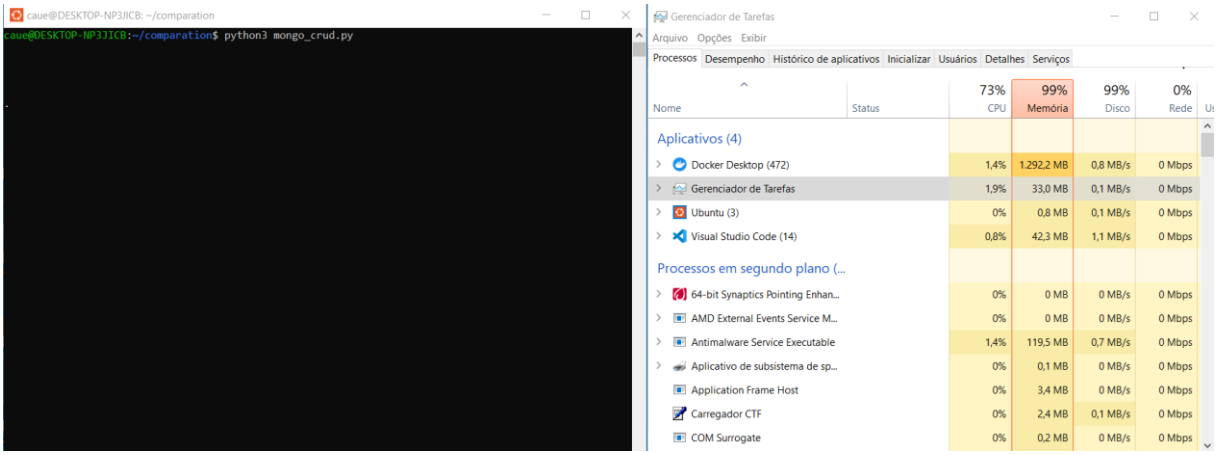
- **Configuração do Keyspace e Criação de Tabela:** session.set_keyspace('mykeyspace'): Define o keyspace a ser utilizado como 'mykeyspace';

- **Session.execute("CREATE TABLE IF NOT EXISTS mytable (id UUID PRIMARY KEY, name text, age int, email text")):** Cria uma tabela chamada 'mytable' se não existir, com colunas para ID, nome, idade e e-mail;
- **Operações CRUD e Medição de Tempo:** Para cada uma das X iterações: Insere registros individualmente e em massa, medindo o tempo de execução; Recupera, atualiza e exclui registros, medindo o tempo de execução para cada operação;
- **Encerramento da Sessão e Cluster:**
  - **session.shutdown():** Encerra a sessão Cassandra;
  - **cluster.shutdown():** Encerra a conexão com o cluster Cassandra;
- **Armazenamento dos Tempos de Execução em JSON:** Os tempos de execução são armazenados em um arquivo JSON chamado 'cassandra_execution_times.json';
- **json.dump(execution_times, json_file):** Salva os tempos de execução no arquivo JSON;
- **Mensagem de Conclusão:**
  - **print ("Tempos de execução salvos em 'cassandra_execution_times.json'."):** Imprime uma mensagem indicando que os tempos de execução foram salvos com sucesso;
- **Em resumo:** Este script Python usa o driver Cassandra para executar operações de inserção, consulta, atualização e exclusão em um banco de dados Cassandra. Ele mede o tempo de execução de cada operação, conecta-se ao cluster, cria um keyspace e uma tabela, executa operações em um número determinado de registros e armazena os tempos de execução em um arquivo JSON.

4 TESTES DE COMPARAÇÃO DE VELOCIDADE CURD

Nesta seção apresenta-se os resultados medidos para cada uma das operações CRUD no MongoDB e no Cassandra para bancos de dados com de diferentes tamanhos: 1 mil, 5 mil, 10 mil, 15 mil, 30 mil, 50 mil, 100 mil, 500 mil e 1 milhão de registros. Inicialmente pretendia-se realizar um teste com 5 milhões de registros no ambiente, entretanto encontrou-se dificuldades decorrentes de limitações de hardware (Figura 7). O disco atingiu sua capacidade máxima, interrompendo o teste, enquanto a limitação de memória sobrecarregou impactou significativamente o desempenho do sistema, comprometendo a integridade dos resultados do teste. Portanto esses resultados foram descartados.

Figura 7 – Ambiente em falha ao executar 5 milhões de registros



Fonte: o autor.

Quadro 1 – Testes executados com 1 MIL Registros

Operações	MongoDB Tempo	Cassandra Tempo
Insert_time	[1.074234962463379] Segundos	[1.7077536582946777] Segundos
Retrieve_time	[5.38825988769531e-05] Segundos	[0.003578662872314453] Segundos
Update_time	[0.013610363006591797] Segundos	[0.0019350051879882812] Segundos
Delete_time	[0.011952877044677734] Segundos	[0.0020155906677246094] Segundos

Fonte: o autor.

Quadro 2 – Testes executados com 5MIL Registros

Operações	MongoDB Tempo	Cassandra Tempo
Insert_time	[5.275515556335449] Segundos	[8.804903507232666] Segundos
Retrieve_time	[6.008148193359375e-05] Segundos	[0.00302886962890625] Segundos
Update_time	[0.01546168327331543] Segundos	[0.001991748809814453] Segundos
Delete_time	[0.013863563537597656] Segundos	[0.0016303062438964844] Segundos

Fonte: o autor.

Quadro 3 – Testes executados com 10 MIL Registros

Operações	MongoDB Tempo	Cassandra Tempo
Insert_time	[10.697443962097168] Segundos	[17.320682764053345] Segundos
Retrieve_time	[6.413459777832031e-05] Segundos	[0.003452301025390625] Segundos
Update_time	[0.02134084701538086] Segundos	[0.0021657943725585938] Segundos
Delete_time	[0.02044391632080078] Segundos	[0.0019443035125732422] Segundos

Fonte: o autor.

Quadro 4 – Testes executados com 15 MIL Registros

Operações	MongoDB Tempo	Cassandra Tempo
Insert_time	[15.839984893798828] Segundos	[27.27610445022583] Segundos
Retrieve_time	[4.363059997558594e-05] Segundos	[0.0029523372650146484] Segundos
Update_time	[0.028970003128051758] Segundos	[0.0019028186798095703] Segundos
Delete_time	[0.02740645408630371] Segundos	[0.0017631053924560547] Segundos

Fonte: o autor.

Quadro 5 – Testes executados com 20 MIL registros

Operações	MongoDB Tempo	Cassandra Tempo
Insert_time	[22.097910165786743] Segundos	[36.225056409835815] Segundos
Retrieve_time	[4.1961669921875e-05] Segundos	[0.0037071704864501953] Segundos
Update_time	[0.04205822944641113] Segundos	[0.0022084712982177734] Segundos
Delete_time	[0.03989672660827637] Segundos	[0.0018525123596191406] Segundos

Fonte: o autor.

Quadro 6 – Testes executados com 30 MIL registros

Operações	MongoDB Tempo	Cassandra Tempo
Insert_time	[32.279165267944336] Segundos	[53.468770265579224] Segundos
Retrieve_time	[6.508827209472656e-05] Segundos	[0.003318309783935547] Segundos
Update_time	[0.057303428649902344] Segundos	[0.002096891403198242] Segundos
Delete_time	[0.05384349822998047] Segundos	[0.0019388198852539062] Segundos

Fonte: o autor.

Quadro 7 – Testes executados com 50 MIL registros

Operações	MongoDB Tempo	Cassandra Tempo
Insert_time	[53.113603353500366] Segundos	[89.37787795066833] Segundos
Retrieve_time	[4.38690185546875e-05] Segundos	[0.0030670166015625] Segundos
Update_time	[0.0830670166015625] Segundos	[0.0019822120666503906] Segundos
Delete_time	[0.08675789833068848] Segundos	[0.0019991397857666016] Segundos

Fonte: o autor.

Quadro 8 – Testes executados com 100 MIL registros

Operações	MongoDB Tempo	Cassandra Tempo
Insert_time	[107.657053232193] Segundos	[175.02129101753235] Segundos
Retrieve_time	[4.38690185546875e-05] Segundos	[0.0037217140197753906] Segundos
Update_time	[0.1477670669555664] Segundos	[0.001995086669921875] Segundos
Delete_time	[0.1327371597290039] Segundos	[0.0016050338745117188] Segundos

Fonte: o autor.

Quadro 9 – Testes executados com 500 MIL registros

Operações	MongoDB Tempo	Cassandra Tempo
Insert_time	[566.4717690944672] Segundos	[915.0653872489929] Segundos
Retrieve_time	[0.0004210472106933594] Segundos	[0.015033960342407227] Segundos
Update_time	[0.4446830749511719] Segundos	[0.0023746490478515625] Segundos
Delete_time	[0.4963996410369873] Segundos	[0.0020093917846679688] Segundos

Fonte: o autor.

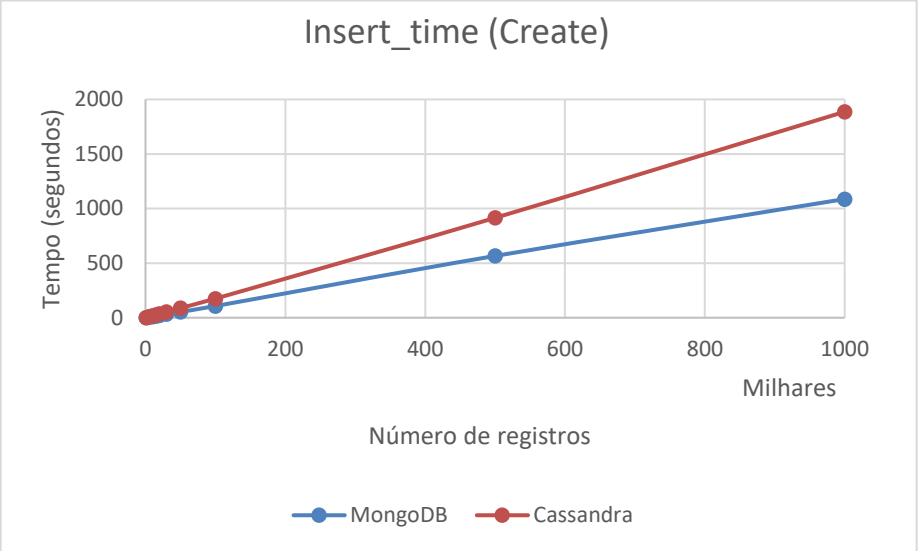
Quadro 10 – Testes executados com 1 MILHÃO de registros

Operações	MongoDB Tempo	Cassandra Tempo
Insert_time	[1087.6862428188324] Segundos	[1887.938758134842] Segundos
Retrieve_time	[0.0005974769592285156] Segundos	[0.03384590148925781] Segundos
Update_time	[1.0379424095153809] Segundos	[0.0025911331176757812] Segundos
Delete_time	[1.3596007823944092] Segundos	[0.0019474029541015625] Segundos

Fonte: o autor.

4.1 Operações de Inclusão (*Create*)

Gráfico 1 – Comparação de performance para inserção de dados com diferentes números de registros

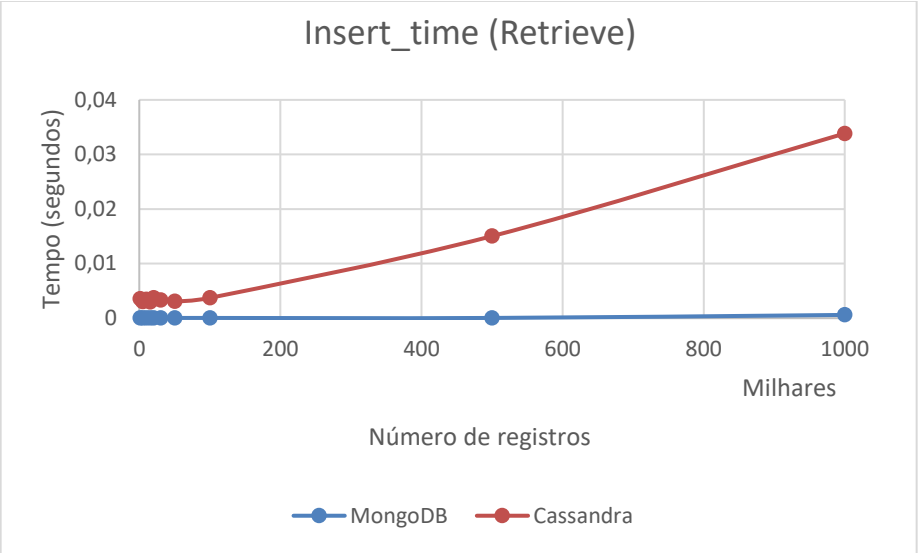


Fonte: o autor.

Como demostrando no Gráfico 1, para operações de inclusão de novos registros (*Create* do CRUD), o tempo de execução tanto do MongoDB como do Cassandra cresce de forma linear com o número dos registros. Além disso, o MongoDB apresenta um desempenho médio 65% superior ao Cassandra em todos os cenários analisados.

4.2 Operações de Leitura (*Read*)

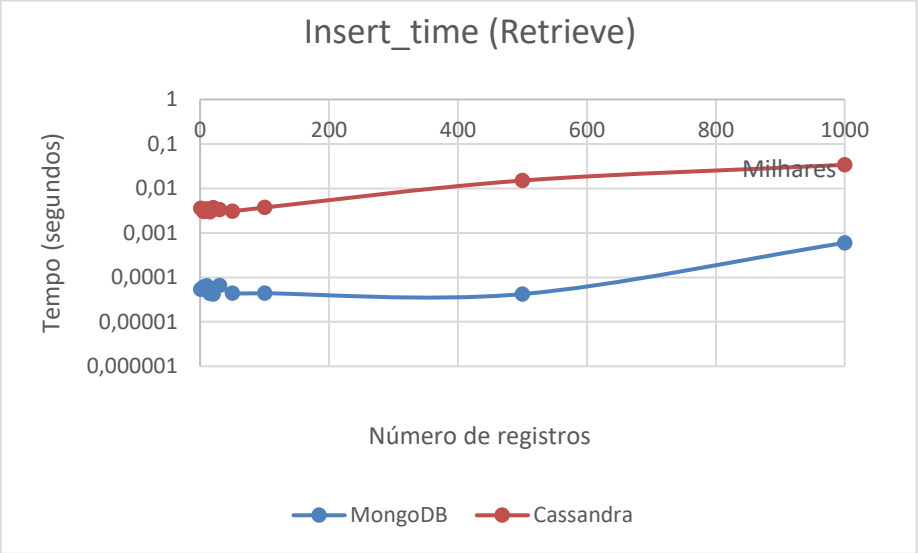
Gráfico 2 – Comparação de performance para recuperação de dados com diferentes números de registros



Fonte: o autor.

Os Gráficos 2 e 3 mostram os dados operações de leitura de registros (*Read* do CRUD), porém o Gráfico 2 está em escala decimal enquanto o Gráfico em 3 está em escala logarítmica. Para menos de 200 mil registros, percebe-se que o tempo de execução parece ser relativamente constante tanto no MongoDB como no Cassandra, apresentando pequenas variações que podem ser resultantes de flutuações do ambiente da execução. Nota-se que a partir de 200 mil registros, o tempo de execução começa a crescer para o Cassandra, mas ainda se mantem relativamente constante no Mongo. Há um incremento perceptível no tempo de execução para 1 milhão de registros no MongoDB (1000%, exponencial) enquanto no Cassandra o tempo de aumenta em 50% (linear). Ainda assim, o MongoDB mostra-se sempre uma ou duas ordens de grandeza mais rápido que o Cassandra em todos os cenários analisados.

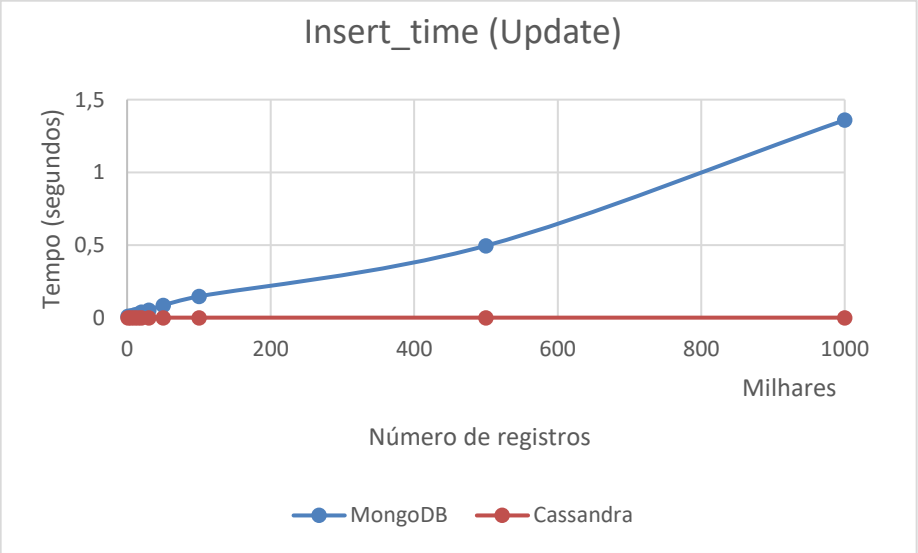
Gráfico 3 – Comparação de performance para recuperação de dados com diferentes números de registros (Escala Logarítmica)



Fonte: o autor.

4.3 Operações de Atualização (Update)

Gráfico 4 – Comparação de performance para atualização de dados com diferentes números de registros



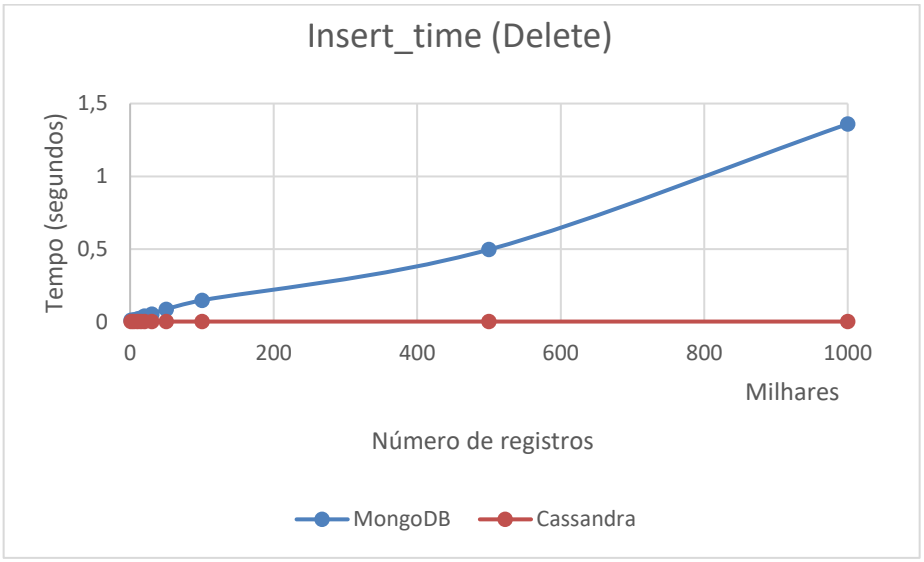
Fonte: o autor.

Como demostrando no Gráfico 4, para operações de atualização de registros (*Update* do CRUD), o tempo de execução do MongoDB cresce de forma linear com o número dos registros. Entretanto, no Cassandra, o tempo de execução se mantem praticamente constante, variando de 0,0019 segundos para 1 mil registros para 0,0025

segundos com 1 milhão de registros. O Cassandra mostra-se entre 2 a 4 ordens de grandeza mais eficiente que o MongoDB em todos os cenários analisados.

4.4 Operações de Remoção (Delete)

Gráfico 5 – Comparação de performance para remoção de dados com diferentes números de registros



Fonte: o autor.

Como demostrando no Gráfico 5, para operações de atualização de remoção (*Delete* do CRUD), o tempo de execução do MongoDB cresce de forma relativamente linear com o número dos registros. Entretanto, no Cassandra, o tempo de execução se mantem constante, apresentando apenas flutuações. O Cassandra mostra-se entre 2 a 4 ordens de grandeza mais eficiente que o MongoDB em todos os cenários analisados.

5 CONCLUSÃO

Com base nos testes realizados, pode-se concluir que a seleção do SGDB ideal depende das características intrínsecas da aplicação a ser construída, pois o MongoDB mostrou-se superior ao Cassandra nas operações de inclusão e recuperação de dados. Entretanto, o Cassandra mostrou-se mais eficiente nas operações de remoção e atualização.

Para uma aplicação de armazenamento de logs remoto, em que milhares de mensagens de log são geradas por segundo e armazenas em um banco de dados SGDB, o MongoDB seria uma escolha mais adequada pois ele é em média 65% mais eficiente que o Cassandra. O MongoDB também seria mais adequado para uma aplicação em grandes quantidades de dados são recuperados em uma única busca, como por exemplo uma lista de contatos telefônicos.

Já para uma aplicação como um sistema de monitoramento em tempo real, que precisa manipular grandes volumes de dados de sensores IoT em constante atualização, o Cassandra tende a ser superior ao MongoDB. Com conjuntos de dados extensos e operações frequentes de atualização à medida que novos dados de sensores são constantemente recebidos, o desempenho superior do Cassandra em operações de atualização e remoção torna-se crucial. Sua arquitetura distribuída e foco em escrever eficientemente em grande escala se alinham melhor com a necessidade de processar e armazenar dados em tempo real.

À primeira vista pode parecer que há um empate já que um dos SGBD mostrou-se melhor em duas das quatro operações CRUD, porém pode-se apontar as seguintes limitações e observações:

- Para recuperação de registros, o tempo de execução do MongoDB pareceu cresceu de maneira exponencial entre 500 mil e 1 milhão; enquanto o Cassandra cresceu de maneira linear. Se tivesse sido possível realizar testes com 5 milhões ou mais registros, talvez o tempo de execução do MongoDB ultrapassasse o do Cassandr;
- No caso da recuperação de registros, foi realizada uma recuperação completa de todos os dados existentes no banco. Se fosse realizada uma recuperação parcial dos dados (por exemplo, apenas um registro), é possível que o resultado fosse diferente e o Cassandra fosse mais eficiente;

- Os testes realizados não exploraram o paralelismo dos SGBD, realizando as operações CRUD sequenciais em um único thread. Se múltiplos threads tivessem sido utilizados, o resultado poderia ter sido diferente.

Como trabalho futuros propõe-se:

a) ampliar o número de registros utilizados nos testes;
b) estender os testes para incluir o paralelismo nas operações CRUD;
c) criar testes com operações mais complexas, sobretudo na recuperação e remoção de registros;
d) replicar os testes em uma plataforma Cloud para aumente sua fidelidade com um ambiente real de produção.

REFERÊNCIAS

6 SENSE. **Best NoSQL databases software in 2023**. [*S. l.*], 2023. Disponível em: https://6sense.com/tech/nosql-databases. Acesso em: 27 nov. 2023.

CARDOSO, Ricardo Manuel Fonseca. **Base de dados NoSQL**. 2012. Dissertação (Mestrado em Engenharia Informática) – Instituto Superior de Engenharia do Porto, Porto, 2012. Disponível em: https://core.ac.uk/download/pdf/302865845.pdf. Acesso em: 29 nov. 2023.

CARPENTER, Jeff; HEWITT, Eben. **Cassandra**: the definitive guide. 2nd ed. Sebastopol: O'Reilly Media, 2019.

CATARINO, Marino H. **Estrutura de banco de dados para big data**. São Paulo: Senac, 2023.

CHODOROW, Kristina; DIROLF, Michael. **MongoDB**: the definitive guide. Sebastopol: O'Reilly Media, 2013.

COSTA, Kaique J. *et al.* Estudo comparativo entre banco de dados NoSQL distribuídos. *In*: ENCONTRO DA ESCOLA REGIONAL DE ALTO DESEMPENHO DE SÃO PAULO, 13., 2022, São Paulo. **Anais** [...]. São Paulo: SBC, 2022. Disponível em: https://sol.sbc.org.br/index.php/eradsp/article/view/21917/21740. Acesso em: 29 nov. 2023.

CUNHA, Fernando. O que é o banco de dados MongoDB?. **Mestres da Web**, [*s. l.*], 6 dez. 2022. Disponível em: https://www.mestresdaweb.com.br/tecnologias/o-que-e-o-banco-de-dados-mongodb. Acesso em: 29 nov. 2023.

LAKSHMAN, Avinash; MALIK, Prashant. Cassandra: a decentralized structured storage system. **ACM SIGOPS Operating Systems Review**, New York, v. 44, n. 2, p. 35-40, 2010. DOI: https://doi.org/10.1145/1773912.1773922.

MATIOLA, Guilherme Schlindwein. **Aplicativo móvel para controle do processamento de dados de amostras de solo com integração da tecnologia RFId**. 2023. Trabalho de Conclusão de Curso (Graduação em Engenharia de Controle e Automação) – Universidade Federal de Santa Catarina, Blumenau, 2023. Disponível em: https://repositorio.ufsc.br/handle/123456789/248788. Acesso em: 29 nov. 2023.

MEMBRE, Peter; HAWKINS, Tim. **The definitive guide to MongoDB**. New York: Apress, 2020.

PERKINS, Luc; REDMOND, Eric; WILSON, Jim. **Seven databases in seven weeks**: a guide to modern databases and the NoSQL movement. [*S. l.*]: Pragmatic Bookshelf, 2018.

SADALAGE, Pramod; FOWLER, Martin. **NoSQL distilled**: a brief guide to the emerging World of polyglot persistence. Melbourne: Addison-Wesley Professional, 2012.

SILVA, Laís Bethânia Brito. **Análise comparativa de benchmarks para banco de dados NoSQL orientados a grafos de propriedades**. 2021. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de São Carlos, São Carlos, 2021. Disponível em: https://repositorio.ufscar.br/bitstream/handle/ufscar/16139/Dissertcao_final_Lais_Silva.pdf?sequence=1&isAllowed=y. Acesso em: 29 nov. 2023.

WHITFIELD, Matt. Is SQL dead, yet? NoSQL's rise in popularity. **Orange Matter**, [*s. l.*], 1 July 2020. Disponível em: https://orangematter.solarwinds.com/2020/07/01/is-sql-dead-yet-nosqls-rise-in-popularity/. Acesso em: 27 nov. 2023.