Uma comparação de performance entre SQLite e MongoDB em Bancos de Dados Massivos

Bruno Gomes Resende
Departamento de Ciência da Computação
Universidade de Brasília - UnB
Brasília, Brasil
bruno.resende@aluno.unb.br

Matheus Luis Teixeira Rosa
Departamento de Ciência da Computação
Universidade de Brasília - UnB
Brasília, Brasil
matheusrosa@unb.br

Andrei Lima Queiroz
Departamento de Ciência da Computação
Universidade de Brasília - UnB
Brasília, Brasil
andreiqueiroz@unb.br

***Resumo*—Dado o cenário atual de massiva geração de dados em diferentes contextos organizacionais, faz-se necessário a adoção de tecnologias que viabilizem o armazenamento e recuperação de informações de maneira ágil e confiável. Este trabalho tem como objetivo realizar uma comparação, em termos de custo computacional, no uso dos paradigmas Relacional e de Documentos para Bancos de Dados Massivos. Para tal, foram utilizados dados abertos da Agência Nacional do Petróleo, contendo a série histórica de preços de combustíveis automotivos, totalizando cerca de 1 milhão de registros, bem como os Bancos de Dados SQLite sob o paradigma Relacional e MongoDB sob o paradigma de Documentos. Foram avaliados os tempos computacionais de ingestão, deleção e recuperação de dados em ambas as tecnologias de armazenamento. Foi possível observar um desempenho sensivelmente melhor ao utilizar o MongoDB, com uma redução de cerca de 83% no tempo de ingestão de dados, 81% no tempo de consulta em resultados massivos e 99% no tempo de deleção. Todavia, o SQLite se mostrou mais adequado no trato de consultas com agregações, cálculos e ordenações.**

Keywords—NoSQL, MongoDB, SQLite, SQL, Big Data

***Abstract*— Given the current mass scenario of data generation in different organizational contexts, it is necessary to adopt technologies that enable the storage and recovery of information in an agile and reliable way. This work aims to make a comparison, in terms of computational cost, in the use of relational paradigms and documents for massive databases. To this end, open data from the National Petroleum Agency, containing the historical series of automotive fuel prices, totaling about 1 million records, as well as the SQLite databases under the relational paradigm and Mongodb under the document paradigm. Computational times of intake, deletion and data recovery in both storage technologies were evaluated. It was possible to observe a significantly better performance when using MongoDB, with a reduction of about 83% in data intake time, 81% in consultation time in massive results and 99% at deletion time. However, SQLite was more appropriate in dealing with aggregations, calculations and ordinations.**

Keywords—NoSQL, MongoDB, SQLite, SQL, Big Data

I. INTRODUÇÃO

Atualmente as bases de dados organizacionais vêm alcançando dimensões cada vez maiores, exigindo a adoção de tecnologias que viabilizem o armazenamento e recuperação de informações relevantes ao contexto de negócio de forma ágil e confiável. Para tratar a questão do volume dos dados, um dos 5 V's do Big Data [1], diferentes paradigmas para bancos de dados têm surgido em complemento ao paradigma Relacional, dentre eles o paradigma de NoSQL (*Not only SQL*) como o banco de dados orientado a Documentos. Todavia, para se obter uma arquitetura de solução adequada ao contexto de negócio, é necessária uma correta alocação de recursos tecnológicos ao propósito da solução ao qual o recurso está alocado.

Neste trabalho iremos avaliar duas tecnologias de bancos de dados, sendo elas o SQLite sob o paradigma Relacional e o MongoDB sob o paradigma orientado a Documentos, de forma a explorar suas características e avaliar suas performances em um contexto de negócio de dados analíticos. Para tal, foi utilizada uma base de dados aberta, contendo a série histórica de preços de combustíveis automotivos, proveniente da Agência Nacional do Petróleo e disponível em [2], contendo cerca de 2 milhões de registros e realizado um experimento prático de ingestão, deleção e recuperação de dados em ambas as tecnologias, de forma a comparar seus desempenhos.

O restante deste trabalho está organizando da seguinte forma: Sessão II. descreve e discute os trabalhos relacionados. Sessão III. apresenta uma revisão teórica sobre as tecnologias utilizadas. Sessão V. apresenta e discute o estudo de caso e sessão VI. conclui o artigo e apresenta direcionamentos para trabalhos futuros.

II. TRABALHOS RELACIONADOS

O trabalho apresentado em [3] discute o modelo de dados NoSQL, tipos de armazenamento, características e funcionalidades de cada tipo de armazenamento de dados, linguagem de consulta utilizada, vantagens e desvantagens de NoSQL sobre bancos de dados relacionais e perspectivas futuras. Apesar de NoSQL ter evoluído rapidamente, ainda está atrasado em relação a adoção pelos usuários quando comparado com os bancos de dados relacionais. A principal razão para isso é que os usuários são familiarizados com a linguagem SQL, enquanto bancos de dados NoSQL não possuem uma linguagem de consulta padrão. A maioria dos bancos de dados NoSQL possuem sua própria linguagem, como CQL para Cassandra, MongoQL para MongoDB etc. Portanto torna-se difícil que os usuários troquem entre fornecedores NoSQL, sendo necessário uma linguagem de consulta padrão como a SQL para todos os bancos NoSQL. Um esforço coletivo para criar a linguagem padronizada UnQL – Unstructured Query Language, está em desenvolvimento pelos criadores do CouchDB e SQLite.

O trabalho apresentado em [4] tem o objetivo de comparar bancos de dados NoSQL com banco de dados Relacionais e verificar quais características dos bancos de dados relacionais também estão presentes nos bancos de dados NoSQL. São levadas em consideração as diferenças significativas entre cada categoria de banco de dados NoSQL - chave-valor, armazenamento em documentos, orientado a colunas e orientado a grafos. Os resultados da comparação entre as características dos bancos de dados tradicionais e bancos de dados NoSQL mostraram que nenhum dos produtos considerados apresentavam mais de 50% das características dos bancos tradicionais. Como nenhum dos produtos verificados apresentavam mais de 50% das características de um banco de dados tradicional, os produtos categorizados

como bancos de dados NoSQL seriam melhor denominados armazenamento de dados não relacional.

O trabalho apresentado em [5] busca investigar o desempenho de alguns bancos de dados NoSQL e bancos de dados SQL sob o enfoque do armazenamento por chave-valor. Foram comparadas operações de leitura, escrita, exclusão e instanciação implementadas por bancos de dados NoSQL chave-valor e bancos de dados realcionais, também foi comparada operação de iteração por todas chaves. O trabalho avaliou o tempo das operações e realizou comparativo entre os bancos de dados. Os resultados mostraram que nem todos os bancos de dados NoSQL funcionaram melhor do que os SQL e alguns foram muito piores. O desempenho das operações variou conforme o banco de dados. Alguns são lentos para instanciar, mas rápidos para ler, escrever e excluir. Outros são rápidos para instanciar, mas lentos nas outras operações. E há pouca correlação entre desempenho e o modelo de dados que cada banco de dados utiliza.

III. REVISÃO DA LITERATURA

A. *Banco de Dados Relacional – SQLite*

SQLite [11] é uma biblioteca da linguagem C que implementa um banco de dados SQL transacional, autocontido, sem configurações e sem servidor. Seu código fonte está em domínio público e gratuito para finalidades privadas e comerciais. SQLite tem implementações para várias linguagens de programação como C, C++, Basic, C#, Python, Java etc. Assim, tornando sua utilização mais acessível e também está disponível em sistemas operacionais embarcados como iOS, Android, Symbian OS e outros devido seu tamanho compacto que pode ser inferior a 500KB e facilidade de uso.

O formato do arquivo SQLite é multiplataforma e a escrita e leitura dos dados é realizada diretamente dentro da sua própria estrutura em arquivos comuns no disco. Essas características tornaram o banco de dados uma escolha popular para dispositivos com restrições de memória e armazenamento. Em geral, o SQLite apresenta boa performance, com execução rápida mesmo em ambientes com pouco poder de processamento e confiabilidade.

As principais razões para o uso e adoção do SQLite são [9]:

- Alta compatibilidade: não há dificuldades de compatibilidade quanto ao ambiente, sistema operacional e aplicações que serão integradas com o SQLite, sendo apenas necessário uma forma de vincular essa aplicação com bibliotecas da linguagem C.
- Domínio público: pode ser utilizado em qualquer aplicação, com qualquer finalidade e em diversas situações. Isso aumenta muito a sua compatibilidade e suporte.
- Dispensa de configuração: devido não utilizar um servidor e sua arquitetura adequada para essa finalidade, dispensa a realização de configurações iniciais, sendo suficiente sua inicialização para uso.
- Autocontido: precisa de suporte mínimo dos sistemas operacionais ou de bibliotecas externas. Assim, funciona muito bem em diversos tipos de aplicações embutidas, como smartphones, consoles e aparelhos diversos.
- Tabelas dinâmicas: permite que o usuário insira qualquer valor na coluna que quiser, sem fazer nenhuma distinção da categoria de dado em questão.
- Conexões únicas: apenas uma conexão é necessária para acessar diversos arquivos de bancos de dados, favorecendo uma operação mais rápida e organizada.

B. *Banco de Dados NoSQL Orientado a Documentos – MongoDB*

O termo Bancos de dados NoSQL pode ser interpretado de diferentes maneiras. Uma interpretação se refere literalmente a bancos de dados que não utilizam a linguagem SQL, há também uma interpretação que significa Not Only SQL, são bancos de dados que não usam SQL e também não são baseados no modelo relacional, assim o termo mais adequado seria bancos de dados não relacionais. Além disso, não se baseiam nos modelos hierárquicos nem de redes, modelos predecessores aos relacionais. São um tipo completamente novo de bancos de dados.

O surgimento dos bancos de dados NoSQL começou no início do século 21 com o desenvolvimento de aplicações para Web e, principalmente, aplicações do Google. As primeiras publicações sobre o tema foram em 2003, vinculada ao Google File System [6]. Surgiram então publicações relacionadas a outros sistemas do Google, como MapReduce [7] e Bigtable [8]. As aplicações Web do Google foram seguidas pelo Yahoo, Amazon, Facebook, Netflix e outros, que levaram a criação de novos bancos de dados NoSQL.

As principais razões para o surgimento e rápido desenvolvimento desse tipo de armazenamento são:

- Armazenamento e processamento de grande volume de dados: a quantidade de fotos, vídeos e dados geográficos armazenados por aplicações aumenta a cada dia.
- Acesso em tempo real: os dados armazenamentos precisam estar disponíveis em tempo real a partir de qualquer localidade do mundo.
- Flexibilidade: capacidade de alterar com rapidez e facilidade a estrutura dos dados
- Armazenamento de dados não estruturados: dados não estruturados ou semi-estruturados também devem ser armazenados junto dos dados estruturados.
- Escalabilidade: é necessário ter facilidade na escalabilidade de aplicações e dados armazenados. Bancos de dados tradicionais não atendem todos os requisitos.

Os diferentes modelos de armazento dos bancos de dados NoSQL são:

- Chave-valor: este é um dos modelos mais simples de armazenamento de dados. Os registros são pares chave-valor e são armazenados com a unicidade das chaves. Os dados são acessados através dos valores das chaves. É adequado para armazenar grandes volumes de dados e fornece acesso rápido pela chave. Existem diferentes variedades, dependendo

da memória em que os dados são armazenados, a classificação das chaves e a consistência dos dados. Os bancos mais populares são Redis, Memcached, Berkeley DB e Oracle NoSQL.

- Orientado a documentos: são projetados especificamente para armazenamento de documentos, os formatos costumam ser XML, JSON, BSON e outros. Os dados são semiestruturados e contêm atributos com pares de nome-valor. Os dados são acessados tanto por meio dos valores das chaves quanto pelos valores dos atributos. Eles são adequados para armazenar documentos de texto, XML e outros dados semiestruturados. Os produtos mais famosos desse tipo são MongoDB, Amazon Dynamo DB, Couchbase e CouchDB.

- Orientado a colunas: diferente dos bancos de dados relacionais que são orientados a linhas, esse tipo de modelo de dados é orientado a colunas. Isso torna possível expandir facilmente a estrutura de dados adicionando novas colunas. Eles são adequados para grandes volumes de dados distribuídos. Os representantes mais conhecidos são Cassandra, HBase e Google Bigtable.

- Orientado a grafos: os dados são representados por uma estrutura semelhante a um grafo, com seus nós representando os objetos e seu conjunto de atributos, e as arestas do grafo representando as conexões entre os objetos. Eles são adequados para representar dados quando as conexões entre os objetos são particularmente importantes para a modelagem. Os produtos mais famosos desse tipo são Neo4j, Titan, Giraph. Existem também produtos baseados em dois ou mais modelos de dados diferentes. Tais são OrientDB, ArangoDB e muitos outros.

Segundo o Teorema CAP [10], sistemas distribuídos não são capazes de atender ao mesmo tempo mais de duas das seguintes características: Consistência, Disponibilidade e Tolerância a Particionamento. Enquanto os bancos de dados tradicionais estão focados em garantir as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), os bancos de dados NoSQL costumam priorizar a alta disponibilidade e o particionamento, perdendo a consistência dos dados. Isso resulta em sistemas com características chamadas BASE - Basically Available, Soft-state, Eventually Consistent. Esses sistemas, ao mesmo tempo que enfrentam alguns problemas, criaram outros.

O MongoDB [15] tem código-fonte aberto licenciado pela GNU AGP 3.0, possui alta performance, não possui esquemas e foi escrito em C++. O projeto foi iniciado em 2007, mas o banco de dados somente foi concluído em 2009 e lançando sua primeira versão. Diversas linguagens e plataformas já possuem drivers para o MongoDB, entre elas: C, C#, C++, Java, JavaScript, PHP, Python e outras. Além disso, o MongoDB possui binários para diversas plataformas como Windows, Mac OS X, Linux e Solaris.

IV. Estudo de Caso

Nesta seção iremos descrever o experimento comparativo realizado entre os bancos de dados SQLite e MongoDB, especificando a metodologia e métrica utilizada, justificar a escolha tecnológica, descrever a base de dados massiva objeto do estudo, o ambiente de execução, os procedimentos testados e os resultados obtidos.

A. Metodologia

De forma a realizar o estudo comparativo, serão executados processos de ingestão, consulta e deleção de dados em ambos os bancos de dados. Os testes serão conduzidos por um total de 30 repetições cada, onde serão coletadas as médias dos tempos computacionais de ingestão, consulta e deleção e comparados ao término do experimento. Os indicadores serão apresentados com um intervalo de confiança de 95%. Os dados foram obtidos em formato .csv e carregados em um Pandas Dataframe previamente aos processos de ingestão, consulta e deleção nos Bancos de Dados selecionados. A implementação do experimento pode ser consultada em https://github.com/brunogresende/ppca-bdm.

B. Escolha das Tecnologias

A motivação para escolha das tecnologias adotadas neste trabalho deu-se devido à grande simplicidade e confiabilidade na utilização das ferramentas representantes dos diferentes modelos de armazenamento de dados. O banco de dados SQLite, é um dos mais compactos e de fácil utilização dentre os bancos relacionais, integrado à linguagem Python por meio da biblioteca SQLAlchemy, é um banco robusto, rápido, com grande capacidade de processamento e testado em diversas aplicações desde embarcadas a sistemas de missão crítica. Sua principal característica de portabilidade entre ambientes pode ser um recurso interessante no uso para o processamento de dados massivos em processos de geração de modelos analíticos e na mineração de dados de forma local diretamente na máquina do usuário. O banco de dados MongoDB, um dos bancos não relacionais mais populares e com uma comunidade ativa de usuários e desenvolvedores, possui grande flexibilidade, integração com Python por meio da biblioteca PyMongo, também é de fácil utilização e robustez, capaz de atender a altas cargas de trabalho, distribuindo os dados em vários computadores e com suporte a replicação, garantindo a disponibilidade dos dados, mesmo em caso de falha de um computador.

C. Descrição da Base de Dados

Para a condução deste estudo comparativo foi selecionada no Portal de Dados Abertos do Governo Federal uma base de dados da Agência Nacional do Petróleo com histórico de preços de combustíveis nas diversas regiões do Brasil. A base selecionada conta com registros entre o segundo semestre de 2019 e primeiro semestre de 2022, totalizando 1.011.250 registros de coleta de preços em pontos de revenda de combustível no país. O dicionário de dados desta base pode ser verificado a seguir:

- Região – Sigla
- Estado – Sigla
- Município
- Revenda
- CNPJ da Revenda
- Nome da Rua
- Número Rua
- Complemento
- Bairro
- Cep
- Produto
- Data da Coleta
- Valor de Venda
- Valor de Compra
- Unidade de Medida
- Bandeira

D. *Ambiente de Execução*

O experimento foi executado a partir de um ambiente Python 3.9.12, utilizando-se das bibliotecas sqlalchemy 2.0.3[13] e pymongo 4.3.3[14]. A biblioteca sqlalchemy implementa o mapeamento objeto-relacional possibilitando o experimento em SQLite. A biblioteca pymongo implementa ferramentas para lidar com o servidor MongoDB 6.0.4[15] instalado localmente para este estudo. A máquina onde foram instalados os componentes e performado o experimento conta com um sistema operacional Windows 10 Home Single Language 64bits 22H2, processador AMD Ryzen 5 3500X 6-Core Processor 3.95 GHz e 16GB de memória RAM disponível.

E. *Definição dos Comandos de Teste*

O experimento foi conduzido realizando comandos de INSERT e DELETE no SQLite para ingestão e deleção de registros na base de dados. De forma análoga no MongoDB foram usados os comandos insert_many() e drop_collection() do pacote pymongo para ingestão e deleção de registros na base de dados. Foram também executadas três consultas, partindo de uma consulta menos elaborada, uma QUERY 1 com simples filtro para os dados da região sudeste, a QUERY 2 com uma agregação da média do valor de venda de combustíveis em todo território nacional e a QUERY 3, mais complexa com uma agregação do valor médio de venda de combustíveis, agrupado por região e ordenado de forma ascendente.

F. *Execução SQLite*

Para a inserção de dados no SQLite, as colunas do dataset foram mapeadas em uma classe "Coleta", e, de forma iterativa incluídas na sessão aberta na base de dados, por fim, para gravação definitiva foi executado o commit na sessão, como podemos verificar no trecho de código a seguir.

```
#Create objects
tempo_ini = time.time()
for index, row in df.iterrows():
    session.add(Coleta(
        id = index,
        regiao = row['Regiao - Sigla'],
        estado = row['Estado - Sigla'],
        municipio = row['Municipio'],
        revenda = row['Revenda'],
        cnpj = row['CNPJ da Revenda'],
        rua = row['Nome da Rua'],
        numero = row['Numero Rua'],
        complemento = row['Complemento'],
        bairro = row['Bairro'],
        cep = row['Cep'],
        produto = row['Produto'],
        data = row['Data da Coleta'],
        valor_venda = row['Valor de Venda'],
        valor_compra = row['Valor de Compra'],
        unidade_medida = row['Unidade de Medida'],
        bandeira = row['Bandeira']
    ))
# Commit changes to database
session.commit()
tempo_fim = time.time()
print('### SQL ### ', df.shape[0], 'registros.')
print("Tempo de inserção: {:.2f} segundos".format((tempo_fim - tempo_ini)))
```

A inserção total dos registros no SQLite foi realizada em tempo médio de 168,258131 segundos, com confiança de 95% em um intervalo de 167,899891 a 168,616370 segundos. O teste de deleção foi realizado em tempo médio de 0,172093 segundos, com confiança de 95% em um intervalo de 0,171286 a 0,172899 segundos conforme verificado no trecho de código a seguir.

```
# Delete database
tempo_ini = time.time()
Base.metadata.drop_all(engine)
tempo_fim = time.time()
print('### SQL DELETE ### ')
print("Tempo de delete: {:.5f} segundos".format((tempo_fim - tempo_ini)))
deleteSQLite.append(tempo_fim - tempo_ini)
```

Para o teste de performance na consulta denominada QUERY 1, foi executado um SELECT simples, listando todos os campos e registros cuja região fosse 'SE' (sudeste), conforme verificado no trecho de código a seguir.

```
query = """SELECT * FROM coleta AS c WHERE c.regiao = 'SE';"""
tempo_ini = time.time()
result = pd.read_sql_query(query, session.bind)
tempo_fim = time.time()
print('### SQL ### ', result.shape[0], 'registros.')
print("Tempo de consulta: {:.5f} segundos".format((tempo_fim - tempo_ini)))
```

A consulta QUERY 1 retornou 491.866 registros em um tempo de médio de 3,721677 segundos, com confiança de 95% em um intervalo de 3,677840 a 3,765514 segundos. Em seguida foi realizado o teste de consulta denominado QUERY 2, executando uma agregação pela média AVG() dos preços de venda, conforme observado no trecho de código a seguir.

```
query = """SELECT AVG(c.valor_venda) FROM coleta AS c ;"""
tempo_ini = time.time()
result = pd.read_sql_query(query, session.bind)
tempo_fim = time.time()
print(i)
print('### SQL - preço médio de venda ### ', result.shape[0], 'registro(s).')
print(result['AVG(c.valor_venda)'].values[0])
print("Tempo de consulta: {:.5f} segundos".format((tempo_fim - tempo_ini)))
```

A consulta QUERY 2 retornou a média em um tempo médio de 0,294327 segundos, com confiança de 95% em um intervalo de 0,273684 a 0,314970 segundos. Por fim, foi realizado o teste de consulta denominado QUERY 3, executando uma agregação pela média AVG() dos preços de venda, agrupados por região e ordenados de forma ascendente, conforme verificado no trecho de código a seguir.

```
query = """ SELECT c.regiao, AVG(c.valor_venda) FROM coleta AS c
            GROUP BY c.regiao ORDER BY AVG(c.valor_venda) ASC;"""
tempo_ini = time.time()
result = pd.read_sql_query(query, session.bind)
tempo_fim = time.time()
print(i)
print('### SQL - preço médio de venda agrupado por região ### ', result.shape[0], 'registro(s).')
print(result)
print("Tempo de consulta: {:.5f} segundos".format((tempo_fim - tempo_ini)))
```

A consulta QUERY 3 retornou em um tempo médio de 0,771643 segundos, com confiança de 95% em um intervalo de 0,769300 a 0,773986 segundos.

G. *Execução MongoDB*

Para a inserção do mesmo dataset no MongoDB, foi utilizada a função insert_many do pacote pymongo, convertendo o Pandas Dataframe para um dicionário.

```
#Create objects
tempo_ini = time.time()
db.coleta.insert_many(df.to_dict('records'))
tempo_fim = time.time()
print('### NoSQL ### ', df.shape[0], 'registros.')
print("Tempo de inserção: {:.2f} segundos".format((tempo_fim - tempo_ini)))
```

A inserção total dos registros no MongoDB foi realizada em 30,342353 segundos, com confiança de 95% em um intervalo de 30,117551 a 30,567156 segundos. O teste de deleção foi realizado em tempo médio de 0,000867 segundos, com

confiança de 95% em um intervalo de 0,000738 a 0,000996 segundos conforme verificado no trecho de código a seguir.

```
#Drop collection
tempo_ini = time.time()
db.drop_collection(name_or_collection='coleta')
tempo_fim = time.time()
print(i)
print('### NoSQL DELETE ### ')
print("Tempo de delete: {:.5f} segundos".format((tempo_fim - tempo_ini)))
```

Para o teste de performance nas consultas, foram executadas as mesmas consultas performadas no SQLite, confome podemos observar nos trechos de código a seguir.
QUERY 1:

```
find = {'Regiao - Sigla':'SE'}
tempo_ini = time.time()
count = db.coleta.count_documents(find)
tempo_fim = time.time()
print('### NoSQL ### ', count, 'registros.')
print("Tempo de consulta: {:.5f} segundos".format((tempo_fim - tempo_ini)))
```

A consulta QUERY 1 no MongoDB foi realizada em 0,791922 segundos, com confiança de 95% em um intervalo de 0,790025 a 0,793818 segundos, retornando os mesmos 491.866 registros.
QUERY 2:

```
tempo_ini = time.time()
result = db.coleta.aggregate([{'$group': {'_id':'null', 'avg_val':{'$avg':'$Valor de Venda'}}}])
tempo_fim = time.time()
print(i)
print('### NoSQL - preço médio de venda ### ')
for r in result:
    print(r['avg_val'])
print("Tempo de consulta: {:.5f} segundos".format((tempo_fim - tempo_ini)))
```

A consulta QUERY 2 no MongoDB foi realizada em 1,024759 segundos, com confiança de 95% em um intervalo de 1,022978 a 1,026540 segundos.
QUERY 3:

```
tempo_ini = time.time()
result = db.coleta.aggregate([{'$group': {'_id':'$Regiao - Sigla','avg_val':{'$avg':'$Valor de Venda'}}},
                              {"$sort": {'avg_val': 1}}])
tempo_fim = time.time()
print(i)
print('### NoSQL - preço médio de venda ### ')
for r in result:
    print(r)
print("Tempo de consulta: {:.5f} segundos".format((tempo_fim - tempo_ini)))
```

A consulta QUERY 3 no MongoDB foi realizada em 1,111994 segundos, com confiança de 95% em um intervalo de 1,110717 a 1,113271 segundos.

V. CONCLUSÃO

A partir dos experimentos realizados neste trabalho, sobre as tecnologias de Banco de Dados Relacionais e Orientados a Documentos, SQLite e MongoDB respectivamente, visando avaliar sua adequabilidade e performance na ingestão, deleção e consulta de grandes volumes de dados podemos verificar que o processo de inserção de dados utilizando-se o MongoDB performou em tempo computacional médio 82% menor do que no SQLite. Para o processo de deleção, os resultados foram favoráveis ao MongoDB, performando em tempo computacional médio 99% menor do que no SQLite. Para os processos de consulta, o MongoDB retornou o resultado na QUERY 1 em tempo computacional médio 79% menor do que o SQLite. Todavia na QUERY 2, um pouco mais complexa, o tempo médio de resposta do MongoDB foi 248% maior do que o procedimento no SQLite e, por fim, na QUERY 3, a de maior complexidade testada, o MongoDB retornou em tempo médio 44% maior do que o SQLite para o mesmo procedimento. Com isso, concluímos que para os procedimentos de ingestão, deleção e consultas simples cujo resultado é massivo, o MongoDB se mostra mais adequado no trato para grandes massas de dados com características analíticas e de séries temporais, em detrimento à facilidade de configuração e manipulação de ambiente apresentada pelo SQLite. O SQLite mostra-se mais adequado para consultas que envolvem agregações e ordenações. Sugere-se como trabalhos futuros a evolução do experimento em ambientes distribuídos, bem como com o uso de outras tecnologias e paradigmas de Bancos de Dados Massivos.

REFERÊNCIAS

[1] Ishwarappa and Anuradha, J. (2015). A brief introduction on big data 5vs characteristics and hadoop technology. volume 48, pages 319–324.

[2] Dados Abertos, https://dados.gov.br/dados/conjuntos-dados/srie-histrica-de-preos-de-combustveis-e-de-glp

[3] Nayak, Ameya, Anil Poriya, and Dikshay Poojary. "Type of NOSQL databases and its comparison with relational databases." International Journal of Applied Information Systems 5.4 (2013): 16-19.

[4] Radoev, Mitko. "A comparison between characteristics of NoSQL databases and traditional databases." Computer Science and Information Technology 5.5 (2017): 149-153.

[5] Li, Yishan, and Sathiamoorthy Manoharan. "A performance comparison of SQL and NoSQL databases." 2013 IEEE Pacific Rim conference on communications, computers and signal processing (PACRIM). IEEE, 2013.

[6] S. Ghemawat, H. Gobioff, S. Leung, The Google File System, 19th ACM Symposium on Operating Systems Principles, Lake George, 2003.

[7] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, 6th Symposium on Operating System Design and Implementation, San Francisco, 2004.

[8] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, R. Gruber, Bigtable: A Distributed Storage System for Structured Data, 7th Symposium on Operating System Design and Implementation, Seattle, 2006.

[9] Bhosale, S. T., Tejaswini Patil, and Pooja Patil. "Sqlite: Light database system." Int. J. Comput. Sci. Mob. Comput 44.4 (2015): 882-885.

[10] A. Fox, E. Brewer, Harvest, Yield and Scalable Tolerant Systems, Proc. 7th Workshop

[11] SQLLite. [citação 2023 Fev]. Disponível em: https://sqlite.org.

[12] MongoDB, http://www.mongodb.org/about/introduction/

[13] SQLAlchemy, https://www.sqlalchemy.org/

[14] PYMongo, https://pymongo.readthedocs.io/en/stable/

[15] MongoDB, https://www.mongodb.com/