Universidade Federal de São Carlos– UFSCar
Centro de Ciências Exatas e de Tecnologia– CCET
Departamento de Computação– DC
Programa de Pós-Graduação em Ciência da Computação– PPGCC

Caio Lazarini Morceli

Uma Avaliação Experimental entre MongoDB e Mongochain utilizando o *benchmark* YCSB+T

São Carlos
2024

Caio Lazarini Morceli

Uma Avaliação Experimental entre MongoDB e Mongochain utilizando o *benchmark* YCSB+T

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Software e Banco de Dados

Orientador: Ricardo Rodrigues Ciferri

São Carlos
2024

Aos meus pais, os quais sempre se dedicaram a me proporcionar a melhor educação e condições de vida possíveis e que, também, sempre me apoiaram nos momentos de dificuldade, então, dedico a vocês essa nova conquista como forma de gratidão.

Agradecimentos

Primeiramente a Deus por me dar saúde e coragem para enfrentar os desafios durante a vida e também por proporcionar oportunidades maravilhosas em meu caminho como a realização deste trabalho.

A todos os Técnicos Administrativos da Universidade Federal de São Carlos que realizaram um trabalho de qualidade para que todo o andamento relacionado a este curso fosse possível.

A todos os Docentes do departamento de computação da Universidade Federal de São Carlos que demonstraram suas excelentes capacidades em relação à docência e a todas as valiosas dicas dentro do ambiente acadêmico.

Ao professor Doutor Ricardo Rodrigues Ciferri, que desde o primeiro momento esteve disposto a me orientar, que teve paciência para me direcionar no caminho correto deste trabalho e que fez o possível para que eu obtivesse sucesso neste árduo percurso.

À professora Doutora Valéria Cesário Times, que fez parte desta pesquisa colaborando com diversas dicas, direcionamentos e importantes ensinamentos acadêmicos.

Aos meus familiares, amigos e todas as pessoas próximas que torcem por mim, acreditam em meu potencial e que desejam o meu bem e a minha felicidade.

Resumo

As propriedades Atomicidade, Consistência, Isolamento e Durabilidade (ACID) encontradas em um Sistema Gerenciador de Banco de Dados (SGBD) normalmente são relacionadas aos Banco de Dados (BD) relacionais. Por outro lado, o surgimento do paradigma Não Apenas SQL (NoSQL) chegou para atender as demandas que os BDs relacionais deixavam a desejar, tais como escalabilidade, disponibilidade e facilidade em tratar um volume maior de dados. Por consequência, o ecossistema NoSQL prosperou se aproveitando dos conceitos Basicamente Disponível, Estado Leve e Consistência Eventual (BASE) que privilegiam a escalabilidade e disponibilidade. Porém, a utilização desses conceitos também trouxe desvantagens, e a mais considerável foi a de crucificar a consistência do modelo ACID para priorizar a disponibilidade e o desempenho. Esse contraste gerou novos nichos de pesquisas sobre essa problemática. Dessa forma, o Mongochain foi criado na busca de acoplar os modelos ACID e BASE para usufruir de seus benefícios. Esse *software* utilizou uma tecnologia inovadora, a *blockchain*, a qual possui as características necessárias para garantir ACID e BASE, pois consegue executar transações confiáveis e com transparência em uma rede distribuída e descentralizada. Além do Mongochain, o MongoDB também trouxe melhorias com o passar do tempo. Uma delas foi o amparo a transações que preservam as propriedades ACID em múltiplos documentos. Em adição, o MongoDB disponibiliza vários níveis de consistência para se trabalhar com a persistência de dados, porém não há um estudo específico que analisa o desempenho desses níveis. Portanto, este trabalho busca realizar um estudo sobre os níveis de consistência do MongoDB, checando quais deles garantem a propriedade de consistência inclusa no modelo ACID e avaliar a corretude do Mongochain. Com isso, foi necessária a utilização de um *benchmark* que permitisse realizar testes de desempenho e consistência, os quais são encontrados no YCSB+T. Por fim, os resultados mostram que os níveis de consistência mais fortes do MongoDB e o Mongochain conseguiram garantir a consistência dos dados.

Palavras-chave: Consistência. MongoDB. Mongochain. Análise de Desempenho.

Abstract

The Atomicity, Consistency, Isolation and Durability (ACID) properties found in a Database Management System (DBMS) are normally related to relational Databases (DB). On the other hand, the emergence of the Not Just SQL (NoSQL) paradigm arrived to meet the demands that relational DBs left something to be desired, such as scalability, availability and ease of handling a larger volume of data. Consequently, the NoSQL ecosystem has thrived by taking advantage of the Basically Available, Light State and Eventual Consistency (BASE) concepts that prioritize scalability and availability. However, the use of these concepts also brought disadvantages, the most considerable of which was that it crucified the consistency of the ACID model to prioritize availability and performance. This contrast generated new research niches on this issue. Therefore, Mongochain was created in an attempt to couple the ACID and BASE models to enjoy their benefits. This software used an innovative technology, blockchain, which has the necessary characteristics to guarantee ACID and BASE, as it can execute reliable and transparent transactions in a distributed and decentralized network. In addition to Mongochain, MongoDB has also brought improvements over time. One of them was support for transactions that preserve ACID properties in multiple documents. In addition, MongoDB provides several consistency levels for working with data persistence, but there is no specific study that analyzes the performance of these levels. Therefore, this work seeks to carry out a study on MongoDB's consistency levels, checking which of them guarantee the consistency property included in the ACID model and evaluating the correctness of Mongochain. Therefore, it was necessary to use a benchmark that would allow performance and consistency tests to be carried out, which are found in YCSB+T. Finally, the results show that the stronger consistency levels of MongoDB and Mongochain were able to guarantee data consistency.

Keywords: Consistency. MongoDB. Mongochain. Performance Analysis.

Lista de ilustrações

Lista de tabelas

Lista de siglas

2L Protocolo de Bloqueio de Duas Fases

ACID Atomicidade, Consistência, Isolamento e Durabilidade

API Interface de Programação de Aplicação

BD Banco de Dados

BDD Banco de Dados Distribuídos

BASE Basicamente Disponível, Estado Leve e Consistência Eventual

BSON Binary JSON

CAP Consistência, Disponibilidade e Tolerância à Partição

CRUD *Create*, *Read*, *Update* e *Delete*

ID Identificador

JSON Notação de Objeto JavaScript

MVC Modelo, Visão e Controle

MVCC Controle de Concorrência Multiversão

NoSQL Não Apenas SQL

PACELC se Partição, então Disponibilidade ou Consistência, senão Latência ou Consistência

PBFT Tolerância Prática a Falhas Bizantinas

PoS Prova de Participação

PoW Prova de Trabalho

SGBD Sistema Gerenciador de Banco de Dados

SGBDR Sistema Gerenciador de Banco de Dados Relacional

SI Isolamento Instantâneo

SQL Linguagem de Consulta Estruturada

XML Linguagem de Marcação Extensível

YCSB Yahoo! Cloud Serving Benchmark

Sumário

Capítulo 1

Introdução

Este capítulo apresenta a contextualização sobre a importância de se avaliar e comparar os desempenhos de aplicações que têm o objetivo de gerenciar transações em um ambiente distribuído. Além disso, está em evidência o uso da tecnologia *blockchain*, a qual vem sendo utilizada de forma conjunta com aplicações que realizam o gerenciamento de transações. Sendo assim, será vista a motivação para a comparação entre os sistemas investigados nesta pesquisa de Mestrado. Será avaliada a corretude com relação à consistência dos dados do *framework* denominado Mongochain, o qual, atualmemte, é uma solução do estado da arte na integração entre as propriedades Atomicidade, Consistência, Isolamento e Durabilidade (ACID), Basicamente Disponível, Estado Leve e Consistência Eventual (BASE) e *blockchain* para o gerenciamento de transações em um ambiente distribuído. O problema de pesquisa foi elaborado e os objetivos foram descritos para a elucidação do problema. Com isso, será justificado o uso de um *software* de *benchmark* que será utilizado com o objetivo de comparar o desempenho do *framework* Mongochain e o Sistema Gerenciador de Banco de Dados (SGBD) Não Apenas SQL (NoSQL) MongoDB. Por fim, é descrita a estrutura desta dissertação de Mestrado.

1.1 Contextualização

Fornecer consistência forte aos dados armazenados por meio de transações que preservam as propriedades ACID são características essenciais oferecidas por um Sistema Gerenciador de Banco de Dados Relacional (SGBDR). Entretanto, essa atividade não é uma tarefa simples de se realizar quando se trata da execução de transações em sistemas distribuídos compostos por *clusters* de computadores, os quais têm facilidade em escalar horizontalmente com a adição de mais nós em sua infraestrustura. Desta forma,

os desenvolvedores de *software* optam por utilizar um novo paradigma de banco dados denominado de SGBD NoSQL, tal como MongoDB, para tratar estes dados com maior disponibilidade e com melhor desempenho. Em contrapartida, ao acatar os conceitos BASE, a consistência forte encontrada no modelo ACID é prejudicada. Pritchett (2008) explica que independente da estratégia de escalonamento horizontal adotada, os desenvolvedores são obrigados a escolher entre consistência e disponibilidade.

Apesar disso, o SGBD NoSQL MongoDB, de acordo com a sua documentação oficial, registra que a partir de sua versão 4.0 garante as propriedades ACID para operações de escrita e leitura em múltiplos documentos. Ademais, existem alguns níveis de consistência nesse SGBD que podem ser configurados para manter uma consistência forte ou eventual. Estes níveis de consistência também são alvos de estudo nesta pesquisa.

Outra tecnologia que está em pleno crescimento e vem sendo abordada de forma frequente na literatura é a *blockchain*. Tecnologia essa que é uma estrutura de dados que mantém o registro das transações em uma rede descentralizada e distribuída (ELROM, 2019). De acordo com Chowdhury et al. (2018), a *blockchain* vem sendo utilizada em diferentes domínios devido aos benefícios que ela garante, como exemplo armazenamento de dados distribuídos e auditoria imutável. A *blockchain* é uma tecnologia que se encontra no estado da arte e ainda está em seu estágio inicial de desenvolvimento, fazendo com que governos e institutos de pesquisa invistam recursos para novas descobertas e avanços (CHOWDHURY et al., 2018). Além da *blockchain* permitir que novos dados sejam adicionados ao banco de dados, também garante que todos os nós da rede possuam exatamente os mesmos dados e que esses dados sejam protegidos de qualquer modificação (MUZAMMAL; QU; NASRULIN, 2019).

1.2 Motivação

Considerando o modelo BASE, a consistência encontrada em um SGBD NoSQL é apontada como eventual, pois prioriza a disponibilidade de um sistema em detrimento da consistência forte que é encontrada com o uso das propriedades ACID. Indrawan-Santiago (2012) explica que para permitir que uma grande quantidade de dados seja processada em um ambiente distribuído, é aceitável relaxar a propriedade de consistência. Dessa forma, não se pode garantir que a escrita de um dado gere leituras atualizadas do mesmo dado por outros processos simultâneos. Considerando tal prerrogativa, pesquisadores buscaram realizar estudos com o objetivo de atingir a integração entre os modelos ACID e BASE para manter a consistência, escalabilidade e disponibilidade dos dados ao executar transações.

O SGBD NoSQL MongoDB é um SGBD que busca atender os requisitos de consistência forte que são garantidos pelas propriedades ACID por meio de níveis de consistência que podem ser configurados para atingir tal objetivo. Além disso, a tecnologia *blockchain* surgiu como uma nova maneira de se trabalhar com dados dentro de um ambiente

descentralizado. Sendo assim, Sousa (2020) apresentou um novo *framework*, o qual se encontra no estado da arte, denominado Mongochain, que busca atingir a integração entre as propriedades ACID e BASE adicionando uma *blockchain*.

Considerando esta problemática envolvendo as propriedades ACID e BASE e levando em consideração que a *blockchain* pode garantir essas propriedades na realização de transações em um ambiente distribuído e descentralizado de *clusters* de computadores, é importante investigar o estado da arte e verificar a corretude de novos sistemas que têm a proposta de unir ACID, BASE e *Blockchain* para se ter mais clareza se a propriedade de consistência enontrada no modelo ACID é de fato garantida nesses *softwares*. Além disso, realizar a comparação desses sistemas com outros SGBDs se torna relevante, pois faz sentido descobrir quais são os gargalos existentes entre os níveis de consistência destes SGBDs e também verificar o impacto que uma rede *blockchain* integrada a um Banco de Dados Distribuídos (BDD) terá com relação ao seu desempenho.

A partir desses avanços, é relevante investigar qual o impacto existente entre os níveis de consistência fornecidos pelo MongoDB e suas garantias de manutenção das propriedades ACID e, além disso, avaliar a corretude do *framework* Mongochain relacionada à consistência de dados para validar a sua proposta. Um grande desafio na realização desta pesquisa de Mestrado é lidar com todas as tecnologias anteriormente descritas, as quais já são complexas quando usadas isoladamente, e muito mais complexas quando usadas conjuntamente. Outro desafio consiste em compreender e implementar a solução em um ambiente distribuído real composto por um *cluster* de computadores.

1.3 Objetivos

Para se obter um resultado relativo à questão de pesquisa, o objetivo geral desta pesquisa de Mestrado é realizar uma investigação a respeito dos níveis de consistência e garantias da propriedade de consistência encontrada no modelo ACID fornecidas pelo SGBD MongoDB e pelo *framework* Mongochain. Este objetivo é alcançado a partir do atendimento dos objetivos específicos relatados na sequência desta seção.

Neste sentido, um importante questionamento é o seguinte: "o SGBD MongoDB e o *framework* Mongochain garantem a propriedade de consistência encontrada no modelo ACID em um ambiente distribuído composto por um *cluster* de computadores, de acordo com as suas configurações padrão?". Outro questionamento é o seguinte: "Quais são os impactos existentes quando uma *blockchain* é inserida no *framework* Mongochain?". Dessa forma, entende-se a viabilidade de se investigar o impacto existente entre os níveis de consistência oferecidos pelo MongoDB e suas garantias de conservação da propriedade de consistência encontrada no modelo ACID. Além disso, esta pesquisa de Mestrado irá analisar a corretude do *framework* Mongochain para validar a sua proposta no ponto de vista de consistência dos dados. Essa corretude significa esclarecer se, de fato, o

Mongochain garante a propriedade de consistência proposta em seu projeto.

1.3.1 Objetivos Específicos

- ❏ Analisar a qualidade e o desempenho dos níveis de consistência do SGBD NoSQL MongoDB para determinar a garantia da propriedade de consistência encontrada nas propriedades ACID.
- ❏ Realizar a integração do *framework* Mongochain ao *benchmark* YCSB+T.
- ❏ Executar experimentos, por meio do uso do *benchmark* YCSB+T, para comparar o desempenho do SGBD MongoDB e o *framework* Mongochain em termos de tempo de execução.
- ❏ Analisar a qualidade dos resultados gerados pelo *benchmark* YCSB+T para avaliação da corretude do *framework* Mongochain no que se refere à consistência dos dados.

1.4 Hipóteses de Pesquisa

A partir dos objetivos citados nas seções 1.3 (Objetivos) e 1.3.1 (Objetivos Específicos), é possível validar as seguintes hipóteses no decorrer desta pesquisa:

- ❏ O SGBD MongoDB garante consistência forte quando utiliza os seus níveis de consistência mais restritos, por meio da escolha adequada de valores para os parâmetros *write concern*, *read concern*, e *read preference*;
- ❏ O SGBD MongoDB não garante consistência forte quando utiliza os seus níveis de consistência mais básicos, por meio da escolha dos valores para os parâmetros *write concern*, *read concern* e *read preference*;
- ❏ O desempenho do MongoDB melhora ou piora conforme os níveis de consistência adotados, quanto mais forte os parâmetros para consistência, pior o seu desempenho;
- ❏ É possível testar e validar o *framework* Mongochain por meio do uso do *benchmark* YCSB+T para verificar a garantia de consistência dos dados.

1.5 Organização do Trabalho

Além deste Capítulo 1, esta monografia de dissertação de Mestrado está organizada em 6 capítulos da seguinte forma:

- Capítulo 2: Descreve a fundamentação teórica composta de conceitos sobre sistemas transacionais, propriedades ACID, armazenamentos de dados distribuídos, SGBD NoSQL MongoDB, tecnologia *blockchain* e análise de desempenho.
- Capítulo 3: Descreve o *framework* Mongochain, objeto de estudo desta pesquisa de Mestrado.
- Capítulo 4: Exibe os trabalhos relacionados que buscam avaliar os níveis de consistência do SGBD NoSQL MongoDB ou comparam o seu desempenho por meio do uso de uma ferramenta de *benchmark*.
- Capítulo 5: Aborda os resultados obtidos através dos experimentos realizados com o SGBD MongoDB.
- Capítulo 6: Aborda os resultados obtidos através dos experimentos realizados com o *framework* Mongochain.
- Capítulo 7: Conclusão descrevendo as contribuições desta pesquisa e as indicações para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Os princípios teóricos que formam a base para a compreensão desta pesquisa de Mestrado encontram-se neste capítulo. Com base na revisão da literatura, foi realizada a fundamentação teórica constando os seguintes tópicos: Sistemas transacionais, Bancos de Dados Distribuídos (BDD), SGBD MongoDB, Sistemas Descentralizados e Análise de Desempenho.

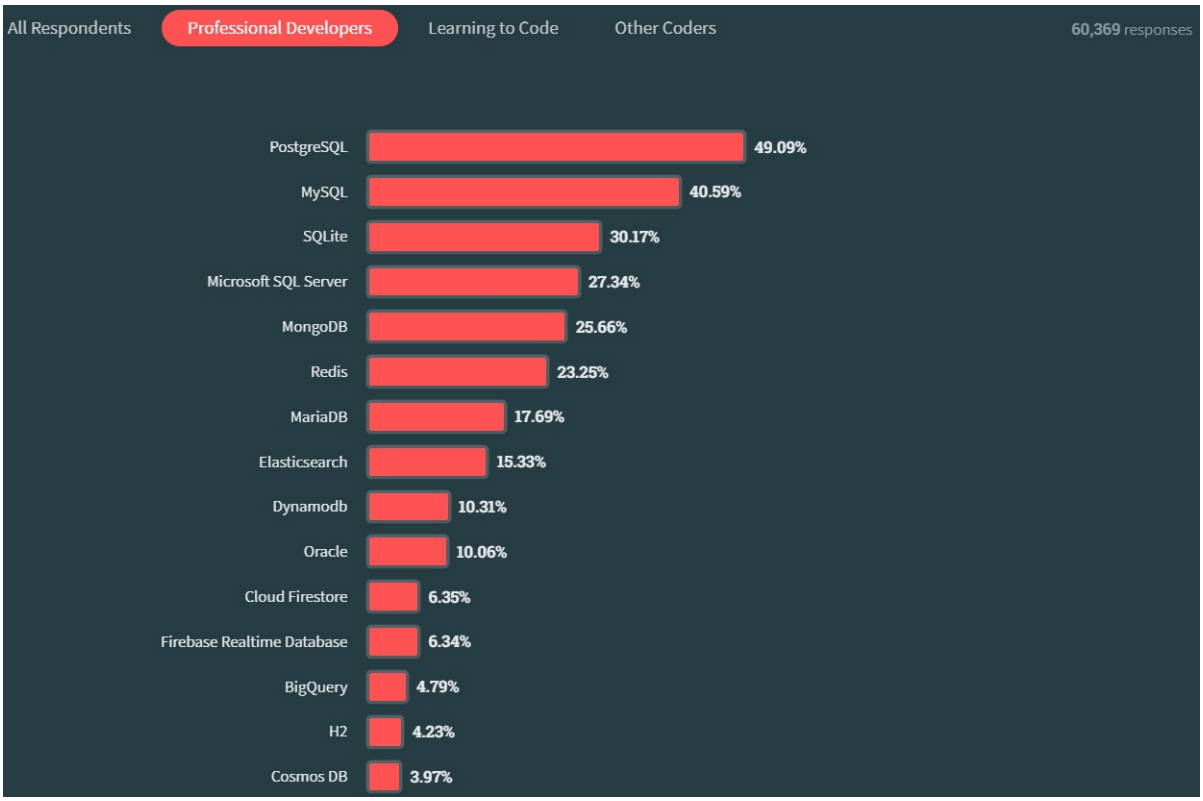
2.1 Sistemas Transacionais

Quando se fala em sistemas transacionais é importante relembrar como se deu a origem desse tipo de sistema. Antes desse avanço tecnológico, diversas informações como registros policiais, informações de bibliotecas, fichas médicas de pacientes e diversos outros tipos de informações eram armazenados de forma manual, sem uma técnica aprimorada para que todo o arquivamento e trabalho de recuperação dos registros fosse facilitado. Com a chegada dos SGBDs, o tratamento da informação teve uma melhora significativa e, dessa forma, foi estabelecido um novo padrão para gerenciamento de registros. Desde então, os SGBDs se popularizaram entre os desenvolvedores de *software* e, dentre eles, têm-se como mais populares os SGBDs relacionais, modelo esse criado por Codd (1970) [1]. É exibido na figura 1 o gráfico que aborda um *ranking* dos SGBDs mais usados de acordo com uma pesquisa realizada no site *StackOverflow*, o qual é um ambiente que tem seu principal conteúdo relacionado à programação de sistemas e SGBDs.

O gráfico mostra que apesar do surgimento de outros paradigmas diferentes do modelo relacional em SGBD (*e.g.,* NoSQL), este modelo continua imperando nas primeiras posições por possuir características importantes no tratamento de dados, tais como con-

[1] CODD, Edgar. Frank. A relational model of data for large shared data banks.

Figura 1 – Gráfico com o ranking de utilização dos SGBDs



Fonte: (STACKOVERFLOW, 2023)

seguir manter a integridade referencial, realizar transações que garantem a consistência dos dados e possuir a Linguagem de Consulta Estruturada (SQL) por padrão, a qual é utilizada para realizar operações de leitura e escrita na base de dados. Um dos objetos de estudo desta pesquisa de Mestrado é a transação. Sendo assim, serão apresentados os conceitos, o funcionamento e o porquê da sua utilização tanto para SGBDs relacionais quanto para SGBDs NoSQL.

Date (2003) explica que uma transação é uma operação lógica de trabalho, a qual não envolve necessariamente apenas uma única operação sendo realizada no Banco de Dados (BD). Comandos de BD como consultas, inserções, remoções e alterações podem estar dentro de transações que normalmente são implementadas por um programa de computador (ELMASRI; NAVATHE, 2011). Além disso, Silberschatz, Korth e Sudarshan (2006) relatam que, na visão de um usuário, esse conjunto de operações realizadas aparentam ser uma única entidade. Dessa forma, o usuário acredita ter feito apenas uma única tarefa no sistema. Um exemplo simples de transação é a transferência de valor monetário de uma determinada conta para outra. Esse processo é constituído de duas operações de *update* no BD. Ou seja, o valor é debitado do saldo de uma conta onde se ocorre a primeira atualização e, na outra conta, ocorre uma atualização de crédito referente ao débito realizado

anteriormente. Na figura 2, pode-se observar, em pseudocódigo, o funcionamento de uma transação simples considerando um ambiente bancário.

Figura 2 – Transação em pseudocódigo

```
BEGIN TRANSACTION;

UPDATE conta_cliente 123{
   SALDO <= SALDO - $ 100
};

UPDATE conta_cliente 456{
   SALDO <= SALDO + $ 100
};


IF houve algum erro THEN
   GO TO DESFAZER;
END IF;


COMMIT;                    /* Término bem sucedido */
GO TO FINALIZAR;


DESFAZER:
   ROLLBACK;               /* Término malsucedido */


FINALIZAR:
   RETURN;
```

Fonte: Adaptado de Date (2003)

Nesse exemplo, o início da transação encontra-se na operação *BEGIN TRANSACTION*. A partir desse momento, a sequência de operações realizadas encontra-se dentro de um contexto transacional. Dessa forma, as duas operações seguintes de *update* são imperceptíveis para o usuário. Por fim, a operação de *COMMIT* ou *ROLLBACK* finaliza a transação. A operação de *commit* significa que o término de uma transação foi realizado com sucesso, o que corresponde a um ponto em que o BD está em um estado confiável e correto, enquanto *rollback* assinala um término malsucedido e desfaz todas as operações realizadas dentro do contexto transacional, devolvendo o estado do BD igual ao que estava em *BEGIN TRANSACTION* (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

Elmasri e Navathe (2011) explicam que as transações devem possuir diversas propriedades que normalmente são chamadas pelo acrônimo ACID. Na seção seguinte, essas propriedades são abordadas com mais clareza de detalhes.

2.1.1 Propriedades ACID

É natural que diversos usuários de uma aplicação *Web* acessem concomitantemente um mesmo item de dados. Dessa forma, um sistema de processamento de transações aceita que diversas transações sejam executadas ao mesmo tempo. Essa situação faz com que muitas transações atualizem dados simultaneamente, o que aumenta consideravel-

mente as complicações relativas à consistência dos dados (SILBERSCHATZ; KORTH; SUDARSHAN, 2006). Além disso, não se pode descartar as possibilidades de falhas do sistema. Silberschatz, Korth e Sudarshan (2006) explicam que um sistema de computador está suscetível a falhas pois se encontra em um ambiente envolvido por *hardware* e *software* que podem apresentar anomalias. Geralmente, nessas imperfeições estão inclusas falhas no próprio computador, tais como um problema de *hardware* ou de rede, falhas na própria transação, tais como no caso de uma divisão por zero ou um estouro de inteiro, falhas de disco e também falhas catastróficas como uma queda brusca de energia (ELMASRI; NAVATHE, 2011). Para Özsu e Valduriez (2011), um sistema resiliente é aquele que suporta falhas do sistema e pode continuar a fornecer serviços mesmo quando ocorrem essas anomalias.

Considerando essa problemática, é necessário que o SGBD possua as propriedades ACID para garantir que os dados fiquem íntegros (SILBERSCHATZ; KORTH; SUDARSHAN, 2006). A seguir, cada característica do modelo ACID é abordada.

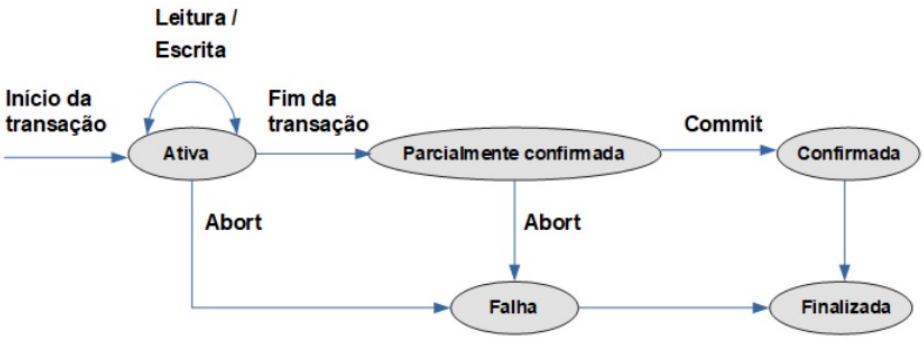
- Atomicidade

  A atomicidade refere-se ao fato de que uma transação é tratada como uma unidade atômica de operação em que todas as ações da transação são concluídas e caso alguma delas não seja, então todas as ações realizadas são desfeitas (ÖZSU; VALDURIEZ, 2011).

  Pode-se exemplificar a atomicidade de acordo com a figura 2 exibida anteriormente. O SGBD não deve permitir que apenas uma das atualizações seja executada e a outra não, pois essa operação deixaria o BD em um estado inconsistente, já que a conta de destino não teria o saldo atualizado com o valor transferido da conta remetente. De acordo com Silberschatz, Korth e Sudarshan (2006), esse tipo de situação reflete a um estado irreal do BD, sendo chamado de estado inconsistente. Dessa forma, um sistema que permite o gerenciamento de transações garante que se durante a realização de algumas operações na transação ocorrer uma falha que antecede o fim da transação, então todas as atualizações serão desfeitas (DATE, 2003).

  Para Silberschatz, Korth e Sudarshan (2006), é uma responsabilidade do próprio SGBD garantir a atomicidade e essa garantia vem por meio de um componente denominado gerenciador de transação. Date (2003) relata que para entender o modo como esse componente funciona, é essencial compreender que a operação de *COMMIT* informa ao gerenciador de transações que uma transação foi concluída com sucesso e, ao contrário disso, a operação de *ROLLBACK* informa que houve uma falha na transação e que o BD pode estar em um estado inconsistente e que todas as operações realizadas até então devem ser desfeitas. A figura 3 exibe um diagrama ilustrando os estados em que uma transação se encontra.

Figura 3 – Estados de uma transação



Fonte: Adaptado de (ELMASRI; NAVATHE, 2011)

Neste diagrama, é possível notar que as operações de leitura e escrita de uma transação se encontram entre os estados de *BEGIN* e *END TRANSACTION*. Enquanto a transação não termina, é possível ocorrer um erro e ser abortada executando um *ROLLBACK* e, por outro lado, caso tudo ocorra sem problemas, o *COMMIT* é executado.

Conforme Silberschatz, Korth e Sudarshan (2006), cada estado tem o seguinte significado:

- **Ativa**

  Enquanto está em execução, a transação permanece neste estado.

- **Parcialmente confirmada**

  Estado posterior à execução da última instrução.

- **Falha**

  Estado onde uma transação se encontra após a descoberta de que uma execução normal não pode continuar.

- **Abortada**

  Estado em que o *rollback* foi executado e o BD foi restaurado ao estado prévio da transação

- **Confirmada**

  Término bem sucedido da transação.

❑ Consistência

Problemas com consistência de dados podem resultar em distúrbios e complicações substanciais em aplicações *Web*. Um exemplo simples é quando um *site* de vendas possui um produto em estoque. Porém, no instante em que um usuário executa uma consulta para a listagem de produtos, este produto aparece zerado. Compreender

os conceitos básicos referentes à integridade dos dados e como ela funciona é o início para manter a qualidade e coerência do BD.

Normalmente fica a cargo dos desenvolvedores criarem regras de integridade dos dados. Entretanto, também é possível que o SGBD imponha regras de integridade. Caso alguma regra seja violada, isso implica que os dados nunca serão consistentes (TIWARI, 2011). Para Elmasri e Navathe (2011), um estado consistente na base de dados significa que essas regras restritivas são respeitadas e são especificadas no esquema do BD. Sendo assim, quando uma transação é realizada, ela transforma um estado anterior do BD que se encontrava correto em outro estado coerente e íntegro.

- Isolamento

  Quando transações são executadas paralelamente, o isolamento acontece. Esse fato significa que o sistema garante que uma transação não terá conhecimento da execução de outra. De acordo com Ramakrishnan e Gehrke (2003), o controle do limite de exposição de uma determinada transação às ações de outras transações que estão sendo executadas no mesmo momento fica a cargo do nível de isolamento utilizado para o controle de concorrência.

  Em um SGBD que contém diversas transações ocorrendo simultaneamente, caso não exista um controle nos dados que estão sendo compartilhados, as transações poderão encontrar estados intermediários inconsistentes, os quais ocorrem por tarefas sendo realizadas por outras transações naquele momento e, dessa forma, atualizações incorretas podem ser resultadas aos dados armazenados na base de dados (SILBERSCHATZ; KORTH; SUDARSHAN, 2006). Considerando tal evento, os SGBDs precisam estar preparados para lidar com isso. É necessário que os SGBDs ofereçam mecanismos para realizar o isolamento entre transações concomitantes. Elmasri e Navathe (2011) explicam que a propriedade de isolamento é determinada pelo mecanismo de controle de concorrência do SGBD.

  Em uma seção posterior neste trabalho será abordada com mais clareza a questão do controle de concorrência.

- Durabilidade

  Um SGBD está propenso a falhas a qualquer momento, inclusive após o término bem sucedido de uma transação. Mesmo nesse cenário, a propriedade de durabilidade garante que todas as atualizações que já foram confirmadas por uma transação persistam na base de dados. Conforme Özsu e Valduriez (2011), a durabilidade está relacionada à propriedade das transações a qual garante que, a partir do momento em que uma transação é finalizada com sucesso, seus resultados são permanentes e não é possível que uma falha posterior no BD afete a transação concluída. Portanto,

o SGBD garante que transações finalizadas com sucesso serão persistentes mesmo após possíveis falhas subsequentes.

É observável que o modelo ACID é muito rígido e estrito, voltado para a consistência forte e segurança, o que, de certa forma, impacta no desempenho. Muitas aplicações, depois do advento da *Internet*, buscam uma relação de custo-benefício entre desempenho e segurança. Também devido a esse fato, houve o surgimento dos SGBDs NoSQL e do modelo BASE. Considerando isso, tem-se também como objeto de estudo nesta dissertação o SGBD NoSQL *MongoDB* que será abordado posteriormente em outra seção.

2.1.2 Concorrência

O controle de concorrência é extremamente importante para tratar da integridade dos dados transacionados no BD. Em um ambiente multiusuário, operações concorrentes podem abalar a consistência ou integridade da base de dados. Diante disso, os mecanismos de controle de concorrência são necessários para preservar a consistência diante do acesso concorrente aos recursos (MENASCÉ; NAKANISHI, 1982). Além disso, conforme Ramakrishnan e Gehrke (2003), ações de diferentes transações são intercaladas pelo SGBD para melhorar o desempenho, porém nem todas devem ser permitidas. Berenson et al. (1995) explicam que níveis de isolamento mais baixos aumentam a concorrência de transações, mas aumentam o risco de deixar o BD em um estado inconsistente, já que as transações concorrentes executadas em um nível de isolamento inferior podem acessar transações que ainda não fizeram o *commit*. Esses níveis de isolamento são muito criteriosos, pois dependendo do nível de isolamento que estiver configurado no SGBD ou na sessão em que uma determinada transação estiver acontecendo, o comportamento dessa transação pode variar muito e os impactos serão diferentes (RAMAKRISHNAN; GEHRKE, 2003). A partir disso, podem acontecer algumas anomalias quando as transações são executadas sem o devido gerenciamento tais como leitura suja, atualização perdida, leitura não repetitiva e fantasma (BERENSON et al., 1995). Na sequência, as situações que podem desencadear essas anomalias são apresentadas:

- Leitura suja

  A transação lê dados escritos por uma sessão concorrente que ainda não foi concluída. Nesse caso, é possível haver duas sessões trabalhando com os mesmos dados e as operações sendo realizadas em cima desses dados, as quais ainda não foram finalizadas por nenhuma sessão, ficam disponíveis para que uma dessas sessões possa realizar a leitura dos dados onde a sessão oposta ainda não terminou a operação. Isso gera uma quebra de atomicidade, pois uma sessão acaba interferindo na operação da outra causando inconsistência nos dados.

- Atualização Perdida

A atualização perdida ocorre quando uma transação T1 realiza a leitura de um item de dados e, na sequência, uma transação T2 atualiza o mesmo item de dados, então T1 atualiza o item de dados e realiza o *commit*. Logo, isso acarreta em uma perda de atualização.

- Leitura não repetitiva

  Ao tentar ler novamente dados lidos anteriormente, descobre-se que esses dados foram alterados. Nesse caso, uma segunda sessão está realizando a manipulação (*UPDATE*) desses mesmos dados de forma concomitante. Isso ocasiona a releitura de dados com valores diferentes.

- Fantasma

  Difere da leitura não repetitiva, pois ao executar novamente uma consulta, o conjunto de dados retornados não contém o mesmo conjunto de dados. Pode ocorrer de serem retornados mais ou menos registros do que a consulta feita anteriormente. Nesse caso, uma segunda sessão está realizando a manipulação (*INSERT / DELETE*) desses mesmos dados de forma concorrente. Isso ocasiona a releitura de dados com valores diferentes.

Ramakrishnan e Gehrke (2003) explicam que as anomalias exibidas anteriormente representam a composição de quatro níveis de isolamento. São eles: *SERIALIZABLE*, *REPEATABLE READ*, *READ COMMITTED* e *READ UNCOMMITTED*. Estes níveis são utilizados para especificar a magnitude de vulnerabilidade referente às violações de consistência para quando as transações concorrentes forem executadas. A tabela 1 exibe estes níveis e as suas relações com as anomalias relatadas.

Tabela 1 – Níveis de isolamento e suas respectivas anomalias

Nível	**Leitura Suja**	**Leitura Não Repetitiva**	**Fantasma**
READ UNCOMMITTED	É possível	É possível	É possível
READ COMMITTED	Não	É possível	É possível
REPEATABLE READ	Não	Não	É possível
SERIALIZABLE	Não	Não	Não

Fonte: Adaptado de (RAMAKRISHNAN; GEHRKE, 2003)

Serializable é o nível mais estrito, pois não permite nenhum tipo de anomalia visto anteriormente. Nesse nível de isolamento não existe a concorrência das transações em cima de um mesmo item de dados, pois a transação realiza o bloqueio no item de dados desejado. Em contrapartida, essa escolha desencadeia uma queda de desempenho devido ao fato de que se outra transação quiser utilizar o mesmo item de dados, deverá aguardar o a liberação do bloqueio feito pela transação anterior. Dessa forma, conforme as transações

chegam ao BD, é necessário que cada uma espere a finalização das transações anteriores para poder iniciar a sua execução.

No nível de isolamento *Repeatable Read*, nenhuma outra transação poderá ler os dados que foram lidos ou escritos por outras transações até que elas terminem. Deixa-se de existir bloqueios compartilhados, logo, todos os bloqueios são exclusivos para cada transação. Sendo assim, quaisquer duas transações que estejam concorrendo aos mesmos dados terão que aguardar o término da outra transação para fazer o novo bloqueio. Isso não significa que este nível de isolamento bloqueie totalmente os dados, apenas não permite a concorrência aos mesmos dados concomitantemente.

O nível *Read Commited* já possui isolamento com uma segurança maior que o nível *Read Uncommited*, pois permite apenas que os dados possam ser lidos após outras transações que manipularam esses mesmos dados terminarem com *commit* ou *rollback*. Neste nível já existe uma presença de bloqueios mais eficientes no BD, porém ainda não é o nível mais seguro.

Por fim, o nível com menos segurança no quesito consistência de dados é chamado de *Read Uncommited*, pois permite a ocorrência de todos os tipos de anomalias. Esse tipo de transação permite a leitura de alterações realizadas em um registro por outra transação concorrente, pois não obtém bloqueios compartilhados antes de realizar a leitura. Em contrapartida, oferece a capacidade de diversas transações trabalharem de forma concorrente em cima de um mesmo item de dados.

Para que um SGBD trabalhe com transações, é necessário a utilização do mecanismo de controle de concorrência. Sendo assim, Date (2003) relata que uma forma desses problemas serem resolvidos é denominado bloqueio, o qual representa uma ideia de que quando uma transação está interessada em um item de dados, este bloqueio garante que o registro não será alterado por uma transação concorrente enquanto a transação dona deste bloqueio estiver ativa. Considerando tal fato, duas abordagens podem ser admitidas: pessimista e otimista. Conforme Menascé e Nakanishi (1982), um mecanismo orientado a bloqueio pode ser considerado pessimista quando os recursos do BD são bloqueados mesmo que as transações não entrem em conflito com outras transações e, por outro lado, a abordagem otimista se baseia na ideia de que os conflitos acontecem com pouca frequência e as ações para preservar a integridade do BD devem ser tomadas apenas quando os problemas ocorrerem.

Tratando-se de bloqueios, um dos protocolos mais utilizados é o Protocolo de Bloqueio de Duas Fases (2L), em que Silberschatz, Korth e Sudarshan (2006) explanam que é demandado que cada transação solicite o bloqueio e desbloqueio em duas fases. Diante disso, a primeira fase é denominada fase de crescimento, a qual a transação somente pode obter bloqueios sem realizar a liberação. Já a segunda fase, denominada fase de encolhimento, apenas faz o contrário. Ou seja, apenas pode realizar a liberação e não pode obter novos bloqueios. Apesar de permitir a serialização, o uso de bloqueios pode

causar o problema de *deadlock*. De acordo com Elmasri e Navathe (2011), o *deadlock* é uma situação onde um conjunto de transações aguarda por um item bloqueado por outra transação criando um ciclo dependente denominado impasse. Em outras palavras, uma transação T1 aguarda um item bloqueado por uma transação T2 enquanto a transação T2 aguarda por um item bloqueado por uma ação T3 e essa ação T3 aguarda por um item bloqueado pela transação T1. Esse ciclo faz com que nenhuma das transações consiga avançar para a realização do desbloqueio dos recursos, caracterizando uma situação de impasse. Dessa forma, a ideia do protocolo otimista serve para que os impasses aconteçam com pouca frequência e as ações para preservar a integridade do BD sejam tomadas, tais como um *rollback*.

Outros protocolos que são muito utilizados são chamados de protocolo de Controle de Concorrência Multiversão (MVCC) e protocolo de *timestamp*. Ramakrishnan e Gehrke (2003) relatam que o MVCC tem como proposta manter diversas cópias das versões do BD utilizando o protocolo de *timestamp*. O *timestamp* identifica exclusivamente uma transação no SGBD por meio de uma marcação de data e hora e não faz a utilização de bloqueios, o que evita a consequência de *deadlocks* (ELMASRI; NAVATHE, 2011). Além disso, controla a ideia de serialidade das transações, buscando garantir que uma transação nunca espere para ler um item do BD.

Baseado nos protocolos de MVCC e *timestamp*, Berenson et al. (1995) apresentaram um nível de isolamento chamado Isolamento Instantâneo (SI) que se encontra entre os níveis *Read Commited* e *Repeatable Read*. Como cada transação possui um registro lógico de data e hora, a abordagem do SI consegue realizar o controle de concorrência entre as transações a partir de um carimbo de tempo de início e de fim da transação. Ou seja, quando ocorrer de alguma transação concorrente sobrepor o intervalo de início e fim de outra transação e ambas estiverem realizando a escrita em cima de um mesmo item de dados, uma delas será abortada (ALOMARI et al., 2008).

2.1.3 Recuperação de Falhas

Em um sistema sempre existe a possibilidade de ocorrência de falhas, principalmente quando se trata de um sistema distribuído em que diversos nós fazem parte do ambiente. Falhas de *hardware*, falhas de rede e falhas de particionamento são alguns dos motivos para a falha de um sistema. Considerando essas possibilidades, é importante possuir uma estratégia para a recuperação de falhas. Para atingir tal objetivo, Elmasri e Navathe (2011) relatam que as informações a respeito das mudanças aplicadas pelas diversas transações em cima dos itens de dados devem ser mantidas pelo sistema.

A propriedade de atomicidade das transações requer que os efeitos de todas as transações confirmadas e nenhum dos efeitos de transações incompletas sejam refletidos nos itens de dados acessados. Coulouris et al. (2011) explanam que esta propriedade pode ser descrita em termos de dois aspectos: durabilidade e atomicidade de falha. A durabilidade

consiste no ato dos objetos estarem salvos permanentemente no SGBD e, dessa forma, ficando disponíveis independentemente do sistema falhar ou não. Já a atomicidade de falha roga que quaisquer efeitos das transações sejam atômicos independentemente de travamento do sistema. Esses dois requisitos não são independentes um do outro e podem ser manipulados pelo mesmo mecanismo, denominado gerenciador de recuperação de falhas. Para Coulouris et al. (2011), as tarefas de um gerenciador de recuperação de falhas são:

- ❏ fornecer a durabilidade, ou seja, deixar os itens de dados salvos em armazenamento permanente, para as transações confirmadas;
- ❏ conseguir restaurar os itens de dados após a ocorrência de uma falha;
- ❏ reconhecer o arquivo de recuperação existente para prover um melhor desempenho da recuperação; e
- ❏ recuperar o espaço de armazenamento;

Portanto, quando se trata de falhas ocorridas devido às transações, o sistema deve possuir a capacidade de manter o BD consistente com os dados antes da ocorrência da falha (ELMASRI; NAVATHE, 2011).

2.2 Bancos de Dados Distribuídos

Com toda a demanda crescente de digitalização e migração de serviços para nuvem, o aumento da carga de dados imposta aos servidores *Web* e, por consequência dos servidores de bancos de dados, é notória. Sendo assim, o que dá norte aos estudos dos bancos de dados distribuídos é a escalabilidade. A escalabilidade é uma métrica utilizada para medir como o aumento de recursos computacionais impacta no desempenho do sistema (FOWLER, 2002). Dentro dessa característica, Fowler (2002) aborda dois modelos conhecidos para a obtenção da escalabilidade.

- ❏ Escalabilidade vertical: é uma arquitetura centralizada, em que o servidor central de banco de dados tem a sua capacidade incrementada. Logo, espaço em disco, espaço em memória RAM e capacidade de processamento são potencializados com um *hardware* mais potente.
- ❏ Escalabilidade horizontal: é uma arquitetura distribuída em que, diferente do tipo de escalabilidade anterior, o foco não está voltado ao aumento do potencial de *hardware* do servidor central de banco de dados. O foco está no incremento de servidores, os quais não têm a necessidade de serem tão potentes quanto um servidor que escala

de forma vertical. Porém, seu sucesso se encontra na distribuição de tarefas de processamento entre os nós do ambiente distribuído. Além disso, tudo isso é feito sem que o usuário perceba a existência dessa distribuição de tarefas. Tanenbaum e Steen (2007) relatam que essa arquitetura distribuída compreende um conjunto de computadores independentes que aparecem como um único sistema harmonioso para o usuário.

De acordo com Sadalage e Fowler (2013), o modelo de escalabilidade vertical necessita de um custo alto para tornar o servidor potente o suficiente para atender o aumento do volume de dados e do número de clientes. Sendo assim, faz sentido escalar de maneira horizontal, pois, por meio de um *cluster*, é possível adicionar mais nós compostos por um *hardware* menos potente, demandando um custo bem inferior (TIWARI, 2011). Considerando tal prerrogativa, os principais BDD se encontram dentro do movimento NoSQL, o qual é descrito com mais detalhes na sequência.

2.2.1 SGBD NoSQL

De acordo com Sharma e Dave (2012), o termo NoSQL foi utilizado em sua primeira vez por Strozzi (1998) [2] referindo-se a um BD relacional que suprimiu o uso da linguagem SQL. A partir disso, com um aumento gigantesco de dados, condicionado pela ascensão da tecnologia que facilitou o relacionamento dos usuários tanto na tecnologia *Web* quanto na móvel, originou-se a necessidade da utilização de um estilo de BD propício para as novas demandas. Atualmente, uma geração nova de SGBDs vêm se espalhando, aumentando seu uso e especialização. Tais SGBDs são conhecidos como BD NoSQL. Este termo se refere à abordagem de um tipo novo de SGBD que não possui base no modelo tradicional de BD relacional (CORONEL; MORRIS, 2017).

Os SGBDs NoSQL têm o objetivo de propiciar maior escalabilidade e disponibilidade. Dessa forma, Mccreary e Kelly (2014) explicam que esses SGBDs executam com bom desempenho em *clusters* de computadores, o que acaba incorrendo em uma alternativa interessante para trabalhar com cargas de dados mais volumosas típicas de *big data*. Tal fato é decorrente de seu esquema ser flexível, não possuindo a obrigação de ter dados normalizados, questões a respeito de integridade referêncial ou quaisquer outras delimitações observadas nos armazenamentos de dados relacionais. Tiwari (2011) explica que um SGBDR consegue escalar caso tenha suas tabelas desnormalizadas, elimine restrições e relaxe a garantia de transações, porém essas alterações levam um SGBDR a se parecer a um produto NoSQL. Porém, este não é o propósito original para o qual um SGBDR foi projetado.

Algumas categorias do ecossistema NoSQL, tais como chave-valor, documentos e família de colunas possuem uma orientação de modelo de dados em que Sadalage e Fowler

[2] $http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/$

(2013) denominam como orientação agregada. Além dessas três categorias, também existem os SGBDs NoSQL de grafos, os quais são baseados na teoria dos grafos. Dessa forma, tendo como base os nós como entidade, atributos sendo representados por propriedades e arestas representando relacionamentos, que são o foco deste tipo de SGBD NoSQL (SHARMA; DAVE, 2012).

A orientação agregada adota uma abordagem diferente da encontrada no modelo relacional tradicional. Enquanto o modelo relacional armazena as informações em uma estrutura de dados mais limitada, denominada como tupla (*i.e.,* linha), o agregado permite uma manipulação mais complexa dos dados permitindo que estruturas de dados aninhadas existam (SADALAGE; FOWLER, 2013). Dessa forma, um registro mais complexo contendo listas e outras estruturas internas se torna possível.

Outro ponto importante a ser considerado no ambiente NoSQL são suas limitações. Este tipo de SGBD acaba por facilitar o trabalho com relação a volumes massivos de dados. Porém, isso incorre em um custo. O custo dessa flexibilidade elimina o poder da integridade transacional, integridade referencial e indexação flexível (TIWARI, 2011).

Na sequência, a categoria de SGBD NoSQL baseada em agregados descrita em Sadalage e Fowler (2013) e que é utilizada nesta pesquisa é abordada.

2.2.2 Orientado a Documentos

Meier e Kaufmann (2019) explicam que os SGBDs NoSQL orientados a documentos são livres de esquema e, dessa forma, é permitida uma extrema flexibilidade para o armazenamento de uma ampla gama de dados. Além disso, são constituídos por coleções, as quais armazenam tipos de dados distintos em formato Linguagem de Marcação Extensível (XML) ou Notação de Objeto JavaScript (JSON). Além disso, os documentos são autodescritos, uma estrutura de dados baseada em uma árvore hierárquica, e podem ser constituídos por mapas, coleções e valores escalares (SADALAGE; FOWLER, 2013).

Para Hecht e Jablonski (2011), é necessário que as chaves referentes aos pares chave-valor que dão forma ao documento em um SGBD NoSQL orientado a documentos sejam únicas. Uma chave única, denominada Identificador (ID), pertence a cada documento. Considerando essa prerrogativa, isso torna possível a identificação exclusiva de um documento dentro de uma coleção de documentos. De forma distinta a um BD chave-valor, o qual só permite a busca pela chave, um SGBD orientado a documentos permite que uma consulta consiga ser realizada pelo seu valor. Outra característica de SGBDs NoSQL orientados a documentos é o armazenamento de dados aninhados (*i.e.,* dados complexos).

Os SGBDs NoSQL orientados a documentos têm bastante utilização de forma conjunta com tecnologias atuais tais como a Interface de Programação de Aplicação (API), a qual é uma forma de executar o percurso de dados entre clientes e servidores. Além disso, Lerman (2011) ressalta que, com a ressalva de alguns modelos, é utilizado, mais comumente, o

Protocolo de Transferência de Hipertexto (HTTP) para acessar os dados e se utiliza de uma API para muitas linguagens de programação.

Na figura 4, um exemplo de documento com uma estrutura aninhada é exibido.

Figura 4 – Exemplo de documento aninhado

```
{
  "_id": ObjectId("3gfdvfgf1234876nhgv52030f25"),
  "nome": "Alberto",
  "sobrenome": "Silva",
  "email": "albertosilva@email.com",
  "meus_caes": [
    {
      "nome": "Jimmy",
      "raca": "Beagle",
      "sexo": "masculino"
    },
    {
      "nome": "Skeeter",
      "raca": "Vira-Lata",
      "sexo": "masculino"
    }
  ]
}
```

Fonte: Próprio Autor (2022)

O documento exibido representa uma estrutura aninhada em que é possível observar um relacionamento existente entre uma pessoa e seus cães. Esse tipo de estrutura é possível em se tratando de documentos. Já no caso de um BD relacional, esse relacionamento entre pessoa e cão provavelmente seria dividido em duas tabelas as quais se conectariam por intermédio da integridade referencial de chave estrangeira.

Um dos principais SGBDs NoSQL orientado a documentos utilizado no mercado é o MongoDB, o qual também é objeto de estudo desta pesquisa de Mestrado e será abordado com mais clareza de detalhes em uma seção posterior. Outros SGBDs NoSQL orietandos a documentos são: RavenDB e CouchDB.

2.2.3 Conceito BASE

O termo BASE aborda a ideia de que uma aplicação deve funcionar teoricamente a todo momento, sendo assim, basicamente disponível. Além disso, não possuir a necessidade constante de ser fortemente consistente, ou seja, adotar um estado leve e posteriormente se tornar consistente, o que configura a consistência eventual. Para Pritchett (2008), o modelo BASE configura-se como antagonista ao modelo ACID, pois este é pessimista e tem a finalidade de garantir a consistência ao fim de cada operação. Já o seu oposto, BASE, adota uma postura otimista em que aceita que o BD atinja a consistência de

maneira eventual. Em adição, o estado suave e a consistência eventual são técnicas que funcionam bem quando há partições e assim aumentam a disponibilidade de acesso aos dados (BREWER, 2012).

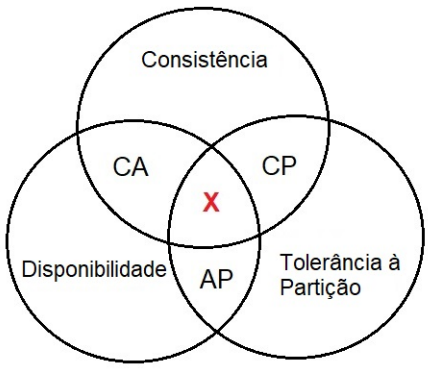
Por outro lado, o modelo BASE não aborda propriedades tais quais as descritas em ACID. O conceito BASE aponta que se deve planejar um sistema com o intuito de tolerar inconsistências momentâneas quando se deseja privilegiar a disponibilidade. Dessa forma, Fox et al. (1997) explicam que a semântica de dados do BASE é mais fraca que a encontrada em ACID e que tem como consequência uma compensação entre a troca da consistência transacional por disponibilidade.

2.2.4 Teorema CAP

O teorema Consistência, Disponibilidade e Tolerância à Partição (CAP) adota um tipo particular de coerência aos sistemas distribuídos, restringindo as características de transação nesses sistemas a três propriedades. São elas: consistência, disponibilidade e tolerância à partição. Em seu teorema, o entendimento pretendido por Brewer afirma que é impossível alcançar simultaneamente as três propriedades. Considerando tal fato, apenas duas dessas propriedades conseguem ser alcançadas ao mesmo tempo (FOX; BREWER, 1999).

Na figura 5, é possível verificar como as propriedades se relacionam no teorema CAP.

Figura 5 – Diagrama de Venn para o Teorema CAP



Fonte: Próprio Autor

2.2.5 Consistência

O entendimento sobre consistência de dados está atribuído ao estado interno de um sistema de armazenamento. Considerando um ambiente de dados distribuídos, a consis-

tência é alcançada quando todas as cópias de um dado escrito se encontram no mesmo estado. Dessa forma, se uma operação de leitura for realizada, qualquer réplica retornará o mesmo resultado (TANENBAUM; STEEN, 2007). Ainda conforme Tanenbaum e Steen (2007), quando uma operação de atualização é requisitada, alterando o estado de um dado, ela deve ser espalhada para todas as réplicas do *cluster*, o que pode comprometer o desempenho. Nesse sentido, o teorema CAP relata que é possível relaxar a forte consistência para obter uma garantia de disponibilidade para um sistema distribuído (FOX; BREWER, 1999).

De acordo com Vogels (2009), é possível calcular a consistência de um sistema a partir dos seguintes parâmetros: N = número de nós que possuem as cópias dos dados; W = número de nós que necessitam confirmar o recebimento do dado antes de uma atualização ser finalizada; R = número de réplicas que são contatadas no momento em que um dado é acessado por meio de uma operação de leitura. Caso o sistema mantenha o cálculo $W + R > N$, então a garantia de consistência será garantida. Considerando tal cálculo, Kleppmann (2017) explica que é esperado um valor atualizado, pois ao mínimo um dos R nós que estão sendo acessados contém o conteúdo atualizado.

2.2.6 Disponibilidade

Um sistema que possui disponibilidade tem a capacidade de se manter operacional e tolerante a falhas. Conforme Pritchett (2008), se o ambiente suporta falhas parciais sem o comprometimento total do sistema, então pode ser considerado como um sistema que alcança a disponibilidade. Dessa forma, Sadalage e Fowler (2013) explicam que, em um sistema distribuído, uma disponibilidade maior é alcançada com as técnicas de fragmentação e replicação dos dados. A partir dessas técnicas, é possível alcançar a tolerância de falhas caso algum nó do ambiente distribuído falhe. Com isso, o sistema continuará operando com alguma réplica que possua os recursos indisponíveis (STONEBRAKER, 2010). Já no caso de um sistema centralizado isso não é possível de acontecer, pois, caso o nó único do sistema falhe, o sistema fica totalmente inoperante.

2.2.7 Tolerância à Partição

Os sistemas distribuídos, por serem mais complexos, estão propensos a diferentes tipos de problemas: perda de mensagens, falhas de rede, quebra de *hardware*, etc. Quanto mais nós no sistema, aumenta-se a probabilidade de problemas.

Uma falha de rede pode incorrer em um particionamento do *cluster* dividindo, por exemplo, os nós em dois grupos e deixando esses grupos sem comunicação. De acordo com Stonebraker (2010), o objetivo de um sistema tolerante à partição é permitir que o processamento do ambiente continue em cada subgrupo. Apesar da perda de comunicação com os demais nós, os diferentes nós do *cluster* continuam em funcionamento.

2.2.8 Teorema PACELC

A construção do teorema se Partição, então Disponibilidade ou Consistência, senão Latência ou Consistência (PACELC) foi motivada a partir do teorema CAP. Diante da repercussão do teorema CAP com base em projetos dos SGBDs distribuídos, também conhecidos como SGBD NoSQL, o PACELC busca elucidar pontos limitantes encontrados no teorema CAP. O principal ponto a ser considerado e abordado é uma relação custo-benefício existente entre consistência e latência que não é questionada no teorema CAP. A construção do teorema PACELC apresenta maior influência dessa relação custo-benefício nos SGBDs distribuídos e argumenta que ela está presente mesmo quando não há partições de rede, separando-a do conceito encontrado no teorema CAP. Desta forma, o teorema PACELC é mais completo que o teorema CAP.

Conforme Abadi (2012), a consideração de que um SGBD distribuído que reduz a consistência quando não há partição faça isso por conta de uma deliberação embasada no teorema CAP é incorreta. Para este discernimento, é necessário compreender o conceito de latência, a qual é o período mínimo exigido para que uma resposta seja obtida a partir de uma requisição na rede (FOWLER, 2002). Para Abadi (2012), a latência é sem sombra de dúvidas a mesma coisa que a disponibilidade, considerando que uma rede que pode perder mensagens não possui diferença de uma que atrasa a entrega de mensagens indefinidamente. Portanto, L (*latency*) é, de certa forma, sinônimo de A (*availability*) pois um sistema indisponível incorre em uma latência exageradamente alta.

Uma condição para a elevação da disponibilidade é que o sistema faça a replicação de dados. Conforme um SGBD distribuído realiza a replicação, uma relação custo-benefício entre consistência e latência fica evidente. A partir dessa prerrogativa, três alternativas são possíveis para estabelecer a replicação dos dados: o sistema realiza a propagação para todas as réplicas ao mesmo tempo; primeiramente para um nó denominado mestre ou líder; ou primeiramente para um nó escolhido arbitrariamente (ABADI, 2012).

2.3 SGBD MongoDB

O MongoDB é um SGBD NoSQL cuja orientação é baseada em documentos. O armazenamento de dados ocorre em uma forma binária de JSON conhecida como Binary JSON (BSON) [3], o qual foi projetado para representar os dados JSON de forma mais compacta e eficiente, utilizando uma codificação melhor para o tratamento de números e outros tipos de dados (HARRISON; HARRISON, 2021).

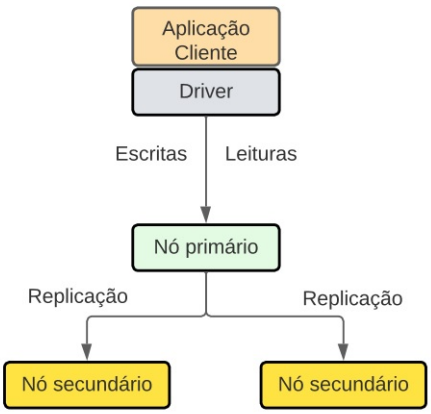
Sua constituição é formada por coleções e documentos. Dessa forma, Matallah, Belalem e Bouamrane (2017) explicam que o MongoDB possui coleções e um conjunto de documentos que pertencem a uma coleção, o que, de forma análoga, assemelha-se às tabelas de um BD relacional.

[3] https://bsonspec.org/

2.3.1 Processo de Replicação de Dados

Schultz, Avitabile e Cabral (2019) relatam que para atingir alta disponibilidade, o MongoDB permite acessar o BD como um grupo de nós que trabalham juntos, denominado *replica set*, atuando como um conjunto de consenso em que cada nó participante do *replica set* mantém uma réplica lógica do estado do BD. Esses *replica sets* utilizam um protocolo de consenso baseado em líder único, no qual um nó é utilizado como líder (*i.e.*, nó primário) e recebe gravações enquanto os nós seguidores (*i.e.*, nós secundários) ficam com cópias de *backup* (KLEPPMANN, 2017). As gravações vindas dos clientes a este nó primário são colocadas em um *log* de replicação chamado *oplog*. Este *oplog* é uma coleção que fica armazenada no BD local do nó primário e contém as informações referentes a como aplicar uma operação de banco de dados. Ou seja, as alterações que vão sendo realizadas na base de dados ficam registradas no *oplog*. Schultz, Avitabile e Cabral (2019) explicam que o *oplog* é utilizado pelos nós secundários para que a replicação dos dados ocorra, uma vez que os nós secundários executam as instruções contidas no *oplog* para manter a consistência com o nó primário. Sendo assim, a existência principal dos *replica sets* é para que o MongoDB forneça suporte à alta disponibilidade, permitindo que um *cluster* possua tolerância a falhas. Na figura 6 a arquitetura de um *replica set* é exibida.

Figura 6 – *Replica Set* no MongoDB



Fonte: Adaptado de (MONGODB, 2022b)

2.3.2 Níveis de Consistência do MongoDB

O MongoDB permite que os desenvolvedores realizem ajustes de consistência nas operações realizadas através de alguns parâmetros que são passados no momento de uma operação. Tais parâmetros podem ser combinados para que se reflita uma integridade

mais forte ou mais fraca no BD, tornando possível a configuração de alguns níveis de consistência.

Os níveis de consistência do MongoDB são encontrados nos parâmetros *write concern*, definido como *w*, e *read concern*. Para Harrison e Harrison (2021), o controle de como os dados são gravados e lidos nos nós do *cluster* são feitos por esses dois parâmetros. Além disso, parâmetros como *journal* e *read preference* também podem ser utilizados para realizar uma combinação do ajuste de consistência. Todos esses parâmetros podem ser definidos para cada operação do BD.

2.3.3 Write Concern e Journal

Para *write concern*, o valor definido pode ser numérico (*e.g.*, w: 1) ou *majority*. Conforme Harrison e Harrison (2021), o parâmetro *write concern* é responsável por controlar a quantidade de nós que devem confirmar a operação em um *cluster* antes de retornar a resposta ao usuário. Quando *write concern* é numérico, isso significa que um número determinado de nós do *cluster* deve reconhecer a gravação. Com relação ao valor *majority*, significa que a maioria dos nós deve reconhecer a gravação. Ou seja, se um *cluster* possui três nós, logo, dois deles devem confirmar a gravação antes de retornar ao usuário em uma configuração *w: majority*.

Dentro da configuração *write concern*, para que uma confirmação seja reconhecida, por padrão, o MongoDB possui uma configuração denominada *journal*, a qual é um arquivo de diário no disco do nó primário. Quando o *journal* está definido como *j: true* (*i.e.*, sua configuração padrão), é exigido que as gravações sejam persistidas no diário antes de serem concluídas. Caso esteja definido como *false*, a gravação é reconhecida apenas por chegar até o servidor, ou seja, sem realizar a escrita no diário. Nessa configuração, pode ocorrer a perda de dados enquanto a gravação ocorre no nó primário e uma falha aconteça. Dessa forma, a partir dessas definições, parte do ajuste da consistência é alcançado, pois configura o SGBD para garantir a escrita em um determinado número de nós no *cluster*.

2.3.4 Read Concern

De acordo com Matallah, Belalem e Bouamrane (2017), o parâmetro *readConcern* estabelece a durabilidade dos dados além de, em certas ocasiões, também determinar a consistência do que é retornado pelo servidor.

Conforme a documentação oficial do MongoDB (2022c) , para *read concern*, o valor definido pode ser algum dos seguintes: *local, available, majority, snapshot* e *linearizable*. Matallah, Belalem e Bouamrane (2017) explanam que a partir do momento em que a transação de uma operação é confirmada (*i.e.*, feito o *commit*), ela se torna uma operação confirmada localmente. Ou seja, confirmada apenas no nó local. A partir do momento em que a gravação consta no *oplog* e no BD, sua replicação pode ser executada para os nós

secundários. Dessa forma, quando atinge a condição necessária definda por *write concern* (*e.g.*, *w: majority*) e realiza o *commit* local na maioria dos nós, torna-se majoritariamente confirmada, significando uma durabilidade permanente no *replica set*.

Quando utilizado o *read concern available*, os dados retornados do servidor não possuem a garantia de que foram gravados na maioria dos nós do *replica set*. Já a opção *snapshot* consegue garantir que os clientes vejam um *snapshot* íntegro dos dados. Por fim, resta a configuração *linearizable*. Tal opção, quando combinada com *write concern w: majority*, garante devolver o efeito da escrita majoritária mais atualizada antes da operação de leitura inciar. Essa opção possui um efeito de linearização das operações, refletindo em uma garantia de consistência mais forte.

Quando comparado aos níveis de isolamento relatados por Berenson et al. (1995), Matallah, Belalem e Bouamrane (2017) relatam que os níveis *read concern local* e *majority* podem ser considerados análogos ao níveis de isolamento SQL *READ UNCOMMITTED* e *READ COMMITTED*. Isso acontence pelo fato de que o nível *read concern local* pode, em determinados casos, visualizar dados que ainda não foram confirmados pela maioria. Já uma operação que utiliza *read concern majority* é mais semelhante a uma operação em caminho de *commit* no isolamento SQL padrão.

2.3.5 Read Preference

A configuração do parâmetro *read preference* é responsável por indicar o local em que um cliente envia requisições de leitura. No MongoDB, tais requisições de leitura são enviadas ao nó primário por padrão (*i.e.*, *readPreference*: *primary*). Além dessa opção, é possível definir outras configurações para esse parâmetro, tais como direcionar as requisições de leitura apenas para um nó secundário, para um nó secundário apenas quando o nó primário estiver indisponível ou, por fim, para o servidor que estiver mais próximo. De acordo com Harrison e Harrison (2021), a configuração de *read preference* também influencia o SGBD a alterar a magnitude de consistência oferecida, considerando que quando as leituras são direcionadas ao nó primário isso consiste em uma consistência maior e, quando enviadas a um nó secundário, o resultado é uma consistência eventual.

A tabela 2 ilustra as configurações para *read preference*.

Tabela 2 – Níveis de *Read Preference* no MongoDB

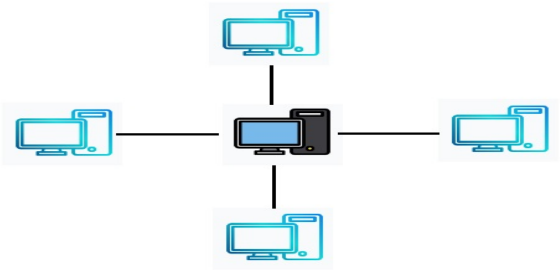
Read Preference	**Comportamento**
primary	Direciona todas as leituras ao nó primário.
primaryPrefered	Direciona, quando possível, as leituras ao nó primário.
secondary	Direciona todas as leituras ao nó secundário.
secondaryPrefered	Direciona, quando possível, as leituras ao nó secundário.
nearest	Direciona as leituras ao nó mais próximo.

Fonte: Adaptado de (MONGODB, 2022a)

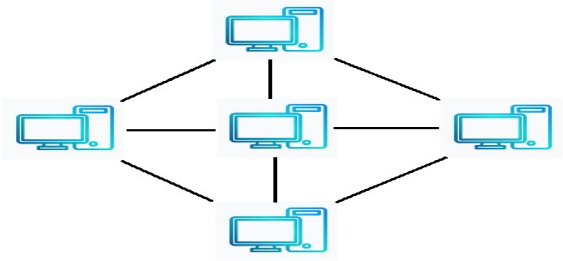
2.4 Sistemas Descentralizados

Ao contrário dos sistemas distribuídos centralizados em que Tanenbaum e Steen (2007) relatam que é um modelo em que existem clientes solicitando um serviço e apenas um servidor central atendendo essa demanda, um sistema descentralizado é um outro tipo de sistema distribuído o qual Raval (2016) explica que nenhum nó possui a atribuição de passar tarefas para outros nós constituintes da rede. Sendo assim, um sistema descentralizado não possui um único nó responsável em se comunicar com os demais na tratativa de gerenciamento de dados. Os nós trabalham na mesma rede, no entanto não possuem apenas um servidor de gerenciamento central. Na figura 7, é exibida a diferença na arquitetura entre um sistema distribuído centralizado e descentralizado.

Figura 7 – Diferenças entre sistema distribuído centralizado e descentralizado



(a) Sistema distribuído centralizado com um servidor de gerenciamento de dados central.



(b) Sistema distribuído descentralizado sem servidor de gerenciamento de dados central.

Fonte: Próprio Autor

Uma categoria nova de sistema descentralizado, denominada *blockchain*, que também baliza a fundamentação teórica deste trabalho é uma inovação tecnológica que foi apresentada pelo pseudônimo Satoshi Nakamoto [4], em 2008, por meio da criação da moeda digital *Bitcoin*. Na sequência essa tecnologia é abordada com mais detalhes.

[4] Nakamoto, Satoshi. Bitcoin: A peer-to-peer electronic cash system.

2.4.1 Tecnologia Blockchain

De acordo com Singhal, Dhameja e Panda (2018), a *blockchain* é um exemplo de sistema distribuído descentralizado. Dessa forma, a criptomoeda *Bitcoin* abriu caminho no uso da *blockchain* por intermédio do uso de uma rede *peer-to-peer*, ou seja, um ambiente descentralizado. Sendo assim, por meio da *blockchain*, o *Bitcoin* conseguiu atender sua ideia principal, a qual é a realização digital de transações monetárias de uma forma que envolva confiança e segurança no processo (ULRICH, 2014). Com isso, sabe-se que a *blockchain* tem sua execução baseada na confiabilidade e colaboração dos participantes ao executar transações de forma transparente, já que podem sofrer auditoria dos próprios nós da rede.

Laurence (2017) aborda a *blockchain* como sendo constituída por três principais pontos:

- Bloco (*Block*)

  Local em que uma lista de transações fica registrada. Ou seja, no caso do *Bitcoin*, dados como remetente, destinatário e valor transferido são partes de uma transação.

- Corrente (*Chain*)

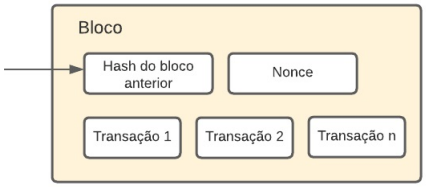
  Uma estrutura criptográfica, denominada *hash*, a qual realiza a interligação dos blocos. Essa função *hash* tem o objetivo de criar um algoritmo complexo que envolve dados de tamanhos distintos em uma *string* que contempla um tamanho fixo.

- Rede

  Local onde estão os nós que compõem a rede da *blockchain*. São os dispositivos que realizam as validações dos blocos por meio de uma camada de consenso e que estão espalhados por todo o mundo.

Na figura 8, é exibida a composição de um bloco da *blockchain*.

Figura 8 – Estrutura de um bloco da *blockchain*



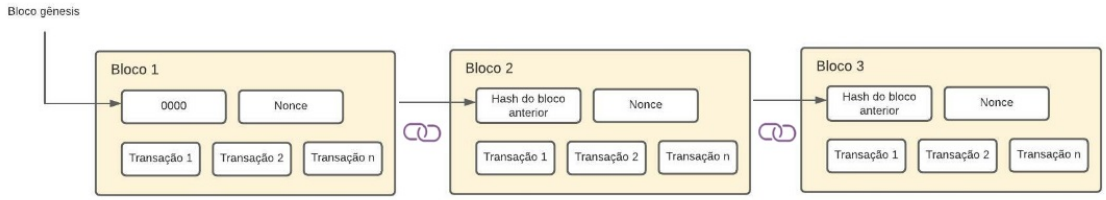
Fonte: Adaptado de (NAKAMOTO, 2008)

Conforme exibido na figura 8, a lista de transações é o conjunto dos dados contidos no bloco e cada bloco possui um valor *hash*, *hash* esse que é construído a partir dos dados do

bloco atual, do *hash* proveniente do bloco anterior e de um *nonce*, o qual Singhal, Dhameja e Panda (2018) relatam que é um valor aleatório que pode ser utilizado apenas uma vez. Portanto, cada bloco possui o *hash* do seu antecessor e o valor zero é correspondente ao *hash* do bloco inicial (*i.e.*, bloco gênesis). Singhal, Dhameja e Panda (2018) explicam que o bloco gênesis é o primeiro bloco da *blockchain* (*i.e.* bloco 0) e pode ser considerado como um bloco especial, já que não possui referência de nenhum bloco anterior.

O funcionamento e estrutura da *blockchain* é mostrado na figura 9.

Figura 9 – Funcionamento da *blockchain*



Fonte: Adaptado de (NAKAMOTO, 2008)

As transações são registradas em um livro razão (*i.e. blockahin*). Estrutura essa que possui características como descentralização, não adulteração, rastreabilidade e consistência (GAI et al., 2020). Para atingir tal objetivo, todos os nós da rede precisam chegar a um consenso sobre quais blocos da cadeia de blocos são válidos ou não. No caso de um bloco ser adulterado ou falsificado, tal bloco é rejeitado pelos outros participantes da rede. Dessa forma, cada nó possui o registro total da rede e precisa entrar em consenso com os demais nós a respeito dos dados que vão sendo adicionados (ANTONOPOULOS, 2017).

Conforme Gai et al. (2020), os dados podem ser armazenados diretamente nos nós e, dessa maneira, a durabilidade da transação é atingida por meio da participação dos nós da rede e pelo mecanismo de consenso, o qual garante a veracidade dos dados baseado na verificação. Partindo dessa premissa, a camada de consenso tem como objetivo principal fazer com que todos os nós participantes da rede cheguem a um acordo a respeito da consistência do livro razão (SINGHAL; DHAMEJA; PANDA, 2018). Elrom (2019) explica que esse acordo é necessário pois, a partir do momento em que se lida com pares não confiáveis em uma rede, estabelecer regras que garantam a segurança e integridade do livro razão contra fraudes é essencial.

Alguns dos principais mecanismos de consenso atuais relatados por Singhal, Dhameja e Panda (2018) são: Prova de Trabalho (PoW), Prova de Participação (PoS) e Tolerância Prática a Falhas Bizantinas (PBFT). Na sequência, seus conceitos são abordados:

- ❏ PoW

É o padrão adotado por diversas plataformas de criptomoedas. Tal mecanismo tem a característica de evitar o gasto duplo, o qual significa que um usuário da rede com más intenções realiza duas transações com os mesmos fundos e não assume ter feito a primeira (RAVAL, 2016). Para evitar tal ação, Elrom (2019) relata que o PoW é um problema matemático complexo em que um nó minerador precisa resolver. Ao realizar as validações necessárias e encontrar a solução, esse nó torna-se apto a criar o próximo bloco da cadeia e adicioná-lo à *blockchain*.

- ❑ PoS

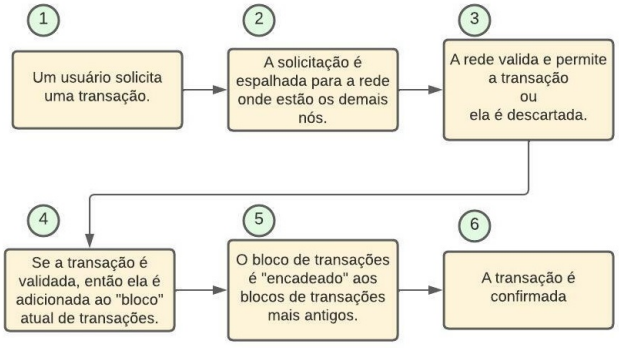
  Neste mecanismo, o poder computacional do PoW para a geração do novo bloco é substituído pelo poder de participação de um nó na rede. Muzammal, Qu e Nasrulin (2019) explicam que esse nó é recompensado baseado na sua participação no sistema (*i.e.*, possuir maiores quantidades de criptomoedas). Isso significa que a probabilidade de um nó minerador gerar um novo bloco coincide com o tamanho de sua participação, ou seja, quanto maior a quantidade de criptomoedas que um nó possui, maior é a chance dele anexar o próximo bloco à *blockchain*. Para Raval (2016), isso acontence para fornecer mellhor proteção a ataques maliciosos, uma vez que quanto mais criptomoedas um nó possuir, maior será o investimento para garantir a estabilidade da rede.

- ❑ PBFT

  Nesse mecanismo de consenso, todos os nós possuem estados internos próprios e, conforme recebem requisições, suas peculiaridades são compartilhadas com os demais integrantes para se alcançar um consenso e espalhar esse resultado para todos os nós da rede (SINGHAL; DHAMEJA; PANDA, 2018).

Na Figura 10, é ilustrado como uma *blockchain* chega a um consenso.

Figura 10 – Criação de consenso na *blockchain*



Fonte: Adaptado de (LAURENCE, 2017)

2.5 Análise de Desempenho

Soluções robustas de *hardware* e *software* agregadas de um elevado poder computacional foram disponibilizadas por meio da evolução computacional e tecnológica advindas de toda a demanda crescente e complexa da digitalização de serviços. Considerando tais acontecimentos, o estudo sobre os aspectos e funcionamento de um computador foram impactados, o que, de certa forma, tornou inevitável a dispensa de recursos que conseguissem avaliar e comparar tais soluções. O motivo para isso foi o aumento de produtos computacionais lançados por empresas diferentes que possuem funcionalidades similares.

Para tal propósito, um dos artifícios mais utilizados é denominado como análise de desempenho. Empregada em diversas áreas científicas, tem o objetivo de comparar o desempenho das funcionalidades estipuladas pelos produtos analisados. Esse desempenho pode ser exposto por meio de medidas escalares (*e.g. tempo e distância*). Dessa forma, produtos semelhantes podem ser facilmente aferidos utilizando operadores simples, tais como maior, menor e igual (CIFERRI, 1995).

Por dentro das subáreas da ciência da computação as quais usam a técnica de análise de desempenho, é possível salientar: teoria de computação, arquitetura de computadores, sistemas operacionais e banco de dados.

Nessa conjuntura, essa técnica auxilia na resolução de tópicos como:

- ❏ Analisar a capacidade total de um produto;
- ❏ Analisar uma funcionalidade específica de um produto;
- ❏ Analisar a relação custo x benefício de um produto; e
- ❏ Realizar a comparação de tecnologias distintas;

Segundo Ciferri (1995), para aplicar a análise de desempenho, é utilizado um dos seguintes modelos:

- ❏ Modelo Analítico:

  Possui seu embasamento relacionado à obtenção de um grupo de equações matemáticas e algoritmos, os quais têm medidas de desempenho relacionadas a parâmetros do sistema que está sendo avaliado.

- ❏ Modelo de Simulação:

  No mundo real, é considerada a entrada e saída de dados como parte. Tal modelo deve possuir a capacidade de realizar a geração de dados de saída por meio de cálculos ou inferências sobre os dados de entrada.

- Modelo Experimental:

  É utilizado o sistema em avaliação para a obtenção dos resultado de desempenho. Em consequência disso, tem-se resultados mais confiáveis advindos da análise de desempenho. Encontram-se duas técnicas pertencentes a esse modelo: monitoração e *benchmark*. Na técnica de monitoração, recursos de análise estatística em disponibilidade nos produtos avaliados são utilizados. Em contrapartida, a técnica de *benchmark* é operada executando um conjunto padrão e fixo de testes, os quais podem ser aplicados na comparação de sistemas distintos, considerando que tais testes não sofrem variação e possuem uma definição clara.

2.5.1 Técnica de *Benchmark*

A técnica de *benchmark* é amplamente usada para a análise de desempenho de sistemas computacionais. Ela mede de maneira padronizada as principais funções dos sistemas escolhidos para análise.

De acordo com Bog (2013), quatro requisitos devem ser cumpridos por um *benchmark*:

- Relevância:

  Compreendida como a medição de desempenho de sistemas dentro de um domínio característico em que as métricas tem o objetivo de auxiliar na comparação dos sistemas analisados;

- Portabilidade:

  Compreende a facilidade de implementação do *benchmark* em diversos e diferentes sistemas e arquiteturas;

- Escalabilidade:

  Considerando distintos volumes de dados, a escalabilidade compreende a capacidade do *benchmark* realizar a medição do comportamento do sistema. Ou seja, é a capacidade de medir como o sistema se comporta em termo de desempenho em função de uma carga baixa, média, ou alta de dados e consultas.

- Simplicidade:

  Os resultados provenientes do *benchmark* devem ser facilmente interpretados.

Ainda conforme Bog (2013), alguns outros aspectos também são valiosos para um *benchmark*, tais como repetibilidade, justiça, verificabilidade e economia. Para o requisito de repetibilidade, o *benchmark* necessita gerar sempre o mesmo tipo de resultado em cada vez que realizar sua execução sobre a mesma base de dados. Para justiça, os sistemas que sofrerem comparação necessitam ter a capacidade de participar de forma igual dos experimentos. A verificabilidade consiste no resultado final do *benchmark* ser interpretado

de maneira real. Por fim, o requisito de economia compreende a acessibilidade financeira para a sua execução.

Para atingir os objetivos relativos a esta pesquisa de Mestrado, foram estudados o *benchmark* YCSB e seu derivado YCSB+T. Esse último *benchmark* será usado para analisar a garantia de consistência da proposta do MongoChain e dos diferentes níveis de consistência do SGBD NoSQL MongoDB.

2.5.2 Yahoo! Cloud Serving Benchmark

O Yahoo! Cloud Serving Benchmark (YCSB) é um *software* cliente desenvolvido na linguagem de programação Java que é utilizado para gerar os dados a serem carregados em um determinado BD escolhido pelo usuário e gerar as operações que compõem uma determinada carga de trabalho escolhida pelo usuário. Além disso, um de seus recursos principais é a possibilidade de ser extensível, oferecendo suporte à definição de novas cargas de trabalho com facilidade. Foi criado com a finalidade de realizar comparações no desempenho da nova era de SGBDs focados em trabalhar com serviços em nuvem. Muitos desses sistemas são denominados como de armazenamento de chave-valor ou reconhecidos também como SGBDs NoSQL. Esses sistemas têm objetivos em comum, tais como ter uma adaptabilidade acentuada de dados sob demanda, também denominada elasticidade e escalabilidade horizontal.

É complexo de se avaliar sistemas diferentes quando se busca compreender o desempenho de vários sistemas, pois alguns sistemas buscam otimizar as escritas enquanto outros buscam a otimização das leituras. Isso se torna um desafio, pois os desenvolvedores acabam por relatar desempenhos específicos de acordo com cargas de trabalho particulares para cada um de seus sistemas. Cooper et al. (2010) explicam que existe dificuldade em encontrar uma comparação exata quando se trata de números diferentes baseados em cargas de trabalho diferentes e que, dessa forma, os desenvolvedores têm a necessidade de realizar manualmente a comparação entre sistemas diferentes. Considerando essa prerrogativa, o *benchmark* YCSB foi elaborado com a ideia de se ter uma ferramenta de *benchmark* padrão para auxiliar na avaliação de diferentes sistemas idealizados para trabalhar em nuvem.

A arquitetura é composta por uma camada executora da carga de trabalho, *threads* do cliente, uma camada de estatísticas e uma camada que representa a interface de BD. A operação básica é que a camada executora da carga de trabalho executa uma série sequencial de operações (*read, write, update, scan*) fazendo chamadas para a camada de interface do BD, tanto para carregar o BD (*i.e.*, fase de carregamento) quanto para executar a carga de trabalho (*i.e.*, fase de transação). Além disso, também realiza a medição da latência e rendimento alcançados de suas operações, relatando tais dados para o módulo de estatísticas. Ao fim do experimento, o módulo de estatísticas agrupa

tais medições e disponibiliza algumas informações como média, latências do 95° e 99° percentil.

Para configurar o funcionamento, o cliente YCSB recolhe uma série de propriedades definidas por chave-valor e essas propriedades estão definidas em dois grupos:

- Propriedades da carga de trabalho

  São propriedades que definem a carga de trabalho independente do BD utilizado. Um exemplo são as proporções definidas de leituras e gravações a serem realizadas no BD.

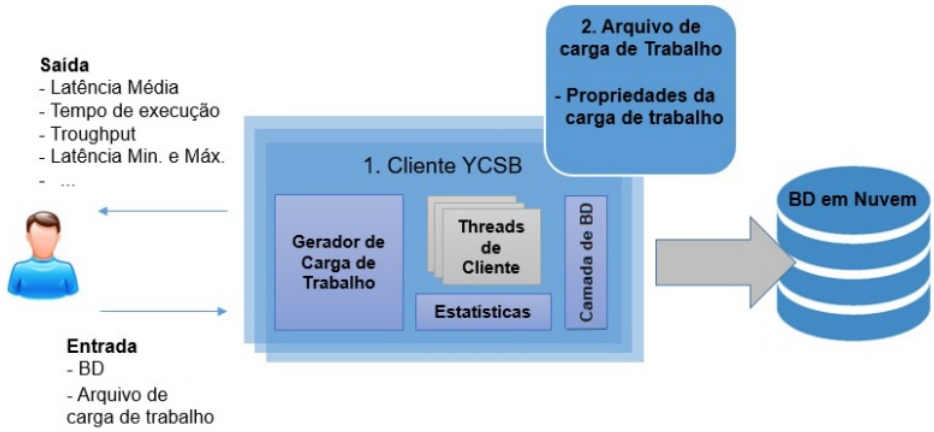
- Propriedades de tempo de execução

  São as propriedades específicas para um determinado experimento. Um exemplo disso é a camada de interface de BD que é passada como parâmetro (e. g. MongoDB) e também as propriedades utilizadas para inicializar essa camada.

Sendo assim, é possível haver arquivos de propriedades de carga de trabalho que permanecem estáticos e servem para avaliar diversos BDs. Em contrapartida, as propriedades de tempo de execução também podem ficar nos arquivos de propriedades, no entanto vão variar de acordo com o experimento e BD.

Na figura 11, a arquitetura do cliente YCSB é exibida.

Figura 11 – Arquitetura do cliente YCSB



Fonte: Adaptado de (COOPER et al., 2010)

O YCSB é composto por duas camadas de *benchmark* que visam avaliar o desempenho e a escalabilidade dos sistemas de serviço em nuvem. Uma das camadas é a camada de desempenho, a qual se concentra na latência das requisições enquanto o BD está sob carga. De acordo com Cooper et al. (2010), a latência é uma métrica importante a ser avaliada, considerando que normalmente se tem um usuário impaciente esperando o

carregamento de uma página *Web*, por exemplo. Essa camada tem o objetivo de verificar a compensação existente no SGBD medindo a latência conforme o rendimento é aumentado até o momento em que o SGBD fica saturado e o desempenho para de aumentar. Ou seja, o *hardware* é mantido constante enquanto o tamanho da carga de trabalho aumenta.

Para conduzir a camada de desempenho do *benchmark*, é necessário possuir o gerador de carga de trabalho, exibido na figura 11, o qual atende aos propósitos de definição e carregamento do conjunto de dados e também de execução de operações em cima desses conjuntos de dados enquanto realiza a medição de desempenho que é trabalhada pela camada de estatísticas.

A outra camada é a camada de escalabilidade, a qual tem a intenção de verificar o aspecto de escalabilidade do SGBD. Dessa forma, essa métrica é utilizada para verificar o comportamento da aplicação conforme a carga de dados aumenta. A camada de escalabilidade verifica o impacto no desempenho de acordo com o aumento do número de máquinas no sistema. Ou seja, permite saber como o BD se comporta conforme o número de máquinas aumenta. Para essa verificação, um determinado número de servidores é carregado com dados e então a carga de trabalho é executada. Na sequência, a mesma tarefa é realizada com a adição de mais máquinas no *cluster*. Primeiramente os dados são excluídos e então mais máquinas são adicionadas para refazer os testes. Por fim, a carga de trabalho é executada novamente e os resultados são coletados. Caso o SGBD tenha boas propriedades de escalabilidade, o desempenho (e. g. latência) deve permanecer constante conforme o número de servidores, quantidade de dados e o rendimento oferecido escalam proporcionalmente.

O *benchmark* YCSB, por padrão, utiliza algumas cargas de trabalho para avaliar aspectos de desempenho diferentes de um sistema, chamado YCSB *Core Package*. Conforme Cooper et al. (2010), cada carga de trabalho configura uma combinação específica de operações de leitura e escrita, distribuição de requisições e volume de dados. Características que são utilizadas para avaliar sistemas em um ponto específico no espaço de desempenho. Além disso, a aplicação gera uma tabela de registros, cada uma com vários campos, onde o registro é identificado por uma chave primária definida como algo aproximado da *string "user234213"*.

Na figura 12 é exibida a saída de *console* ao ser realizada a fase de carregamento do cliente YCSB utilizando o SGBD MongoDB que está em um *cluster* na nuvem contendo 3 máquinas, sendo uma delas primária e duas secundárias. Além disso, está sendo utilizada uma carga de trabalho padrão do *benchmark* que realiza mil operações de *insert*. No caso, foram feitas mil operações de *insert* com sucesso em um tempo de execução de 167.787 milissegundos. Além da latência mínima, média e máxima, latência do nonagésimo quinto e nonagésimo nono percentil, também é relatado o *throughput*, totalizando aproximadamente 5,96 operações por segundo.

Cada operação no armazenamento de dados é escolhida aleatoriamente entre *insert*,

Figura 12 – *Output* da fase de carregamento do YCSB

```
[OVERALL], RunTime(ms), 167787
[OVERALL], Throughput(ops/sec), 5.959937301459589
[TOTAL_GCS_PS_Scavenge], Count, 1
[TOTAL_GC_TIME_PS_Scavenge], Time(ms), 5
[TOTAL_GC_TIME_%_PS_Scavenge], Time(%), 0.002979968650729794
[TOTAL_GCS_PS_MarkSweep], Count, 0
[TOTAL_GC_TIME_PS_MarkSweep], Time(ms), 0
[TOTAL_GC_TIME_%_PS_MarkSweep], Time(%), 0.0
[TOTAL_GCs], Count, 1
[TOTAL_GC_TIME], Time(ms), 5
[TOTAL_GC_TIME_%], Time(%), 0.002979968650729794
[CLEANUP], Operations, 1
[CLEANUP], AverageLatency(us), 151360.0
[CLEANUP], MinLatency(us), 151296
[CLEANUP], MaxLatency(us), 151423
[CLEANUP], 95thPercentileLatency(us), 151423
[CLEANUP], 99thPercentileLatency(us), 151423
[INSERT], Operations, 1000
[INSERT], AverageLatency(us), 166551.36
[INSERT], MinLatency(us), 145152
[INSERT], MaxLatency(us), 2942975
[INSERT], 95thPercentileLatency(us), 220927
[INSERT], 99thPercentileLatency(us), 267007
[INSERT], Return=OK, 1000
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 02:48 min
Finished at: 2022-04-26T21:09:45-03:00
Final Memory: 9M/243M
------------------------------------------------------------------------
```

Fonte: Próprio Autor

update, *read* e *scan*, onde a opção de *insert* insere um novo registro, *update* atualiza um registro alterando o valor de um campo, *read* realiza a leitura de um registro e *scan* verifica os registros em ordem, começando de uma chave de registro escolhida aleatoriamente. O número de registros a serem verificados também é escolhido aleatoriamente.

Para gerar a carga de trabalho, o cliente YCSB realiza escolhas aleatórias sobre qual operação executar, qual registro gravar ou ler e a quantidade de registros a serem verificados, por exemplo. Sendo assim, o *software* utiliza alguns tipos de distribuições de dados que são: *uniform*; *zipfian*; *latest*; e *multinomial*. A distribuição *uniform* representa uma mesma probabilidade dos registros de um BD serem escolhidos. Já a distribuição *zipfian*, baseada na lei de Zipf, adota uma distribuição mais específica, a qual busca valorizar um registro popular em todo universo dos dados, do mais velho ao mais novo. Essa distribuição simula uma certa imprevisibilidade no acesso a dados mais velhos. Um registro mais popular terá duas vezes mais acessos do que um registro com menos popularidade e assim por diante. A distribuição *latest* se refere a uma consulta que será realizada nos dados mais recentes. É como a distribuição *zipfian*, porém os registros inseridos por último são os mais populares. Por fim, a distribuição multinomial permite que se faça a verificação de vários eventos dentro de um mesmo espaço amostral. É possível especificar

as probabilidades para cada item.

Sobre sua extensibilidade, o YCSB permite que as camadas geradora de carga de trabalho e de interface de BD sejam sobrescritas. Dessa forma, Cooper et al. (2010) explicam que é possível para os usuários definirem novos pacotes de duas formas: uma delas é definindo diferentes parâmetros para serem usados no executor principal de carga de trabalho, denominado *"CoreWorkload"*, e a segunda abordagem consiste de se codificar uma nova classe executora de carga de trabalho com seus parâmetros atrelados.

2.5.3 Yahoo! Cloud Serving Benchmark com Camada Transacional

Frameworks de *benchmark* de serviços em nuvem tais como YCSB, são projetados para avaliar o desempenho de SGBDs NoSQL distribuídos, os quais em um primeiro momento não permitiam transações. Sendo assim, os *benchmarks* utilizam operações que acabam não constando dentro do escopo das transações. Entretanto, implementações mais recentes de sistemas NoSQL distribuídos começaram a oferecer recursos de transação para atender a novas demandas em escala *Web*. Dessa forma, Dey et al. (2014) relatam que esse fato abriu uma nova oportunidade de implementação desses tipos de *benchmarks*.

Os problemas referentes ao suporte a transações em SGBDs NoSQL foram identificados e assim uma nova extensão do *benchmark* YCSB surgiu. A partir disso, o Yahoo! Cloud Serving Benchmark com Camada Transacional (YCSB+T) foi criado, o qual envolve operações de BD realizadas dentro do escopo de transações (DEY et al., 2014). Nessa extensão, foi incluído um estágio de validação, o qual tem a finalidade de detectar anomalias do BD que resultam da carga de trabalho executada.

Dey et al. (2014) explicam que o desempenho bruto e a escalabilidade são o foco principal do *benchmark* YCSB, mas que a correção das transações não é validada como parte do *benchmark* e, além disso, as operações não se delimitam nas transações, considerando que alguns SGBDs NoSQL podem não possuir o suporte transacional e nem garantir a consistência dos dados. Diante desse cenário, a proposta do YCSB+T busca continuar com a flexibilidade existente no YCSB permitindo que o usuário possa escolher a interface de BD a ser implementada. Ademais, permite que suas operações sejam agrupadas no escopo de transação. Também há um estágio de validação que é utilizado para especificar as avaliações de consistência de dados conduzidas no BD depois da conclusão de uma carga de trabalho. Esse estágio de validação fica responsável pela detecção de anomalias de transação.

O YCSB+T possui duas novas camadas que têm como objetivo avaliar a sobrecarga transacional relacionada ao *throughput* do armazenamento de dados e a consistência transacional do BD a partir do resultado da execução da carga de trabalho do *benchmark*. As duas novas camadas são denominadas camada de sobrecarga transacional e camada de

consistência.

A camada de sobrecarga transacional busca representar o aspecto de sobrecarga das transações e verificar o efeito das operações que são executadas dentro e fora do ambiente transacional no BD. O foco principal é em realizar a medição dessa sobrecarga no contexto transacional. Apesar do foco no contexto transacional, esta camada realiza as medidas de latência no contexto não transacional também. Dessa forma, a latência de operações de varredura do BD e das operações de *Create*, *Read*, *Update* e *Delete* (CRUD) são medidas. Além disso, são medidas as latências para os métodos *start()*, *abort()* e *commit()* nos dois modos. A medição dessas operações transacionais fica sob a responsabilidade da camada que gera a carga de trabalho. O que torna possível medir esta sobrecarga transacional é justamente esta latência das operações de varredura e de CRUD em companhia da latência das operações de *start()*, *abort()* e *commit()* recuperadas (DEY et al., 2014).

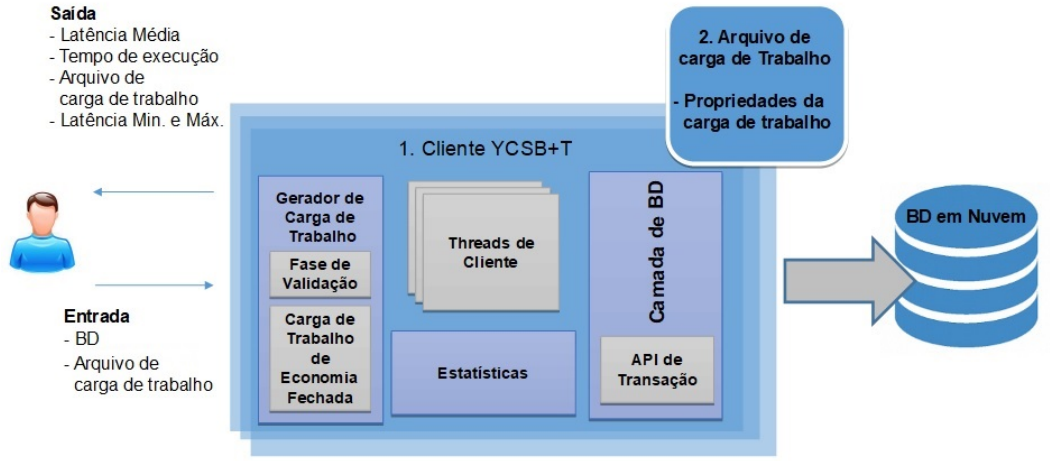
A camada de consistência é utilizada para verificar a consistência do BD após a execução de uma carga de trabalho correspondente à fase de transação do *benchmark*. Ou seja, é necessário que a fase de carregamento já tenha sido realizada anteriormente. A ideia do contexto transacional para executar as operações no BD existe para que a garantia das propriedades ACID e a consistência sejam preservadas. Considerando isso, esta camada foi projetada para identificar anomalias de consistência nos dados enquanto a carga de trabalho é executada. Para atingir esse objetivo, um passo de validação do BD foi implementado na camada de geração de carga de trabalho. Tal validação realiza uma varredura em todos os registros da base de dados e faz uma verificação no conteúdo do BD.

Na arquitetura YCSB+T, uma nova carga de trabalho denominada carga de trabalho de economia fechada foi implementada e possui um método adicional, chamado *validate()*, o qual é chamado dentro da classe de carga de trabalho após uma carga de trabalho ser executada no BD na fase de transação para que a consistência seja validada. Esse comportamento é referente ao estágio de validação abordado anteriormente. Na figura 13, a arquitetura do YCSB+T é exibida.

É possível verificar que novas cargas de trabalho, neste caso a carga de trabalho de economia fechada, ficam inseridas dentro da camada geradora de carga de trabalho. Dessa forma, este *framework* que é uma extensão do YCSB inclui a nova carga de trabalho possuindo o método *validate()* que é um método sobrecarregado da classe abstrata de carga de trabalho, sendo implementado pela classe derivada. Por fim, tem-se o estágio de validação acompanhando a nova carga de trabalho.

A carga de trabalho de economia fechada tem a ideia de simular uma economia onde a quantia total de dinheiro não entra ou sai do sistema enquanto a avaliação é realizada. É um número de contas determinado de forma prévia e um montante total de dinheiro que é distribuído de maneira igualitária entre as contas. A fase de carregamento desta carga de trabalho gera chaves até um número que é determinado em um dos parâmetros da carga de trabalho. Nesse sentido, cada chave simboliza o número de uma conta e o montante

Figura 13 – Arquitetura do cliente YCSB+T



Fonte: Adaptado de (DEY et al., 2014)

total é definido por outro parâmetro passado à carga de trabalho. Para atribuir o saldo inicial de cada conta é realizado um cálculo simples, onde o montante total é dividido pelo número total de contas e, dessa forma, é gerado um saldo inicial igualitário. Já a fase de transação retém a responsabilidade de escolher uma das operações de BD e chamá-la de acordo com a combinação de operações escolhidas. Tais operações são as seguintes e possuem as seguintes tarefas:

- *doTransactionInsert()*

  Faz a criação de uma conta nova incluindo um saldo inicial que é referente ao saldo capturado da operação.

- *doTransactionDelete()*

  Faz a leitura de uma conta, adiciona o valor do saldo que foi lido à conta que foi criada utilizando *doTransactionInsert()* e, na sequência, apaga o registro de conta lido previamente.

- *doTransactionRead()*

  Faz a leitura de um grupo de saldos de contas especificados pelo gerador de chaves do YCSB+T.

- *doTransactionScan()*

  Faz uma varredura no BD e realiza a busca dos registros utilizando uma chave de início e o número de registros.

- *doTransactionUpdate()*

Faz a leitura de um registro e atualiza seu saldo somando 1. O valor utilizado na atualização é advindo das operações de exclusão.

- *doTransactionReadModifyWrite()*

  Inicia realizando a leitura de dois registros. Na sequência subtrai 1 de uma conta e adiciona à outra. Por fim, faz a gravação dos dois registros.

Por fim, na fase de validação, cada transação que ocorre não deve alterar a soma total do saldo de todas as contas. Caso isso ocorra, significa que houve inconsistência durante a realização das operações. Esse estado errôneo do BD é exibido a partir da diferença existente entre a soma total das contas no início da execução da carga de trabalho e a soma total ao fim do procedimento.

2.6 Considerações Finais

Este capítulo abordou os principais conceitos para compreender as tecnologias estudadas para a construção desta pesquisa de Mestrado. Explicou a importância do ambiente transacional para a execução de transações de BD, as propriedades ACID, o contraste dessas propriedades em relação ao modelo BASE, o qual se encontram os SGBDs NoSQL. Em adição, por meio dos teoremas CAP e PACELC, explanou as limitações e compensações existentes na utilização dessas tecnologias. O SGBD MongoDB também foi esclarecido por meio de seu modelo de replicação e de seus níveis de ajuste de consistência. Por fim, esclareceu a teoria da tecnologia *blockchain*, a qual ainda se encontra em processo de amadurecimento tecnológico e está embutida no *framework* Mongochain, e abordou os conceitos referentes à análise de desempenho, a qual possui a técnica de *benchmark* que é utilizada nesta pesquisa com as explicações a respeito do *benchmark* YCSB e seu derivado YCSB+T.

No capítulo seguinte será abordado o Mongochain, o qual é um dos trabalhos que serão investigados para a realização da coleta de dados para análise e verificação de sua corretude no quesito consistência de dados.

Capítulo 3

Framework Mongochain

Conforme explicado anteriormente nas Seções 2.1.1 (Propriedades ACID) e 2.2.3 (Conceito BASE), os modelos ACID e BASE possuem ideias contrárias em relação ao gerenciamento de transações. Considerando tal fato, Sousa (2020) explica que existe um campo aberto de pesquisa para se construir ferramentas híbridas que atendam diferentes domínios contemplando concomitantemente os benefícios desses dois paradigmas para o gerenciamento de transações além de oferecer forte segurança e transparência no processo. Dessa forma, sua proposta de pesquisa busca preencher essa lacuna por meio do desafio da criação de um *framework* (*i.e.*, Mongochain) que forneça os mecanismos necessários para providenciar dados consistentes, escaláveis, disponíveis, seguros e transparentes em um ambiente distribuído composto por um *cluster* de computadores. Para isso, foi necessário o uso de algumas tecnologias que são abordadas na sequência.

Pelo fato do Mongochain ser um *framework*, é interessante compreender que essa abordagem é uma estrutura ou um conjunto de ferramentas, bibliotecas e convenções que fornecem uma base para o desenvolvimento de *softwares*. A partir disso, o processo de criação de aplicativos é facilitado ao oferecer uma arquitetura organizada e pré-definida, permitindo que os desenvolvedores se concentrem mais na lógica específica da aplicação do que em questões mais gerais e repetitivas que normalmente aparecem na implementação de *softwares*. O fato de ser reutilizável permite que os desenvolvedores possam aproveitar soluções existentes para problemas comuns, economizando tempo e esforço no desenvolvimento de aplicativos. Em sua pesquisa, Sousa (2020) aborda a capacidade de extensão e reutilização de outras duas soluções criadas derivadas do Mongochain *(i.e., MongoChainScheduleHealth* que é utilizado para agendamentos de consultas médicas e *MongoChainMarketPlace* que é utilizado para o comércio de produtos automotivos).

Sendo assim, neste capítulo é abordado de forma mais clara o *framework* Mongochain.

São esclarecidas quais as tecnologias foram utilizadas para a construção do *software*, quais as suas funcionalidades e como o sistema se comporta para realizar o gerenciamento de transações.

3.1 Tecnologias Utilizadas

O Mongochain é um *framework* programável que propõe estender as funcionalidades encontradas no SGBD MongoDB com a adição de uma *blockchain* permissionada. Ou seja, os nós que participam dessa rede são selecionados pelo desenvolvedor. Portanto, tem-se a integração dos modelos ACID e BASE juntamente com os recursos provenientes de uma *blockchain* permissionada (SOUSA, 2020).

Para a criação do Mongochain, as seguintes tecnologias foram adotadas:

- ❏ Linguagem de programação *JavaScript* para a escrita do código;
- ❏ Arquitetura Modelo, Visão e Controle (MVC) para modulação e implementação de regras de negócio;
- ❏ API para o atendimento de requisições externas;
- ❏ SGBD NoSQL orientado a documentos MongoDB;
- ❏ MongoDB Atlas[1], o qual é o serviço em nuvem do MongoDB; e
- ❏ Uma *blockchain* permissionada;

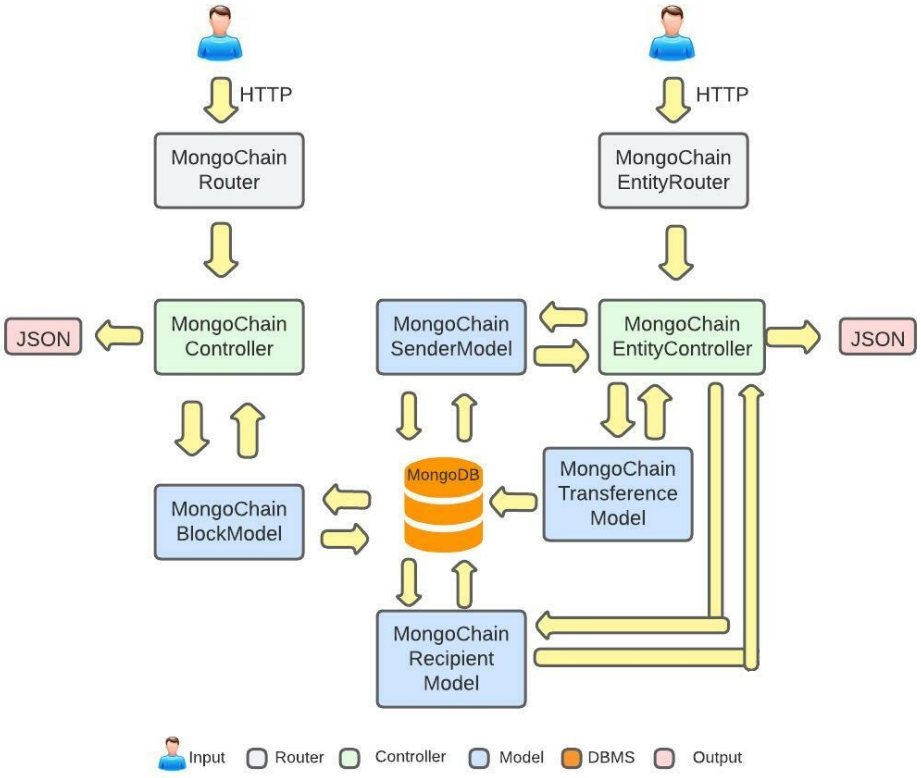
Com a junção deste conjunto de tecnologias, a construção do Mongochain se tornou possível. Na sequência, o comportamento e as funcionalidades encontradas no *framework* são abordados.

3.2 Comportamento e Funcionalidades

Com a extensão do SGBD MongoDB incluindo uma *blockchain* permissionada, a ideia do Mongochain é que ele seja um *framework* em que seus usuários tenham um aumento de produtividade por consequência de poder reutilizar partes centrais do sistema e conseguir estender suas funcionalidades para atender a uma demanda ou regra de negócio específica (SOUSA, 2020).

Sua construção é baseada no modelo arquitetural MVC, o qual permite uma modulação eficaz do sistema, fazendo com que a codificação de suas funcionalidades consiga fluir de forma compreensível para outros desenvolvedores. Para compreender sua estrutura, a

Figura 14 – Arquitetura presente no Mongochain



Fonte: Adaptado de (SOUSA, 2020)

figura 14 mostra a arquitetura do Mongochain e, a partir disso, é possível esclarecer com mais detalhes as funcionalidades do sistema.

A princípio, duas rotas são responsáveis por tratar os *inputs* (*i.e.*, MongoChainRouter e MongoChainEntityRouter) provenientes das requisições realizadas pelos clientes da aplicação. A partir disso, o Mongochain utiliza do *Middleware Pattern* existente no *framework* Express [2] como base da aplicação. Quando as rotas são requisitadas, Sousa (2020) esclarece que o *Middleware Pattern* entra em ação realizando a interceptação das chamadas às rotas a partir dos parâmetros *req* e *res* das requisições, os quais são feitos para fazer a manipulação dos dados encontrados na requisição e resposta. A manipulação dessas requisições se inicia repassando seus dados para os *Controllers* que, internamente, acabam por criar e registrar operações de leitura e escrita de acordo com as tarefas solicitadas pelo *framework*. Operações essas que posteriormente são registradas na *blockchain* e no SGBD MongoDB.

O Mongochain, em sua forma padrão de implementação, traz um ambiente onde são

[1] https://www.mongodb.com/atlas/database
[2] *https* : *//expressjs.com/pt* − *br/*

simuladas transações de valores monetários entre dois participantes da rede. Ou seja, um usuário remetente transfere determinado valor para um usuário destinatário da aplicação. Dessa forma, o *framework* possui 4 coleções em que os dados são armazenados no MongoDB. Tais coleções são exibidas na sequência:

- *blocks*:

  É a coleção onde a *blockchain* completa fica armazenada. Aqui é possível visualizar cada bloco que foi minerado e integrado à rede descentralizada. Cada documento pertencente a essa coleção é um bloco da *blockchain* e aninhado a cada documento consiste um *array* de objetos que simbolizam o grupo de transações realizadas em cada bloco. A figura 15 ilustra o bloco gênesis armazenado no MongoDB.

Figura 15 – Bloco gênesis armazenado no MongoDB

Fonte: Adaptado de (SOUSA, 2020)

- *senders*:

  É a coleção onde os usuários remetentes de uma transação ficam armazenados. Aqui cada usuário possui um atributo denominado *amount*, o qual significa o seu saldo disponível.

- *recipients*:

  É a coleção onde os usuários destinatários de uma transação ficam armazenados. Aqui cada usuário também possui um atributo denominado *amount*, o qual significa o seu saldo disponível.

- *transferences*:

  É o local onde as transações referentes às transferências entre remetentes e destinatários ficam armazenadas. Aqui é possível identificar quem enviou e para quem foi enviado através dos *ids* de remetentes e destinatários, respectivamente, e a quantidade que foi enviada através do atributo *amount*.

A figura 16 exibe as coleções existentes no Mongochain em sua implementação padrão.

Figura 16 – Coleções existentes no Mongochain

blocks	*senders*	*recipients*	*transferences*
block: Object	*id*: ObjectId *name*: String *amount*: Number *createdAt*: Date	*id*: ObjectId *name*: String *amount*: Number *createdAt*: Date	*senderId*: ObjectId *recipientId*: ObjectId *amount*: Number *status*: String *createdAt*: Date

Fonte: Adaptado de (SOUSA, 2020)

Sua funcionalidade inicial e fundamental é permitir a criação de uma *blockchain* a partir do momento em que o usuário realiza a requisição específica que faz o *framework* gerar o primeiro bloco (*i.e.*, bloco gênesis) e deixá-lo armazenado no MongoDB enquanto também fica registrado na rede descentralizada. Ademais, outras funcionalidades são os cadastros de usuários remetentes e destinatários. São eles que realizam as operações de transferência de valores no sistema.

A adição de novos nós na rede é outra funcionalidade existente no Mongochain. Dessa forma, aumenta-se o tamanho da rede tornando-a distribuída e descentralizada. Quando um novo nó é adicionado, é feito um *broadcast* na rede para que esse novo nó seja adicionado à *blockchain* de cada máquina pertencente ao ambiente descentralizado. Portanto, para que os integrantes da rede consigam verificar todos os outros nós existentes, o Mongochain possui a funcionalidade de consultar toda a *blockchain*. Em consequência disso, cada nó consegue visualizar de forma transparente os outros nós e as transações realizadas na cadeia de blocos.

Além disso, considerando esse funcionamento em um ambiente distribuído, é possível a ocorrência de falhas nos nós da rede. É possível acontecer alguma indisponibilidade ou também alguma falha para incluir novos blocos na *blockchain*. Ademais, Sousa (2020) relata que pode ocorrer de algum nó mal-intencionado tentar burlar a rede descentralizada com o envio de dados fraudulentos, tendo a intenção de ludibriar os demais participantes de que as transações que vão sendo realizadas são legítimas e corromper o sistema. Para inviabilizar esses tipos de atividades espúrias, através de outro *middleware*, o Mongochain possui a funcionalidade que realiza uma validação de consenso na rede, a qual tem o tamanho da *blockchain* como regra. Assim, a partir desse algoritmo, é possível que qualquer nó consiga garantir o fluxo de processamento em cima da *blockchain* original evitando contratempos indesejados.

Com relação às transferências realizadas dentro do *framework*, o Mongochain disponibiliza essa funcionalidade permitindo que um usuário remetente envie valores a um destinatário. Ao ocorrer esse processo de transferência, as operações que vão sendo realizadas são distribuídas na rede (*i.e.*, é realizado o *broadcast* para os demais nós participantes) e aguardam serem validadas para entrarem de fato na *blockchain*. É o processo de mineração o responsável por isso, o qual é realizado através do algoritmo PoW. A partir do momento em que um novo bloco é minerado, significa que tais operações são válidas e o bloco em que estão contidas é acoplado à rede. Dessa forma, o bloco é persistido na *blockchain* e no MongoDB concomitantemente. Por fim, as demais funcionalidades encontradas são as exibições das transferências realizadas por determinados remetentes e também por determinados destinatários por meio de seus *ids*. Essas são as operações que ficam armazenadas na coleção *transferences*.

3.3 Considerações Finais

Neste capítulo, foi abordado de forma mais clara o *framework* Mongochain. A sua motivação que inclui as lacunas existentes entre os modelos ACID e BASE , as tecnologias utilizadas para a construção e desenvolvimento do *framework*, o comportamento do sistema e suas funcionalidades de acordo com sua arquitetura foram esclarecidas. Com isso, a maneira como o Mongochain integra os modelos ACID e BASE em conjunto de uma rede *blockchain* descentralizada para realizar o gerenciamento de transações foi abordada e esclarecida para uma melhor compreensão do *framework*.

Capítulo 4

Trabalhos Relacionados

Neste capítulo, são abordados os trabalhos que se encontram no estado da arte e possuem relação com esta pesquisa de Mestrado. Tais trabalhos, em sua maioria, são referentes a SGBDs NoSQL, tendo como alvo principal o SGBD MongoDB. Dessa forma, buscando relatar informações a respeito dos aspectos de consistência existentes no SGBD alvo. Ou seja, são abordados trabalhos que, de alguma forma, buscam relatar avaliações dos níveis de consistência que podem ser configurados dentro do MongoDB. Além disso, os principais trabalhos que realizam algum tipo de avaliação de desempenho do SGBD MongoDB utilizando algum *benchmark* também são abordados.

4.1 *Strings* de busca

Para a busca de trabalhos relacionados a esta pesquisa, foram utilizadas duas *strings* de busca. A primeira com o objetivo de encontrar trabalhos que, de algum modo, realizam avaliações de consistência baseadas em algumas configurações dos níveis de consistência permitidos pelo MongoDB. A segunda é baseada na busca de trabalhos que utilizam o *benchmark* YCSB ou sua extensão, YCSB+T, para avaliar o desempenho do SGBD MongoDB, analisando, por exemplo, métricas como a latência das cargas de trabalho. Diante disso, utilizando os operadores lógicos *AND* e *OR*, essas *strings* foram combinadas para buscar trabalhos no idioma português e inglês entre os anos 2017 e 2022 nas plataformas Springer Link[1], IEEEXplore Digital Library[2], ACM Digital Library[3], Scopus[4] e DBLP:

[1] https://link.springer.com/
[2] https://ieeexplore.ieee.org/Xplore/home.jsp
[3] https://dl.acm.org/
[4] https://www.scopus.com/

computer science bibliography[5]. As sintaxes de busca que foram utilizadas em cada plataforma são as seguintes:

- **Springer Link**

  (MongoDB AND (YCSB OR YCSB+T OR YCSB-T OR evaluation OR benchmark OR assessment OR performance))
  e
  (MongoDB AND (consistency OR integrity OR levels OR tunable OR adjustable))

- **IEEEXplore Digital Library**

  ("Document Title":"MongoDB") AND ("All Metadata":"YCSB"OR "All Metadata": "YCSB+T"OR "All Metadata":"YCSB-T"OR "All Metadata":"evaluation"OR "All Metadata":"benchmark"OR "All Metadata":"assessment"OR "All Metadata": "performance")
  e
  ("Document Title":"MongoDB") AND ("All Metadata":"consistency"OR "All Metadata":"integrity"OR "All Metadata":"levels"OR "All Metadata":"tunable"OR "All Metadata":"adjustable")

- **ACM Digital Library**

  Title:(Mongodb) AND (Abstract:(ycsb) OR Abstract:(ycsb+t) OR Abstract:(ycsb-t) OR Abstract: (benchmark) OR Abstract:(evaluation) OR Abstract:(assessment) OR Abstract:(performance))
  e
  Title:(MongoDB) AND (Abstract:(consistency) OR Abstract:(integrity) OR Abstract:(levels) OR Abstract:(tunable) OR Abstract:(adjustable))

- **Scopus**

  (TITLE (Mongodb) AND TITLE-ABS-KEY (ycsb) OR TITLE-ABS-KEY (ycsb AND t) OR TITLE-ABS-KEY (ycsb-t) OR TITLE-ABS-KEY (benchmark) OR TITLE-ABS-KEY (evaluation) OR TITLE-ABS-KEY (assessment) OR TITLE-ABS-KEY (performance)) AND PUBYEAR > 2017
  e
  (TITLE (Mongodb) AND TITLE-ABS-KEY (consistency) OR TITLE-ABS-KEY (integrity) OR TITLE-ABS-KEY (levels) OR TITLE-ABS-KEY (tunable) OR TITLE-ABS-KEY (adjustable)) AND PUBYEAR > 2017

- **DBLP**

[5] https://dblp.uni-trier.de/

MongoDB ycsb|evaluation|benchmark|assessment|performance
e
MongoDB consistency|integrity|levels|tunable|adjustable

Na tabela 3, os resultados encontrados baseados nessas *strings* de busca são exibidos. Ademais, uma observação a ser feita é de que, em algumas bases de dados, os mesmos trabalhos são retornados.

Tabela 3 – Resultados das buscas nas bases de dados acadêmicas

Base de dados	**Resultado**	**Estudos selecionados**
Springer Link	43	3
IEEEXplore Digital Library	32	1
ACM Digital Library	13	3
Scopus	94	5
DBLP	25	7
Total sem repetições	**198**	**10**

Fonte: Próprio Autor

4.2 Critérios para escolha e exclusão de trabalhos relacionados

A quantidade de pesquisas sobre questões relacionadas ao desempenho e consistência do SGBD MongoDB é grande. Sendo assim, como requisitos para inclusão de trabalhos com melhor exatidão, qualidade e precisão do tema, foram adotados os filtros de período de publicação e idioma. Além disso, foram utilizadas as combinações de palavras-chaves abordadas na seção 4.1 (*Strings* de busca), as quais especificam que o foco da pesquisa são trabalhos que avaliam os níveis de consistência ou que utilizam um dos *benchmarks* (*i.e.*, YCSB ou YCSB+T) para os experimentos baseados no desempenho do SGBD MongoDB. Tais características permitiram delimitar um escopo mais preciso para abordar trabalhos correlatos.

Por outro lado, como abordado anteriormente, os requisitos de exclusão permitiram eliminar trabalhos um pouco mais antigos, anteriores ao ano de 2017, e também trabalhos em que não são utilizados os idiomas português ou inglês. Ademais, trabalhos fora dos escopos das palavras chaves também foram eliminados.

4.3 Descrição dos estudos selecionados

4.3.1 Matallah, Belalem e Bouamrane (2017)

Na pesquisa de Matallah, Belalem e Bouamrane (2017), é observado que existe uma dificuldade para escolher uma solução para um problema específico quando se trata do ecossistema NoSQL. O número elevado de soluções leva os autores a citarem o *benchmark* YCSB como uma ferramenta poderosa e relevante no quesito avaliação de BD em nuvem, tornando-se um padrão de ferramenta de *benchmark* para soluções NoSQL da indústria. Considerando tal prerrogativa, escolhem dois SGBDs NoSQL, MongoDB e HBase, para realizar testes de desempenho com o objetivo de auxiliar os agentes interessados em computação em nuvem para uma melhor tomada de decisão na hora de selecionar uma das opções NoSQL.

Relativo à análise experimental, os autores utilizaram como referência um trabalho anterior de outro autor como base de parâmetros para a realização dos testes. Dessa forma, além das 6 cargas de trabalho existentes no YCSB, também foram utilizadas duas cargas de trabalho adicionais encontradas no trabalho anterior em que os autores se utilizaram para comparação. Essas duas outras cargas de trabalho são as cargas de trabalho G (*i.e.*, *Update mostly*) e H (*i.e.*, *Update only*).

Em seus resultados, os autores observaram que para as operações de *insert* relativas ao carregamento da base de dados o SGBD HBase é mais eficiente que o SGBD MongoDB por conta de não necessitar de muita memória para a execução dessa carga de trabalho. Relativo às demais cargas de trabalho, seus resultados mostram que em cargas de trabalho onde parte das operações são de leitura, o SGBD MongoDB supera o SGBD HBase em consequência da sua melhor otimização para essas operações. Em contrapartida, o SGBD HBase se sobressai nas operações de escrita pelo mesmo motivo de possuir melhor otimização nesse contexto e, além disso, por conta do SGBD MongoDB utilizar um mecanismo de bloqueio para realizar atualizações, retardando o tempo de execução.

4.3.2 González-Aparicio et al. (2017)

González-Aparicio et al. (2017) explicam que os SGBD NoSQL não exigem consistência forte nos dados e nem oferecem suporte a transações. Dessa forma, relatam que a implementação de transações não é uma funcionalidade trivial para um SGBD NoSQL, considerando que tais SGBDs são projetados para tratarem das três propriedades do universo *Big Data* (*i.e.*, Volume, Variedade e Velocidade) e que, portanto, tais propriedades dificultam a implementação de transações no contexto NoSQL. Com isso, desenvolvem esquemas transacionais que fornecem facilidades de transação aos BDs NoSQL. Em adição, investigam o processamento de transações de dois SGBDs NoSQL (*i.e.*, MongoDB e Riak) e realizam testes com essas ferramentas. Baseados nesses novos esquemas propos-

tos, utilizam a mesma ferramenta de *benchmark* usada nesta pesquisa de Mestrado (*i.e.*, a extensão do *benchmark* YCSB, YCSB+T) para fazerem suas avaliações e analisar as questões de consistência e eficiência dos dados.

Neste trabalho, para a realização dos testes, os autores criaram um cenário de caso de uso onde utilizam os esquemas transacionais propostos para ambos SGBDs e realizam as avaliações do nível de consistência alcançado em termos de quantidade de acordo com a número de operações realizadas em seus experimentos. Além disso, também relatam a incorrência de sobrecarga no desempenho devido ao sistema de processamento de transações.

4.3.3 Huang et al. (2019)

Huang et al. (2019) utilizam o MongoDB Atlas, o qual é uma plataforma de BD como serviço oferecida pelo MongoDB e tem sua hospedagem em nuvem, para realizarem testes de acordo com algumas configurações que envolvem a manutenção da consistência dos dados nesse SGBD. Para tal experiência, os autores descrevem a propriedade de consistência do BD de acordo com a probabilidade de um valor desatualizado ser consumido por uma leitura, ou seja, a leitura de um valor diferente daquele que foi definido pela última escrita do BD. Além disso, utilizam um aplicativo de *benchmarking* que é constituído de um leitor e um gravador, os quais se comunicam com o *cluster* e realizam as operações que buscam medir a consistência nos experimentos.

Neste estudo, o *cluster* do MongoDB Atlas está definido no modo *Replica Set* com três máquinas, contendo um nó primário e dois secundários, e apenas o parâmetro *read preference* é utilizado para *benchmarking* da consistência. Já o parâmetro *write concern* é padrão (w: 1) para todas as variações dos testes. Sendo assim, três variantes de *read preference* são utilizadas, sendo elas o valor padrão (*primary*) e os valores alternativos *secondary* e *nearest*. Além disso, também exibem dados referentes à latência de escrita e leitura em seus testes.

Relativo aos resultados dos experimentos, os autores concluíram que, sobre a latência de leitura, as três opções de *read preference* obtiveram resultados parelhos. Relatam que isso pode ocorrer em consequência de que a máquina que realiza o *benchmarking* se encontra na mesma região que a instância do MongoDB Atlas. Com relação às questões de consistência, os autores relatam que para as leituras configuradas com *read preference primary* não foram encontradas inconsistências. Entretanto, quando não é selecionada a opção *primary*, é possível encontrar leituras inconsistentes.

4.3.4 Schultz, Avitabile e Cabral (2019)

Neste trabalho, Schultz, Avitabile e Cabral (2019) explicam que os dados coletados para análise incorporam os parâmetros *readConcern* e *writeConcern*. Os autores realizam

um levantamento de quais são os níveis *readConcern* e *writeConcern* mais utilizados pelos desenvolvedores de *software* e relatam os padrões utilizados pelos usuários na escolha do nível de consistência adotado em suas aplicações. Nesses casos, no momento da escolha entre consistência ou latência, os autores explicam que existe a preferência dos desenvolvedores em utilizar garantias de durabilidade mais fortes, porém, em consequência do custo da maior latência, escolhem garantias de consistência mais fracas. Essa consideração é baseada no motivo das falhas serem raras e a perda de gravação ser relativamente pequena. Dessa forma, a tolerância relativa a uma pequena perda de gravações é aceita pelos desenvolvedores em troca de um melhor desempenho relacionado à latência. Ademais, os autores realizam essa comparação de latência executando testes em operações com os níveis *writeConcern w: 1* e *w: majority* em um *replica set* de três nós e sinalizando que todas as operações devem gravar a escrita no diário (*i.e.*, *journal* = *true*).

Em seus experimentos, os autores exibem melhores resultados referentes à latência quando níveis mais baixos de *write concern* são utilizados. Quando um nível de consistência mais forte é utilizado, sua latência também é superior.

4.3.5 Kamsky (2019)

Neste trabalho, Kamsky (2019) realiza extensas modificações no benchmark TPC-C para deixá-lo adequado a realizar testes que medem o desempenho de transações no SGBD MongoDB. A autora explica que tais modificações foram originadas pelo fato de que, no ano de 2011, quando o *framework* PyTPCC foi criado, ainda não havia suporte necessário para o tratamento de transações em sua implementação original.

A autora realiza os testes utilizando a plataforma de BD como serviço fornecida pelo MongoDB, MongoDB Atlas, com diferentes instâncias existentes dentro do ambiente (*e.g.*, instâncias M50, M60 e M80) variando alguns parâmetros de níveis de consistência encontrados no MongoDB, tais como *readPreference*, *readConcern* e *writeConcern*. Porém, não fica claro como a combinação entre os valores dos parâmetros é utilizada, considerando que a autora relata explicitamente apenas algumas configurações dos níveis de consistência (*e.g.*, *writeConcern* = 1 e *writeConcern* = majority) neste trabalho.

4.3.6 Araujo et al. (2021)

Araujo et al. (2021) abordam que os SGBDs NoSQL são essenciais para o tratamento massivo de dados produzidos no dia a dia e, dessa forma, realizam dois experimentos com o objetivo de comparar o desempenho de dois desses SGBDs mais utilizados pelos desenvolvedores, MongoDB e Cassandra. O segundo experimento foge do escopo de *benchmark* utilizado neste trabalho. Porém, o primeiro experimento retrata a utilização da ferramenta estudada nesta pesquisa de Mestrado, YCSB, utilizando três de suas seis cargas de trabalho para a execução dos testes de desempnho.

Diante disso, realizaram experimentos utilizando o MongoDB no modo *sharded*, o qual não é o modo padrão fornecido gratuitamente pelo MongoDB (*i.e.*, modo *Replica Set*) e, além disso, foram utilizadas as cargas de trabalho A (*i.e.*, *Update heavy*), B (*i.e.*, *Read heavy*) e C (*i.e.*, *Read only*) do YCSB para a realização dos testes.

Como resultados do primeiro experimento utilizando o YCSB e suas cargas de trabalho, os autores chegaram à conclusão de que o SGBD Cassandra possui um maior desempenho de maneira geral, pois em todos os testes conseguiu ter uma latência menor e um *throughput* maior no experimento.

4.3.7 Seghier e Kazar (2021)

Seghier e Kazar (2021) descrevem um trabalho o qual colocam em evidência os SGBDs NoSQL, relatando que tais SGBDs possuem um melhor desempenho no tratamento de cargas de trabalho mais elevadas levando em consideração aplicativos que trabalham com *Big Data* e utilizam um ambiente em nuvem. A partir desse contexto, os autores utilizam três SGBDs NoSQL bem conhecidos (*i.e.*, Redis, MongoDB e Cassandra) para a execução de testes e têm como objetivo analisar de maneira geral e mais específica diversas cargas de trabalho oferecidas pelo *benchmark* YCSB.

Como metodologia para o levantamendo de dados e comparação de desempenho, os autores utilizaram o *benchmark* YCSB. Dessa forma, utilizaram em seus testes todas as cargas de trabalho que o *benchmark* oferece (*i.e.*, *Workload* A, *Workload* B, *Workload* C, *Workload* D, *Workload* E e *Workload* F) para avaliar os diferentes cenários e auxiliar desenvolvedores e pessoas interessadas a escolherem o BD mais apropriado para as suas necessidades.

Com relação aos resultados, os autores mostram que na fase de carregamento o SGBD MongoDB teve melhor desempenho em relação aos SGBDs Redis e Cassandra. Entretanto, em operações de leitura, o SGBD Redis tem o melhor desempenho por possuir uma otimação de leitura melhor que a dos SGBDs MongoDB e Cassandra.

4.3.8 Matallah, Belalem e Bouamrane (2020)

Matallah, Belalem e Bouamrane (2020) continuam suas pesquisas a respeito do desempenho de alguns dos SGBDs NoSQL mais utilizados do mercado. Em um trabalho anterior (*i.e.*, *Experimental comparative study of NoSQL databases: HBASE versus MongoDB by YCSB*), os mesmos autores já estudaram o desempenho do MongoDB realizando uma comparação perante o SGBD HBase. A extensão dessa pesquisa segue neste trabalho onde a quantidade de SGBDs a serem testados é maior. Além do SGBD HBase, também testam e comparam o desempenho do MongoDB frente aos SGBDs Cassandra, Redis, Couchbase e OrientDB.

Os autores abordam que existem mais de trezentas e cinco soluções NoSQL de diferentes arquiteturas disponíveis no mercado de tecnologia da informação. Dessa maneira, buscam aumentar a quantidade de SGBDs a serem testados e avaliados, trazendo mais resultados para a comunidade científica e para profissionais da área, tornando-os aptos a tomarem melhores decisões na escolha do SGBD NoSQL a ser utilizado de acordo com a demanda de trabalho.

Como este trabalho é uma extensão de um trabalho anterior dos mesmos autores, os testes de desempenho dos SGBDs selecionados continuam sendo gerados utilizando o *benchmark* YCSB em uma única máquina física. Ademais, as seis cargas de trabalho existentes do *benchmark* são utilizadas no experimento executando mil operações cada uma.

4.3.9 Ouyang, Wei e Huang (2021)

Ouyang, Wei e Huang (2021) abordam que o SGBD MongoDB é um dos pioneiros a oferecer um modelo de consistência denominado consistência causal. Entretanto, por ser um modelo de consistência complexo, questionam se esse modelo de consistência foi implementado corretamente. Em adição, comentam que a consistência causal é permitida nas sessões do cliente e pode ser combinada com a consistência configurável do MongoDB, a qual permite a seleção da compensação existente entre latência e consistência no nível de operação do BD.

Os autores relatam que outra equipe, denominada Jepsen, realizou testes referentes à consistência causal do MongoDB. Porém, faltando informações referentes a termos de especificação e qual variante de consistência causal foi utilizada, motivo pelo qual a credibilidade dos resultados foi prejudicada. Além disso, informam que existe mais de uma variante de consistência causal, a qual não foi informada no teste anterior.

Com relação ao cenário de testes, utilizam o *benchmark* YCSB para geração de dados e o *framework* de testes Jepsen para analisar as três possíves variações de consistência causal juntamente de níveis de consistência diferentes do MongoDB.

4.3.10 Martins et al. (2021)

Martins et al. (2021) abordam um trabalho realizando uma avaliação entre três dos SGBDs NoSQL mais populares entre os desenvolvedores, MongoDB, CouchDB e Couchbase. Para isso, utilizam dois balizadores para medir a competência dos SGBDs. O primeiro é a metodologia OSSpal, a qual serve para comparar as bases de dados de acordo com critérios de qualidade geral, instalação, usabilidade, robustez, segurança e escalabilidade. O segundo é o *benchmark* YCSB, o qual os autores utilizam para obterem medidas de desempenho, verificando o tempo de execução e *throughput* em dois cenários de testes. Um deles representa a fase de carregamento do BD (*i.e.*, fase de *load* do *benchmark*) e

o outro é referente a uma das cargas de trabalho fornecidas pelo YCSB, a qual contém uma proporção de cinquenta por cento de leitura e cinquenta por cento de escrita (*i.e.*, *Workload* A *Update heavy*) em suas operações.

Para o ambiente dos experimentos, os autores utilizaram instalações locais de cada BD para a aplicação do *benchmark*. Além disso, para cada cenário de teste, a carga de trabalho foi executada em cima de três volumes de BD diferentes. Sendo os testes feitos em cima de um volume de base de dados de mil, quinze mil e cinquenta mil registros e operações respectivamente. Dessa forma, o *benchmark* YCSB foi executado e utilizado para a geração de informações relativas a cada um desses cenários.

Os autores relatam em seus experimentos que o SGBD MongoDB tem o melhor desempenho tanto em tempo de execução quanto em *throughput* na fase de *load* do BD e também utilizando a carga de trabalho A do YCSB.

4.4 Características dos estudos selecionados

Nos estudos que se referem especificamente às métricas de desempenho, a coleta de dados e as avaliações a respeito do desempenho dos SGBDs NoSQL estudados foram realizadas por meio da utilização de *benchmarks* e de ambientes de implantação diferentes, variando entre o ambiente local e em *cluster*. Com relação aos estudos que abordam as características de consistência encontradas no SGBD MongoDB, é possível encontrar níveis de integridade diferentes de acordo com a configuração de parâmetros passados para as operações que são realizadas no BD. Tais estudos se diferenciam ao focarem suas pesquisas em variações específicas, não envolvendo todos os principais parâmetros fornecidos pelo SGBD. Dessa forma, é interessante esclarecer quais configurações esses trabalhos utilizaram e quais ferramentas de *benchmark* foram adotadas para a coleta de dados a respeito dos experimentos realizados. Além disso, também são expostas as configurações adotadas por este trabalho para mostrar as diferenças entre os estudos correlatos e a contribuição desta pesquisa para o estado da arte.

Diante disso, a tabela 4 exibe as características referentes aos trabalhos relacionados e a esta pesquisa de Mestrado.

Tabela 4 – Estudos selecionados e suas características

Estudo	**Utiliza o YCSB ou YCSB+T**	**Ambiente utilizado**	**Parâmetros de ajuste de consistência do MongoDB**	**Comparativo realizado**
Matallah, Belalem e Bouamrane (2017)	YCSB	Local	N/D	MongoDB e HBase
González-Aparicio et al. (2017)	YCSB+T	*Cluster*	N/D	MongoDB e Riak
Huang et al. (2019)	Não	*Cluster*	*Read Preference*	MongoDB e MongoDB
Schultz, Avitabile e Cabral (2019)	Não	*Cluster*	*Write Concern*	MongoDB e MongoDB
Kamsky (2019)	Não	*Cluster*	*Write Concern*, *Read Concern* e *Read Preference*	MongoDB e MongoDB
Araujo et al. (2021)	YCSB	*Cluster*	N/D	MongoDB e Cassandra
Seghier e Kazar (2021)	YCSB	Local	N/D	MongoDB, Redis, Cassandra
Matallah, Belalem e Bouamrane (2020)	YCSB	Local	N/D	MongoDB, HBase, Cassandra, Redis, Couchbase, OrientDB
Ouyang, Wei e Huang (2021)	YCSB	*Cluster*	N/D	MongoDB e MongoDB
Martins et al. (2021)	YCSB	Local	N/D	MongoDB, CouchDB, Couchbase
Morceli (2024)	YCSB+T	*Cluster*	*Write Concern*, *Read Concern* e *Read Preference*	MongoDB e Mongochain

Fonte: Próprio Autor

Capítulo 5

Avaliações Experimentais do SGBD MongoDB

Este capítulo apresenta a infraestrutura física utilizada para os experimentos desta pesquisa e o método de coleta de dados dos diferentes níveis de consistência que podem ser combinados para o SGBD MongoDB na seção 5.1 (Infraestrutura Física do Ambiente e Metodologia de Coleta de Dados). Esta metodologia permite a geração dos diversos conjuntos de dados atrelados, respectivamente, às variações de configurações de consistência do MongoDB escolhidas para análise. Além disso, a avaliação de desempenho das cargas de trabalho escolhidas para os experimentos são abordadas na seção 5.2 (Avaliação de Desempenho do SGBD MongoDB) e a avaliação de consistência realizada com a carga de trabalho de economia fechada do *benchmark* YCSB+T é descrita na seção 5.3 (Avaliação de Consistência do SGBD MongoDB).

Com relação aos resultados de todos os experimentos feitos nesta pesquisa e às implementações referentes às produções técnicas realizadas, tudo foi disponibilizado a partir de um *link*[1] para acesso remoto ao gerenciador de repositório de *software* Gitlab.

5.1 Infraestrutura Física do Ambiente e Metodologia de Coleta de Dados

A análise experimental foi feita utilizando o cliente YCSB+T em uma máquina física: Core i7 9750H, CPU 2.60GHz, 16GB de RAM, HD de 1 TB e SSD de 128 GB. O sistema operacional usado foi o Windows 10 Home de 64 bits.

[1] https://gitlab.com/morcelicaio/experimento-mongodb-mongochain

A estrutura para a realização dos experimentos é sustentada pela plataforma em nuvem oferecida pelo MongoDB, denominada MongoDB Atlas. Esta plataforma fornece dois modos para a construção do *cluster* onde o SGBD MongoDB é executado. Um deles é o modo *Replica Set*, o qual é o padrão disponibilizado de forma gratuita e que é utilizado nesta pesquisa. A outra opção disponibilizada é o modo *Sharded*, o qual é um recurso pago e permite a funcionalidade de fragmentação dos dados da aplicação. Tal recurso estende o modo *Replica Set*.

Além dos modos disponíveis para a configuração do ambiente, o MongoDB Atlas também oferece configurações distintas para as instâncias que fazem parte do *cluster* (*i.e.*, os nós do *cluster*). Dessa forma, a instância disponibilizada de forma gratuita e que foi utilizada é chamada *M0*, a qual possui seus recursos computacionais compartilhados com outros usuários, tais como memória RAM e CPU virtual. Por outro lado, também são oferecidas instâncias pagas (*e.g.*, *M10, M20, M30, M40, M50, M60, M80, M140, M200, M300, M400* e *M700*), as quais possuem recursos computacionais dedicados exclusivamente para o usuário e possuem características superiores de hardware.

Diante desse contexto, foram selecionadas as opções gratuitas para o *cluster* por razões financeiras. Dessa forma, é utilizado o SGBD MongoDB em sua plataforma em nuvem, MongoDB Atlas, com a configuração de um *cluster* no modo *Replica Set* possuindo três instâncias *M0* sendo elas um nó primário e dois nós secundários para a execução dos experimentos.

O *benchmark* usado é o YCSB+T que deriva da versão 0.17.0 do YCSB. O pacote de *benchmark* fornece um conjunto de cargas de trabalho padrão que podem ser executadas da seguinte forma:

- Carga de Trabalho A (*Update Heavy*): Constituída por uma proporção de 50 % de leitura e 50 % de atualização;

- Carga de Trabalho B (*Read Mostly*): Constituída por uma proporção de 95 % de leitura e 5% de atualização;

- Carga de trabalho C (*Read Only*): Constituída por uma proporção de 100 % de leitura;

- Carga de Trabalho D (*Read latest*): Constituída por uma relação de 95 % de leitura e 5% de *insert*;

- Carga de Trabalho E (*Short Ranges*): Constituída por uma proporção de 95 % *scan* e 5% *insert*;

- Carga de Trabalho F (*Read-modify-write*): Constituída por uma relação de 50 % de leitura e 50 % de leitura-modificação-escrita;

Dessa forma, foram carregados 1000 registros gerados pelo arquivo de propriedades padrão YCSB+T através da classe de carga de trabalho *CoreWorkload*. Os tempos de execução das cargas de trabalho obtidas durante as operações de leitura, escrita e atualização são apresentados. Todas as cargas de trabalho executam 1.000 operações e ao fim do experimento exibem os dados resultantes da execução. Na seção 5.2 (Avaliação de Desempenho do SGBD MongoDB), é descrita uma avaliação do desempenho das cargas de trabalho escolhidas para os experimentos.

5.2 Avaliação de Desempenho do SGBD MongoDB

Para a realização da avaliação de desempenho das diversas combinações de parâmetros possíveis no SGBD MongoDB em união com as diversas cargas de trabalho do *benchmark*, foram necessárias diversas baterias de testes. Isso porque os experimentos englobam 7 configurações de testes diferentes possuindo 3 combinações de testes distintas em cada configuração, ou seja, em cada configuração é feita a troca dos valores dos parâmentros para as operações do MongoDB. Algumas das 7 configurações de testes são ilustradas juntamente com suas possíveis combinações de parâmetros na figura 17.

Figura 17 – Configurações de testes para o SGBD MongoDB

Configuração de Testes
Quando readConcern = local

Configuração de Teste 1
Combinações possíveis

w	readPreference	readConcern
1	primary	local
majority	primary	local
3	primary	local

Configuração de Teste 5
Combinações possíveis

w	readPreference	readConcern
1	secondary	local
majority	secondary	local
3	secondary	local

Configuração de Testes
Quando readConcern = majority

Configuração de Teste 2
Combinações possíveis

w	readPreference	readConcern
1	primary	majority
majority	primary	majority
3	primary	majority

Configuração de Teste 4
Combinações possíveis

w	readPreference	readConcern
1	secondary	majority
majority	secondary	majority
3	secondary	majority

Fonte: Próprio autor

Para cada uma das possíveis combinações dentro de cada configuração de teste, foram realizadas 5 baterias de testes. Dessa forma, executando-se 105 baterias de testes para cada carga de trabalho (*i.e.,* A até F) e 630 baterias de testes no total.

Para a verificação de desempenho das possíveis combinações de teste, foi utilizada a métrica de tempo de execução (*i.e., runtime*) fornecida pelo *benchmark*. Sendo assim, os experimentos puderam fornecer os seguintes dados para análise:

- **Listagem geral das combinações de testes em cada carga de trabalho iniciando com os melhores desempenhos até a finalização com as piores performances**;

  Com esses dados é possível realizar o levantamento dos demais tópicos presentes, pois a listagem colabora com a organização dos dados e oferece maior facilidade e clareza para as análises.

- **Diferença de desempenho entre a combinação com melhor desempenho na carga de trabalho e a combinação com menor desempenho**;

  Com esses dados foi possível identificar a diferença de desempenho existente entre as combinações com o melhor e o pior desempenho. Dessa forma, os experimentos indicaram uma variação de performance entre o melhor e o pior desempenho de até 13,40 % entre as cargas de trabalho A, B, C, D e F do *benchmark*, as quais realizam operações de escrita e leitura de dados em diferentes proporções. Para a carga de trabalho E, a diferença encontrada foi de 29,66 %. Essa maior diferença se deve ao fato dessa carga de trabalho possuir a maioria de suas operações sendo de *scan* na base de dados e de não possuir nenhuma configuração prévia específica de índice no MongoDB para a realização das varreduras.

  Tais dados permitem analisar que a diferença de desempenho não é significativa o suficiente para impedir que o desenvolvedor de um sistema escolha uma combinação de parâmetros para o MongoDB que forneça uma maior garantia de consistência para os dados. Na figura 18 os resultados referentes à carga de trabalho A do *benchmark* são exibidos.

- **Diferença de desempenho entre as combinações possíveis dentro de uma mesma configuração de teste**;

  Com esses dados foi possível identificar a diferença de desempenho existente entre as combinações presentes em uma mesma configuração de teste. Nesses casos, é possível checar a influência que a variação do parâmetro de escrita *w* exerce em uma mesma configuração de teste.

  A partir disso, os experimentos realizados indicaram uma variação de desempenho para as configurações de teste entre as cargas de trabalho A, B, C, D e F do *ben-*

Figura 18 – Combinações de parâmetros possíveis no SGBD MongoDB para testes com a carga de trabalho A

Combinação de Parâmetros	Runtime (ms)	Diferença de desempenho
W: 1 , RC: local , RP: secondary	142689.6	Valor de referência
W: 1 , RC: maj , RP: secondary	143339.6	0.45%
W: 1 , RC: snapshot , RP: secondary	143990.2	0.90%
W: maj , RC: local , RP: secondary	145282.2	1.78%
W: maj , RC: maj , RP: secondary	148701.6	4.04%
W: maj , RC: snapshot , RP: secondary	149228.8	4.38%
W: 1 , RC: local , RP: primary	149299.4	4.43%
W: 1 , RC: majority , RP: primary	149354.6	4.46%
W: 1 , RC: snapshot , RP: primary	151930.8	6.08%
W: 1 , RC: linearizable , RP: primary	152282.2	6.30%
W: 3 , RC: local , RP: secondary	153101.6	6.80%
W: maj , RC: local , RP: primary	153103.1	6.80%
W: maj , RC: maj , RP: primary	153869.2	7.27%
W: maj , RC: snapshot , RP: primary	154510.2	7.65%
W: maj , RC: linearizable , RP: primary	154839.8	7.85%
W: 3 , RC: maj , RP: secondary	154847.8	7.85%
W: 3 , RC: snapshot , RP: secondary	155528.6	8.26%
W: 3 , RC: local , RP: primary	156567	8.86%
W: 3 , RC: majority , RP: primary	157831.6	9.59%
W: 3 , RC: snapshot , RP: primary	159859	10.74%
W: 3 , RC: linearizable , RP: primary	164777.4	13.40%

Fonte: Próprio autor

chmark de até 7,58 %. Para a carga de trabalho E, a diferença encontrada foi de 20,55 %.

Os dados presentes para análise permitem avaliar que a influência no desempenho referente ao parâmetro *w* entre as combinações de uma mesma configuração de teste não é expressiva o suficiente para impossibilitar que o desenvolvedor de um sistema escolha uma combinação de parâmetros para o MongoDB que tenha uma maior garantia de consistência dos dados. A Figura 19 mostra os resultados das configurações de teste na carga de trabalho F do *benchmark*.

5.2.1 Influência dos parâmetros das combinações em relação ao desempenho

A partir da listagem com o desempenho de todas as combinações possíveis em todas as cargas de trabalho da camada de desempenho do *benchmark*, é possível verificar a influência que os parâmetros *w*, *readConcern* e *readPreference* exercem nos resultados. Como cada carga de trabalho possui uma variação diferente na proporção entre escritas e leituras, alguns parâmetros vão ser mais influentes nas cargas de trabalho em que se

Figura 19 – Combinações de parâmetros possíveis no SGBD MongoDB para testes com a carga de trabalho F

Configuração de teste 1	Diferença de desempenho%
W: 1, RC: local, RP: primary	Valor de referência
W: maj, RC: local, RP: primary	0.27
W: 3, RC: local, RP: primary	2.08

Configuração de teste 2	Diferença de desempenho%
W: 1, RC: majority, RP: primary	Valor de referência
W: maj, RC: maj, RP: primary	0.36
W: 3, RC: majority, RP: primary	2.25

Configuração de teste 3	Diferença de desempenho%
W: 1, RC: snapshot, RP: primary	Valor de referência
W: maj, RC: snapshot, RP: primary	0.64
W: 3, RC: snapshot, RP: primary	2.31

Configuração de teste 4	Diferença de desempenho%
W: 1, RC: maj, RP: secondary	Valor de referência
W: maj, RC: maj, RP: secondary	0.92
W: 3, RC: maj, RP: secondary	3.11

Configuração de teste 5	Diferença de desempenho%
W: 1, RC: local, RP: secondary	Valor de referência
W: maj, RC: local, RP: secondary	2.36
W: 3, RC: local, RP: secondary	4.99

Configuração de teste 6	Diferença de desempenho%
W: 1, RC: linearizable, RP: primary	Valor de referência
W: maj, RC: linearizable, RP: primary	0.86
W: 3, RC: linearizable, RP: primary	4.22

Configuração de teste 7	Diferença de desempenho%
W: 1, RC: snapshot, RP: secondary	Valor de referência
W: maj, RC: snapshot, RP: secondary	0.64
W: 3, RC: snapshot, RP: secondary	2.31

Fonte: Próprio autor

predominam as leituras enquanto que o parâmetro *w* terá mais influência nas cargas de trabalho com mais operações de escrita. Além disso, o local de preferência das leituras no *cluster* (*i.e., readPreference*) também exerce influência no desempenho. Dessa forma foi possível chegar às seguintes conclusões:

- **Em geral combinações com *readPreference secondary* possuem melhor desempenho**.

  Na maioria dos casos, as combinações que utilizam *read preference: secondary* apresentam melhor desempenho comparadas às combinações com *read preference: primary*. Isso acontece pelo fato da carga de leitura ser distribuída entre os nós secundários. Quando é utilizado *read preference: primary*, o nó primário recebe as requisições, o que ocasiona maior latência e consequentemente um menor desempenho na execução da carga de trabalho.

  Independentemente da carga de trabalho utilizada, esse padrão se repete. A concentração de leituras realizadas no nó primário diminui o desempenho. Por outro lado, o direcionamento de leitura aos nós secundários realiza um maior balanceamento de carga, evitando que as requisições sejam direcionadas apenas para um dos nós secundários. Em questão de desempenho, os experimentos evidenciaram uma melhora na performance das combinações de parâmetros quando *read preference: secondary*. Isso pode ser observado na figura 18 exibida anteriormente neste capítulo.

- **Nem sempre combinações com *readPreference: secondary* vão ter melhor desempenho**

  Nem sempre as combinações que utilizam um *read preference: secondary* vão apresentar melhor desempenho comparadas às combinações com *read preference: primary*. Isso irá depender de algumas variáveis importantes para o desempenho da execução da carga de trabalho. Uma delas é a própria carga de trabalho escolhida pelo usuário. Algumas são compostas pelo conjunto de operações de 95 % de leitura e 5 % de escrita. Nesses casos, uma combinação que utiliza *read preference: secondary* tende a possuir um desempenho melhor. Entretanto, quando a carga de trabalho é mais equilibrada, como no caso da proporção de 50 % de leitura e 50 % de escrita, as variáveis *write concern* e *read concern* influenciam no resultado final.

- ***w* influencia na latência de escrita e *read concern* na latência de leitura**

  Junto do parâmetro *read preference*, as combinações contém os parâmetros *write concern* e *read concern*. Nos experimentos realizados, é possível observar combinações onde *read preference: primary* tiveram um melhor desempenho comparado a combinações com *read preference: secondary*. É possível observar que quando uma carga de trabalho é balanceada entre 50 % de leitura e 50 % de escrita, uma combinação que possua maiores níveis de *write concern* e *read concern* combinados com *read preference: secondary* terão um desempenho inferior comparado a algumas combinações com *read preference: primary* que possuem níveis inferiores de *write concern* e *read concern*.

  Apesar da escolha de um *read preference: secondary*, quando se tem um nível mais elevado de *write concern* e *read concern*, essas variáveis elevam a latência da execução da carga de trabalho. Isso pode ocorrer pelo fato das operações de escrita terem que ser replicadas para a maioria ou todos os nós do *cluster* em algumas das combinações (*i.e., write concern > 1*) ou também pelo fato das leituras terem que confirmar o dado mais atualizado para a maioria dos nós do *cluster* (*i.e., read concern > local*).

- **Diferentes combinações de parâmetros podem ter desempenhos similares quando a relação custo-benefício entre *write concern* e *read concern* é avaliada**

  Nesse experimento foi evidenciado que algumas combinações possuem desempenhos bastante similares. Isso ocorre pelo fato de suas combinações de parâmetros, apesar de diferentes, possuírem características que tornam a performance da execução da carga de trabalho parelha.

  Não é trivial fazer uma avaliação de desempenho quando diversas variáveis estão envolvidas na execução de uma operação de BD em uma determinada carga de

trabalho, entretanto, com a compreensão de que um nível mais elevado de *write concern* aumenta o tempo de resposta de uma escrita ao cliente e de que um nível mais elevado de *read concern* aumenta o tempo de resposta de uma leitura, é possível identificar nos experimentos que existe uma similaridade entre determinadas combinações de parâmetros. Isso não significa que só por terem desempenhos parecidos o usuário pode escolher qualquer uma inconsequentemente. Um dos objetivos desse estudo é o de analisar e trazer clareza sobre os dados das performances das combinações existentes no MongoDB. Cabe ressaltar que a responsabilidade fica a cargo do desenvolvedor sobre entender qual é a configuração que se adequa melhor ao seu modelo de negócios, considerando que cada aplicação possui suas particularidades. Algumas podem precisar de uma maior performance e disponibilidade dos dados, sendo assim adotando níveis menores de *write concern* e *read concern* junto de um *read preference: secondary*. Por outro lado, outras aplicações podem tolerar um certo atraso nas respotas quando possuem a demanda de dados mais consistentes. Nesses casos podem utilizar maiores níveis de *write concern* e *read concern* unidos a um *read preference primary*. Ou seja, existirá uma decisão a ser tomada entre desempenho ou consistência dos dados.

Outra questão encontrada dentro das aplicações são suas proporções de leituras e escritas. Algumas aplicações utilizam demasiadamente as operações de leitura. Logo, é possível que o sistema não tenha uma demanda muito alta por dados sempre atualizados de forma imediata. Dessa forma, é viável pensar em configurações para o BD que tenham melhor performance tais como *read preference: secondary* e *read concern: local*. Em contrapartida, outras aplicações têm a necessidade de dados mais precisos, o que tem por consequência configurações de BD mais pesadas. Nesses casos, observa-se uma queda no desempenho que está atrelada a maiores níveis de *write concern* e *read concern* que são adotados para perceber uma melhora na consistência dos dados.

5.3 Avaliação de Consistência do SGBD MongoDB

Para a realização dos testes de consistência com o SGBD MongoDB, foi utilizada a camada de consistência do *benchmark* YCSB+T. Essa camada tem o objetivo de verificar a consistência da base de dados depois da carga de trabalho ser executada. A partir disso é possível verificar as hipóteses de pesquisa descritas neste trabalho, as quais validam se o SGBD MongoDB garante consistência forte quando utiliza os seus níveis de consistência mais restritos, se não garante consistência forte quando utiliza os seus níveis de consistência mais fracos e também validar a questão de desempenho, descrevendo se quanto maior é o nível de consistência, menor é a performance.

Para atingir essa verificação, a carga de trabalho de economia fechada tem a ideia

de simular uma economia fechada em que a quantia total armazenada em cada conta não entra ou sai do sistema enquanto o experimento é realizado. Por fim, a carga de trabalho, por meio de sua fase de validação *(i.e.,* fase que verifica a integridade do BD ao fim do teste), informa se o BD alterou seu estado consistente inicial para um estado de inconsistência ou se continuou íntegro da mesma forma em que se iniciou.

A princípio, para cada uma das combinações de valores dos parâmetros de consistência do SGBD MongoDB *(i.e.,* 21 combinações), foram realizadas 10 baterias de testes, totalizando, ao fim, 210 execuções da carga de trabalho de economia fechada. A figura 20 exibe o desempenho e o comportamento em relação à questão de integridade da base de dados de cada uma das combinações e quais delas apresentaram um estado inconsistente do BD ao fim dos experimentos.

Figura 20 – Resultados da avaliação de desempenho e consistência no SGBD MongoDB com a carga de trabalho de economia fechada

Combinação de parâmetros	Runtime (ms)	Apresentou inconsistência	Diferença de desempenho
W: 1 , RC: local , RP: secondary	384705.2	X	Valor de referência
W: 1 , RC: maj , RP: secondary	387850.3	X	0.81%
W: 1 , RC: snapshot , RP: secondary	390127.2	X	1.39%
W: maj , RC: local , RP: secondary	390386.6	X	1.46%
W: maj , RC: maj , RP: secondary	394993.7	X	2.60%
W: maj , RC: snapshot , RP: secondary	395646.5	X	2.77%
W: 1 , RC: local , RP: primary	396385.2		2.95%
W: 1 , RC: majority , RP: primary	400797.9		4.02%
W: 1 , RC: snapshot , RP: primary	400998.8		4.06%
W: 1 , RC: linearizable , RP: primary	401253.2		4.12%
W: maj , RC: local , RP: primary	402992.0		4.54%
W: maj , RC: maj , RP: primary	403519.9		4.66%
W: maj , RC: snapshot , RP: primary	403774.0		4.72%
W: maj , RC: linearizable , RP: primary	403817.5		4.73%
W: 3 , RC: local , RP: secondary	410597.4	X	6.31%
W: 3 , RC: maj , RP: secondary	410719.4	X	6.33%
W: 3 , RC: snapshot , RP: secondary	414328.0	X	7.15%
W: 3 , RC: local , RP: primary	416585.7		7.65%
W: 3 , RC: majority , RP: primary	420838.7		8.59%
W: 3 , RC: snapshot , RP: primary	423971.6		9.26%
W: 3 , RC: linearizable , RP: primary	425501.2		9.59%

Fonte: Próprio autor

Os dados mostram que parte das combinações com valores de parâmetros mais frágeis de consistência obtiverem melhores resultados no quesito desempenho. Entretanto, também é mostrado que são combinações de valores que não garantem a consistência dos dados.

Outras combinações tais como *W:3, RC: local, RP: secondary*, *W:3, RC: majority, RP: secondary* e *W:3, RC: snapshot, RP: secondary* possuem parâmetros que ajudam a

diminuir a existência de inconsistência nos dados, o que também acarreta na diminuição da performance, entretanto também não conseguiram garantir a integridade total do BD ao fim dos experimentos. Isso acontece devido ao fato da preferência de leitura *(i.e., read preference*) nas operações da carga de trabalho estar definida como *secondary*. Ou seja, apesar de possuir maiores garantias na escrita, isso não é o suficiente para a garantia de dados consistentes em um ambiente em *cluster*, corroborando também com os experimentos apresentados pela pesquisa feita em Huang et al. (2019) que mostram a existência de inconsistências quando *read preference* é definido como *secondary*.

Capítulo 6

Avaliação Experimental do Mongochain

Este capítulo apresenta de forma mais detalhada como foi realizada a integração do *framework* Mongochain ao *benchmark* YCSB+T na seção 6.1 (Integração do Mongochain ao YCSB+T), a infraestrutura física utilizada para os experimentos desta pesquisa em conjunto com o método de coleta de dados para a avaliação da corretude do Mongochain com relação à consistência na seção 6.2 (Infraestrutura Física do Ambiente e Metodologia de Coleta de Dados) e, por fim, os resultados dos testes de consistência e desempenho são exibidos, respectivamente, nas seções 6.3 (Avaliação de Consistência do Mongochain) e 6.4 (Avaliação de Desempenho do Mongochain).

6.1 Integração do Mongochain ao YCSB+T

Para que a realização dos testes fosse possível, foi necessária a integração do Mongochain ao *benchmark* YCSB+T. Essa motivação se dá porque o *framework* não pertence ao rol de bases de dados catalogadas pelo *benchmark* em sua documentação e o seu uso é fundamental para que os dados experimentais sejam gerados. Essa atividade era um dos desafios desta pesquisa, considerando o fato de se estar trabalhando com duas tecnologias distintas (*i.e.*, linguagens de programação Java e Javascript), cada uma com suas particularidades e formas de implementação diferentes.

Pelo fato do YCSB+T ser uma extensão do YCSB, o protocolo adotado para a integração de uma nova interface de base de dados ao *benchmark* segue os padrões encontrados no YCSB. A princípio, em sua implementação, o YCSB possui um pacote onde inclui a codificação das classes referentes às interfaces de BD disponíveis para testes. Essa im-

plementação possui uma classe abstrata denominada DB, a qual possui as propriedades e métodos de manipulação de dados (*e.g., insert, read, update, delete*) a serem implementados pelas interfaces de BD herdeiras. A partir dessas premissas, uma nova classe dentro do pacote de interfaces de BD denominada *MongochainClient* foi criada. Nas figuras 21 e 22 são ilustrados trechos das implementações encontrados nas classes DB e *MongochainClient*.

Figura 21 – Implementação da classe abstrata DB do YCSB

```
public abstract class DB {

    /**
     * Initialize any state for this DB.
     */
    public void init() throws DBException{ }

    /**
     * Cleanup any state for this DB.
     */
    public void cleanup() throws DBException{ }


    public abstract Status read(String table, String key, Set<String> fields, HashMap<String,ByteIterator> result

    public abstract Status scan(String table, String startkey, int recordcount, Set<String> fields, Vector<HashMa

    public abstract Status update(String table, String key, HashMap<String,ByteIterator> values);

    public abstract Status insert(String table, String key, HashMap<String,ByteIterator> values);

    public abstract Status delete(String table, String key);
}
```

Fonte: Próprio Autor

Com a classe *MongochainClient* criada, a implementação dos métodos de acesso ao *framework* foi codificada de acordo com as chamadas de API encontradas no Mongochain. Sendo assim, torna-se possível a comunicação do *benchmark* com o Mongochain para a execução dos experimentos.

6.2 Infraestrutura Física do Ambiente e Metodologia de Coleta de Dados

A análise experimental foi feita utilizando o cliente YCSB+T na versão 0.17.0 em uma máquina física: Core i7 9750H, CPU 2.60GHz, 16GB de RAM, HD de 1 TB e SSD de 128 GB. O sistema operacional usado foi o Windows 10 Home de 64 bits. Para a execução dos testes, foi adotado um *cluster* na plataforma em nuvem gratuita do MongoDB, MongoDB Atlas, no modo *Replica Set* possuindo 3 instâncias *M0*, as quais possuem seus recursos computacionais compartilhados com outros usuários da infraestrutura em nuvem, sendo elas um nó primário e dois nós secundários.

Figura 22 – Trechos da implementação da classe *MongochainClient* no YCSB+T

```
public class MongochainClient extends DB {
    /** A singleton Mongochain API instance. */
    private static ApiMongochain mongochain;
    private static final AtomicInteger INIT_COUNT = new AtomicInteger(0);

    @Override
    public void init() throws DBException {
        INIT_COUNT.incrementAndGet();

        if(mongochain != null){
            return;
        }

        mongochain = new ApiMongochain();
    }
```

(a) *MongochainClient extends* DB.

```
@Override
public Status read(String table, String key, Set<String> fields, HashMap<String, Integer> result) {
    return mongochain.readAccount(key, result);
}


@Override
public Status update(String table, String key, HashMap<String, Integer> values) {
    return mongochain.updateAccount(key, values);
}

@Override
public Status insert(String table, String key, HashMap<String, Integer> values) {
    try {
        return mongochain.insertNewAccount(key, values);

    } catch (Exception e) {
        e.printStackTrace();
        return Status.ERROR;
    }
}
```

(b) Implementação dos métodos *read, update e insert.*

Fonte: Próprio Autor

Inicialmente, para os testes de consistência, foram carregados 1.000 registros gerados pelo arquivo de propriedades padrão do YCSB+T utilizando a carga de trabalho de economia fechada (*i.e.*, *Closed Economy Workload*). Essa etapa de carregamento gerou as 1.000 contas com seus respectivos saldos simulando a economia fechada que é retratada pelo *benchmark*.

Após a fase inicial de carregamento das contas, o ambiente fica preparado para que as operações subsequentes da fase de transação sejam realizadas.

6.3 Avaliação de Consistência do Mongochain

Esta bateria de testes tem por objetivo verificar a corretude do Mongochain com relação à consistência e validar a hipótese de pesquisa de que esse *framework* consegue manter

a integridade dos dados em sua implementação original. Para isso, foram realizadas 10 execuções do *benchmark* tendo o *framework* Mongochain como alvo.

A camada de consistência do YCSB+T realiza operações na base de dados simulando uma economia fechada. Dessa forma, são realizadas operações durante a execução do *benchmark* que simulam transferências de valores monetários entre contas. Ao fim do experimento, o YCSB+T informa se a base de dados escolhida para o teste (*i.e.*, Mongochain) se encontra dentro ou fora de seu estado consistente, considerando que as somas totais dos valores iniciais e finais das contas devem ser as mesmas. Isso é feito através da etapa de validação ao fim de cada um dos testes realizados pelo *benchmark*.

O Mongochain foi testado de acordo com sua implementação padrão. Ao fim das 10 execuções realizadas pelo *benchmark*, o Mongochain apresentou um estado consistente da base de dados em 100 % dos experimentos. Na figura 23, é exibido um trecho de um dos testes realizados que vai ao encontro da hipótese de pesquisa, mostrando que o Mongochain passou pela avaliação sem apresentar inconsistência na base de dados ao fim das baterias de testes executadas pelo *benchmark*. Isso é representado com o parecer positivo do *benchmark* (*i.e., Database validation succeeded*) mostrando que a base de dados foi testada e passou com sucesso pela fase de validação. Ou seja, ao término do experimento, a soma total dos valores de todas as contas continuou com a mesma quantidade a qual estava no início. Isso significa que a economia fechada permaneceu íntegra, não constando acréscimos e nem decréscimos de valores.

6.4 Avaliação de Desempenho do Mongochain

Em relação à avaliação de desempenho do *framework* Mongochain frente às diversas cargas de trabalho da camada de desempenho do *benchmark* (*i.e., Core Workload*), as baterias de testes são menores quando colocadas ao lado dos experimentos realizados com o SGBD MongoDB. Isso acontece devido ao fato de que o Mongochain possui a sua implementação original de parâmetros (*i.e., write concern, read concern* e *read preference*) das operações. Dessa forma, diferentemente do grande número de combinações utilizdas no MongoDB, os experimentos em cada carga de trabalho foram focados na originalidade da implementação do Mongochain.

Para cada uma das cargas de trabalho (*i.e., workloads* A até F), foram realizadas 5 baterias de testes, totalizando a execução de 30 baterias de testes ao final. Para a verificação de desempenho foi utilizada a métrica de tempo de execução (*i.e., runtime*) fornecida pelo *benchmark*.

Com esses dados coletados, foi possível verificar a performance do Mongochain em cada carga de trabalho e também avaliar a sobrecarga existente no desempenho quando relacionado aos experimentos realizados no SGBD MongoDB na seção 6.2 (Avaliação de Desempenho do SGBD MongoDB). Os resultados provenientes de algumas das cargas

Figura 23 – Teste de consistência do Mongochain através do *benchmark* YCSB+T

```
Database validation succeeded

[OVERALL], RunTime(ms), 641523.0
[OVERALL], Throughput(ops/sec), 1.5587905655759808
[TX-READMODIFYWRITE], Operations, 506.0
[TX-READMODIFYWRITE], AverageLatency(us), 1097405.7233201582
[TX-READMODIFYWRITE], MinLatency(us), 913408.0
[TX-READMODIFYWRITE], MaxLatency(us), 3932159.0
[TX-READMODIFYWRITE], 95thPercentileLatency(us), 1809407.0
[TX-READMODIFYWRITE], 99thPercentileLatency(us), 2891775.0
[TX-READMODIFYWRITE], Return=OK, 506
[TX-READ], Operations, 494.0
[TX-READ], AverageLatency(us), 174488.35627530364
[TX-READ], MinLatency(us), 150656.0
[TX-READ], MaxLatency(us), 599039.0
[TX-READ], 95thPercentileLatency(us), 205567.0
[TX-READ], 99thPercentileLatency(us), 559615.0
[TX-READ], Return=OK, 494
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 13:32 min
```

Fonte: Próprio Autor

de trabalho que mostram a diferença de desempenho entre Mongochain e MongoDB são exibidos na figura 24.

Figura 24 – Diferenças de desempenho entre Mongochain e MongoDB

Workload A			
	MongoDB (ms)	Mongochain (ms)	Diferença de desempenho
Pior desempenho	164777.4	244392.2	**32.58%**
Melhor desempenho	142689.6	244392.2	**41.61%**

Workload B			
	MongoDB (ms)	Mongochain (ms)	Diferença de desempenho
Pior desempenho	168834.2	221453.2	**31.74%**
Melhor desempenho	151185.4	221453.2	**38.88%**

Workload C			
	MongoDB (ms)	Mongochain (ms)	Diferença de desempenho
Pior desempenho	162111.2	229193.17	**29.27%**
Melhor desempenho	149438.8	229193.17	**34.80%**

Workload D			
	MongoDB (ms)	Mongochain (ms)	Diferença de desempenho
Pior desempenho	163508.4	242502.33	**32.57%**
Melhor desempenho	149688.8	242502.33	**38.27%**

Fonte: Próprio Autor

Esses resultados atingem outro objetivo desta pesquisa que está relacionado à verificação da sobrecarga de desempenho do *framework* quando posto ao lado das combinações de parâmetros do MongoDB que fornecem maior consistência (*i.e.,* combinações que mostraram ter os desempenhos inferiores). A ideia de focar a comparação dos resultados do Mongochain com o que o MongoDB oferece de maior consistência é devido ao fato da

proposta do Mongochain de garantir a consistência dos dados. Dessa forma, é conveniente entender qual é a sobrecarga presente que o Mongochain apresenta quando é deparado com o MongoDB em sua combinação mais forte de consistência.

Sendo assim, os resultados encontrados entre todas as cargas de trabalho mostram que, no pior caso, houve uma sobrecarga de 37,52 % no desempenho do Mongochain, o que significa que dependendo do contexto da aplicação essa sobrecarga ainda permite a utilização do *framework*, considerando ainda que ele funciona trazendo os benefícios de uma *blockchain* e que garante a consistência dos dados. Já quando é colocado ao lado dos melhores desempenhos do MongoDB essa diferença de desempenho aumenta, entretanto, não é uma relação interessante considerando que os experimentos com o MongoDB mostraram que as combinações de testes que possuem os melhores desempenhos não garantem a consistência dos dados.

Capítulo 7

Conclusão

Esta pesquisa teve como um de seus objetivos esclarecer questões a respeito dos níveis de consistência encontrados no SGBD MongoDB, os quais são utilizados pelos desenvolvedores para manipular o desempenho e a garantia de consistência que pode ser oferecida na base de dados. Essa configuração é realizada através de alguns parâmetros (*e.g.*, *write concern, read concern* e *read preference*) que podem ser passados no momento em que as operações de banco de dados são realizadas. A partir disso, o *benchmark* YCSB+T, através de suas cargas de trabalho predefinidas, foi utilizado para realizar os experimentos e levantamento de dados para as avaliações de desempenho e de garantia de consistência desses níveis de consistência encontrados neste SGBD NoSQL.

Em adição aos experimentos realizados no MongoDB, também foi realizada a integração do *framework* Mongochain com o YCSB+T, possibilitando, dessa forma, verificar o seu desempenho e sua corretude com relação à consistência de dados. Tais tarefas permitiram verificar de maneira padronizada (*i.e.*, utilizando as cargas de trabalho do YCSB+T) a diferença de desempenho existente entre o SGBD MongoDB e o *framework* Mongochain. Com esses experimentos realizados, foi possível verificar que o Mongochain possui uma sobrecarga natural de desempenho frente ao MongoDB, entretanto, apresentou um desempenho aceitável considerando que funciona trazendo os benefícios encontrados em uma *blockchain* e que também garante a consistência dos dados.

7.1 Contribuições

O SGBD MongoDB possui uma grande variedade de níveis de consistência que podem ser configurados para que os desenvolvedores de *software* tomem a melhor decisão no momento de escolher as regras mais adequadas para a sua aplicação. Dessa forma, esta

pesquisa de Mestrado teve como um de seus objetivos realizar uma análise crítica em cima desses níveis para um maior esclarecimento com relação as suas performances, o que facilita aos desenvolvedores de *software* uma melhor tomada de decisão em relação ao desempenho e consistência de dados para a construção de suas aplicações.

Além disso, outro objetivo desta pesquisa era o de validar a proposta do Mongochain verificando se é possível que este *framework* funcione mantendo a preservação da consistência dos dados.

Com esses objetivos atingidos, foi possível realizar as seguintes contribuições nesta pesquisa:

- ❏ Analisar o desempenho dos níveis de consistência do SGBD NoSQL MongoDB;
- ❏ Verificar quais níveis de consistência do SGBD NoSQL MongoDB conseguem preservar a consistência dos dados;
- ❏ Realizar a integração do *framework* Mongochain ao *benchmark* YCSB+T;
- ❏ Comparar o desempenho do SGBD NoSQL MongoDB e do *framework* Mongochain em termos de tempo de execução;
- ❏ Validar a proposta do Mongochain verificando a corretude do *framework* no que se refere à consistência dos dados; e
- ❏ Publicar um artigo no evento International Conference on Information Technology - New Generations (ITNG) sobre o desempenho do SGBD MongoDB e seus níveis de consistência (MORCELI; TIMES; CIFERRI, 2023).
- ❏ Disponibilizar as produções técnicas de código-fonte no gerenciador de repositório de *software* Gitlab[1].

7.2 Trabalhos Futuros

Os experimentos realizados nesta pesquisa foram feitos utilizando a infraestrutura em nuvem oferecida pelo MongoDB, MongoDB Atlas. Tal infraestrutura disponibiliza recursos grátis, porém limitados, como por exemplo o número de nós no *cluster*. Diante disso, recomenda-se a realização de mais testes com um maior número de nós, dando a possibilidade de se investigar quais os impactos relacionados às questões de desempenho e consistência podem ocorrer. Além disso, também é interessante realizar os experimentos em um ambiente dedicado, porém também é necessário se atentar aos custos relacionados a um melhor ambiente para os experimentos.

[1] https://gitlab.com/morcelicaio/experimento-mongodb-mongochain

Relacionado ao *benchmark*, é possível realizar mais experimentos alterando o volume de operações das cargas de trabalho utilizadas nesta pesquisa para verificar o impacto causado na base de dados relativo ao desempenho. Por fim, também é possível aumentar o número de nós na *blockchain* do Mongochain, verificando as mudanças que podem ocorrer em relação à performance do *framework*.

Referências

ABADI, D. Consistency tradeoffs in modern distributed database system design: Cap is only part of the story. **Computer**, v. 45, n. 2, p. 37–42, 2012.

ALOMARI, M. et al. The cost of serializability on platforms that use snapshot isolation. In: **2008 IEEE 24th International Conference on Data Engineering**. [S.l.: s.n.], 2008. p. 576–585.

ANTONOPOULOS, A. M. **Mastering Bitcoin: Programming the Open Blockchain**. 2nd. ed. [S.l.]: O'Reilly Media, Inc., 2017. ISBN 1491954388.

ARAUJO, J. M. A. et al. Comparative performance analysis of nosql cassandra and mongodb databases. In: **2021 16th Iberian Conference on Information Systems and Technologies (CISTI)**. [S.l.: s.n.], 2021. p. 1–6.

BERENSON, H. et al. A critique of ansi sql isolation levels. **SIGMOD Rec.**, Association for Computing Machinery, New York, NY, USA, v. 24, n. 2, p. 1–10, may 1995. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/568271.223785>.

BOG, A. **Benchmarking Transaction and Analytical Processing Systems: The Creation of a Mixed Workload Benchmark and Its Application.** [S.l.]: Springer Publishing Company, Incorporated, 2013.

BREWER, E. Cap twelve years later: How the "rules"have changed. **Computer**, v. 45, n. 2, p. 23–29, 2012.

CHOWDHURY, M. J. M. et al. Blockchain versus database: A critical analysis. In: **2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)**. [S.l.: s.n.], 2018. p. 1348–1353.

CIFERRI, R. R. **Um benchmark voltado a analise de desempenho de sistemas de informações geograficas**. Dissertação (Mestrado) — Universidade Estadual de Campinas, Instituto de Matematica, Estatistica e Ciencia da Computação, Campinas, Março 1995.

CODD, E. F. A relational model of data for large shared data banks. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 13, n. 6, p. 377–387, jun 1970. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/362384.362685>.

COOPER, B. F. et al. Benchmarking cloud serving systems with ycsb. In: **Proceedings of the 1st ACM Symposium on Cloud Computing**. New York, NY, USA: Association for Computing Machinery, 2010. (SoCC '10), p. 143–154. ISBN 9781450300360. Disponível em: <https://doi.org/10.1145/1807128.1807152>.

CORONEL, C.; MORRIS, S. **Database Systems: Design, Implementation, and Management**. 13th. ed. Boston, MA, USA: Cengage Learning, 2017. ISBN 9781337627900.

COULOURIS, G. et al. **Distributed Systems: Concepts and Design**. 5th. ed. USA: Addison-Wesley Publishing Company, 2011. ISBN 0132143011.

DATE, C. **An Introduction to Database Systems**. 8. ed. USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0321197844.

DEY, A. et al. Ycsb+t: Benchmarking web-scale transactional databases. In: **2014 IEEE 30th International Conference on Data Engineering Workshops**. [S.l.: s.n.], 2014. p. 223–230.

ELMASRI, R.; NAVATHE, S. **Fundamentals of Database Systems**. 6th. ed. USA: Pearson, 2011. ISBN 1484248465.

ELROM, E. **The Blockchain Developer: A Practical Guide for Designing, Implementing, Publishing, Testing, and Securing Distributed Blockchain-Based Projects**. 1st. ed. USA: Apress, 2019. ISBN 13:978-0-136-08620-8.

FOWLER, M. **Patterns of Enterprise Application Architecture**. USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321127420.

FOX, A.; BREWER, E. Harvest, yield, and scalable tolerant systems. In: **Proceedings of the Seventh Workshop on Hot Topics in Operating Systems**. [S.l.: s.n.], 1999. p. 174–178.

FOX, A. et al. Cluster-based scalable network services. **SIGOPS Oper. Syst. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 31, n. 5, p. 78–91, oct 1997. ISSN 0163-5980. Disponível em: <https://doi.org/10.1145/269005.266662>.

GAI, R. et al. A summary of the research on the foundation and application of blockchain technology. **Journal of Physics: Conference Series**, IOP Publishing, v. 1693, n. 1, p. 012025, dec 2020. Disponível em: <https://dx.doi.org/10.1088/1742-6596/1693/1/012025>.

GONZáLEZ-APARICIO, M. T. et al. Transaction processing in consistency-aware user's applications deployed on nosql databases. **Hum.-Centric Comput. Inf. Sci.**, Springer-Verlag, Berlin, Heidelberg, v. 7, n. 1, dec 2017. ISSN 2192-1962. Disponível em: <https://doi.org/10.1186/s13673-017-0088-3>.

HARRISON, G.; HARRISON, M. **MongoDB Performance Tuning: Optimizing MongoDB Databases and their Applications**. USA: Apress Berkeley, CA, 2021. ISBN 978-1484268780.

HECHT, R.; JABLONSKI, S. Nosql evaluation: A use case oriented survey. In: **2011 International Conference on Cloud and Service Computing**. [S.l.: s.n.], 2011. p. 336–341.

HUANG, C. et al. Data consistency properties of document store as a service (dsaas): Using mongodb atlas as an example. In: NAMBIAR, R.; POESS, M. (Ed.). **Performance Evaluation and Benchmarking for the Era of Artificial Intelligence**. Cham: Springer International Publishing, 2019. p. 126–139. ISBN 978-3-030-11404-6.

INDRAWAN-SANTIAGO, M. Database research: Are we at a crossroad? reflection on nosql. In: **2012 15th International Conference on Network-Based Information Systems**. [S.l.: s.n.], 2012. p. 45–51.

KAMSKY, A. Adapting tpc-c benchmark to measure performance of multi-document transactions in mongodb. **Proc. VLDB Endow.**, VLDB Endowment, v. 12, n. 12, p. 2254–2262, aug 2019. ISSN 2150-8097. Disponível em: <https://doi.org/10.14778/3352063.3352140>.

KLEPPMANN, M. **Designing Data-Intensive Applications**: The big ideas behind reliable, scalable, and maintainable systems. Sebastopol, CA, USA: O'Reilly Media, Inc., 2017. ISBN 978-1449373320.

LAURENCE, T. **Blockchain For Dummies**. [S.l.]: For Dummies, 2017. ISBN 1119365597.

LERMAN, J. **Data Points - What the Heck Are Document Databases?** 2011. Disponível em: <http://msdn.microsoft.com/en-us/magazine/hh547103.aspx.> Acesso em: 16/09/2022.

MARTINS, P. et al. Mongodb, couchbase, and couchdb: A comparison. In: **Trends and Applications in Information Systems and Technologies**. [S.l.]: Springer International Publishing, 2021. p. 469–480. ISBN 978-3-030-72651-5.

MATALLAH, H.; BELALEM, G.; BOUAMRANE, K. Experimental comparative study of nosql databases: Hbase versus mongodb by ycsb. **Computer Systems Science and Engineering**, v. 32, p. 307–317, 07 2017.

_____. Evaluation of nosql databases: Mongodb, cassandra, hbase, redis, couchbase, orientdb. **International Journal of Software Science and Computational Intelligence**, v. 12, p. 71–91, 10 2020.

MCCREARY, D.; KELLY, A. **Making Sense of NoSQL: A GUIDE FOR MANAGERS AND THE REST OF US**. [S.l.]: Manning Publication, 2014. ISBN 9781617291074.

MEIER, A.; KAUFMANN, M. **SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management**. [S.l.: s.n.], 2019. ISBN 978-3-658-24548-1.

MENASCÉ, D. A.; NAKANISHI, T. Optimistic versus pessimistic concurrency control mechanisms in database management systems. **Information systems**, Elsevier, v. 7, n. 1, p. 13–27, 1982.

MONGODB. **Read Preference**. 2022. Disponível em: <https://www.mongodb.com/docs/manual/core/read-preference/>. Acesso em: 25/03/2022.

_____. **Replication**. 2022. Disponível em: <https://www.mongodb.com/docs/manual/replication/>. Acesso em: 25/03/2022.

_____. **Transactions**. 2022. Disponível em: <https://www.mongodb.com/docs/manual/core/transactions/>. Acesso em: 24/03/2022.

MORCELI, C. L.; TIMES, V. C.; CIFERRI, R. R. A performance analysis of different mongodb consistency levels. In: LATIFI, S. (Ed.). **ITNG 2023 20th International Conference on Information Technology-New Generations**. Cham: Springer International Publishing, 2023. p. 395–401. ISBN 978-3-031-28332-1.

MUZAMMAL, M.; QU, Q.; NASRULIN, B. Renovating blockchain with distributed databases: An open source system. **Future generation computer systems**, Elsevier, v. 90, p. 105–117, 2019.

NAKAMOTO, S. A peer-to-peer electronic cash system. 2008. Disponível em: <https://bitcoin.org/bitcoin.pdf>. Acesso em: 10/10/2022.

OUYANG, H.; WEI, H.; HUANG, Y. Checking causal consistency of mongodb. In: **Proceedings of the 12th Asia-Pacific Symposium on Internetware**. New York, NY, USA: Association for Computing Machinery, 2021. (Internetware '20), p. 209–216. ISBN 9781450388191. Disponível em: <https://doi.org/10.1145/3457913.3457928>.

ÖZSU, M. T.; VALDURIEZ, P. **Principles of Distributed Database Systems**. 3rd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2011. ISBN 1441988335.

PRITCHETT, D. Base: An acid alternative: In partitioned databases, trading some consistency for availability can lead to dramatic improvements in scalability. **Queue**, Association for Computing Machinery, New York, NY, USA, v. 6, n. 3, p. 48–55, may 2008. ISSN 1542-7730. Disponível em: <https://doi.org/10.1145/1394127.1394128>.

RAMAKRISHNAN, R.; GEHRKE, J. **Database Management Systems**. 3rd. ed. [S.l.]: McGraw-Hill Higher Education, 2003. ISBN 978-0072465631.

RAVAL, S. **Decentralized applications: harnessing Bitcoin's blockchain technology**. [S.l.]: O'Reilly Media, Inc., 2016. ISBN 9781491924549.

SADALAGE, P. J.; FOWLER, M. **NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence**. [S.l.]: Pearson Education, 2013. ISBN 978-0-321-82662-6.

SCHULTZ, W.; AVITABILE, T.; CABRAL, A. Tunable consistency in mongodb. **Proc. VLDB Endow.**, VLDB Endowment, v. 12, n. 12, p. 2071–2081, aug 2019. ISSN 2150-8097. Disponível em: <https://doi.org/10.14778/3352063.3352125>.

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.

SHARMA, V.; DAVE, M. Sql and nosql databases. **International Journal of Advanced Research in Computer Science and Software Engineering**, v. 2, p. 20–27, 2012.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. **Database System Concepts**. 5th. ed. [S.l.]: Elsevier, 2006. ISBN 978-0-07-295886-7.

SINGHAL, B.; DHAMEJA, G.; PANDA, P. **Beginning Blockchain**. [S.l.]: Apress, 2018. ISBN 978-1-4842-3443-3.

SOUSA, C. M. V. **MONGOCHAIN**: um framework para implementação de sistemas transacionais. Dissertação (Mestrado) — Universidade Federal de Pernambuco, Centro de Informática, Recife, 2020.

STACKOVERFLOW. **Most popular technologies**. 2023. Disponível em: <https://survey.stackoverflow.co/2023/#most-popular-technologies-database-prof>. Acesso em: 07/02/2023.

STONEBRAKER, M. In search of database consistency. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 10, p. 8–9, oct 2010. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1831407.1831411>.

STROZZI, C. **NoSQL – A relational database management system**. 1998. Disponível em: <http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/>. Acesso em: 14/09/2022.

TANENBAUM, A. S.; STEEN, M. v. **Distributed Systems: Principles and Paradigms (2nd Edition)**. 2nd. ed. [S.l.]: Prentice-Hall, Inc., 2007. ISBN 0132392275.

TIWARI, S. **Professional NoSQL**. GBR: Wrox Press Ltd., 2011. ISBN 9780470942246.

ULRICH, F. **Bitcoin - A moeda na era digital**. [S.l.]: LVM Editora, 2014. ISBN 978-8581190761.

VOGELS, W. Eventually consistent. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 52, n. 1, p. 40–44, jan 2009. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1435417.1435432>.