CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
Recredenciado pela Portaria Ministerial nº 1.162, de 13/10/16, D.O.U nº 198, de 14/10/2016
ASSOCIAÇÃO EDUCACIONAL LUTERANA DO BRASIL

Icaro José Bilinski

DESENVOLVIMENTO DE UMA ESTRUTURA DE ARMAZENAMENTO DE DADOS DE ALTO DESEMPENHO PARA UMA PLATAFORMA DE ANÁLISE E MANIPULAÇÃO DE DADOS

Palmas – TO
2017

Icaro José Bilinski

DESENVOLVIMENTO DE UMA ESTRUTURA DE ARMAZENAMENTO DE DADOS DE ALTO DESEMPENHO PARA UMA PLATAFORMA DE ANÁLISE E MANIPULAÇÃO DE DADOS

Trabalho de Conclusão de Curso (TCC) II elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação / Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Jackson Gomes de Souza.

Palmas – TO
2017

Icaro José Bilinski

DESENVOLVIMENTO DE UMA ESTRUTURA DE ARMAZENAMENTO DE DADOS DE ALTO DESEMPENHO PARA UMA PLATAFORMA DE ANÁLISE E MANIPULAÇÃO DE DADOS

Trabalho de Conclusão de Curso (TCC) II elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação / Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Jackson Gomes de Souza

Aprovado em: 22/06/2017

BANCA EXAMINADORA

_____

Prof. M. Jackson Gomes de Souza
Orientador
Centro Universitário Luterano de Palmas – CEULP

_____

Prof.ª M.ScMadianitaBogo
Centro Universitário Luterano de Palmas – CEULP

_____

Prof.ª M.ScParcilene Fernandes de Brito
Centro Universitário Luterano de Palmas – CEULP

Palmas – TO
2017

RESUMO

Este trabalho aborda os conceitos de Banco de Dados *NoSQL* de alto desempenho para armazenamento de dados para um sistema de análise e manipulação. Propõe também implementar a camada de persistência de dados com o banco de dados *MongoDB* para garantir o alto desempenho com a distribuição dos dados. Este trabalho concebe uma estrutura de armazenamento de dados de alto desempenho para uma plataforma de análise e manipulação de dados fazendo parte de um projeto do grupo de pesquisa de Engenharia Inteligente de Dados do CEULP/ULBRA.

PALAVRAS-CHAVE: Banco de Dados, NoSQL, alto desempenho, análise e manipulação de dados.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE COMANDOS

LISTA DE ABREVIATURAS E SIGLAS

GNU	Licença pública geral
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
SQL	Linguagem de Consulta Estruturada
NoSQL	Não apenas SQL
CAP	Consistência, Disponibilidade e Tolerância a falhas
API	Interface de Programação de Aplicativos
CRUD	Criação, Consulta, Atualização e Destruição de dados
SGBD	Sistemas Gerenciadores de Banco de Dados
BASE	Basicamente disponível, estado leve e eventualmente consistente.
XML	Linguagem de marcação extensível
JSON	Notação de Objeto de *JavaScript*
BSON	JSON binário
MVCC	Controle de concorrência multiversão.
URL	Localizador Padrão de Recursos

SUMÁRIO

1 INTRODUÇÃO

A popularização e a evolução dos serviços da internet aumentaram significativamente nas últimas décadas e pessoas, empresas e organizações produzem agora maiores quantidades de dados. Surgem assim, maiores dificuldades de gerenciamento destas informações através de bancos de dados relacionais, originando a necessidade de criação de novos protótipos e tecnologias para aperfeiçoar o desempenho, a disponibilidade e escalabilidade dos métodos de persistência dos dados.

O SGDB (Sistema de Gerenciamento de Bancos de Dados) com maior popularidade é o relacional (DB-ENGINES, 2017), que faz uso de estrutura de dados em tabelas que contêm linhas e colunas de dados (CALDEIRA et al. 2004 p.34). O SGBD relacional possui identificadores únicos e transações que são: ações de leitura e escrita das tabelas para inserir, remover e alterar dados. Todavia, o modelo relacional não apresenta facilidades para expandir as capacidades de armazenamento e desempenho da base de dados, pois adota o modelo de transição de dados ACID (Atomicidade, Consistência, Isolamento e Durabilidade), desfavorável para o gerenciamento de grandes volumes de dados que exigem agregação de recursos computacionais (LÓSCIO;OLIVEIRA;PONTES,2011).

Assim, empresas de tecnologia, especializada no desenvolvimento de banco de dados, criaram novas formas de persistência de dados. Para melhor atender aplicações que exigem grandes volumes de dados descentralizados e não estruturados. Estas inovações foram publicadas em artigos e disponibilizadas através protótipos sob licença livre para comunidade de desenvolvedores e usuários, e ajudaram a definir características para o novo modelo de banco de dados, tais como: modelagem de dados flexível, bases de dados distribuídas, código aberto, escalável, acesso via API (Interface de Programação de Aplicativos) e replicação nativa (DIANA; GEROSA,2012). Este conjunto de inovações foi batizado de NoSQL (não somente SQL), e não faz uso de SQL (Linguagem de Consulta Estruturada) utilizada nos bancos relacionais (STROZZI,1998). São classificados conforme a estruturação dos dados tendo como

os principais tipos: chave-valor, documentos, colunas e grafos. Alguns bancos de dados NoSQL mais conhecidos são: *MongoDB*[1], *CouchDB*[2], *Redis*[3], *Memcached*[4],*Cassandra*[5] e *Neo4J*[6].

O objetivo deste trabalho é colaborar com desenvolvimento de uma plataforma de análise e manipulação de dados que está sendo desenvolvido pelo Grupo de Pesquisa de Engenharia Inteligente de Dados do CEULP/ULBRA e tem como meta desenvolver uma estrutura de armazenamento de dados de alto desempenho para uma plataforma de análise e manipulação de dados.

[1]https://www.mongodb.com/
[2]couchdb.apache.org/
[3]https://redis.io/
[4]https://memcached.org/
[5] *http://cassandra.apache.org/*
[6]https://neo4j.com/

2 REFERENCIAL TEÓRICO

Neste trabalho são apresentados os conceitos de teorema CAP (Consistência, Disponibilidade e Tolerância a falhas), as diferenças entre os modelos de transição ACID e BASE (Basicamente disponível, estado leve e eventualmente consistente), bem como o conceito de banco de dados NoSQL, suas características e tipos. E por fim o banco de dados *MongoDB*. Tais conceitos e ferramentas são necessários para o entendimento das técnicas para desenvolvimento do modelo de armazenamento de dados para a plataforma de análise e manipulação de dados.

2.1 CAP

O teorema de CAP, segundo Brewer (2012) diz que qualquer sistema de dados distribuídos, pode obter apenas duas das três propriedades, consistência, disponibilidade e tolerância a falhas.

A **consistência** garante que os dados continuem autênticos conforme a última transição executada; seu valor deve permanecer fiel para todos os usuários em tempo real, mesmo em bases distribuídas (WEI, 2009). Um exemplo de consistência ocorre na escrita de dados em memória não volátil, visto que o dado pode ser recuperado caso o sistema seja interrompido.

A **disponibilidade** garante que um sistema se mantenha disponível mesmo em caso de falhas, fornecendo acesso parcial aos dados e às funcionalidades durante um determinado período de tempo. Esta propriedade é fundamental para sistemas que disponibilizam serviços ininterruptamente de forma distribuída (BROWNE, 2009). Um exemplo de disponibilidade ocorre quando dois ou mais computadores exercem as mesmas funções, onde em caso de falha o outro assume a tarefa sem perder a conexão com o cliente.

A **tolerância a falhas** garante que um banco de dados distribuído mantenha as operações durante uma falha de rede, e outras instâncias do banco de dados adotam tal função de forma transparente para o cliente, mantendo o funcionamento da base de dados como um todo em diferentes servidores (GILBERT, 2002). Um exemplo de tolerância a falhas ocorre em nuvem computacional, onde os servidores estão distribuídos em locais distintos, assim as tarefas serão redimensionadas em caso de falhas de conexão.

As combinações entre as propriedades do teorema CAP variam conforme a necessidade da aplicação onde a base de dados distribuída será empregada. Garantir todas as propriedades ao mesmo tempo demanda aumento significativo dos custos e complexidades computacionais.

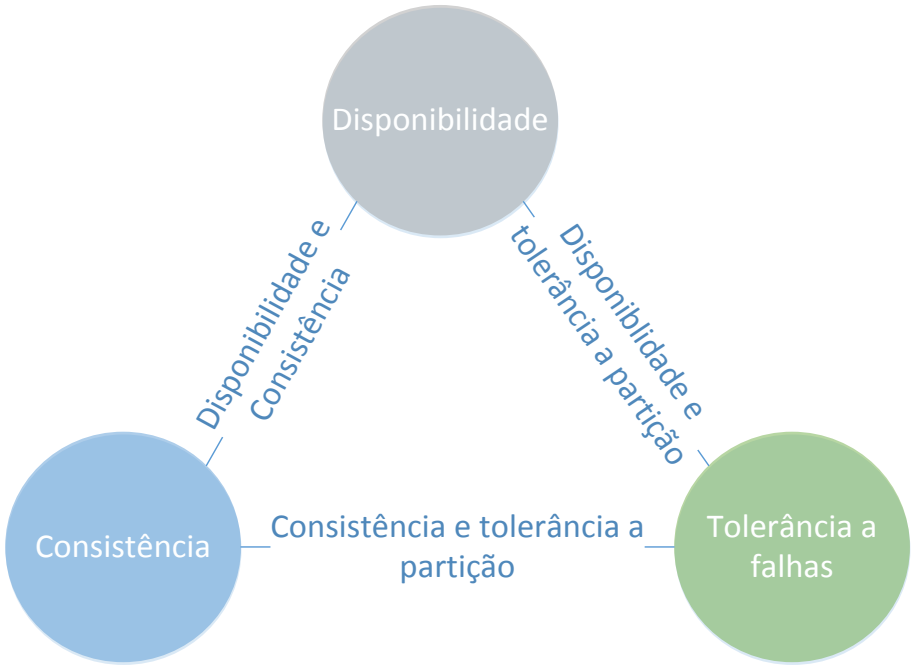
Além destas dificuldades, não há garantia de melhorias (MONSON-HAEFEL, 2009). Para implementar uma base de dados distribuída é necessário definir as três possíveis combinações das propriedades CAP, para melhor atender os requisitos da aplicação que fará uso da base de dados distribuída (POKORNY, 2013). Estas combinações são:

- **Consistência e disponibilidade**: não assevera a replicação de dados, evitando deste modo, um ponto único de falha. Em caso de falhas de conexão de rede, outra instância do banco de dados distribuído passa a atender às requisições do cliente. Esta abordagem é utilizada pelos bancos de dados relacionais. Alguns exemplos de banco de dados que implementam esta combinação são: *MySQL*[7], *PostgresSQL*[8]e *DB2*[9] entre outros.
- **Consistência e tolerância a falhas**: não assegura a disponibilidade dos dados visto que, para manter as bases de dados sincronizadas, é indispensável alterar o estado dos dados para imutável. Logo, clientes não terão acesso à base de dados de forma íntegra. Outro fator que inviabiliza a disponibilidade é o fato de não existir transação distribuída, mas sim, protocolos de consensos que definem a consistência dos dados em base de dados tolerante a falhas. Esta abordagem é utilizada pelos bancos de dados não relacionais (STEPPAT, 2011). Alguns exemplos de banco de dados que implementam esta combinação são: *BigTable*[10], *HBase*[11] e *MongoDB* entre outros.
- **Disponibilidade e tolerância a falhas**: não garante a consistência dos dados. Todas as instâncias do banco de dados aceitam a inserção de dados que posteriormente serão sincronizadas com os demais bancos de dados distribuídos, havendo a possibilidade de conter dados inconsistentes, antes ou durante a sincronização destes. Esta abordagem é utilizada por sistemas que não podem ficar *off line* e são empregados em sistemas de arquivo distribuído, análise de dados e aprendizado de máquina. Alguns exemplos de banco de dados que implementam esta combinação são: *AmazonDynamo*[12], *Cassandra* e *Riak*[13] entre outros. A Figura 1 demonstra o modelo CAP e as relações entre suas propriedades.

---

[7] *https://www.mysql.com/*
[8] *https://www.postgresql.org/*
[9] *https://www.ibm.com/analytics/us/en/technology/db2/*
[10] https://cloud.google.com/bigtable/
[11] https://hbase.apache.org/
[12] https://aws.amazon.com/pt/dynamodb/
[13] http://basho.com/

Figura 1 - Teorema CAP



Conforme apresentado na Figura 1, os círculos representam as propriedades do modelo CAP e as arestas são possíveis combinações entre as propriedades, que garantem a distribuição das bases de dados.

2.2 ACID

Os primeiros SGBDs relacionais foram baseados em sistemas de arquivos, herdando algumas características como: controle de concorrência, segurança, recuperação de falhas e mecanismos para garantir a escrita de dados e controle das restrições de integridade (LÓSCIO; OLIVEIRA; PONTES, 2011). Estas características deram base para definir as regras de transações, que são operações de leitura e escrita realizadas no banco de dados. O SGBD deve garantir as execuções das propriedades ACID em cada transação (LEITE; BONOMO, 2016). As propriedades ACID são: atomicidade, consistência, isolamento e durabilidade.

Atomicidade garante que as transações sejam executadas como um todo, de forma atômica. Caso ocorra algum tipo de erro, tal transação não terá validade e deverá ser desfeita. A exemplo das transações bancárias onde, durante um saque, os valores só serão computados de fato quando o dinheiro é realmente transferido.

C**onsistência** assegura que as transações mantenham o estado do dado persistido no banco de dados, para garantir que um dado tenha somente um valor válido. Voltando ao

exemplo das transações bancárias, os valores deverão ser representados por um único tipo numérico.

Isolamento garante o controle de concorrência dos dados restringindo a uma única transição por vez. Esse dado permanece em um estado bloqueado para outras transições. Por exemplo, em uma transação bancária entre contas, os saldos de ambas as contas permanecerão inalteráveis para garantir que outras transações não alterem esses valores até serem totalmente concluídas.

Durabilidade, ao concluir uma transição de escrita, é responsável por fazer com que o dado permaneça salvo na base de dados e não seja perdido. É semelhante ao que ocorre nos moldes dos depósitos bancários, onde os valores acrescentados devem permanecer registrados para acessos futuros.

Nos bancos de dados relacionais que fazem uso do modelo de transação ACID, a distribuição dos dados é dificultada, pois as propriedades de consistência e isolamento restringem os dados para que não possam ter valores em diferentes nas bases de dados interligadas, isto torna o modelo restritivo para aplicações distribuídas (VIEIRA. 2012).

2.3 BASE

Segundo Abadi (2012), o modelo de transação BASE surgiu como alternativa ao modelo ACID, e define propriedades para manter os dados sempre disponíveis e consistentes em momentos apropriados. Facilita o desenvolvimento de soluções que não podem ser interrompidas durante a expansão da capacidade do banco de dados distribuídos. Alguns exemplos de aplicações são os sistemas de larga escala e nuvens computacionais. As propriedades BASE são: basicamente disponível, estado leve e consistência eventual (DOS SANTOS, 2015).

Basicamente disponível, consiste em disponibilizar os dados a todo momento. A disponibilidade é prioridade durante quaisquer requisições de dados, mesmo em caso de falha de rede, estado inconsistente ou durante alteração de dados. Assim, quando um computador for reiniciado, outro irá assumir suas tarefas automaticamente, por exemplo.

O **estado leve** permite que o sistema seja flexível e possa ser alterado em tempo de execução, de modo que não necessita ser consistente a todo o momento para possibilitar a execução de funções internas. Quando uma base de dados não estiver executando tarefas, ele pode sincronizar ou replicar os dados para outras bases de dados.

A **consistência eventual** permite que o sistema não fique bloqueado durante uma transação e continuará a receber dados sem verificar a consistência dos mesmos. Os dados se tornarão consistentes logo que a replicação e sincronização entre demais bases de dados distribuídas sejam concluídas, deste modo, o sistema torna-se consistente no momento apropriado. Em uma transição, os dados não ficarão isolados e serão criados vários valores cronológicos para tais dados.

O modelo de transação BASE não propõe regras como o modelo ACID, mas sim propriedades para projetar aplicações que permitam inconsistências temporárias quando se quer priorizar a disponibilidade (DIANA; GEROSA, 2012). Alguns bancos de dados *NoSQL* que fazem uso do modelo de transação BASE são: *MongoDB*, *Cassandra* e *Redis* entre outros.

2.4 NoSQL

O termo *NoSQL* representa o banco de dados que não segue os princípios estabelecidos pelo modelo relacional de estruturação de dados. Apresenta diferentes tecnologias de banco de dados desenvolvidas para suprir as demandas das aplicações contemporâneas.

Segundo Tiwari (2011), bancos de dados *NoSQL* são sistemas para gerenciar grandes quantidades de dados com alto desempenho através de diversas estruturas de dados, tais como documentos, gráficos, chave-valor, colunas, dentre outros.

Os Banco de Dados *NoSQL* possuem características para suportar grandes volumes de dados: escalabilidade horizontal, esquema flexível, replicação nativa, API de acesso aos dados e consistência eventual (NOSQL, 2010).

A **escalabilidade horizontal** permite que o banco de dados aumente as capacidades de processamento e armazenamento agregando mais computadores sob demanda, sem interromper o serviço. Os dados e a carga da consulta serão automaticamente distribuídos em todos os servidores, e quando um servidor fica indisponível, será substituído de forma rápida e transparente sem interromper a aplicação.

O **esquema flexível** permite a alteração dos dados sem afetar a estrutura lógica, permitindo mudanças em aplicações sem interromper o funcionamento do banco de dados. Isso torna o desenvolvimento ágil, pois a integração com o código se torna confiável. Regras de validação de dados são aplicadas diretamente no banco de dados, permitindo um maior controle sobre eles (MongoDB, 2017i).

A **replicação nativa** assegura a distribuição dos dados entre instâncias do banco de dados de forma automatizada, para gerar redundância dos dados armazenados. Existem duas

formas de replicação: mestre-escravo; onde instâncias do banco de dados mestre recebe todas as requisições de escrita e replica para os escravos atenderem as requisições de leitura; e mestre-mestre; onde todas as instâncias do banco de dados tem as mesmas atribuições e os dados são sincronizados de forma automática (TOTH, 2011).

A **API de acesso aos dados** permite o acesso aos dados por aplicações de forma rápida e eficiente através de rotinas e padrões de programação (LÓSCIO; OLIVEIRA; PONTES, 2011).

A **Consistência eventual** propicia a sincronia de todas as versões dos dados distribuídos entre réplicas. A sincronização é executada no momento propício para definir um estado válido dos dados, eventualmente, as consultas retornarão o último valor válido. Se não ocorrerem falhas, o tempo de inconsistência será determinado por atrasos de comunicação, carga no sistema e o número de réplicas. Quanto maior a quantidade de réplicas, menor será o tempo de inconsistência dos dados (VOGELS, 2008).

Para melhor aproveitar as características do modelo de distribuído dos *NoSQL* foram implementadas ferramentas e métodos para distribuir operações em paralelo para viabilizar o alto desempenho, como *MapReduce*, *Hash* consistente, controle de concorrência multiversão e relógios vetoriais.

O ***MapReduce*** consiste na composição de duas funções para processar grandes volumes de dados através da fragmentação e distribuições de processamento entre as instâncias do banco de dados. A função *Map* cria índices de resultados de uma consulta local adicionando um identificador único para cada resultado. A função *Reduce* agrega os resultados idênticos em conjuntos para receber o identificador. Um conjunto de identificadores é criado em cada instância e enviada para o cliente que invocou a função de *MapReduce (*ULLMAN, 2012).

O ***Hash* consistente** é um mecanismo de armazenamento e recuperação em dados distribuídos, onde a quantidade de dados está em constante modificação. Uma função *Hash* é executada em todos os dados redundantes. Caso uma instância do banco de dados fique indisponível, outra instância contendo o dado com o mesmo *Hash* passa a responder em seu lugar (WHITE, 2007).

O **MVCC (controle de concorrência multiversão)** é um método de controle de concorrência de acesso simultâneo aos dados. Neste, o banco de dados não exclui dados, mas substitui por uma nova versão e registra os dados antigos como obsoletos, passando a existir várias versões armazenadas, mas apenas uma é a mais recente. Permite que o banco de dados evite a sobrecarga de alocação de memória e disco e, periodicamente o banco de dados exclui dados obsoletos (GAJENDRAN, 2012).

Os **relógios vetoriais** detectam conflitos entre os mesmos dados distribuídos nos casos em que estes dados sofram atualizações cuja ordem não possa ser determinada. As versões conflitantes permanecem salvas enquanto outra instância do banco determine como resolver o conflito, ou seja, transformá-las em uma única versão combinada (POPESCU, 2010).

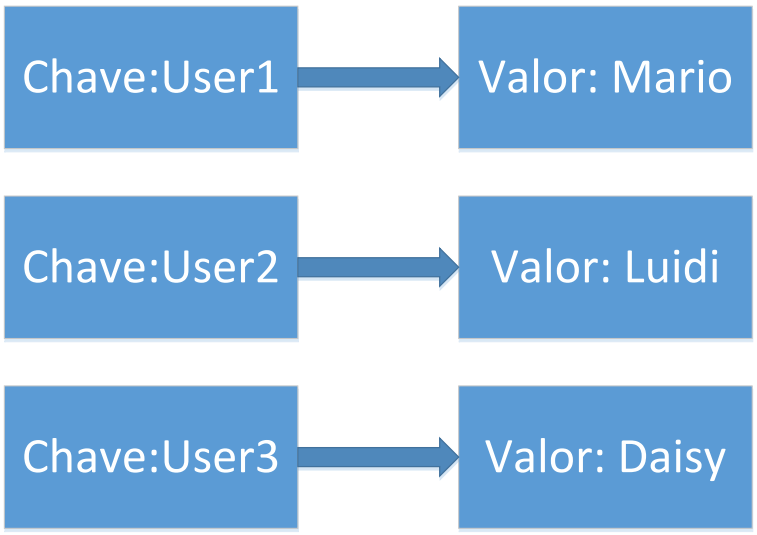
2.5 Tipos de Bancos de dados NoSQL

Existem diversos tipos de banco de dados NoSQL e todos os tipos possuem conceitos e particularidades para atender necessidades específicas. Os principais tipos são: chave-valor, orientado a colunas, orientado a documentos e orientado a grafos (LÓSCIO; OLIVEIRA; PONTES, 2011).

2.5.1 Orientado a Chave-valor

A orientação a chave-valor organiza os dados através de chaves e valores e é livre de esquemas para garantir melhor desempenho em inserções e consulta dos dados. As chaves representam um índice de acesso para um valor armazenado, enquanto que os dados são gerenciados em memória e aceitam persistência em disco. A distribuição dos dados é feita através de cluster (TIWARI, 2012). Este tipo de banco de dados é mais utilizado em sistemas que necessitam de acesso rápido, tais como: cache de sessão, cache de página inteira, aplicativos de fila de mensagens, tabelas de classificação e contagem. A **Erro! Autoreferência de indicador não válida.** demonstra modelo de chaves e valores:

Figura 2 – Modelo de tabela chave/valor



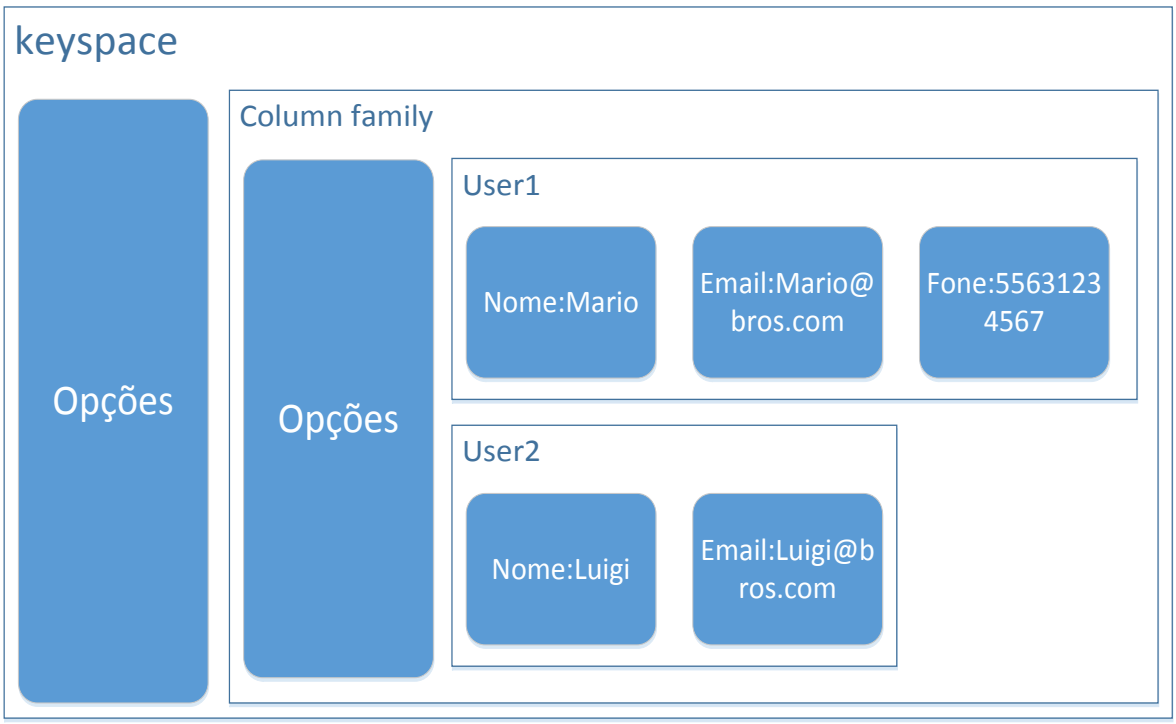
Conforme apresentado na A orientação a chave-valor organiza os dados através de chaves e valores e é livre de esquemas para garantir melhor desempenho em inserções e consulta dos dados. As chaves representam um índice de acesso para um valor armazenado, enquanto que os dados são gerenciados em memória e aceitam persistência em disco. A distribuição dos dados é feita através de cluster (TIWARI, 2012). Este tipo de banco de dados é mais utilizado em sistemas que necessitam de acesso rápido, tais como: cache de sessão, cache de página inteira, aplicativos de fila de mensagens, tabelas de classificação e contagem. A **Erro! Autoreferência de indicador não válida.** demonstra modelo de chaves e valores:

Figura 2, existem três chaves que armazenam valores para usuários de um dado sistema e cada chave armazena um valor. A chave “User1” contém o valor atribuído “Mario”, na chave “User2” está contido o valor “Luidi”, e na chave “User3”, o valor “Daisy”.

2.5.2 Orientado a Colunas

No modelo orientado a colunas os dados são indexados em linhas, colunas e *timestamp.* As linhas e colunas são identificadas por chaves e o *timestamp* é utilizado para gerenciar os vários estados de um dado. Também é possível relacionar em grupo de colunas *(column Family),* com chave para indexar colunas com tipos de dados iguais. São aplicados para armazenar e processar grandes volumes de dados de forma distribuída (SOUZA, 2014). A Figura 3 representa um modelo de colunas.

Figura 3 – Modelo de dados em colunas



Conforme apresentado na Figura 3, o *keyspace* representa uma chave única para administrar a distribuição das *column family* entre as instâncias do banco de dados distribuídas. As opções definem um nível externo de organização, como o nome da aplicação e permissões de acesso das *column family*. Cada *column family* define regras relacionais de armazenamento dos usuários e cada usuário é representado por uma chave. As opções contêm informações sobre a indexação das colunas, quais colunas estão persistidas em disco, quantidade de colunas dentre outras. A chave "User1" contém os dados: "Nome", com o valor "Mario"'; "Email", com valor "Mario@bros.com"; e "Fone", com o valor "55631234567". A chave "User2" apresenta os dados: "Nome", com o valor "Luigi" e "Email", com o valor "Luigi@bros.com".

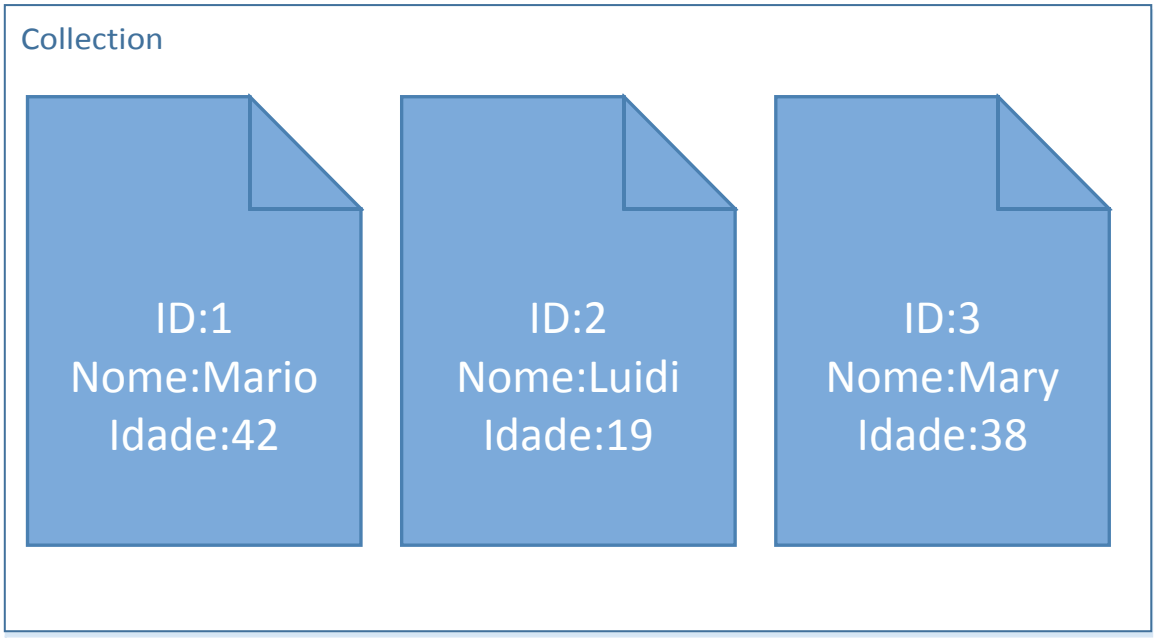
2.5.3 Orientado a Documentos

A estrutura dos dados por documentos contém uma coleção de atributos e valores referenciados por uma chave única. Podem ser adotados modelos de documentos como XML (Linguagem de marcação extensível) ou JSON (Notação de Objeto de *JavaScript*). Não há um esquema de estrutura rígida e podem ser armazenados dados semiestruturados ou não estruturados (MONIRUZZAMAN; HOSSAIN, 2013). Os dados são acessados por chave ou demais registros e um documento pode ser embutido como atributo de outro documento. A

duplicação de dados é permitida para facilitar a distribuição e acesso aos dados (LEAVITT, 2010).

Bancos de Dados NoSQL que seguem esse tipo são utilizados para solução de problemas relativos à tolerância de dados incompletos (NASCIMENTO, 2014). A Figura 4 apresenta um modelo de documento.

Figura 4 - Modelo de documento



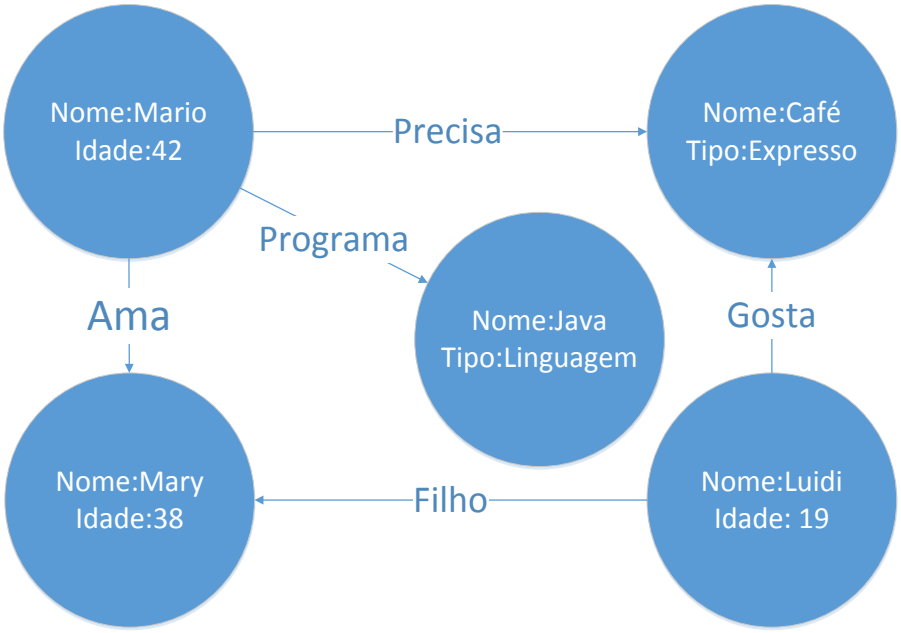
Conforme apresentado na Figura 4, uma "*collection*" armazena um conjunto de documentos de usuários, onde cada documento contém um identificador único representado pelo campo "ID". O documento de identificador 1 armazena a informação "Mario" no campo "Nome", e "42" no campo "Idade". O documento de identificador 2 armazena a informação "Luigi" no campo "Nome", e "19" no campo "Idade". Por fim, o documento de identificador 3 armazena a informação "Mary" no campo "Nome", e "38" no campo "Idade".

2.5.4 Orientado a Grafos

A estruturação orientada a grafos utiliza conceitos de grafos para organizar os dados e possui dois componentes: os nós que representam as entidades e os relacionamentos que contêm informações sobre a relação entre nós. As informações são geradas pela composição de caminhos nos grafos, assim, os dados podem ser interpretados de diferentes formas seguindo

uma analogia a conjuntos. São aplicados em buscas gráficas, rede bayesianas, buscas otimizadas, conectividade e geolocalização (VAN ERVEN, 2015). A Figura 5 ilustra um modelo orientado a grafos.

Figura 5 - Modelo de grafos



Conforme apresentado na Figura 5, os círculos representam entidades do banco de dados e suas informações. As linhas representam os vínculos relacionais entre as entidades. A entidade com os atributos “Nome: Mario” e “Idade: 42” contêm três vínculos: “Precisa”, que está vinculada com a entidade que possui os atributos “Nome: Café” e “Tipo: Expresso”; “Programa” está vinculado com a entidade que possui os atributos “Nome: Java” e “Tipo: Linguagem”; e “Ama”, que está vinculada à entidade que possui os atributos “Nome: Mary” e “Idade: 38”. A entidade com os atributos “Nome: Luidi” e “Idade: 19” contêm dois vínculos: “Gosta”, que faz referência a entidade que possui “Nome: Café” e “Tipo: Expresso”, e “Filho” para a entidade que possui “Nome: Mary” e “Idade: 38”. As entidades que possuem, respectivamente, os atributos “Nome: Mary” e “Idade: 38”, “Nome: Café” e “Tipo: Expresso”, e “Nome: Java” e “Tipo: Linguagem” são passivas, pois não originam nenhum vínculo com as demais entidades.

2.6 MongoDB

O *MongoDB* é um banco de dados *NoSQL* multi-plataforma, foi criado pela empresa de mesmo nome. Armazena dados em documentos *JSON* com estrutura dinâmica, logo não restringe o número de campos ou tipos de valores armazenados. Foi desenvolvido com a linguagem de programação C++ e seu código é livre. Oferece alto desempenho, alta disponibilidade e dimensionamento automático. A manipulação dos dados é feita por funções *Javascript*. As aplicações se comunicam com o *MongoDB* através de drivers específicos para cada linguagem, para gerenciar a iteração com o banco de dados (MongoDB, 2017a). O conteúdo desta seção foi baseado na documentação oficial do *MongoDB*[14].

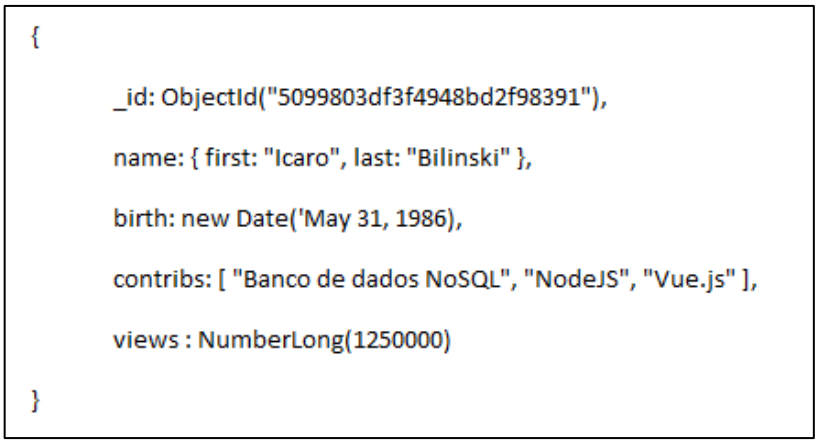
2.6.1 Modelo de documento

O *MongoDB* adota o modelo de armazenamento de documentos baseado na sintaxe do *JSON*, sendo "um padrão leve de intercâmbio de dados projetado para facilitar a leitura e escrita de estruturas de dados". (JSON,2000). São baseadas em um subconjunto da especificação *Javascript* e permitem a construção de coleções de pares de chave e valor, e uma lista ordenada de valores (MDN, 2017).

O *MongoDB* armazena registros de dados em estruturas *BSON* (*JSON* binário), que são representações binárias de documentos *JSON* e podem conter tipos de dados específicos, como *ObjectId*, *Double*, *String* e *Date* dentre outros (MongoDB, 2017b). A Figura 6Figura 5 ilustra um documento *BSON*.

[14] Disponível em <*https://docs.mongodb.com/manual/*> Acesso em abril. 2017.

Figura 6 - Documento BSON



Conforme apresentado na Figura 6, as chaves representam um documento, o atributo "_id" contém um *ObjectId*, que serve como identificação para o documento. O atributo "*name*" contém dois documentos com os atributos "first", com o valor "Icaro" e "last", com o valor "Bilinski". O atributo "birth" contém um objeto tipo *Date* que armazena "May 31,1986". No atributo "contribs" os colchetes representam um conjunto de *Strings* contendo os valores "Banco de dados NoSQL", "NodeJS", "Vue.js". Por fim, o atributo "views" representa um objeto do tipo *NumberLong* contendo o valor "1250000".

Um conjunto de documento com propósitos semelhantes, formam uma coleção de documentos (*collection*) que é equivalente à uma tabela em banco de dados relacionais. As coleções também são livres de esquemas, ou seja, podem armazenar documentos com atributos diferentes, porém, com propósito similar ou relacionado. Uma coleção de documentos é criada quando o primeiro documento é inserido no banco de dados.

2.6.2 ObjectId

Um *ObjectId* é um atributo que geralmente é representado por "_id" e serve como chave primária exclusiva para cada documento no *MongoDB. É* composta por doze *bytes* combinados para gerar um valor único. Os quatro primeiros *bytes* representam os segundos no padrão *Unix*[15], os três próximos *bytes* são os identificadores do computador, os próximos dois *bytes*

[15] Disponível em: <*http://www.opengroup.org/subjectareas/platform/unix*> Acesso em abril. 2017.

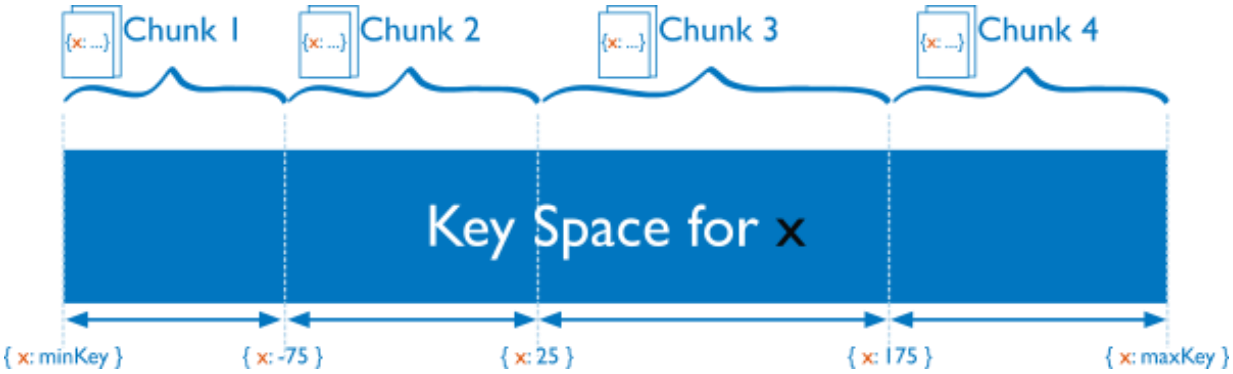
são os identificadores do processo do *MongoDB* e os últimos três bytes representam um valor aleatório.

Se um documento for inserido sem "_id", o *MongoDB* irá gerar automaticamente um *ObjectId* para o campo _id do documento (MongoDB, 2017c).

2.6.3 Chunks

Segundo a documentação oficial do *MongoDB*, MongoDB cf. (2017j) *Chunks* é técnica utilizada pelo *MongoDB* para distribuir os documentos entre outras instâncias do banco de dados. O *MongoDB* divide a coleção de documentos com a atribuição de uma chave de partição em blocos, que é constituída por um campo ou campos imutáveis que existem em cada documento na coleção. Um *chunk* consiste em um subconjunto de dados particionados e uma chave de partição com tamanho definido. A distribuição é executada automaticamente quando um *chunk* ultrapassa o limite de tamanho configurado. Seu tamanho pode ser alterado conforme as necessidades da aplicação, porém quanto menor for o *chunk* maior será o consumo de rede durante a migração. Faz sentido evitar migrações frequentes e desnecessárias às custas de um conjunto de dados ínfimos. A Figura 7 ilustra a distribuição dos *chunk's* entre instancias do MongoDB.

Figura 7 - Distribuição dos *Chunk's* no *MongoDB*



Fonte: Site oficial do MongoDB[16]

Conforme apresentado na Figura 7, o "x" representa a chave utilizada para distribuição dos documentos de uma coleção, os limites de tamanho do *Chunk 1*, iniciam com a chave de valor "minKey" e terminam na chave de valor "-75". O *Chunk 2* inicia com a chave de valor "-

[16] Disponível em: <*https://docs.mongodb.com/manual/core/sharding-data-partitioning*> Acesso em abril. 2017.

76” e termina em “25”. O *Chunk 3* inicia com a chave de valor “26” e termina em “176”. O *Chunk 4* inicia com a chave de valor “176” e termina em “maxKey”.

2.6.4 Replicação

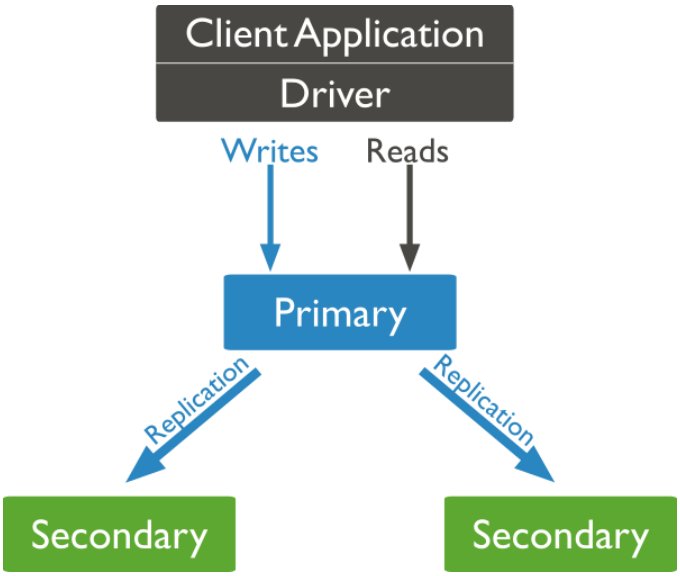
Segundo a documentação oficial do *MongoDB,* MongoDB cf. (2017d) Replicação é um método utilizado para distribuir os dados entre servidores do *MongoDB*, para prover redundância e aumento da disponibilidade de dados. Replicar os dados aumenta o nível de tolerância a falhas, evitando a perda em um único servidor de banco de dados. Em alguns casos, aumenta a capacidade de leitura, pois os clientes podem enviar requisições para servidores diferentes.

A distribuição de servidores replicados em locais distintos favorece as consultas de dados por clientes próximos e melhora a disponibilidade de dados para aplicativos distribuídos. A replicação também pode servir para a recuperação de desastres, relatórios e backup.

O **nó primário** é responsável por receber todas as requisições dos clientes. Um conjunto de réplicas pode ter apenas um nó primário, porém em alguns casos, como falhas de comunicação, outras instâncias secundárias do *MongoDB* podem realizar uma eleição para eleger o novo nó primário e garantir a alta disponibilidade. Todas as operações são armazenadas em um *log* de operações para replicar para os nós secundários. Após a replicação do *log* de operações, as operações são gravadas de modo que os dados dos nós secundários reflitam os dados do primário.

Os **nós secundários** são bloqueados para requisições de escrita, porém, os clientes podem requisitar a leitura dos dados, mas não há certeza de que os dados sejam realmente consistentes com a base de dados do nó primário. A **Erro! Autoreferência de indicador não válida.** ilustra a replicação de dados no *MongoDB.*

Figura 8 - Replicação no MongoDB



Fonte: Site oficial do MongoDB[17]

Conforme demostrado na Segundo a documentação oficial do *MongoDB,* MongoDB cf. (2017d) Replicação é um método utilizado para distribuir os dados entre servidores do *MongoDB*, para prover redundância e aumento da disponibilidade de dados. Replicar os dados aumenta o nível de tolerância a falhas, evitando a perda em um único servidor de banco de dados. Em alguns casos, aumenta a capacidade de leitura, pois os clientes podem enviar requisições para servidores diferentes.

A distribuição de servidores replicados em locais distintos favorece as consultas de dados por clientes próximos e melhora a disponibilidade de dados para aplicativos distribuídos. A replicação também pode servir para a recuperação de desastres, relatórios e backup.

O **nó primário** é responsável por receber todas as requisições dos clientes. Um conjunto de réplicas pode ter apenas um nó primário, porém em alguns casos, como falhas de comunicação, outras instâncias secundárias do *MongoDB* podem realizar uma eleição para eleger o novo nó primário e garantir a alta disponibilidade. Todas as operações são armazenadas em um *log* de operações para replicar para os nós secundários. Após a replicação do *log* de operações, as operações são gravadas de modo que os dados dos nós secundários reflitam os dados do primário.

Os **nós secundários** são bloqueados para requisições de escrita, porém, os clientes podem requisitar a leitura dos dados, mas não há certeza de que os dados sejam realmente

[17] Disponível em: <*https://docs.mongodb.com/manual/replication/*> Acesso em abril. 2017.

consistentes com a base de dados do nó primário. A **Erro! Autoreferência de indicador não válida.** ilustra a replicação de dados no *MongoDB.*
Figura 8, as requisições de escrita e leitura são realizadas diretamente no nó primário que irá replicar o log de operações para demais nós secundários.

2.6.5 Sharding

Segundo a documentação oficial do *MongoDB,* MongoDB cf. (2017e) *Sharding* é um método utilizado para distribuir dados e operações entre vários servidores com o *MongoDB.* Os sistemas que demandam armazenamento de grandes quantidades de dados e operações custosas exigem uma demanda de *hardware* além das capacidades de um único servidor, tais como: consultas compostas que exigem altas taxas de processamento, conjunto de trabalho maiores do que a quantidade de memória do sistema e leituras constantes de unidades de disco. Logo, é necessário aumentar os recursos computacionais requeridos pelo banco de dados. Existem dois métodos para expandir as capacidades computacionais para um sistema, escalabilidade vertical e horizontal.

A **escalabilidade vertical** consiste em aumentar a capacidade de hardware de um único servidor, como usar um processador de potência superior, adicionar memórias e aumentar a capacidade de armazenamento. Esta abordagem é limitada pelas capacidades físicas do servidor, logo um único servidor não será suficientemente capaz de servir cargas de trabalho em expansão, logo, existe um teto máximo para a escalabilidade vertical.

A **escalabilidade horizontal** consiste em dividir o conjunto de dados e tarefas do sistema em vários servidores, agregando mais servidores para aumentar a capacidade conforme a demanda. Ainda que os recursos e capacidades de uma única máquina não sejam suficientes, cada máquina recebe um subconjunto da carga de trabalho global, somando-se uma melhor eficiência em relação à um único servidor de alta velocidade e de alta capacidade. A expansão de recursos requer apenas a adição de servidores sob demanda.

O *MongoDB* implementa escalabilidade horizontal através do método *Sharding*, que consistem em dividir funções entre nós do *sharding* e aumentar as capacidades do banco de dados como um todo, tais funções podem ser exercidas no mesmo servidor. Entretanto, não é uma prática adequada visto que, em caso de falha, pode afetar o banco de dados como um todo. Existem três tipos de nós *shards, config server* e *query routes* em um *sharding.*

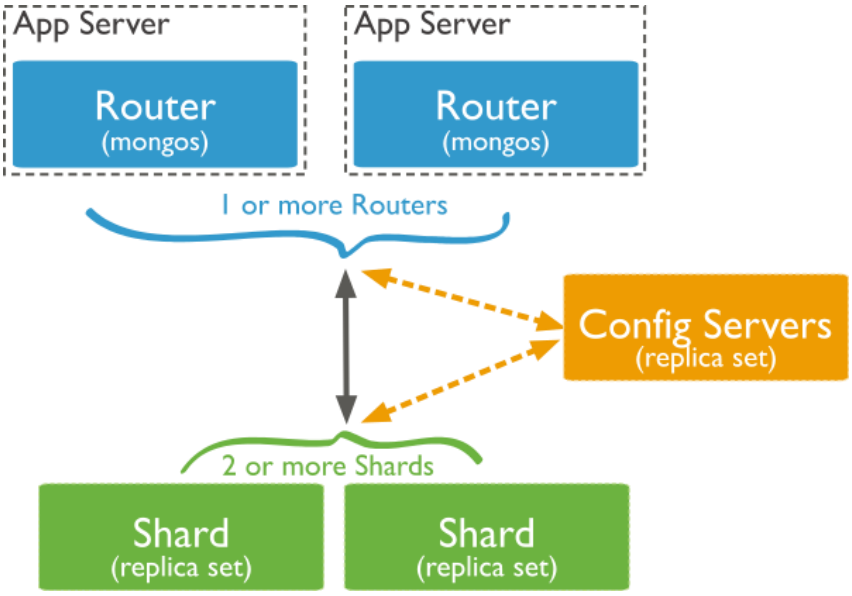
Os ***Shards*** são usados para armazenar réplicas de dados e garantir a alta disponibilidade e redundância. A união dos *shards* mantêm a base de dados como um todo. Os *shards* exercem a função de nó secundário para prover a replicação dos dados no *sharding*.

Os ***Config Servers*** compartilham informações do cluster através de metadados, para direcionar o envio de requisições para os nós com menor tarefa. Os metadados armazenam o estado e organização dos dados em uma lista de chaves dos *chunks* de cada *shard*. Os *config servers* exercem a função de nó primário para gerir a replicação dos dados no *sharding*.

Os ***Query Routers*** são responsáveis pela comunicação com o cliente via API, e distribuem as requisições entre os nós dos *shards*, fornecendo uma interface entre a aplicação e o *sharding*. A

Figura 9 demonstra a arquitetura de um *sharding* no MongoDB.

Figura 9 - *Sharding* no MongoDB



Fonte: Site oficial do MongoDB[18]

Conforme demostrado na

[18] Disponível em: <*https://docs.mongodb.com/manual/sharding/*> Acesso em abril. 2017.

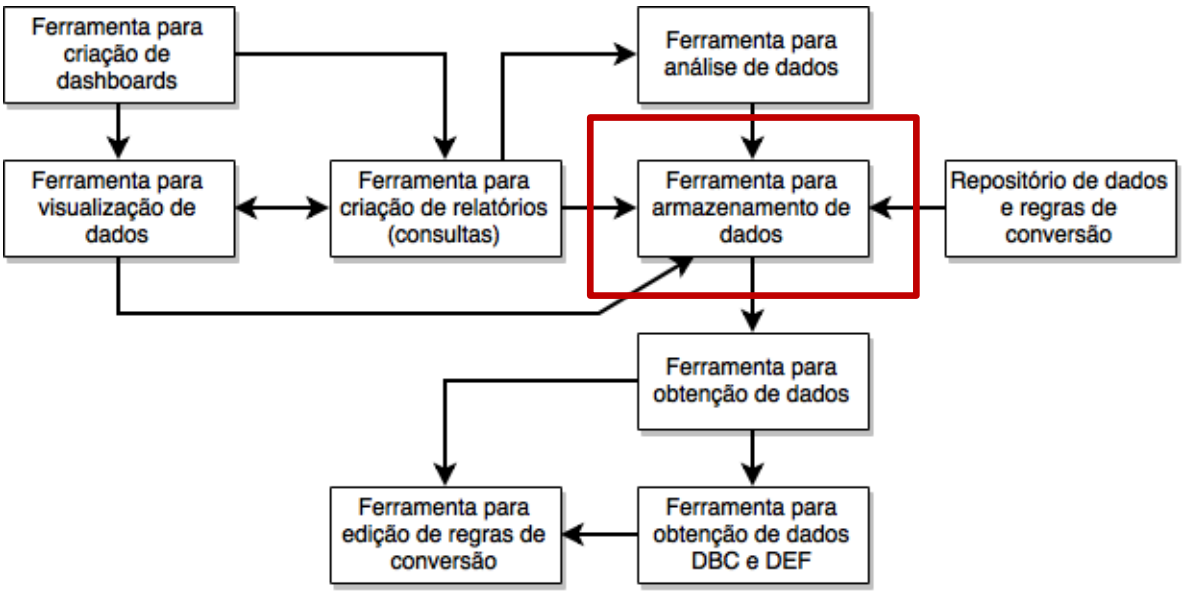
Figura 9, os *query routes*, direcionam as consultam das aplicações para os *config server* para executar as requisições de escrita e direciona as requisições leitura e *log* de operações para os *shards*.

2.7 Plataforma para análise e manipulação de dados

O presente trabalho tem o objetivo de desenvolver uma estrutura de armazenamento de dados de alto desempenho para uma plataforma de análise e manipulação de dados, a fim de obter dados como o processamento dos dados armazenados e gerar resultados das informações obtidas. Tal plataforma está dividida em módulos onde cada módulo contém um conjunto de ferramentas, para executar funções específicas e independes na plataforma.

Os resultados ou saídas do processamento de dados de cada ferramenta, irão servir como entrada de dados para demais módulos contidos na plataforma. A Figura 10 apresenta a plataforma como um todo e o fluxo de informação entre os módulos em questão.

Figura 10 - Relação entre módulos.



Fonte: Imagem cedida pelo o prof. Orientador

Conforme apresentado na Figura 10, os retângulos representam as ferramentas que integram a plataforma em questão e as setas indicam a dependência entre as ferramentas. O destaque em vermelho indica onde está incluso o presente trabalho.

Os principais módulos são detalhados a seguir.

- **A ferramenta para armazenamento de dados** é responsável pela persistência dos dados inseridos e gerados na plataforma, para compartilhar os dados com as demais ferramentas. A fim de garantir tal função, será utilizado o banco de dados NoSQL *MongosDB*.
- **A ferramenta para obtenção de dados** é responsável pela conversão de entradas de dados em um modelo de dados usual da plataforma, através de funcionalidades como entrada de arquivos e de definições preestabelecidas, importação e exportação de dados e criar regras de conversão de dados.
- As **ferramentas de repositório de dados e regras de conversão** compartilham com demais ferramentas da plataforma, os conjuntos de dados e regras para conversão de dados.
- As **ferramentas para criação de relatórios** gerenciam os relatórios para a plataforma, sendo capaz de criar relatórios tabulados, através de filtragens de dados, salvar e compartilhar através de URL (Localizador Padrão de Recursos).
- As **ferramentas para visualização dos dados** permitem visualizar os dados através de gráficos de relatórios salvo em repositório.
- As **ferramentas para criação de *dashboards*** permite visualizar os dados dos relatórios através de *dashboards*.

3 MATERIAIS E MÉTODOS

Nesta seção serão apresentados os materiais e métodos utilizados para desenvolvimento do este trabalho.

3.1 Materiais

A ferramenta escolhida para, desenvolver a estrutura de armazenamento de dados de alto desempenho para uma plataforma de análise e manipulação de dados, foi o banco de dados *MongoDB.* Serão realizados testes para verificar o desempenho e comportamento que simulem as necessidades da plataforma de análise e manipulação de dados.

O banco de dados foi implementado de duas formas diferentes objetivando simular dois ambientes, centralizado e distribuído. Será utilizado o recurso de *sharndig* do *MongoDB* para avalizar o alto desempenho e garantir a disponibilidade dos dados, como requisito da plataforma.

Os testes serão executados através da ferramenta *Yahoo! CloudServing Benchmark*[19] de duas formas, carga e execução de dados, em ambos ambientes implementados. Isto, para testar a carga de inserções e leituras do banco de dados, e gerar uma saída contendo os resultados dos testes.

Para visualizar detalhes da base de dados e o comportamento do banco de dados foi utilizado a ferramenta *MongoDB Compass*[20]. A seguir, cada ferramenta será apresentada com maiores detalhes.

3.1.1 Yahoo! CloudServing Benchmark

Segundo a documentação oficial do YCSB cf. (2017) o YCSB é uma ferramenta de linha de comando que executa testes de cargas de trabalho para avaliar o desempenho de diferentes tipos de banco de dados. Está dividida em duas partes: o cliente que executa a carga de trabalho através de conexão direta com o banco de dados; e arquivos contendo parâmetros de carga de trabalho para serem executados pelo cliente.

---

[19]https://github.com/brianfrankcooper/YCSB/wik
[20]https://www.mongodb.com/products/compass

A ferramenta permite personalizar os testes para melhor atender aos sistemas ou cenários específicos de aplicações que não estão devidamente configuradas por padrão. Da mesma forma, o cliente é extensível para suportar testes em diversos bancos de dados, e permite a integração de conexões com bancos não suportados por padrão.

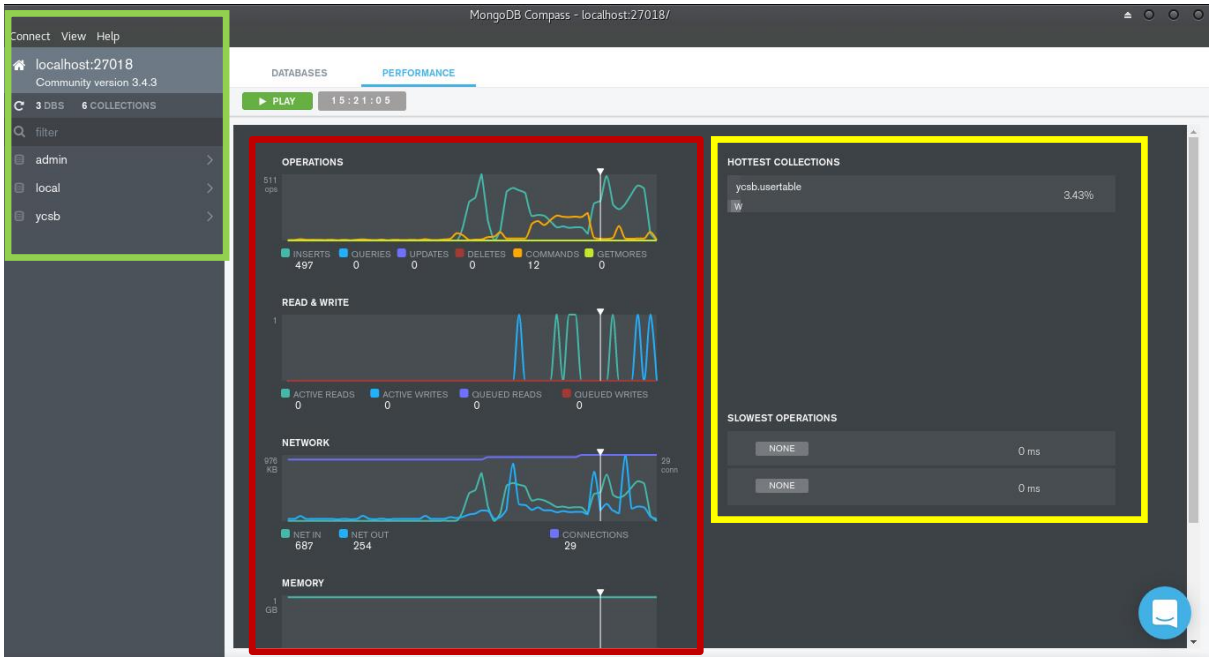
O uso comum da ferramenta é comparar vários sistemas de banco de dados ou configurações diferentes para um mesmo banco de dados. Por exemplo, testes idênticos podem ser executados em vários sistemas no mesmo hardware, no intuito de avaliar qual melhor banco de dados NoSQL, atende as necessidades da aplicação. Tais testes são analisados, através do cruzamento dos resultados, entre latência e a curvas de transferência.

Esta ferramenta será responsável por executar testes, de escriva e leitura no banco de dados *MongoDB*. Tais resultados serão detalhados na seção de resultados e discussões.

3.1.2 MongoDB Compass

É uma ferramenta gráfica que permite visualizar e manipular os dados através das funcionalidades CRUD, auditar consultas para analisar o tempo de execução e detectar possíveis gargalos de desempenho. Esta ferramenta será utilizada para analisar os resultados dos testes dos ambientes programados e fazer considerações sobre pontos específicos encontrados. (MongoDB, 2017e). A **Erro! Autoreferência de indicador não válida.** demonstra a interface do *MongoDB Compass*.

Figura 11- Interface da ferramenta MongoDB Compass



Conforme apresentado na É uma ferramenta gráfica que permite visualizar e manipular os dados através das funcionalidades CRUD, auditar consultas para analisar o tempo de execução e detectar possíveis gargalos de desempenho. Esta ferramenta será utilizada para analisar os resultados dos testes dos ambientes programados e fazer considerações sobre pontos específicos encontrados. (MongoDB, 2017e). A **Erro! Autoreferência de indicador não válida.** demonstra a interface do *MongoDB Compass*.
Figura 11, o destaque verde demonstra a listagem da bases de dados. O destaque em vermelho demostra os gráficos de monitoramento do status do banco de dados, contendo o gráfico de operações de inserção, consultas, atualização, remoção e agregações (*getmores*). O gráfico de operações de escrita e leitura de dados, o gráfico do tráfego de rede com entrada e saída de dados bem como o número de conexões com o banco de dados e a gráfico de consumo de memória. O destaque em amarelo demostra o consumo do recurso pela base dados e porcentagem de tempo de resposta para operações lentas. Esta foi a principal ferramenta utilizada na captação do consumo de recursos físicos pelo banco de dados durante os experimentos realizados nos dois ambientes de teste.

Esta ferramenta será responsável por monitorar o comportamento do *MongoDB*, durante a execução dos testes. Tais resultados serão detalhados na seção de resultados e discussões.

3.2 Métodos

Para elaboração do presente trabalho foram realizados os seguintes passos:

1. Levantamento dos requisitos da plataforma (juntamente com o prof. Orientador);
2. Criação de dois ambientes (Centralizado e Distribuído);
3. Execução dos testes em ambientes implementados e coleta dos resultados;
4. Comparativo dos resultados de desempenho em ambas implementações.

Para facilitar o entendimento, estes passos serão detalhados nos parágrafos seguintes.

No levantamento inicial dos requisitos da plataforma de análise e manipulação de dados foi constatado que a estrutura de armazenamento deve acompanhar o crescimento sob demanda, visto que não há servidores específico para função exclusiva de persistência de dados para a plataforma de análise e manipulação de dados, logo constado que a base de dados deve ser

distribuída e manter-se disponível. Para isso foi definido que a ferramenta utilizada para persistir os dados deveria ser um banco de dados *NoSQL*, pois estes garantem tais recursos, através da escalabilidade horizontal de expansão do banco de dados. Foi escolhido do banco de dados *MongoDB* devido a facilidade para implementar os requisitos necessários da plataforma.

Os ambientes de teste foram executados em um único computador, contendo a seguinte configuração: processor Intel® Core™ i5 M450 2.40Ghz de 2 núcleos e 2 *threads* por núcleo, memória RAM 6.0 Giga bytes e disco de armazenamento de 500 Giga bytes. O *MongoDB* foi instalado sobre o sistema operacional *Manjaro Linux*[21], com a versão do *Kernel* 4.9.32[22] e sistema de arquivos *XFS* como aconselhado na documentação oficial do banco de dados *MongoDB.*

O ambiente centralizado servirá de modelo representativo, para servir de comparação com o ambiente distribuído, a plataforma exige que os dados sejam replicados para garantir que os dados permaneçam disponíveis em caso de sinistro, tais distribuição de tarefas permite a base de dados disseminar funções de leitura e escrita no banco de dados em sincronia com todos os computadores que executam os processos do banco de dados *MongoDB.*

O ambiente distribuído foi executado localmente, devido a indisponibilidade de equipamentos físicos. Onde cada parte do *Sharding* foi representado por processos diferentes executados em portas díspares, visto a necessidade de dispor de cinco processos para realização deste trabalho, afim de simular um sistema contendo de alta disponibilidade.

Os testes foram executados a partir da ferramenta *YCSB*, em duas etapas, na primeira foram carregadas informações para o banco de dados e posteriormente executada a sobrescrita dos dados, o número de operações foi personalizada para melhora a demanda da plataforma. Além disso os processos do banco de dados foram monitorados em tempo de teste com a ferramenta *MongoDB Compass,* para melhor visualizar o comportamento do banco de dados e a distribuição das transações entre os processos da base de dados distribuída.

Com os resultados de desempenho coletados serão comparados, levando em consideração o número de operações, consumo de rede e consumo de memória em cada um dos ambientes. Este comparativo tem como objetivo mostrar a distribuição das tarefas no ambiente distribuído, para comprovar a disponibilidade dos dados entre os processos do banco de dados *MongoDB.*

---

[21] Disponível em: < https://manjaro.github.io/> Acesso em Mai. 2017
[22] Disponível em: <https://www.kernel.org/> Acesso em Jun. 2017

Os resultados obtidos nos experimentos realizados, servirão de apoio para implementar um modelo de persistência de dados de alto desempenho, na plataforma de análise e manipulação de dados do Grupo de Pesquisa de Engenharia Inteligente de Dados do CEULP/ULBRA.

4 RESULTADOS E DISCUSSÃO

O presente trabalho tem como objetivo, testar duas formas de implementar o ambiente de persistência para a plataforma de análise e manipulação dos dados. O ambiente centralizado foi utilizado para demostrar o funcionamento padrão do banco de dados e o ambiente distribuído exemplificou a distribuição das tarefas entre demais processos, através da escalabilidade horizontal, provida pelos bancos de dados *NoSQL* e acrescer o desempenho da persistência dos dados e a disponibilidade dos dados em mais de uma origem.

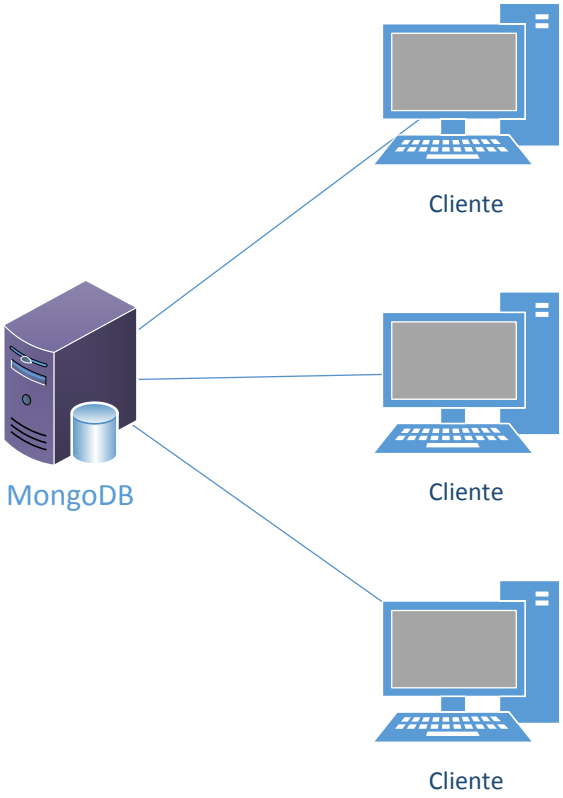
O presente capítulo aborda os resultados alcançados com a implementação da estrutura de armazenamento de dados em alto desempenho. Para critérios comparativos, os testes foram executados em dois ambientes, centralizado e distribuído, no intuito de comprovar os benefícios, do ambiente distribuído como estrutura de alto desempenho para a plataforma. Através da distribuição de funções do banco de dados *MongoDB*, em diversos processos replicado em demais computadores, e garantir a disponibilidade do sistema como um todo.

4.1 Ambiente Centralizado (*Standalone)*

O ambiente centralizado consiste na execução do *MongoDB* em um único processo, onde todas as tarefas serão executadas, os dados não serão replicados e as consultas serão isoladas a um único servidor. A seguir será demostrado a implementação do ambiente de teste centralizado.

A Figura 12 apresenta o ambiente centralizado.

Figura 12 – MongoDB centralizado *(standalone)*



Como demostrado na Figura 12, o *MongoDB* centralizado recebe todas requisições em um único processo. O comando a seguir demostra como foi iniciado o banco de dados MongoDB em modo centralizado.

Comando 1 - Inicia o *MongoDB* em modo centralizado.

```
mongod --port 27017 --dbpath /data/db
```

O Comando 1 inicia um processo do banco de dados *MongoDB*, onde os parâmetros representam as seguintes configurações:

"–port 27017" representa a porta para estabelecer a conexão de rede com os clientes

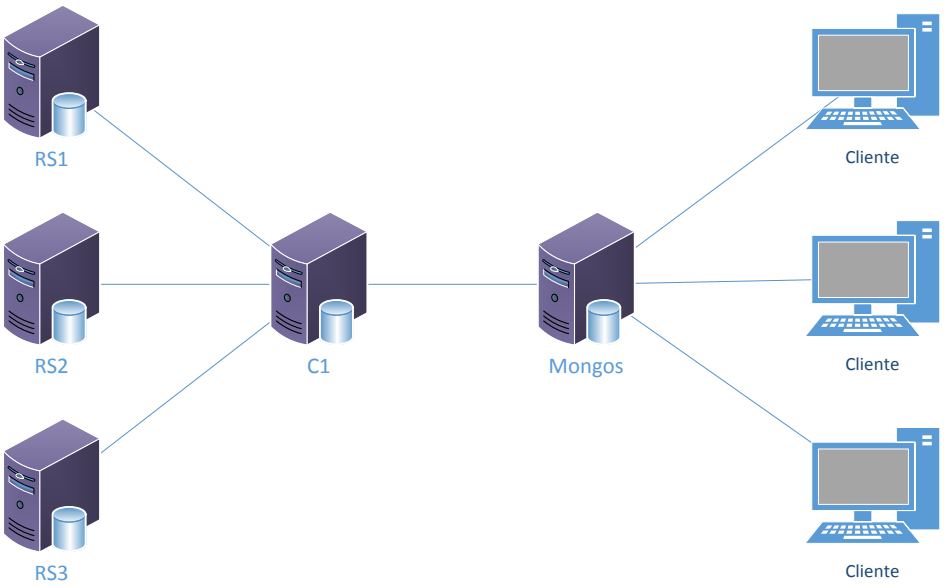
"–dbpath /data/db" identifica o diretório onde serão armazenados dos dados.

4.2 Ambiente Distribuído (*Sharding*)

O ambiente distribuído consiste na execução do *MongoDB* em vários processos, onde determinadas as tarefas serão executadas em diferentes instancias do banco de dados, os dados serão replicados para três processos com a função de *shards*, um processo com função de *config server* e outro processo com a função de *query router*. Tais quebras de funções representa o

aumento de desempenho geral da base de dados, visto que outros computadores somam seus recursos e agregam poder de processamento e disponibilidade para a base de dados. A Figura 13 representa o ambiente o ambiente distribuído.

Figura 13 - *MongoDB* em Sharding



Como demostrado na Figura 13, o *MongoDB query router* (Mongos), recebe as requisições dos clientes e envia para o *config server* (C1), onde será consultado seus metadados para definir qual *shard* (RS1, RS2 ou RS3) responderá tais requisições. A seguir será demostrado a implementação do ambiente de teste distribuído, bem como a inicialização de todos os processos que abrangem o ambiente distribuído.

Comando 2 - Inicia um processo do MongoDB em modo *Config Server*.

```
mongod --configsvr --replSet c1 --enableMajorityReadConcern
--logpath=/data/configdb/log.txt;
mongo localhost:27019 --eval "JSON.stringify(rs.initiate())"
```

O Comando 2 inicia um processo do banco de dados *MongoDB* em modo *Config Server*, onde os parâmetros representam as seguintes configurações:

“–configsvr” define que esta instância de *MongoDB* será o servidor de configuração de um *sharding*.

“—replSet c1” define o nome no conjunto de réplicas **“c1”**.

“--enableMajorityReadConcern” especifica que será um nó primário na replicação dos dados no *sharding.*

“--logpath=/data/configdb/log.txt;” define o local do arquivo de log do processo.

“mongo localhost:27019 --eval "JSON.stringify(rs.initiate())” comando interno do *MongoDB* para iniciar o processo.

Comando 3 - Inicia um processo do MongoDB em modo *Shard replicaset rs1*.

```
mongod --shardsvr --replSet rs1 --dbpath /data/db --logpath=/data/db/log.txt --port 27018;
mongo localhost:27018 --eval "JSON.stringify(rs.initiate())"
```

O Comando 3 inicia um processo do banco de dados *MongoDB* em modo *Shard replicaset rs1*, onde os parâmetros representam as seguintes configurações:

“–shardsvr” define que esta instância de *MongoDB* será um *shard*.

“—replSet rs1” define o nome no conjunto de réplicas **“rs1”.**

“–dbpath /data/db” identifica o diretório onde serão armazenados dos dados.

“--logpath=/data/db/log.txt;” define o local do arquivo de log do processo.

“—port 27018” define a porta de comunicação com demais processos do *sharding*

“mongo localhost:27018 --eval "JSON.stringify(rs.initiate())” comando interno do *MongoDB* para iniciar o processo.

Comando 4 - Inicia um processo do MongoDB em modo *Shard replicaset rs2*.

```
mongod --shardsvr --replSet rs2 --dbpath /data/db2 --logpath=/data/db2/log.txt --port 27028;
mongo localhost:27028 --eval "JSON.stringify(rs.initiate())"
```

O Comando 4 inicia um processo do banco de dados *MongoDB* em modo *Shard replicaset rs2*, onde os parâmetros representam as seguintes configurações:

“–shardsvr” define que esta instância de *MongoDB* será um *shard*.

“—replSet rs2” define o nome no conjunto de réplicas **“rs2”.**

“–dbpath /data/db2” identifica o diretório onde serão armazenados dos dados.

“--logpath=/data/db2/log.txt;” define o local do arquivo de log do processo.

“—port 27028” define a porta de comunicação com demais processos do *sharding*

“mongo localhost:27028 --eval "JSON.stringify(rs.initiate())” comando para iniciar o processo.

Comando 5 - Inicia um processo do MongoDB em modo *Shard replicaset rs3*.

```
mongod --shardsvr --replSet rs3 --dbpath /data/db3 --logpath=/data/db3/log.txt --port 27038;
mongo localhost:27038 --eval "JSON.stringify(rs.initiate())"
```

O Comando 5 inicia um processo do banco de dados *MongoDB* em modo *Shard replicaset rs3*, onde os parâmetros representam as seguintes configurações:

“–shardsvr” define que esta instância de *MongoDB* será um *shard*.

“—replSet rs3” define o nome no conjunto de réplicas **“rs3”.**

“–dbpath /data/db3” identifica o diretório onde serão armazenados dos dados.

“--logpath=/data/db3/log.txt;” define o local do arquivo de log do processo.

“—port 27038” define a porta de comunicação com demais processos do *sharding*

“mongo localhost:27038 --eval "JSON.stringify(rs.initiate())” comando para iniciar o processo.

Comando 6 - Inicia um processo do MongoDB em modo *Query Router.*

```
mongos --configdb c1/localhost --logpath=/data/mongos.log
```

O Comando 6 inicia um processo do banco de dados *MongoDB* em modo *Query Router*, onde os parâmetros representam as seguintes configurações:

“—configdb c1/localhost” a conexão do *query router* com o *config server*.

“--logpath=/data/mongos.log” define o local do arquivo de log do processo.

Comando 7 - Comandos interno do MongoDB *query router* para adicionar os *shards* no *config server*

```
sh.addShard("rs1/localhost:27018")
sh.addShard("rs2/localhost:27028")
sh.addShard("rs3/localhost:27038")
```

O Comando 7 adiciona os *shards* configurados no ambiente distribuído, onde os parâmetros representam as seguintes configurações:

“sh.addShard("rs1/localhost:27018")” adiciona o *sharding rs1* no ambiente distribuído.

“sh.addShard("rs2/localhost:27028")” adiciona o *sharding rs2* no ambiente distribuído.

“sh.addShard("rs3/localhost:27038")” adiciona o *sharding rs3* no ambiente distribuído.

Comando 8 - Comando interno do MongoDB para definir o tamanho do *Chunk’s* para 2 *megabytes*.

```
use config
db.settings.save( { _id:"chunksize", value: 2 } )
```

O Comando 8 define o tamanho dos *Chunk’s* para 2 megabytes. Tal comando foi utilizado para facilitar a replicação dos dados, entre os processos do ambiente distribuído. Onde os parâmetros representam as seguintes configurações:

“use config” define o acesso a base de dados de configuração do *MongoDB query Router*.

“db.settings.save({ _id:"chunksize", value: 2 })” define que o tamanho dos *Chunk’s* será de 2 megabytes.

Comando 9 – Comando de definição da base de dados distribuída.

```
sh.enableSharding("ycsb")
sh.shardCollection("ycsb.usertable", { "_id" : 1 } )
```

O Comando 9 criar uma nova base de dados distribuída e adiciona uma chave para compartilhar os documentos entre os *shards*. Onde os parâmetros representam as seguintes configurações:

“sh.enableSharding("ycsb")” define o nome da base de dados distribuída entre os shards.

“sh.shardCollection("ycsb.usertable", { "_id" : 1 })” define a chave de identificação da base de dados distribuída.

Com o ambiente centralizado e distribuído configurados, foram executados testes em cada ambiente, na próxima seção será melhor detalhado os testes e seus resultados.

4.3 Testes de Ambientes

Através dos conceitos apresentados em capítulos anteriores, o *MongoDB* foi executado de duas formas, em um processo único para simular o ambiente centralizado e em vários processos para simular um ambiente distribuído, no qual cada processo distribuído é responsável por uma função do *sharding*, visto que tais processos podem ser distribuídos para executar em demais computadores em rede.

Os testes foram realizados com a ferramenta *YCSB* e foram divididos em duas etapas. A primeira etapa consistiu no carregamento dos dados que define os dados a serem inseridos. A segunda etapa consistiu na execução dos dados que define as operações a serem executadas em relação ao conjunto de dados. Foi definida a inserção de 10000 registros binários no banco de dados onde o tamanho final da base de dados permaneceu em 10 *MegaBytes.*

Os mesmos testes foram executados em ambos ambientes (centralizado e distribuído) implementados com o *MongoDB.* Após a implementação dos ambientes de testes a ferramenta *YCSB* foi executada para automatizar os procedimentos de inserção e leitura de dados no banco de dados, através de conexão direta com o banco de dados. Em paralelo foi executado a ferramenta *MongoDB Compass* para monitor o comportamento e a quantidade de operações, em ambos ambientes implementados.

Para melhor atender aos requisitos da plataforma a carga de trabalho, executada através da ferramenta *YCSB* foi personalizada com parâmetros adicionais no comando de execução da ferramenta.

Os comandos para executar os testes são:

Comando 10 - Comando de teste de caga de dados

```
./bin/ycsb load mongodb-async -s -P workloads/workloada –p recordcount=10000 –p operationcount=100000
```

O Comando 10 executa a ferramenta de carga de dados na base de dados, através da conexão da com o banco de dados pela porta padrão. Onde os parâmetros representam as seguintes configurações:

“./bin/ycsb” executa a ferramenta *YCSB,* o primeiro parâmetro representa o tipo de teste **“load”** para o teste de carga de dados.

“mongodb-async” define qual será o banco de dados utilizado para fazer a conexão, foi escolhi uma conexão assíncrona com o *MongoDB.*

“-s” garante o status do relatório a cada 10 segundos na saída padrão de erro stderr. Caso o teste seja interrompido por eventual erro.

“-P” é usado para carregar arquivos de propriedades, nesse caso, foi carregado o arquivo workloada, que contém os parâmetros padronizados da ferramenta.

“–p” personaliza as definições do teste em linha de comando, onde **“recordcount=10000”** representa o número de registros carregados no banco de dados inicialmente foi definido para 10000 e **“operationcount=10000”** que representa o número de operações a serem executadas, onde foi definido para 10000.

Comando 11 - Comando de teste de execução de dados

```
./bin/ycsb run mongodb-async -s -P workloads/workloada –p recordcount=10000 –p operationcount=100000
```

O Comando 11 executa a ferramenta de teste de execução de dados na base de dados, através da conexão da com o banco de dados pela porta padrão. Onde os parâmetros representam as seguintes configurações.

“./bin/ycsb” executa a ferramenta *YCSB,* o primeiro parâmetro representa o tipo de teste **“run”** para o teste de execução dos dados.

“mongodb-async” define qual será o banco de dados utilizado para fazer a conexão, foi escolhi uma conexão assíncrona com o *MongoDB.*

“-s” garante o status do relatório a cada 10 segundos na saída padrão de erro stderr. Caso o teste seja interrompido por eventual erro.

“-P” é usado para carregar arquivos de propriedades, nesse caso, foi carregado o arquivo workloada, que contém os parâmetros padronizados da ferramenta.

“–p” personaliza as definições do teste em linha de comando, onde **“recordcount=10000”** representa o número de registros carregados no banco de dados inicialmente foi definido para 10000 e **“operationcount=10000”** que representa o número de operações a serem executadas, onde foi definido para 10000.

As tabelas representam os resultados dos testes gerados pela ferramenta *YCSB.* A seguir as tabelas 1 e 2 contendo os resultados dos testes executados pela ferramenta.

Tabela 1 - Tabela dos resultados do teste de carga de dados

Ambiente do teste de carga	Centralizado	Distribuído	Diferença em porcentagem.
Tempo de execução em segundos	5.22	32.58	524%
Número de operações por segundo	1914	306	-84%

Conforme demostrado na Tabela 1, os resultados do teste de carga no ambiente centralizado, foi executado 524% mais rápido em comparação ao ambiente distribuído. E foi executado em média 84% a mais de operações por segundo em relação ao ambiente distribuído.

Tabela 2 - Tabela dos resultados do teste de execução de dados

Ambiente do teste de execução	Centralizado	Distribuído	Diferença em porcentagem.
Tempo de execução em segundos	4.58	21.30	365%
Número de operações por segundo	2181	469	-78%

Conforme demostrado na Tabela 1, os resultados do teste de execução no ambiente centralizado, foi executados 365% mais rápido que o ambiente distribuído e executou em média 78% a mais operações em um segundo em relação ao ambiente distribuído.

Foi observado que, conforme explicado anteriormente, o ambiente de teste distribuído não apresentou um ganho real de desempenho, devido a forma de implementação através de vários processos em um único computador, sofrendo gargalos de desempenho de escrita e

leitura de dados em disco, visto que tal procedimento deve ser executado com vários computadores em rede, em compensação a replicação dos dados entre processos pode ser demostrada através do monitoramento do banco de dados, e foi comprovado a disponibilidade dos dados entre processos.

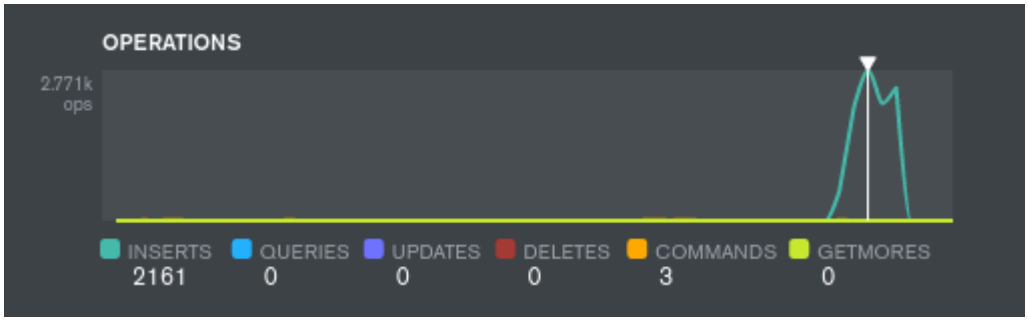
4.3.1 Teste ambiente centralizado

Durante a execução dos testes com a ferramenta *YCSB*, o ambiente do banco de dados distribuído foi monitorado com a ferramenta *MongoDB Compass* e apresentou os resultados descritos a seguir.

A Figura 14

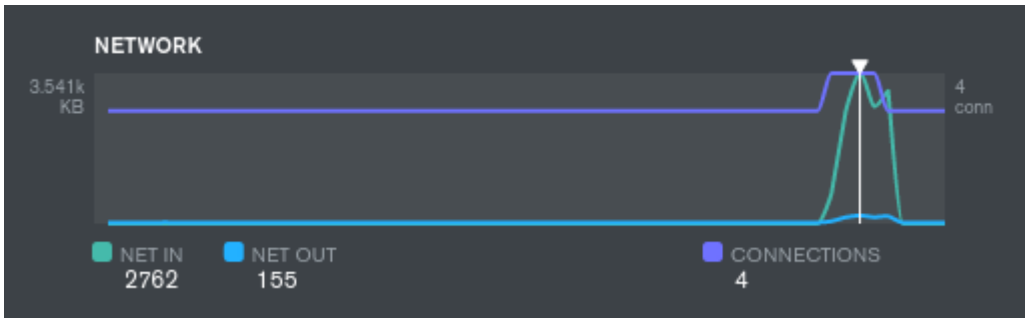
Figura 14 apresenta o gráfico da quantidade de operações executadas durante o teste de carga no ambiente centralizado.

Figura 14 - Gráfico de operações executadas no teste de carga no *MongoDB* centralizado



Conforme ilustrado na Figura 14, o gráfico apresenta a quantidade máxima operações executadas durante o teste de carga de dados, onde foram inseridos 2.161 registros e 3 comandos. A Figura 15 apresenta o gráfico do consumo de rede durante o teste de carga de dados no ambiente centralizado.

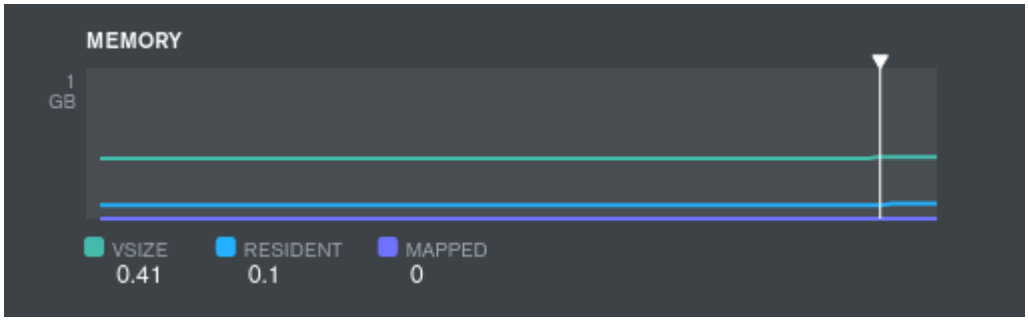
Figura 15 – Gráfico do consumo de rede em teste de carga no *MongoDB* centralizado



Conforme ilustrado na Figura 15, o gráfico mostra o consumo máximo de rede durante o teste de carga de dados, onde foram recebidos 2.7 *megabytes* e enviado 155 *kilobytes* em 4

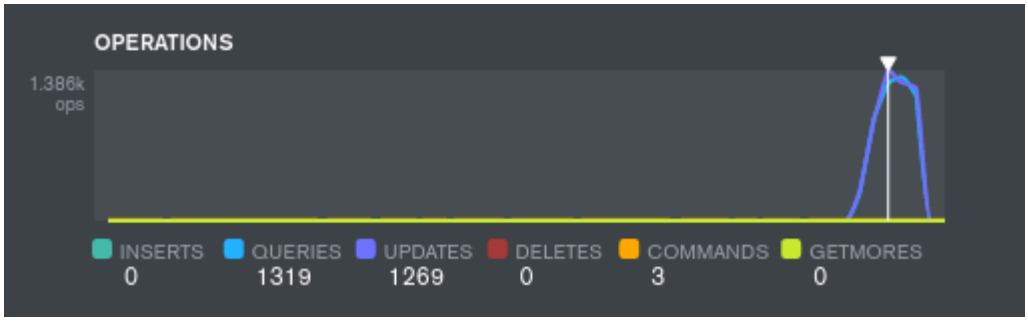
conexões com o banco de dados. A Figura 16 apresenta o gráfico do consumo de memória *RAM* durante o teste de carga de dados no ambiente centralizado.

Figura 16 – Gráfico do consumo de memória em teste de carga no *MongoDB* centralizado



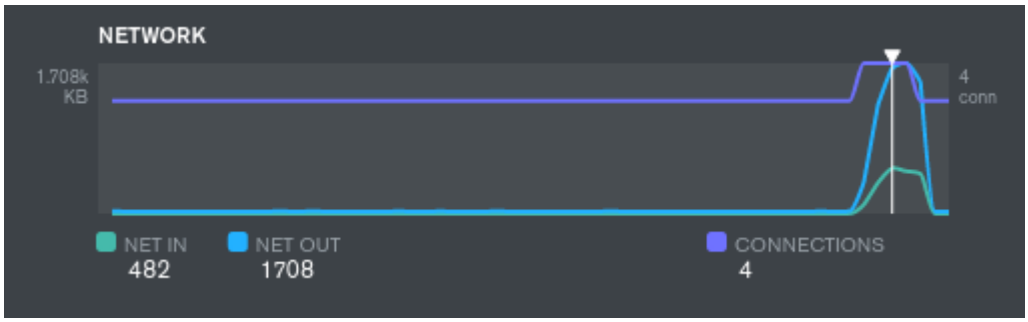
Conforme apresentado na Figura 16, o processo do banco de dados consumiu 41 *megabytes* da memória *RAM* e utilizou 10 *megabytes* como cache para os documentos. A Figura 17 ilustra o gráfico da quantidade de operações executadas durante o teste de execução no ambiente centralizado.

Figura 17 – Gráfico de operações em teste de execução no *MongoDB* centralizado



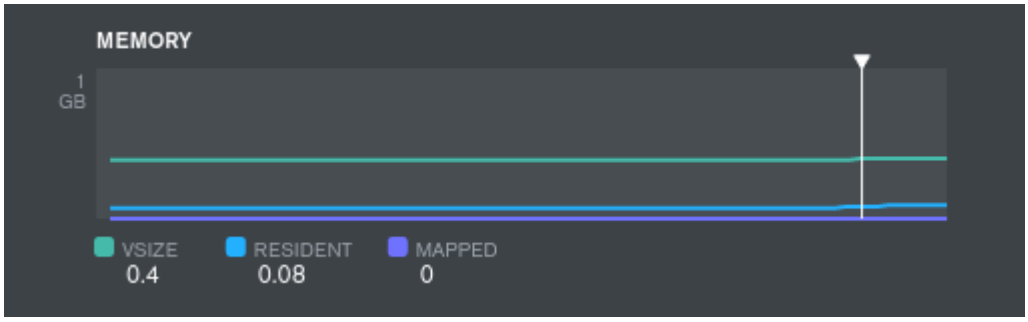
Conforme ilustrado na Figura 17, o gráfico apresenta o máximo operações durante o teste de execução de dados, onde foram executados 1.319 transações de consulta de dados, 1.269 transações de alteração de dados e 3 comandos executados. A Figura 18 mostra o gráfico do consumo de rede durante o teste de execução de dados no ambiente centralizado.

Figura 18 - Gráfico do consumo de rede em teste de execução no *MongoDB* centralizado



Conforme ilustrado na Figura 18, o gráfico apresenta o consumo máximo de rede durante o teste de execução de dados, onde foram recebidos 482 *kilobytes* e enviado 1.708 *megabytes* em 4 conexões com o banco de dados. A Figura 19 apresenta o gráfico do consumo de memória *RAM* durante o teste de execução de dados no ambiente centralizado.

Figura 19 – Gráfico do consumo de memória em teste de execução no *MongoDB* centralizado

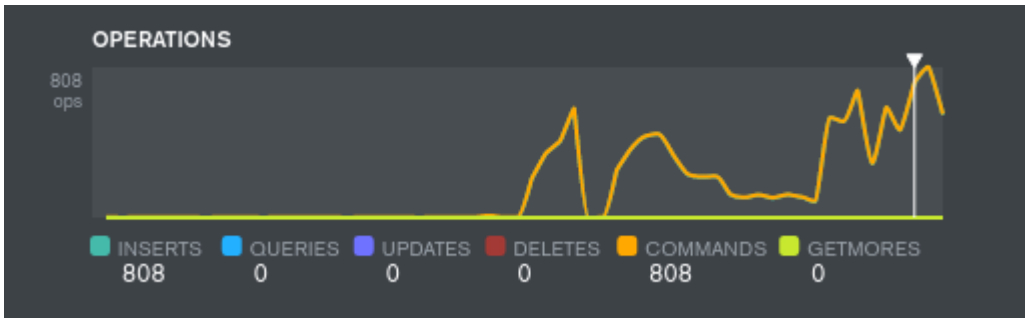


Conforme apresentado na Figura 19, o processo do banco de dados consumiu 40 *megabytes* da memória *RAM* e utilizou 8 *megabytes* para cache de documentos.

4.3.2 Testes ambiente distribuído

Durante a execução dos testes com a ferramenta *YCSB*, o ambiente distribuído foi monitorado com a ferramenta *MongoDB Compass*, e apresentou os seguintes resultados. A Figura 20 ilustra a quantidade de operações executadas durante o teste de carga no ambiente distribuído no processo do *MongoDB Query Router*.

Figura 20 – Gráfico de operações em teste de carga no *MongoDB Query Router*

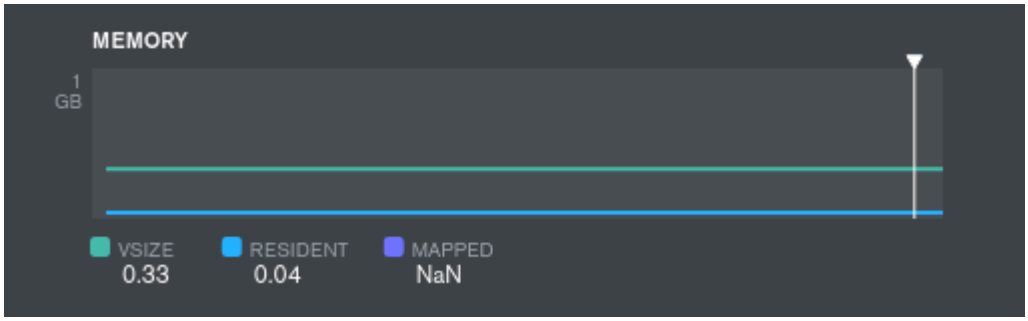


Conforme ilustrado na Figura 20, o gráfico apresenta a quantidade máxima de operações durante o teste de carga de dados, onde foram replicadas 808 transações de inserções de dados e 808 comandos executados de envio de transações para o *config server* (C1). A Figura 21 apresenta o gráfico de consumo de rede durante o teste de carga de dados no ambiente distribuído no *MongoDB Query Router*.

Figura 21- Gráfico do consumo de rede em teste de carga no *MongoDB Query Router*

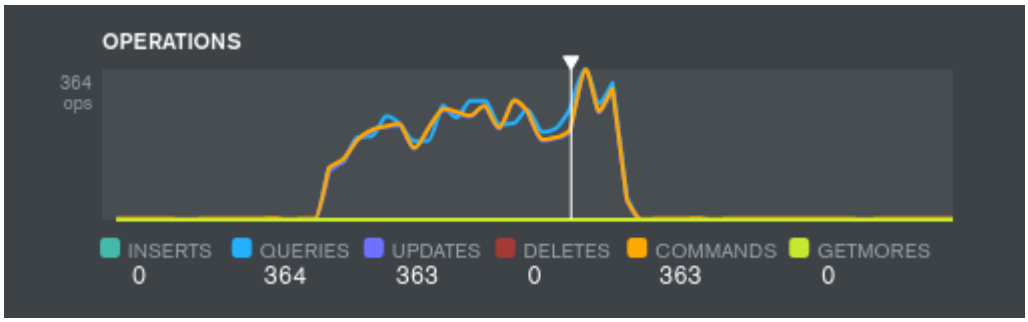
Conforme ilustrado na Figura 21, o gráfico mostra o consumo máximo de rede durante o teste de carga de dados, onde foram recebidos 1.031 megabytes e enviado 69 *kilobytes* em 6 conexões com o banco de dados. A Figura 22 apresenta o gráfico do consumo de memória *RAM* durante o teste de carga de dados no ambiente distribuído no *MongoDB Query Router*.

Figura 22 – Gráfico do consumo de memória em teste de carga no *MongoDB Query Router*



Conforme ilustrado na Figura 22, o processo do banco de dados consumiu 33 *megabytes* da memória *RAM* e utilizou 4 *megabytes* como cache de documentos. A Figura 23 ilustra a quantidade máxima de operações executadas durante o teste de execução no ambiente distribuído no *MongoDB Query Router*.

Figura 23 – Gráfico de operações em teste de execução no *MongoDB Query Router*



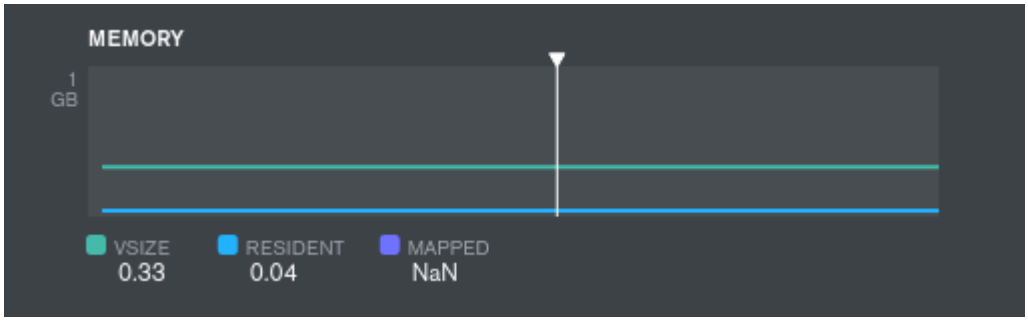
Conforme ilustrado na Figura 23, o gráfico apresenta a quantidade máxima de operações durante o teste de execução de dados, onde foram executadas 364 transações de consulta de dados, 363 transações de atualização de dados em 363 comandos, tais comandos são transações

enviadas para o ***config server***. A Figura 24 apresenta o consumo máximo de rede durante o teste de execução de dados no ambiente distribuído no *MongoDB Query Router*.

Figura 24 – Gráfico do consumo de rede em teste de execução no *MongoDB Query Router*

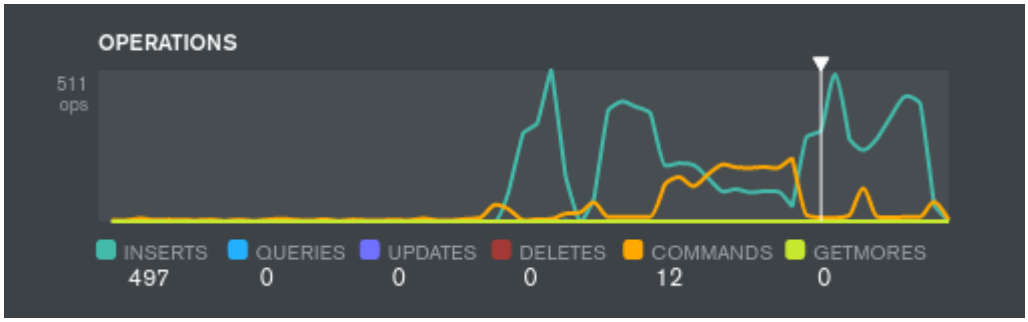
Conforme ilustrado na Figura 24, o gráfico ilustra o consumo máximo de rede durante o teste de execução de dados, onde foram recebidos 137 *kilobytes* e enviado 485 *kilobytes* em 3 conexões com o banco de dados. A Figura 25 apresenta o gráfico do consumo máximo de memória *RAM* durante o teste de execução de dados no ambiente distribuído no *MongoDB Query Router*.

Figura 25 – Gráfico do consumo de memória em teste de execução no *MongoDB Query Router*



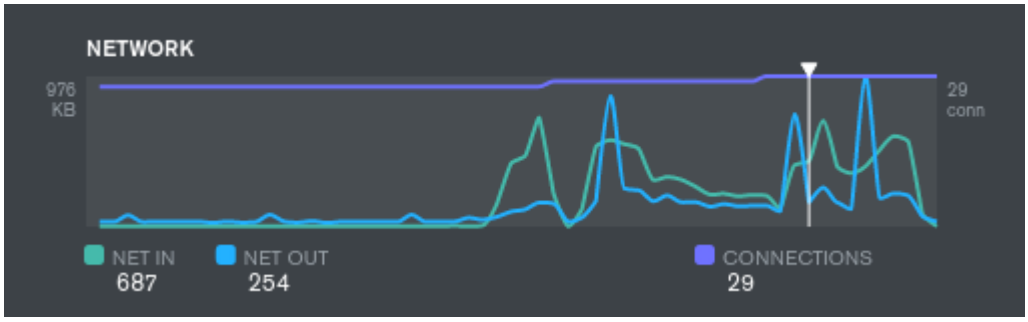
Conforme ilustrado na Figura 25, o processo do banco de dados consumiu 33 *megabytes* da memória *RAM* e utilizou 4 *megabytes* como cache de documentos. A Figura 26 ilustra o gráfico da quantidade máxima de operações executadas durante o teste de carga no ambiente distribuído no processo do *MongoDB Shard RS1*.

Figura 26 – Gráfico de operações em teste de carga no *MongoDB Shard RS1*



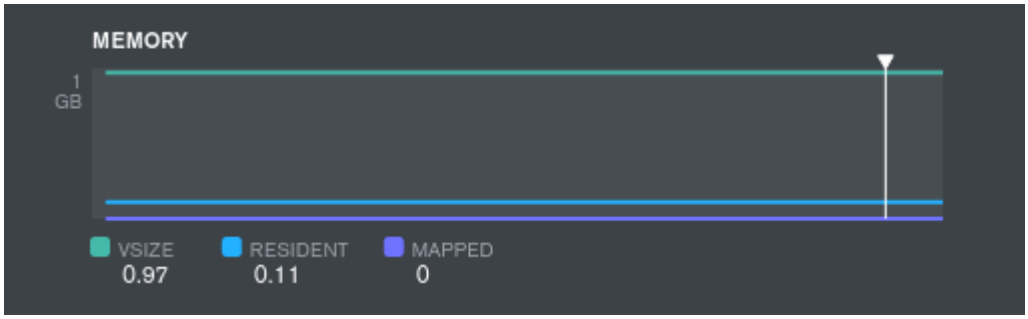
Conforme ilustrado na Figura 26, o gráfico apresenta a quantidade máxima de operações durante o teste de carga de dados, onde 497 transações de inserções de dados foram executas em 12 comandos executados. A Figura 27 apresenta o consumo máximo de rede durante o teste de carga de dados no ambiente distribuído no processo do *MongoDB Shard RS1*.

Figura 27 – Gráfico do consumo de rede em teste de carga no *MongoDB Shard* RS1



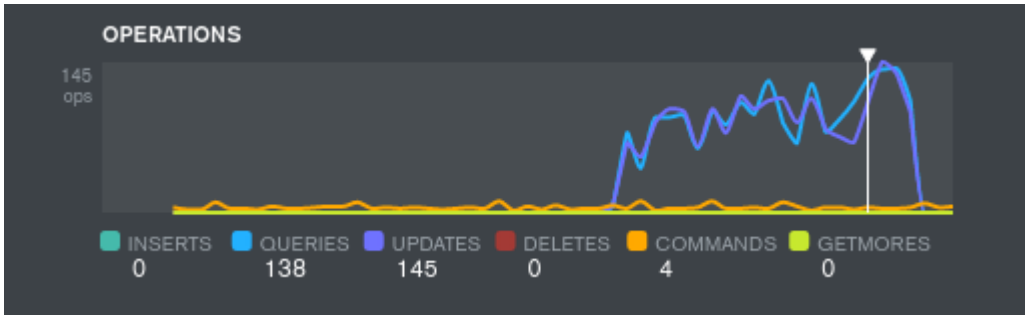
Conforme ilustrado na Figura 27, o gráfico demostra o consumo máximo de rede durante o teste de carga de dados, onde foram recebidos 687 *kilobytes* e enviado 254 *kilobytes* em 29 conexões com o banco de dados. A Figura 28 mostra o gráfico do consumo de memória *RAM* durante o teste de carga de dados no ambiente distribuído no processo do *MongoDB Shard rs1*

Figura 28 – Gráfico do consumo de memória em teste de carga no *MongoDB Query Shard RS1*



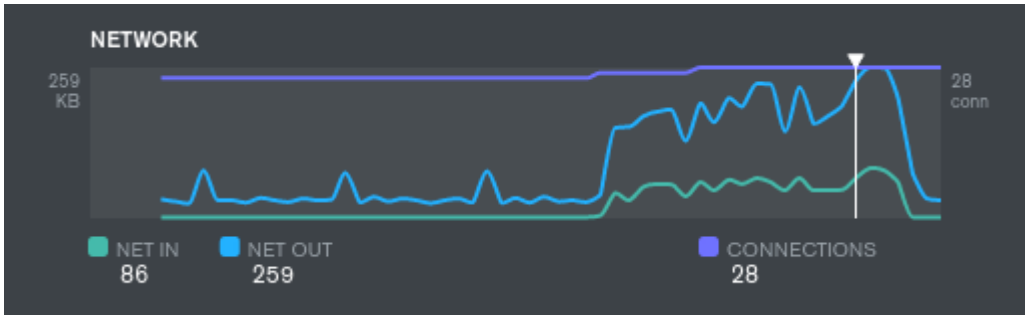
Conforme ilustrado na Figura 28, o processo do banco de dados consumiu 97 *megabytes* da memória *RAM* e utilizou 11 *megabytes* como cache de documentos. A Figura 29 apresenta o gráfico de operações executadas durante o teste de execução no ambiente distribuído no processo do *MongoDB Shard RS1*.

Figura 29 – Gráfico de operações em teste de execução no *MongoDB Shard RS1*



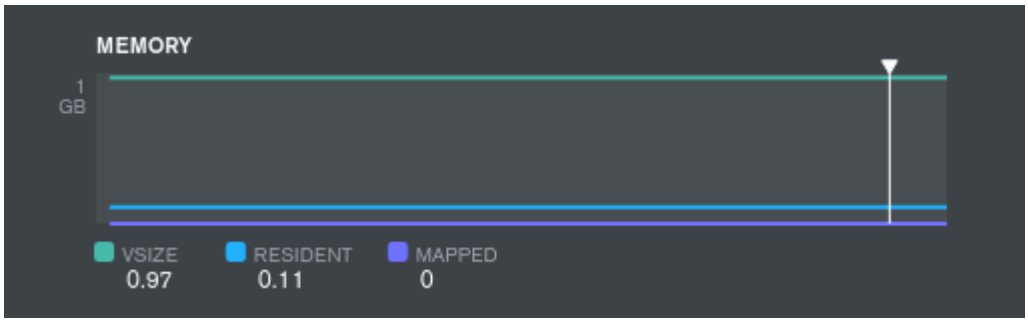
Conforme ilustrado na Figura 29, o gráfico apresenta a quantidade máxima de operações durante o teste de execução de dados, onde foram realizadas 138 transações de consulta de dados, 145 transações de atualização de dados e 4 comandos executados. A Figura 30 ilustra o gráfico do consumo de rede durante o teste de execução de dados no ambiente distribuído no *MongoDB Shard RS1*.

Figura 30 – Gráfico do consumo de rede em teste de execução no *MongoDB Shard RS1*



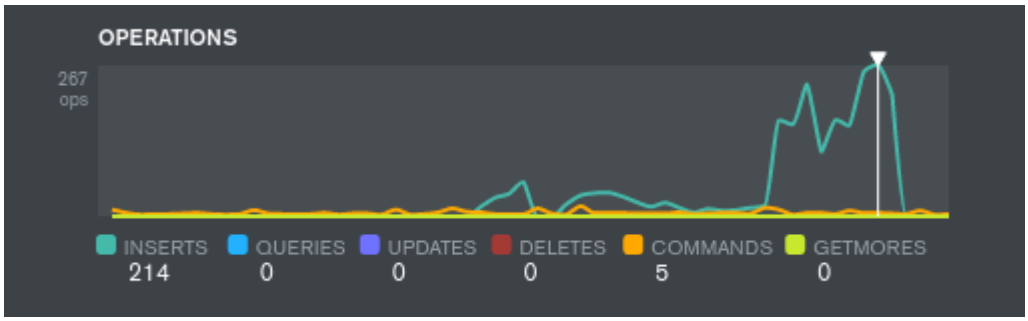
Conforme ilustrado na Figura 30, o gráfico apresenta o consumo máximo de rede durante o teste de execução de dados, onde foram recebidos 86 *kilobytes* e enviado 259 *kilobytes* em 28 conexões com o banco de dados. A Figura 31 mostra o gráfico do consumo de memória *RAM* durante o teste de execução de dados no ambiente distribuído no processo do *MongoDB Shard RS1*.

Figura 31 – Gráfico do consumo de memória em teste de execução no *MongoDB Query Shard RS1*



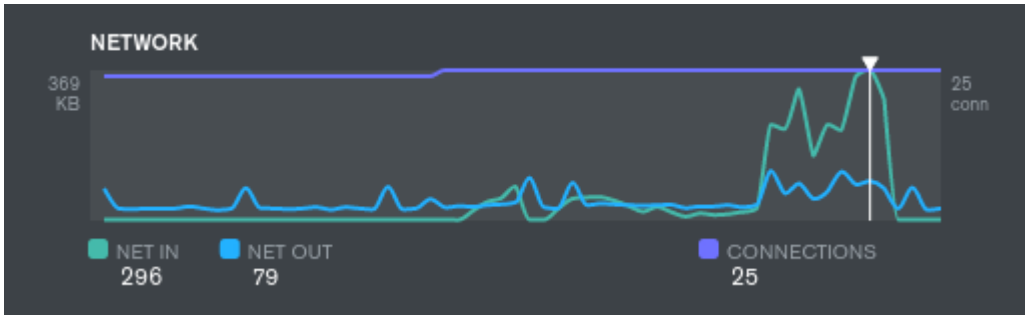
Conforme ilustrado na Figura 31, o processo do banco de dados consumiu 97 *megabytes* da memória *RAM* e utilizou 11 *megabytes* como cache de documentos. A Figura 32 ilustra o gráfico do máximo de operações executadas durante o teste de carga no ambiente distribuído no processo do *MongoDB Shard RS2*.

Figura 32 – Gráfico de operações em teste de carga no *MongoDB Shard RS2*



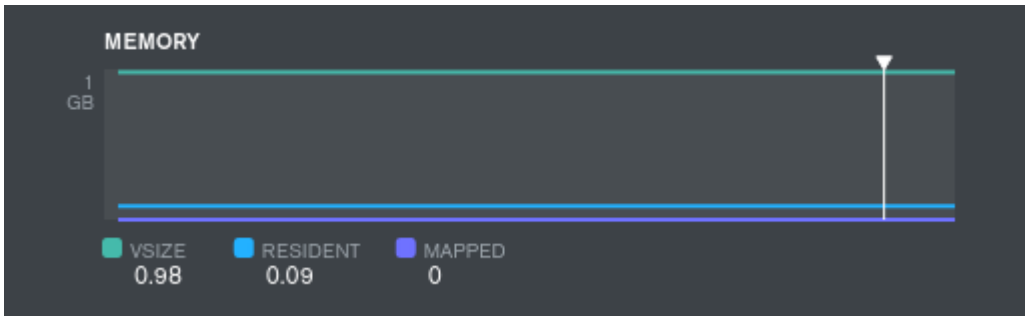
Conforme ilustrado na Figura 32, o gráfico mostra a quantidade máxima de operações durante o teste de carga de dados, onde foram realizadas 214 transações de inserção de dados e 5 comandos executados. A Figura 33 apresenta o gráfico do consumo de rede durante o teste de carga de dados no ambiente distribuído no *MongoDB Shard RS2*.

Figura 33 - Gráfico do consumo de rede em teste de carga no *MongoDB Shard RS2*



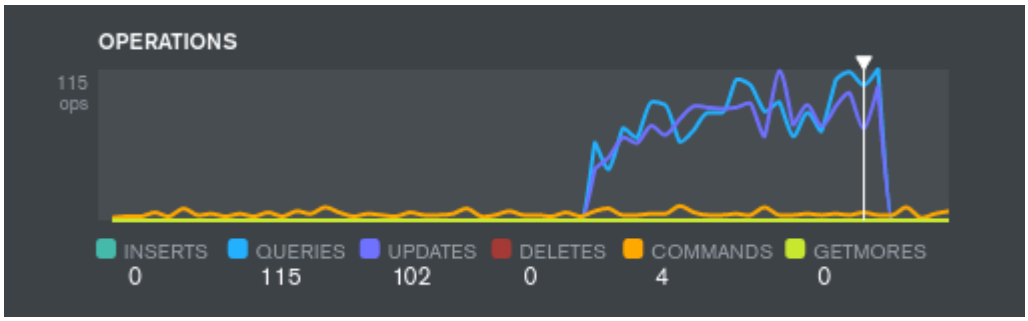
Conforme ilustrado na Figura 33, o gráfico ilustra o consumo máximo de rede durante o teste de carga de dados, onde foram recebidos 296 *kilobytes* e enviado 79 *kilobytes* em 25 conexões com o banco de dados. A Figura 34 apresenta o consumo de memória *RAM* durante o teste de execução de dados no ambiente distribuído no processo do *MongoDB Shard RS2*.

Figura 34 – Gráfico do consumo de memória em teste de carga no *MongoDB Query Shard RS2*



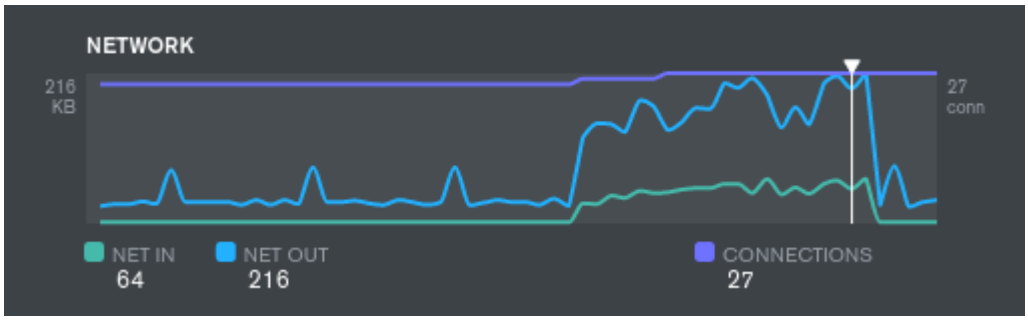
Conforme ilustrado na Figura 34, o processo do banco de dados consumiu 98 *megabytes* da memória *RAM* e utilizou 9 *megabytes* como cache de documentos. A Figura 35 ilustra a quantidade máxima de operações executadas durante o teste de execução no ambiente distribuído no processo do *MongoDB Shard RS2*.

Figura 35 – Gráfico de operações em teste de execução no *MongoDB Shard RS2*



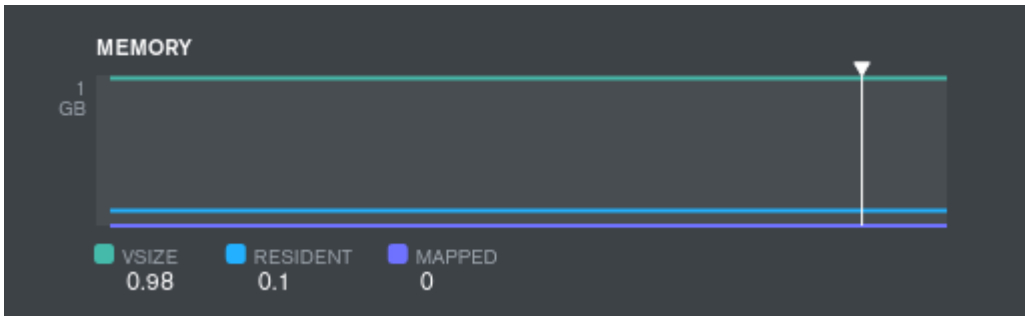
Conforme ilustrado na Figura 35, o gráfico apresenta a quantidade máxima de operações durante o teste de execução de dados, onde foram realizadas 115 transações de consulta de dados, 102 transações de atualizações de dados e 4 comandos executados. A Figura 36 apresenta o consumo de rede durante o teste de execução de dados no ambiente distribuído no *MongoDB Shard RS2*.

Figura 36 – Gráfico do Consumo de rede em teste de execução no *MongoDB Shard RS2*



Conforme ilustrado na Figura 36, o gráfico mostra o consumo máximo de rede durante o teste de execução de dados, onde foram recebidos 64 *kilobytes* e enviado 216 *kilobytes* em 27 conexões com o banco de dados. A Figura 37 apresenta o gráfico do consumo de memória *RAM* durante o teste de execução de dados no ambiente distribuído no processo do *MongoDB Shard RS2*.

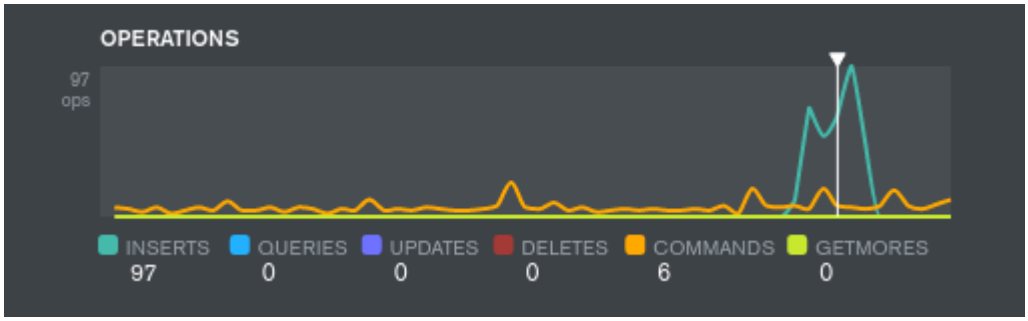
Figura 37 – Gráfico do consumo de memória em teste de execução no *MongoDB Query Shard RS2*



Conforme ilustrado na Figura 37, o processo do banco de dados consumiu 98 *megabytes* da memória *RAM* e utilizou 10 *megabytes* como cache de documentos. A Figura 38 ilustra o

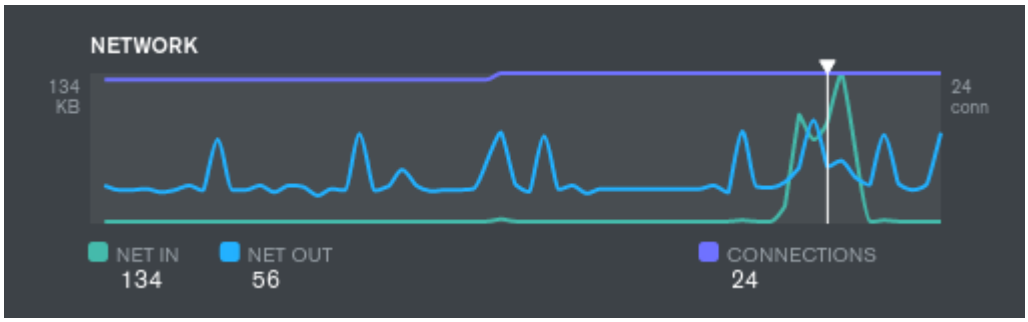
gráfico do máximo de operações executadas durante o teste de carga no ambiente distribuído no processo do *MongoDB Shard RS3*.

Figura 38 – Gráfico de operações em teste de carga no *MongoDB Shard RS3*



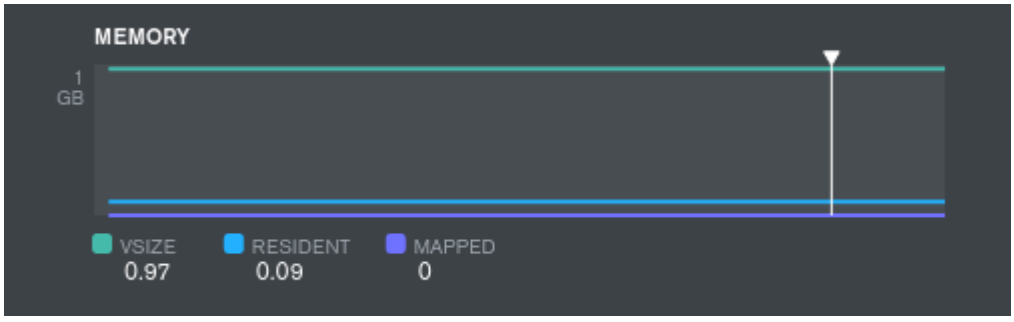
Conforme apresentado na Figura 38, o gráfico apresenta o máximo operações durante o teste de carga de dados, onde foram realizadas 97 transações de inserção de dados e 6 comandos executados. A Figura 39 apresenta o gráfico do consumo de rede durante o teste de carga de dados no ambiente distribuído no *MongoDB Shard RS3.*

Figura 39 – Gráfico do consumo de rede em teste de carga no *MongoDB Shard RS3*



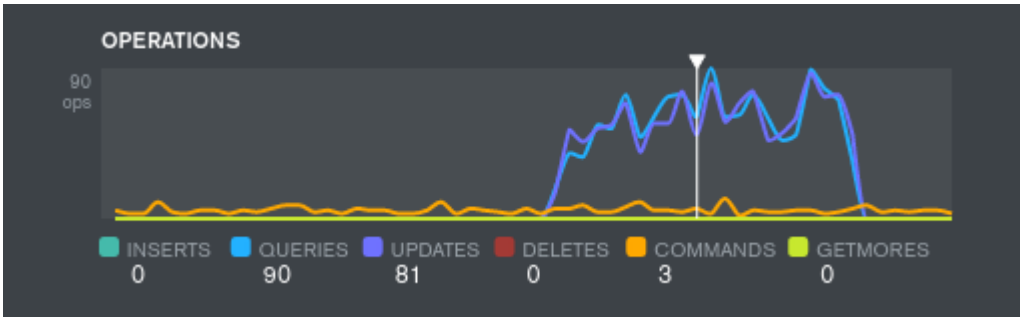
Conforme ilustrado na Figura 39, o gráfico apresenta o consumo máximo de rede durante o teste de execução de dados, onde foram recebidos 134 *kilobytes* e enviado 56 *kilobytes* em 24 conexões com o banco de dados. A Figura 40 apresenta o gráfico do consumo de memória *RAM* durante o teste de carga de dados no ambiente distribuído no processo do *MongoDB Shard RS3.*

Figura 40 – Gráfico do consumo de memória em teste de carga no *MongoDB Query Shard RS3*



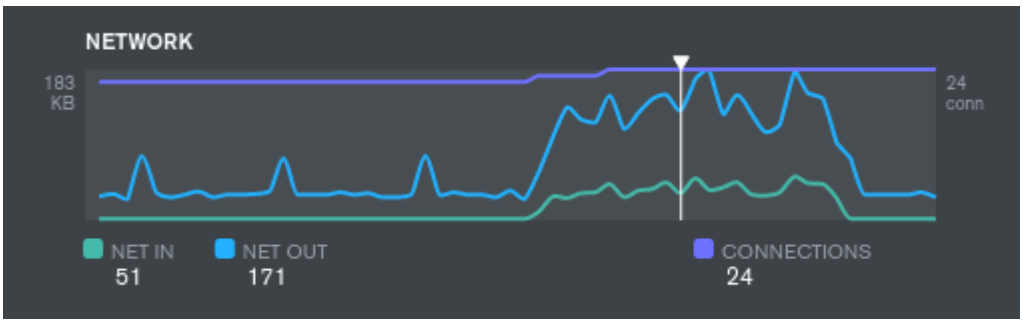
Conforme ilustrado na Figura 40, o processo do banco de dados consumiu 97 *megabytes* da memória *RAM* e utilizou 9 *megabytes* como cache de documentos. A Figura 41 ilustra a quantidade máxima de operações executadas durante o teste de execução no ambiente distribuído no processo do *MongoDB Shard RS3*.

Figura 41 – Gráfico de operações em teste de execução no *MongoDB Shard RS*3



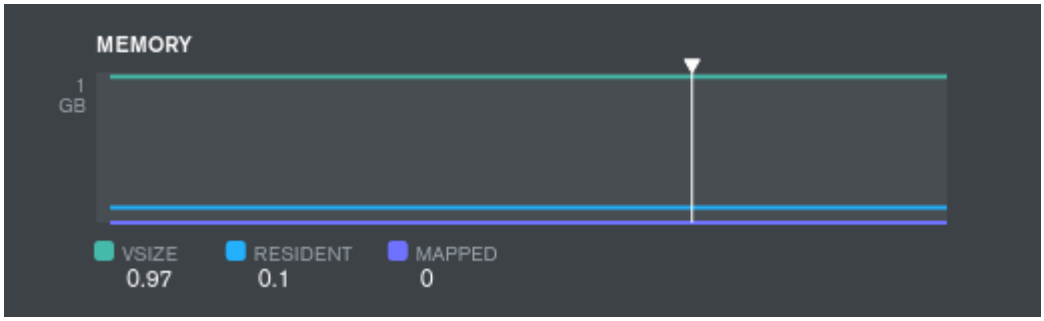
Conforme ilustra a Figura 41, o gráfico apresenta o máximo operações durante o teste de execução de dados, onde foram realizadas 90 transações de consulta de dados, 81 transações de atualização de dados e 3 comandos executados. A Figura 42 apresenta o gráfico do consumo de rede durante o teste de execução de dados no ambiente distribuído no *MongoDB Shard RS3*.

Figura 42 – Gráfico do consumo de rede em teste de execução no *MongoDB Shard RS3*



Conforme ilustrado na Figura 42, o gráfico apresenta o consumo máximo de rede durante o teste de execução de dados, onde foram recebidos 51 *kilobytes* e enviado 171 *kilobytes* em 24 conexões com o banco de dados. A Figura 43 ilustra o consumo de memória *RAM* durante o teste de execução de dados no ambiente distribuído no processo do *MongoDB Shard RS3*.

Figura 43 – Gráfico do consumo de memória em teste de execução no *MongoDB Query Shard RS3*



Conforme ilustrado na Figura 43, o processo do banco de dados consumiu 97 *megabytes* da memória *RAM* e utilizou 10 *megabytes* como cache de documentos.

5 CONSIDERAÇÕES FINAIS

Foi concluído que a utilização da tecnologia de *NoSQL* para o módulo de armazenamento de dados se mostrou vantajoso, pois apresentou algumas características fundamentais de armazenamento de dados para a plataforma, como a escalabilidade horizontal e alto desempenho em consultas distribuídas, porém a forma que foi implementado o ambiente distribuído não se comprovou um real ganho de desempenho na persistência dos dados. A escolha de um banco de dados orientado a documentos permitiu a plataforma trabalhar com vários tipos de dados, devido flexibilização do modelo de dados adotado pelo banco de dados *MongoDB*.

A utilização de um ambiente distribuído em vários computadores executando o *MongoDB* irá garantir o crescimento da plataforma conforme necessário, visto que com a adição de mais computadores ganha-se mais desempenho de processamento e capacidade de armazenamento conforme for adicionado mais computadores, consequentemente melhora-se a disponibilidade dos dados, replicados entre as bases de dados que compõem a camada de persistência para a aplicação. Outra vantagem observada foi a portabilidade do *MongoDB*, visto que facilitou a incorporação computadores com sistemas operacionais heterogêneos no modulo de armazenamento de dados.

Mesmo que os testes realizados tenham demonstrado resultados inferiores no ambiente distribuído, devido a indisponibilidade de equipamentos exclusivo para o trabalho, a implementação do modelo de dados foi realizada em um único computador, porém a distribuição de tarefas e replicação dos dados entre processos se mostrou positiva, ganhando-se disponibilidade dos dados. Em um ambiente contendo vários computadores exclusivo para banco de dados será teoricamente melhor, pois as operações de escrita e leitura serão fragmentadas entre os mesmos, trazendo em uma melhor performance geral da base de dados.

Tais procedimentos de teste comprovou a eficiência do banco de dados *MongoDB*, na distribuição dos dados em diversas base de dados, para aumentar a disponibilidade dos dados e fragmentar o processamento dos dados em diversos computadores, comprovando que o desenvolvimento deste trabalho se torna viável para servir como uma estrutura de armazenamento de dados de alto desempenho em uma plataforma de análise e manipulação de dados.

6 REFERÊNCIAS BILIOGRÁFICAS

DE DIANA, Mauricio; GEROSA, Marco Aurélio. Nosql na web 2.0: Um estudo comparativo de bancos não relacionais para armazenamento de dados na web 2.0. In: **IX Workshop de Teses e Dissertações em Banco de Dados**. 2010.

STROZZI, Carlos,**Nosql-arelationaldatabase management system**. 1998. Disponível em:http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/Home%20Page Acesso em 3 de abril de 2017.

CALDEIRA, Filipe. **Introdução aos Sistemas de Gestão de Bases de Dados Microsoft ACCESS.** 2004. Disponível em: http://www.estgv.ipv.pt/paginaspessoais/jloureiro/gcp_inf22002_2003/sebenta/seb_cap5_2.pdf Acesso em 4 de abril de 2017.

BREWER, Eric. **CAP twelve years later: How the" rules" have changed. Computer**, v. 45, n. 2, p. 23-29, 2012.

POKORNY, Jaroslav. **NoSQL databases: a step to database scalability in web environment. International Journal of Web Information Systems,** v. 9, n. 1, p. 69-82, 2013.

STEPPAT, Nico. **NoSQL – Do teorema CAP para P?(A|C):(C|L).** 2011. Disponível em http://blog.caelum.com.br/nosql-do-teorema-cap-para-paccl/ Acesso em 5 de abril de 2017.

DE SOUZA, Alexandre Morais et al. Critérios para Seleção de SGBD NoSQL: o Ponto de Vista de Especialistas com base na Literatura. **Anais doX Simpósio Brasileiro de Sistemas de Informação** Londrina–PR, Brasil. 27 a 30/05/2014.

MONIRUZZAMAN, A. B. M.; HOSSAIN, Syed Akhter. **Nosql database: New era of databases for big data analytics-classification, characteristics and comparison.** International Journal of Database Theory and Application Vol. 6, No. 4. 2013.

LEAVITT, Neal. **Will NoSQL databases live up to their promise?** Computer, v. 43, n. 2, 2010.

TIWARI, Shashank; **ProfessionalNoSQL**. John Wiley & Sons, 2011. 14 p.

LEITE, Hermano Portella; BONOMO, Igor da Silva. **Análise comparativa de projeto e administração de banco de dados entre os SGBDs Cassandra e MySQL.** 2017. Monografia para Bacharelado em Ciência da Computação. Universidade de Brasília.

TOTH, Renato Molina. **Abordagem NoSQL–uma real alternativo**. Sorocaba, São Paulo, Brasil: abril, v. 13, 2011.

DOS SANTOS, Marcus Vinícius Carli; DE SOUZA, Vanessa Cristina Oliveira. **Amadurecimento, Consolidação e Performance de SGBDs NoSQL-Estudo Comparativo.** XI Brazilian Symposium on Information System, Goiânia-GO, Brasil 26 a 29/05/2015.

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

JSON, **Introdução ao JSON**, 2000. Disponível em: http://www.json.org/json-pt.html Acesso em: 20 de março de 2017.

MDN, **JSON**, 2017. Disponível em: https://developer.mozilla.org/pt-BR/docs/JSON

NOSQL, **NOSQL Databases**, 2010. Disponível em http://nosql-database.org/ Acesso em: 25 de março de 2017.

DB-ENGINES**, DB-Engines Ranking - popularity ranking of database management systems**.2017. Disponível em: https://db-engines.com/en/ranking Acesso em: 31 de maio de 2017

LÓSCIO, Bernadette Farias; OLIVEIRA, Hélio Rodrigues de; PONTES, Jonas César de Sousa. NoSQL no desenvolvimento de aplicações Web colaborativas. **VIII Simpósio Brasileiro de Sistemas Colaborativos**, v. 10, p. 11, 2011.

NASCIMENTO, Matheus Bellio. MongoDB: **Um Estudo Teórico-Prático do Conceito de Banco de Dados NoSQL**, 2014.

ULLMAN, Jeffrey D. **Designing good mapreduce algorithms.**XRDS: Crossroads, The ACM Magazine for Students, v. 19, n. 1, p. 30-34, 2012.

GAJENDRAN, Santhosh Kumar. **A survey on nosql databases.**University of Illinois, 2012.

POPESCU, Alex**. Learn about Conflict Resolution and Vector Clocks**, 2010.Disponívelem: http://nosql.mypopescu.com/post/524128219/learn-about-conflict-resolution-and-vector-clocksAcesso em: 27 de março de 2017.

WHITE, Tom. **ConsistentHashing.** Disponível em http://www.tom-e-white.com/2007/11/consistent-hashing.htmlAcesso em: 11 de maio de 2017.

VAN ERVEN, Gustavo C. Galvão. MDG-NoSQL: **Modelo de Dados para Bancos NoSQL Baseados em Grafos.** 2015. Tese de Doutorado. Universidade de Brasília.

VIEIRA, Marcos Rodrigues et al. **Bancos de Dados NoSQL: conceitos, ferramentas, linguagens e estudos de casos no contexto de Big Data. Simpósio Brasileiro de Bancos de Dados**, 2012.

VOGELS,Werner, **EventuallyConsistent – Revisited**, 2008. Disponível em: http://www.allthingsdistributed.com/2008/12/eventually_consistent.htmlAcesso em: 26 de março de 2017.

SOUSA, **O teorema CAP**, 2010. Disponível em: http://unrealps.wordpress.com/2010/12/28/o-teorema-cap/Acesso em: 10 de maio de 2017.

WEI, Zhou; PIERRE, Guillaume; CHI, Chi-Hung. **Scalable transactions for web applications in the cloud.** In: European Conference on Parallel Processing. Springer Berlin Heidelberg, 2009. p. 442-453.

BROWNE Julian. Brewer's **CAP Theorem**, 2009. Disponível em: http://www.julianbrowne.com/article/viewer/brewers-cap-theorem. Acessoem: 11 de maio de 2017.

GILBERT, Seth; LYNCH, Nancy. **Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services**. AcmSigact News, v. 33, n. 2, p. 51-59, 2002.

MONSON-HAEFEL, Richard. **97 Things Every Software Architect Should Know.** Collective Wisdom from the Experts, O'Reilly Media,1.ed., p. 77, 2009.

GILBERT, Seth; LYNCH, Nancy. **Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.**AcmSigact News, v. 33, n. 2, p. 51-59, 2002.

MongoDB. **Documents - MongoDB Manual 3.4**, 2017a. Disponível em: https://docs.mongodb.com/manual/core/document/, Acessado em 01 jun. 2017 as 12h45m.

MongoDB. **BSON Types - MongoDB Manual 3.4**, 2017b. Disponível em: https://docs.mongodb.com/manual/reference/bson-types/, Acesso em 01 jun. 2017 as 13h45m.

MongoDB. **ObjectId - MongoDB Manual 3.4**, 2017c. Disponível em: https://docs.mongodb.com/manual/reference/method/ObjectId/, acesso em 01 jun. 2017 as 14h45m.

MongoDB. **Replication - MongoDB Manual 3.4**, 2017d. Disponível em: https://docs.mongodb.com/manual/replication/, acesso em 01 jun. 2017 as 14h45m.

MongoDB. **Sharding- MongoDB Manual 3.4**, 2017e. Disponível em: https://docs.mongodb.com/manual/sharding/, acesso em 01 jun. 2017 as 15hm.

MongoDB. **MongoDB Compass | MongoDB**, 2017f. Disponível em: https://www.mongodb.com/products/compass, acesso em 02 jun. 2017 as 16h30m.

MongoDB.**Introduction to MongoDB — MongoDB Manual 3.4,** 2017h. Disponível em: https://docs.mongodb.com/manual/introduction/, acesso em 03 jun. 2017 as 16h30m.

MongoDB. **What is NoSQL?**. 2017i. Disponível em: https://www.mongodb.com/nosql-explained, acesso em 04 jun. 2017 as 16h30m.

MongoDB.**Data Partitioning with Chunks — MongoDB Manual 3.4,** 2017j. Disponível em: https://docs.mongodb.com/manual/core/sharding-data-partitioning/, acesso em 05 jun. 2017 as 16h30m.

YCSB. **Yahoo! Cloud Serving Benchmark**, 2017. Disponível em https://github.com/brianfrankcooper/YCSB, acesso em 07 jun 2017 as 14hrs