UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

RENAN FARAON

BANCOS DE DADOS NEWSQL: CONCEITOS, FERRAMENTAS E COMPARATIVO PARA GRANDES QUANTIDADES DE DADOS

CAXIAS DO SUL
2018

RENAN FARAON

BANCOS DE DADOS NEWSQL: CONCEITOS, FERRAMENTAS E COMPARATIVO PARA GRANDES QUANTIDADES DE DADOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientadora: Prof. Dra. Helena G. Ribeiro

CAXIAS DO SUL

2018

RENAN FARAON

BANCOS DE DADOS NEWSQL: CONCEITOS, FERRAMENTAS E COMPARATIVO PARA GRANDES QUANTIDADES DE DADOS

Trabalho de Conclusão de Curso apresentado como requisito para a obtenção do grau de Bacharel em Ciência da Computação da Universidade de Caxias do Sul.

Aprovado em: 28/11/2018.

Banca Examinadora

_____
Prof. Dra. Helena Graziottin Ribeiro
Universidade de Caxias do Sul

_____
Prof. Dr. Daniel Luis Notari
Universidade de Caxias do Sul

_____
Prof. Me. Alexandre Erasmo Krohn Nascimento
Universidade de Caxias do Sul

RESUMO

Os bancos de dados e os sistemas gerenciadores de bancos de dados se tornaram componentes essenciais no cotidiano da sociedade moderna, devido a grande quantidade de dados gerados diariamente. Os bancos de dados relacionais e NoSQL, bastante utilizados atualmente, muitas vezes não oferecem performance e, ou, confiabilidades desejadas para as novas aplicações que utilizam grandes volumes de dados. Com o intuito de suprir essas demandas surgem os bancos de dados NewSQL. Esses sistemas compartilham grande parte das funcionalidades dos bancos de dados relacionais e dos bancos de dados NoSQL. O trabalho apresenta um estudo comparativo entre as características dos bancos de dados relacional, NoSQL e NewSQL, com base no estudo foi elabora uma avaliação com três sistemas de bancos de dados: MySQL do modelo relacional, Cassandra do modelo NoSQL e MemSQL e NuoDB do modelo NewSQL, utilizando a ferramenta de *benchmark* YCSB para a avaliação.

Palavras-chaves: Bancos de dados, NewSQL, NoSQL, Relacional, MySQL, Cassan NuoDB, MemSQL.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Os bancos de dados e os sistemas de bancos de dados se tornaram componentes essenciais no cotidiano da sociedade moderna (ELMASRI; NAVATHE, 2006). Antigamente as informações eram armazenadas em papéis, e acabava se tornando difícil encontrar as informações necessárias, essa forma de armazenamento evoluiu para arquivos digitais que facilitaram a consulta em alguns aspectos, mas ainda era complicado atualizar e manter os dados na medida em que o seu volume crescia.

Na década de 1970 surgiram os primeiros bancos de dados relacionais, introduzidos por Ted Codd, da IBM, que imediatamente atraíram a atenção em virtude de sua simplicidade e base matemática (ELMASRI, 2011). Um banco de dados relacional armazena dados em um conjunto de relações simples (Oracle, 2018c).

Os bancos de dados relacionais são os mais utilizados desde a sua criação, devido a oferecerem aos usuários confiabilidade, segurança e disponibilidade dentre outras características, permitindo os desenvolvedores de sistema se preocuparem mais com a parte da aplicação (DB-ENGINES, 2018).

A revolução da Internet na década de 1990 aumentou significativamente o acesso direto a bases de dados (SILBERSCHATZ, 2012). Houve um considerável crescimento do volume de dados, e percebe-se que os bancos de dados relacionais estavam chegando ao limite de sua capacidade de processamento e de escalabilidade.

Conforme Brito (2010), as tentativas de encontrar uma alternativa para o modelo relacional acabaram levando os desenvolvedores a uma estratégia em que optaram por se livrar de algumas estruturas do banco de dados relacional, mas, em contrapartida, ganhariam em performance, surgindo assim em 1998 a abordagem NoSQL.

NoSQL foi um termo usado inicialmente para denotar software de gestão de dados que não usa SQL (*Structured Query Language*) para interagir com o banco de dados (LENNON, 2011)

Os bancos de dados NoSQL acabaram ganhando performance e escalabilidade, diferente dos bancos de dados relacionais que estavam tendo problemas. Porém, esses benefícios acabam acarretando a perda das propriedades ACID: atomicidade, consistência, isolamento e durabilidade, as quais o banco de dados relacional tem como suas principais características.

Devido a falta das propriedades ACID nos bancos de dados NoSQL, manten-se uma consistência eventual, segundo Stonebraker (2011) sobrecarregando os aplicativos, pois eles é que deverão verificar a consistência, surge então o NewSQL como uma alternativa que une o melhor dos dois modelos, NoSQL e relacional: a familiaridade e a interatividade do modelo relacional, e a escalabilidade e velocidade do modelo NoSQL.

NewSQL é um termo usado para descrever um novo grupo de bancos de dados que compartilham grande parte da funcionalidade dos bancos de dados relacionais tradicionais do SQL, e que ao mesmo tempo oferecem alguns dos benefícios das tecnologias dos bancos de dados NoSQL (STONEBRAKER, 2011).

1.1 Questão de pesquisa

O NewSQL tende a ser promissor devido a combinar as vantagens dos bancos de dados relacionais, e dos bancos de dados NoSQL. Quais as diferenças entre os bancos de dados relacional, NoSQL e NewSQL, e que podem possibilitar que o NewSQL seja mais promissor para manipular grandes quantidades de dados?

1.2 Objetivos

Verificar através de experimentos quais seriam as vantagens que os bancos de dados NewSQL apresentam em relação aos sistemas de banco de dados relacionais e bancos de dados NoSQL, para processamento de grandes quantidades de dados.

2 BANCO DE DADOS RELACIONAL E NOSQL

Um banco de dados ou base de dados é uma coleção de dados ou informações relacionadas entre si. Elas representam aspectos do mundo real com significado próprio e que desejamos armazenar para uso futuro (GUIMARÃES, 2003).

2.1 BANCO RELACIONAL

O banco de dados relacional foi desenvolvido baseado em um modelo relacional, e na teoria dos conjuntos matemáticos, que armazena dados em um conjunto de relações simples (ORACLE, 2018c)

Codd propôs o modelo de dados relacional em 1970, de forma simples e elegante, na qual é uma coleção de uma ou mais relações, que cada relação consiste em uma coleção de tabelas (SILBERSCHATZ; KORTH; SUDARSHAN, 2012).

2.1.1 **Tabelas**

Segundo a Oracle (2018,c) uma tabela, Figura 1, descreve uma entidade, definida com um nome e um conjunto de colunas e linhas, na qual a IBM (2018) define cada componente da seguinte forma:

a) Tabela – objeto do banco de dados que contém uma coleção de dados, que consistem em linhas e colunas;
b) Coluna – componente vertical da tabela, que possui um nome e um tipo de dados específico, exemplo caractere, decimal, ou número inteiro;
c) Linha – componente horizontal da tabela, que consistem em uma sequência de valores, um para cada coluna da tabela.

Figura 1 - Representação de tabela

TABELA DE CLIENTES							
ID	DESCRICAO	CPF	CNPJ	INSC_EST	ENDERECO	BAIRRO	CIDADE_ID
1	ADILSON DIAS PINHEIRO ME		99999999999999	550042342	RUA CEL BRAULIO DE OLIVEIRA, 1979	CENTRO	15
2	ELIO LEAO	88888888888		ISENTO	ARI ALVES, 144	LIMEIRA	33
3	REDOL ALIMENTOS LTDA		77777777777777	9018695615	RUA FRANCISCO MUNOZ MADRID, 1583	ROSEIRA	16

Fonte: Elaborado pelo autor

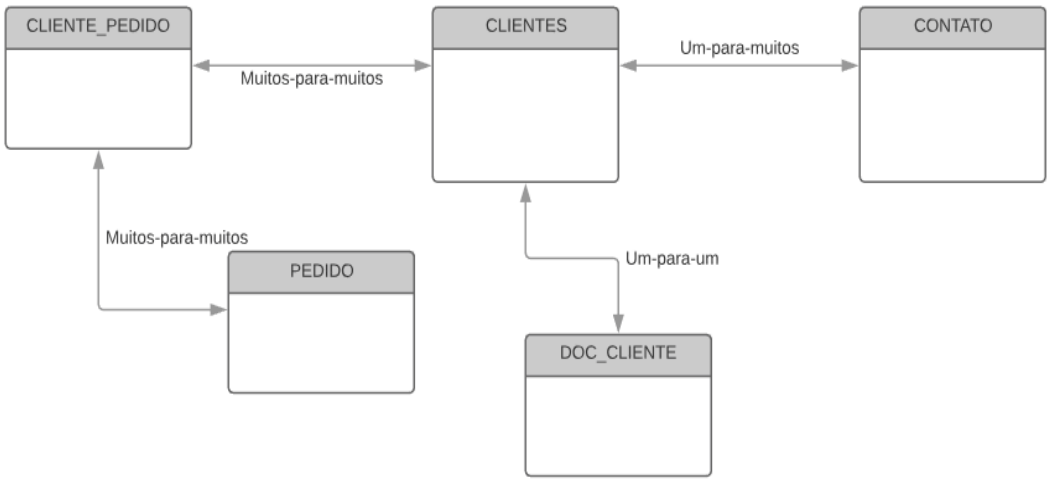
Uma das principais características do modelo relacional segundo Puga, França e Goya (2013), é a possibilidade de relacionar várias tabelas para evitar a redundância no armazenamento de dados, Figura 2. Podendo ocorrer de três formas diferentes:

a) Relacionamento um-para-um;

b) Relacionamento um-para-muitos;

c) Relacionamento muitos-para-muitos.

Figura 2 - Exemplo de relacionamentos



Fonte: Elaborado pelo autor

Para poder determinar essas relações precisamos identificar a linha referenciada, e para isso existem as chaves.

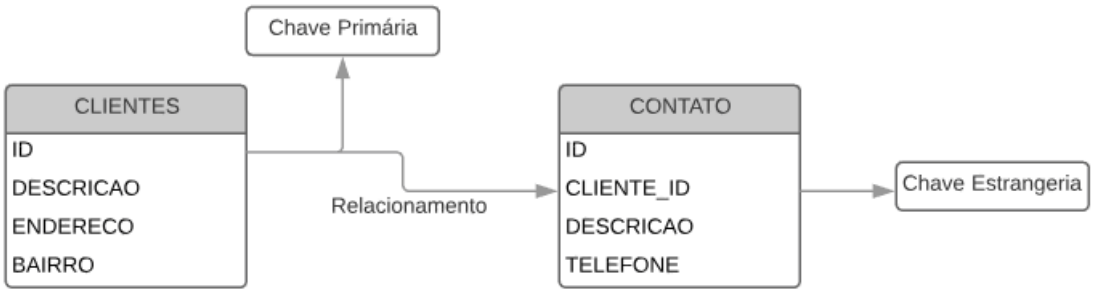
No modelo relacional as chaves, Figura 3, são importantes, pois sua utilização garante que cada linha da tabela seja identificável de modo exclusivo. Elas também são utilizadas para estabelecer relacionamentos entre tabelas e garantir a integridade dos dados (ROB; CORONEL, 2011).

Rob e Coronel (2011) também definem as chaves de banco relacionais como:

a) Chave primária – chave selecionada para identificar exclusivamente todos os outros valores de atributos de uma linha, não podendo conter entradas nulas;

b) Chave estrangeira – atributos em uma tabela cujos valores devem coincidir com uma chave primária de outra tabela, podendo ser nulo.

Figura 3 - Exemplo de chaves



Fonte: Elaborado pelo autor

2.1.2 **Linguagem SQL**

Durante o desenvolvimento dos bancos de dados relacionais, duas linguagens foram propostas para serem utilizadas, Figura 4, álgebra relacional, formada por expressões de álgebra relacional com operados padrões, e a linguagem de cálculo relacional, que é a que mais se aproxima da linguagem SQL (GUIMARÃES, 2003).

Figura 4 - Representação de álgebra relacional e cálculo relacional

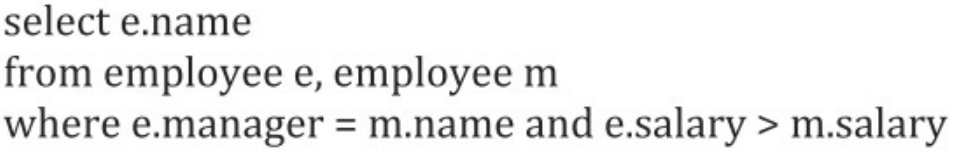
$$\pi_{e.name} (\sigma_{e.salary > m.salary} (\rho_e(employee) \bowtie_{e.manager = m.name} \rho_m(employee)))$$

RANGE employee e;
RANGE employee m;
GET w (e.name): $\exists m((e.manager = m.name) \wedge (e.salary > m.salary))$

Fonte : https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6359709

Com o intuito de padronizar e facilitar as operações com o banco de dados relacional, em 1970 a IBM desenvolveu a linguagem SQL, Figura 5, do inglês, *Structured Query Language* (Linguagem Estruturada para Consulta) como forma de interface padrão para o sistema gerenciador de banco de dados relacional denominado como SYSTEM R, que com o tempo acabou sendo padronizada pela *American National Standards Institute*(ANSI) e pela *International Standards Organization*(ISO) (PUGA; FRANÇA; GOYA, 2013) (ELMARI, 2011).

Figura 5 - Exemplo de linguagem SQL



Fonte: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6359709

Segundo Puga, França e Goya (2013), o SQL possuí três categorias de instruções:

a) DDL – *Data Definition Language*: Utilizada para definição e manutenção da estrutura do banco de dados, conforme o Quadro 1;

Quadro 1 - Instruções DDL

Instruções DDL	
Create	Criação de estruturas de objetos do banco de dados
Alter	Alteração da estrutura de objetos do banco de dados
Drop	Eliminação das estruturas de objetos do banco de dados
Truncate	Exclusão física de linhas de tabelas
Rename	Renomeação de objetos do banco de dados
Comment	Inclusão de comentários aos objetos do banco de dados

Fonte: Puga, França e Goya (2013)

b) DML – *Data Manipulation Language*: Utilizada para manipulação de dados, sendo elas Insert, Update, Delete, Select e Merge. Contendo uma subcategoria chamada DCL – *Data Control Language* que é utilizada para controlar os acessos aos dados que são os comandos *Grant* e *Revoke,* conforme o Quadro 2;

Quadro 2 - Instruções DML

Instruções DML	
Insert	Inserção de dados
Update	Alteração de dados
Delete	Exclusão de dados
Select	Consulta de dados
Merge	Combinação das instruções *insert*, *update* e *delete*
Grant	Atribuição de privilégios aos usuários do banco de dados
Revoke	Revogação de privilégios dos usuários do banco de dados

Fonte: Puga, França e Goya (2013)

c) TCL – *Transact Control Language*: Utilizada para controle de transações, conforme Quadro 3.

Quadro 3 - Instruções TCL

Instruções TCL	
Commit	Confirmação das manipulações
Rolback	Desistência das manipulações
Savepoint	Criação de pontos para o controle das transações

Fonte: Puga, França e Goya (2013)

2.1.3 **Transações**

Um banco de dados relacional pode executar uma ou mais transações, que segundo Elmasri e Navathe (2011), uma transação é um programa em execução que inclui operações de banco de dados, como leitura de dados, inserção de dados, exclusões ou atualizações das informações, que no final da transação, ela deve deixar o banco de dados em um estado válido ou coerente, satisfazendo todas as restrições especificadas no esquema do banco de dados.

Conforme Price (2009), uma transação é definida como uma unidade lógica, que é um grupo de instruções SQL relacionadas que sofrem *commit* ou *rollback*, tendo quatro propriedades fundamentais, conhecidas como propriedade ACID definida por:

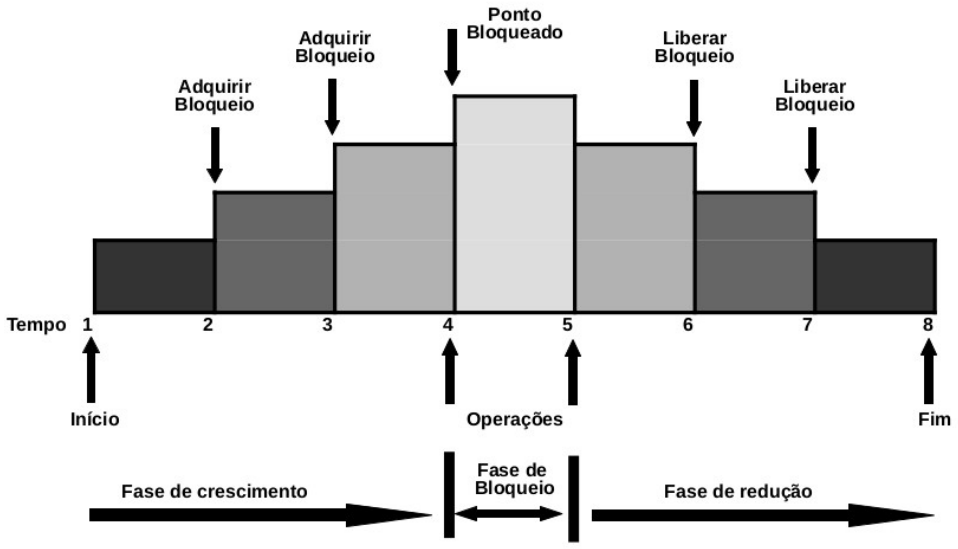
a) Atomicidade – transação atômica significa que as instruções SQL contidas em uma transação constituem uma única unidade de trabalho;

b) Consistência – as transações garantem que o estado consistente, permaneça consistente após a execução da transação, que em caso de falha no decorrer da transação retorna os dados para o estado inicial;
c) Isolamento – uma transação não interfere em outra transação corrente;
d) Durabilidade – se a transação sofre *commit*, as alterações feitas no banco de dados são preservados, mesmo no caso de uma falha e/ou reinício do sistema.

Para Rob e Coronel (2011) o controle de concorrência tem o objetivo de serializar as transações, como se todas as transações fossem executadas uma após a outra, em série, e para isso utiliza formas de bloqueios, na qual garante que somente uma transação tenha acesso ao dado no momento da alteração, conforme a granularidade de bloqueio, que pode ser de banco de dados, tabela, página, linha e campo. Existem três protocolos de bloqueios para garantir a serialização da transação:

a) Bloqueio de duas fases (2PL) – Figura 6, garante a serialização mas não evita *deadlocks*, duas ou mais transações que aguardam indefinidamente que a outra desbloqueie dados, divida em duas fases, a fase de crescimento, fase que a transação adquire todos os bloqueios solicitados sem desbloquear nenhum dado, e a fase de redução que é a fase que a transação libera todos os bloqueios e não obtém mais nenhum bloqueio

Figura 6 - Exemplo de bloqueio de duas fases



Fonte : Adaptado de Rob e Coronel (2011)

b) *Time Stamping* – identificador único e exclusivo a cada transação, que é utilizado para saber qual transação tem prioridade no acesso aos dados;
c) Otimista – baseada no pressuposto de que a maioria das operações não entrem em conflito, a transação é executada sem restrições, passando pelas fases de leitura, na qual a transação lê o banco de dados e executa as operações necessárias e atualiza em uma cópia privada, fase de validação, em que a transação é validada para garantir a integridade e a consistência do banco de dados, se negativo a validação a transação é descartada e reiniciada, e em caso de positivo passa para a fase de gravação, em que as alterações são aplicadas ao banco de dados permanentemente.

O nível de isolamento é o que ajusta o equilíbrio entre desempenho e confiabilidade, os bancos de dados relacionais fornecem 4 níveis diferentes de isolamento:(MYSQL, 2018a)

a) *Repeatable read*: Garantindo que a mesma leitura de um dado através do mesmo *select* se repita, tendo o mesmo resultado para diferentes execuções na mesma transação. Nesse nível de consistência é impossível que ocorra a chamada leitura fantasma, quando entre um *select* e outro ocorre uma atualização nos dados;
b) *Read committed*: A transação sempre irá ler e manipular os dados já comitados por outras transações, caso alguma transação tenha alterado algum dado e não tenha efetuado um *commit*, esse dado não será visto pelas outras transações, ocorrendo a leitura fantasma;
c) *Read uncommitted*: Permite que uma transação possa ler e manipular dados não comitados por outras transações, podendo ocorrer a leitura fantasma e as leituras sujas;
d) *Serializable*: Semelhante ao r*epeatable read*, com a restrição de que as linhas selecionadas por uma transação não podem ser alteradas ou lidas por outra transação até que a primeira seja finalizada.

Segundo Brito(2010) o modelo relacional utiliza a estratégia do escalonamento vertical, que é o aumento da capacidade do servidor em que o banco de dados está instalado, ou até a substituição do mesmo por um melhor, acarretando em um custo muito elevado.

2.1.4 **Vantagens e desvantagens**

A utilização de um banco de dados relacional traz algumas vantagens e desvantagens, podemos citar as seguintes características como vantagens em que a utilização de um banco de dados relacional apresenta:

a) Linguagem SQL – os bancos de dados relacionais utilizam um padrão de linguagem, conhecido como SQL, que facilita toda e qualquer manipulação dos dados;
b) Relacionamento – facilidade em relacionar tabelas, facilitando a combinação de dados a fim de gerar informações mais detalhada, e evitando a redundância de dados com base nas relações entre as tabelas;
c) Esquema definido - garantindo a integridade dos dados;
d) Transações baseadas em ACID – baseado no teorema ACID, o banco de dados relacional garante que todas as transações executadas sejam atômicas, consistentes, isoladas e duráveis.

O banco de dados relacional apresenta algumas limitações, que podemos citar como sendo desvantagens:

a) Escalabilidade – a escalabilidade do banco de dados relacional não é flexível e adaptada para o escalonamento, na qual o modelo relacional utiliza o escalonamento vertical, que muitas das vezes possui um custo elevado, devido a troca do servidor, ou o aumento da capacidade de processamento do servidor, tornando-se uma desvantagem quando falamos em grande quantidades de dados;
b) Dados distribuídos – o modelo relacional pode ter os dados distribuídos, porém não é realizada de maneira tão simples, demandando que os desenvolvedores dos sistemas tratem a distribuição, pois a normalização dos dados define que dados que são utilizados em conjunto sejam armazenados em conjunto;
c) Disponibilidade – como o banco de dados relacional tem dificuldade em distribuir os dados, a disponibilidade fica restrita e de difícil implementação;
d) Performance – com o aumento do volume de dados, o banco de dados relacional acaba tendo uma queda de performance.

2.2 BANCO DE DADOS NOSQL

Conforme Lennon (2011), nos últimos anos, houve um interesse crescente em sistemas de gerenciamento de bancos de dados que diferem do modelo relacional, com isso

apareceu o conceito NoSQL, termo utilizado para denotar softwares de banco de dados que não utilizam SQL para interagir com o banco de dados.

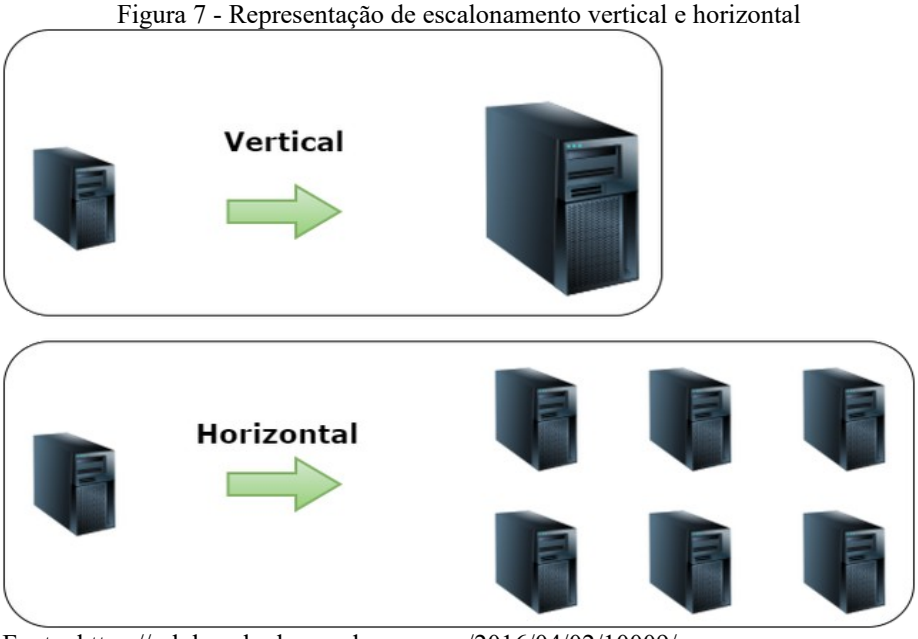
Para Yegulalp (2018) os bancos de dados NoSQL surgiram em resposta as limitações do modelo relacional. Os sistemas NoSQL armazenam e gerenciam os dados de maneira que permitem alta velocidade operacional e grande flexibilidade para os desenvolvedores. Muitos dos sistemas NoSQL tiveram influência de grandes empresas como Google, Amazon, Yahoo e Facebook, que buscavam formas melhores de armazenar ou processar dados massivos.

Uma das características fundamentais do NoSQL é a escalabilidade horizontal, na qual Cattell (2018) define o termo escalabilidade horizontal como a capacidade de distribuir os dados, e a carga das operações simples em muitos servidores.

A definição de escalonamento horizontal, Figura 7, para MongoDB (2018) é:

> *"O* escalonamento horizontal envolve a divisão do conjunto de dados do sistema e o carregamento em vários servidores, adicionando servidores adicionais para aumentar a capacidade conforme necessário. Embora a velocidade ou a capacidade geral de uma única máquina possa não ser alta, cada máquina lida com um subconjunto da carga de trabalho, potencialmente fornecendo melhor eficiência do que um único servidor de alta capacidade e velocidade. A expansão da capacidade da implantação requer apenas a adição de servidores, conforme necessário, o que pode representar um custo geralmente menor do que o hardware de última geração para uma única máquina."

Figura 7 - Representação de escalonamento vertical e horizontal



Fonte: https://ralphavalonbr.wordpress.com/2016/04/02/10009/

O escalonamento horizontal é realizado com o auxílio do algoritmo *Sharding,* dividindo o banco de dados em bancos de dados menores, dados fragmentados, que geralmente contém itens que estão dentro de um intervalo específico (MongoDB, 2018).

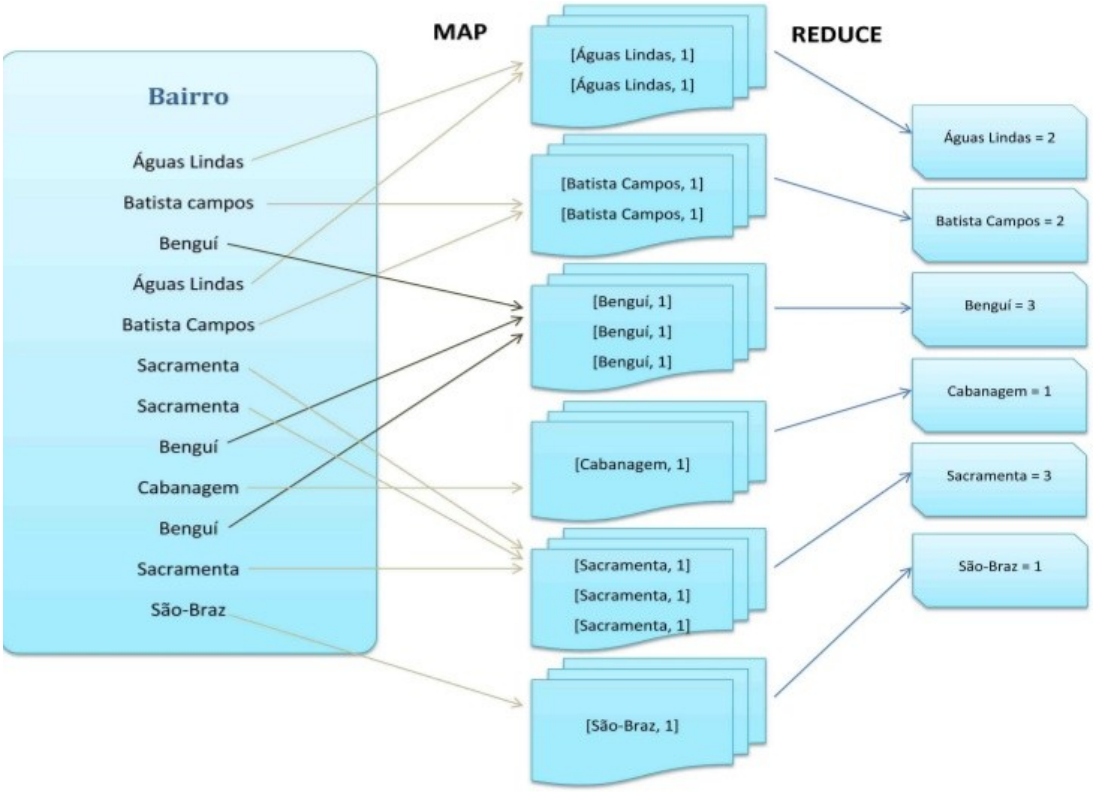
Os bancos de dados NoSQL suportam replicação automática de banco de dados para manter a disponibilidade, em caso de interrupções ou eventos de manutenção planejados. Lóscio, Oliveira e Pontes (2011) definem duas abordagens principais para a replicação:

a) *Master-Slave* (Mestre escravo) – cada escrita no banco de dados resulta em N escritas, onde N é o número de nós escravos. A escrita é feita no nó mestre, e cada escrita é refeita em cada nó escravo. Leitura se torna mais rápida, porém quando tiver um grande volume de dados, a escrita pode se tornar um gargalo;
b) *Multi-Master* (Multi-Mestre) – possui vários nós mestres, diminuindo o gargalo gerado pela escrita, porém pode causar problema de conflito de dados.

Para processar grandes quantidades de dados distribuídos, o NoSQL utiliza o algoritmo desenvolvido pela Google, chamado de *Map-Reduce*, Figura 8, que é dividida nas seguintes etapas segundo Tiwari (2011):

a) *Map* – pega um par de chave/valor em uma coleção de dados e gera novos pares de chave/valor menores para enviar para a etapa *Reduce;*
b) *Reduce* – recebe os pares de chave/valor da etapa *map*, e divide-os entre os nós para serem processados, retornando os valores que serão combinados para obter o resultado final.

Figura 8 - Exemplo de Map-Reduce



Fonte: https://pt.slideshare.net/jgabriellima/bigdata-mapreduce

A maioria dos bancos de dados NoSQL utilizam para controle de concorrência o MVCC, controle de concorrência multiversão (MVCC – *Multiversion Concurrency Control*), onde relaxa-se a consistência para permitir um maior desempenho e as leituras não bloqueiam os dados, permitindo que uma operação de escrita ocorra ao mesmo tempo no mesmo conjunto de dados, criando uma nova versão dos dados. Para lidar com duas ou mais operações de escritas conflitantes, cada processo armazena, além do novo valor, um link para a versão que o processo leu antes. Portanto, os algoritmos no banco de dados ou no lado do cliente têm a possibilidade de resolver valores conflitantes por diferentes estratégias. Alguns modelos de NoSQL também utilizam o bloqueio otimista, no qual antes de os dados alterados sejam confirmado, cada transação verifica se alguma outra transação efetuou alguma modificação no mesmo conjunto de dados, e em caso de conflito a transação será revertida. O bloqueio otimista é mais adequado quando a chance de conflito entre as transações é pequena (HECHT; JABLONSKI, 2011).

As vantagens dos bancos de dados NoSQL não vêm sem custo, geralmente não fornecem o mesmo nível de consistência de dados que o modelo relacional. Segundo Tiwari

(2011), em sistemas distribuídos somente pode se ter dois dos três pilares do teorema CAP, Figura 9, (*Consistency, Availability, Partition Tolerance):*

a) *Consistency* (Consistência) – Significa que todas as operações de leitura e escrita vejam o mesmo estado de dados válidos, sempre dados atualizados;
b) *Availability* (Disponibilidade) – Significa que o sistema deve estar disponível para atender as solicitações quando solicitado;
c) *Partition Tolerance* (Tolerância ao Particionamento) – Refere-se a capacidade de um sistema continuar operando, mesmo depois de um dos nós ficar indisponível por tempo indeterminado.

Bancos de dados NoSQL, fazem com que você tenha que escolher entre quaisquer duas prioridades do CAP, uma vez que todos os três são difíceis de se conseguir em um sistema baseado em nó distribuído:

a) **Sistemas CA (Consistência e Disponibilidade)** – Os sistemas com consistência forte e alta disponibilidade não sabem lidar com uma possível falha, caso ocorra, o sistema inteiro pode ficar indisponível;
b) **Sistemas CP (Consistência e Particionamento)** – Sistemas com consistência forte e tolerância a particionamento é necessário abrir mão da disponibilidade, o dado somente será salvo se todas as réplicas confirmarem a escrita, caso contrário, a escrita não será efetivada;
c) **Sistemas AP (Disponibilidade e Particionamento)** – Sistemas que desejam ter alta disponibilidade e com tolerância ao particionamento, o sistema responderá em caso de falha de comunicação, mas com dados desatualizados.

Figura 9 - Representação do teorema CAP

Fonte: Adaptado de https://dzone.com/articles/quick-notes-what-cap-theorem

A partir desse teorema, os bancos de dados NoSQL utilizam o paradigma BASE, que segundo a Oracle (2018) se caracteriza por:

a) BA – (*Basically Available*) – (Basicamente Disponível) Disponibilidade é prioridade, o sistema deve estar em funcionamento na maior parte do tempo;
b) S – (*Soft-State*) – (Estado Leve) Não precisa ser consistente o tempo todo;
c) E – (*Eventually Consistent*) – (Eventualmente Consistente) Consistente em momento indeterminado, a consistência nem sempre é mantida para todos os nós, os nós podem não ter a mesma versão dos dados.

Tiwari (2011) lista quatro modelos de bancos de dados NoSQL, que são elas: orientado a colunas, orientado a documentos, chave-valor e orientado a grafos.

2.2.1 **Orientado a colunas**

Enquanto o modelo relacional é orientado a linhas, otimizado para armazenar linhas de dados, o banco de dados orientado a coluna, Figura 10, é otimizado para recuperação

rápida de colunas diminuindo a quantidade de dados que é carregado do disco (AMAZON, 2018c).

Cada unidade de dados pode ser pensada como um conjunto de pares chave/valor, onde a unidade é identificada com a ajuda de um identificador primário, muitas vezes referida como a chave primária, ou chamada também de *row-key*. As unidades são armazenadas de maneira ordenada com base na chave de linha (TIWARI, 2011).

As operações de leitura e escrita são atômicas, todas as colunas da linha são consideradas na execução da operação, independente da coluna que está sendo utilizada na operação (LÓSCIO; OLIVEIRA; PONTES, 2011).

Figura 10 - Exemplo de base orientado a colunas

Chave	Nome
0	Paulo
1	Eduardo
2	Ana
3	John
4	Roberta

Chave	Idade
0	45
1	25
2	67
3	33
4	18

Chave	Genero
0	Masculino
1	Masculino
2	Feminino
3	Masculino
4	Feminino

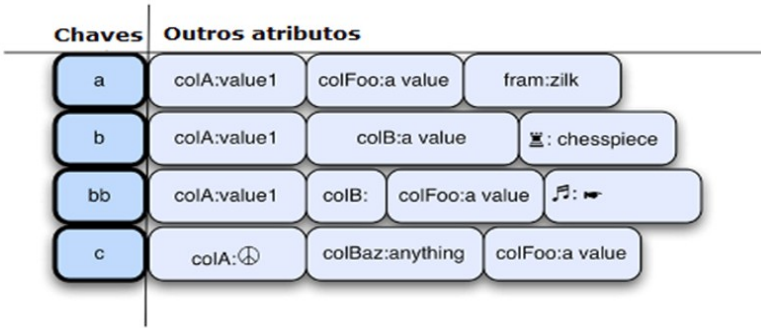
Fonte: Autor

Como exemplo, destaca-se o BigTable da Google, Cassandra do Facebook e Hbase.

2.2.2 **Chave-Valor**

Para Lóscio, Oliveira e Pontes (2011) os registros são representados por um conjunto de chaves, Figura 11, na qual cada chave está associada a um único valor, podendo ser representando como uma tabela *hash.*

Figura 11 - Exemplo de base de dados chave-valor



Fonte: https://imasters.com.br/banco-de-dados/bancos-de-dados-nosql-uma-visao-geral/?trace=1519021197&source=single

Uma chave é a concatenação de um caminho de chave. Todos os registros que compartilham um caminho de chave são co-localizados, dentro de uma coleção co-localizada de caminhos chave, composta pelos caminhos principais e secundários, fornece uma pesquisa rápida e indexada. Já o campo valor é armazenado como uma matriz de bytes (ORACLE, 2018c).

Como exemplo dos bancos de dados NoSQL chave-valor, destacam-se o Dynamo, desenvolvido pela Amazon, Redis, Oracle NoSQL.

2.2.3 **Orientado a documentos**

Conforme Lóscio, Oliveira e Pontes (2011) os bancos de dados orientados a documentos, Figura 12, são considerados como a evolução dos sistemas chave-valor, na qual os dados são armazenados em forma de coleções de documentos, sendo que uma coleção de documentos pode conter um ou mais documentos.

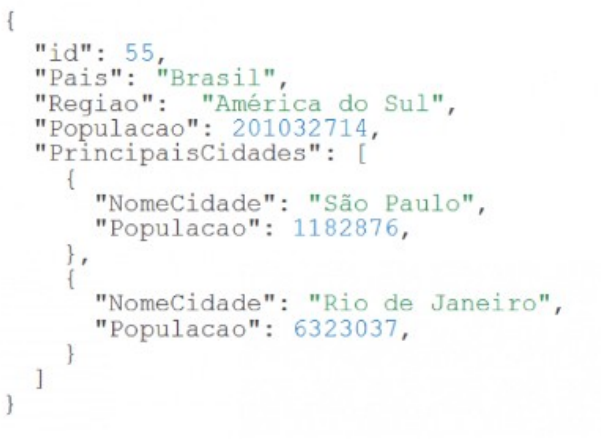
Desenvolvido para armazenar dados semiestruturados, normalmente em formato JSON ou XML (AMAZON, 2018d).

Diferente dos tradicionais bancos de dados relacionais que armazenam os dados em estruturas rígidas, como tabelas, os bancos de dados orientados a documentos armazenam em documentos vagamente definidos, possuindo um campo que serve como identificador único, atributos e campos, sendo possível acrescentar novos atributos aos documentos individuais sem que os demais sejam alterados (LENNON, 2018).

Bancos de dados orientados a documentos, permitem a indexação de documentos não somente pelo seu identificador, mas também permite em suas propriedades (TIWARI, 2011).

Como exemplo deste banco de dados, podemos citar o MongoDB, OrientDB, CouchBase e RavenDB.

Figura 12 - Exemplo de base de dados orientado a documentos



Fonte: https://imasters.com.br/banco-de-dados/bancos-de-dados-nosql-uma-visao-geral/?trace=1519021197&source=single

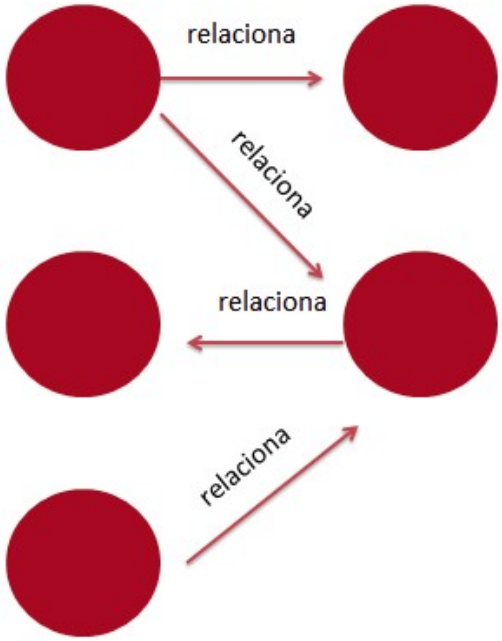
2.2.4 **Orientado a grafos**

Possuindo três componentes básicos: nós, relacionamentos e as propriedades dos nós e relacionamentos, o modelo orientado a grafos, Figura 13, apresenta uma grande vantagem quando utilizado para consultas complexas, tendo um ganho de performance em relação ao modelo relacional (LÓSCIO; OLIVEIRA; PONTES, 2011).

Os nós são as entidades, podendo ter qualquer valor de atributo, chamados de propriedades. Podem ser representandos por um rótulo, que mostram suas diferentes funções no seu domínio, servindo também, para atribuir o índice ou restrição. Já os relacionamentos são as conexões dirigidas, entre duas entidades de nó, sempre tendo uma direção, um tipo, um nó inicial e um nó final. Na maioria dos casos os relacionamentos têm propriedades quantitativas, como pesos, custos, distâncias, entre outras. Mesmo o relacionamento tendo direção, podem ser navegados em qualquer direção (NEO4J, 2018).

Alguns bancos de dados orientados a grafos, são compatíveis com ACID, porém eles acabam não sendo tão eficientes quanto os outros tipos de armazenamentos de dados NoSQL (GROLINGER et al., 2013). Exemplo de bancos de dados NoSQL orientados a grafos são, Neo4J, OrientDB e AllegroGraph.

Figura 13 - Exemplo de base de dados orientado a grafos



Fonte: https://imasters.com.br/banco-de-dados/graphdb-series-o-que-e-um-banco-de-dados-de-grafos/?trace=1519021197&source=single

2.2.5 **Vantagens e desvantagens**

A utilização de um banco de dados NoSQL apresenta algumas vantagens e desvantagens, algumas das vantagens encontradas nos bancos de dados NoSQL podem ser definidas como sendo:

a) Esquema Flexível – os bancos de dados NoSQL não possuem um esquema definido, no momento de sua criação não é necessário definir como será cada campo, podendo ser adicionados ou removidos a qualquer momento, conforme a necessidade;

b) Escalonamento e Desempenho – o escalonamento horizontal foi um dos principais motivos para a criação dos bancos de dados NoSQL, na qual permite aumentar a quantidade de servidores, não necessariamente mais robustos, conforme o volume de dados e as requisições vão crescendo, utilizando o método de *sharding,* que divide as tabelas em diversos servidores;

c) Replicação de Dados - a forma em que o banco de dados NoSQL replica os dados também acaba sendo uma vantagem, pois o mesmo tem suporte nativo a

replicação, sem ter que ser feito manualmente, tendo também a possibilidade de compartilhar os dados de forma automática, o banco de dados distribui os dados e as consultas conforme a quantidade de servidores disponíveis, efetuando o balanceamento automaticamente, e em caso de indisponibilidade de alguns dos servidores, o mesmo é substituído rapidamente por outro servidor disponível, sem que o usuário perceba;

d) Disponibilidade – com a replicação de dados que o NoSQL fornece, acaba favorecendo a alta disponibilidade, pois se um servidor ficar indisponível, automaticamente outro servidor assumirá o lugar do mesmo.

As desvantagens que podemos citar na utilização do banco de dados NoSQL, podem ser definidas da seguinte forma:

a) Linguagem – bancos de dados NoSQL não possuem uma linguagem padrão, dificultando o aprendizado pois cada banco de dados NoSQL utiliza sua própria linguagem, com isso muitas vezes a aplicação acaba fazendo algum processamento para obter os dados necessários;

b) Consistência – como o principal motivo do banco de dados NoSQL é ter a informação distribuída, tendo performance, flexibilidade e disponibilidade, baseado no teorema CAP, o qual afirma que é impossível para um sistema distribuído garantir de forma simultânea consistência, disponibilidade e tolerância ao particionamento. O banco de dados NoSQL utiliza o paradigma BASE, que torna as informações eventualmente consistentes, tem apenas a garantia de que se não houver nenhuma atualização dos dados, todos os acessos terão as informações atualizadas;

c) Esquema Flexível – com o esquema flexível a integridade e consistência dos dados fica sob a responsabilidade da aplicação.

2.3 GRANDES VOLUMES DE DADOS

Vivemos em um mundo cada vez mais voltado para os dados, em que gera-se uma quantidade enorme de dados, Marr (2013), defende que praticamente todas as atividades exercidas acabam deixando um rastro digital, uma vez que todos os recursos que utilizamos de alguma forma estão conectados. Mas para explicar o que é esse grande volume de dados, Sas (2018) cita que big data é um termo utilizado para descrever grandes quantidades de dados, sejam eles estruturados ou não estruturados. A definição de *big data* é baseada em 3

Vs, volume, variedade e velocidade, e segundo Pettey e Goasduff (2018) são definidos da seguinte forma

a) **Volume:** o aumento nos volumes de dados nos sistemas corporativos é causado por volumes de transações, e outros tipos de dados tradicionais. Muito volume é um problema de armazenamento, mas muitos dados também são um problema de análise em massa;
b) **Variedade:** refere-se a dados estruturados ou não estruturados, provenientes de diversos lugares, como: dados tabulares (bases de dados), dados hierárquicos, documentos, e-mail, vídeo, imagens fixas, áudio, dados de cotações de ações, transações financeiras e muito mais;
c) **Velocidade:** significa a velocidade com que os dados estão sendo gerados e a velocidade com que os dados devem ser processados para atender à demanda.

Segundo Gartner (2012) o *big data* cria uma nova camada na economia, baseada toda sobre informações, transformando informações ou dados em receitas.

Muitos autores como Taurion (2013), Marr (2014) defendem que o big data possuí mais 2 Vs:

a) **Veracidade:** significa que os dados devem fazer sentido, que são autênticos;
b) **Valor:** transformar os dados em valor, obter um retorno.

O grande desafio é descobrir como gerenciar e analisar todo o volume de dados que cresce infinitamente todos os dias, mais do que isso, entender esses dados e gerar mais experiência, produtividade, consumo e novos serviços (OLHAR DIGITAL, 2013).

3 NEWSQL

O NewSQL surge a partir da necessidade de ter a consistência dos dados e de poder escalonar mais facilmente o sistema. Esse termo, NewSQL, foi utilizado pelo analista Matt Aslett, para descrever um novo grupo de bancos de dados, na qual o NewSQL é uma classe de sistemas de gerenciamento de bancos de dados relacionais, que procura oferecer o mesmo desempenho escalável do modelo NoSQL, para cargas de trabalho de leitura e gravação no processamento de transações on-line, mantendo as garantias ACID do modelo relacional (VOLTDB, 2018).

Grolinger et al. (2013), define o modelo NewSQL como sendo baseado no modelo relacional, oferecendo uma visão puramente relacional dos dados. Embora os dados possam ser armazenados em forma de tabelas e relações, é interessante observar que as soluções NewSQL podem usar diferentes representações de dados.

Para Ryan (2018), os bancos de dados NewSQL fornecem uma plataforma de banco de dados redesenhado do zero, com a finalidade de processar milhões de transações por segundo, em uma plataforma de hardware possível de se escalonar horizontalmente, rodando quase inteiramente em memória, com vantagens significativas :

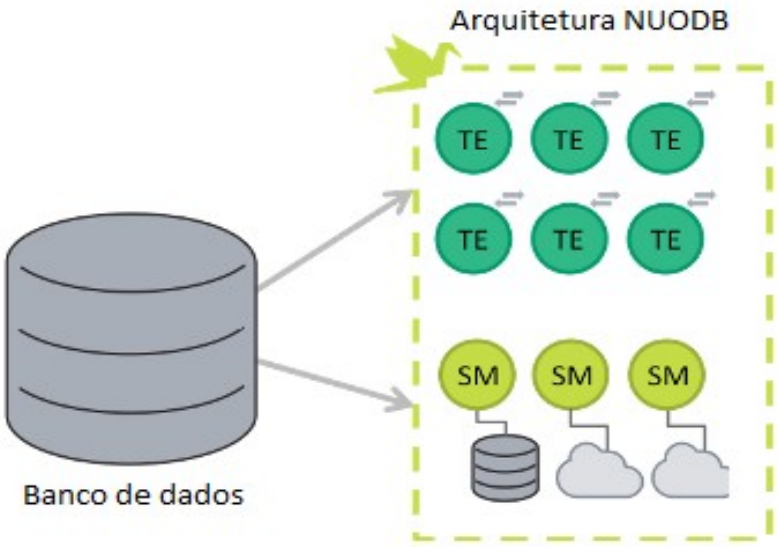
a) Banco totalmente relacional – completo contendo junções e suporte ao padrão SQL;
b) Conformidade ACID – todos os bancos de dados NewSQL suportam totalmente transações ACID;
c) Latência de milissegundos – executam milhões de transações por segundo;
d) Tolerância a falhas – mesmo os dados sendo replicados para diversos servidores de memória, a arquitetura fornece operações 24x7;
e) No local ou na nuvem – podem ser utilizados em máquinas locais, ou na nuvem.

Nas sessões seguintes serão apresentados alguns bancos de dados NewSQL.

3.1 NUODB

O banco de dados NuoDB é uma arquitetura distribuída, Figura 14, dividida em duas camadas, separando o processamento de transações e o gerenciamento de armazenamento, o Mecanismo de Transação (TE) é utilizado para manipular as solicitações dos clientes, e o Gerenciador de Armazenamento (SM), serve para persistir os dados, com uma implementação de armazenamento de chave/valor (NUODB, 2018a).

Figura 14 - Representação da arquitetura do NuoDB



Fonte: https://www.nuodb.com/product-overview

Os aplicativos interagem como sendo um banco de dados relacional único, compatível com ACID, utilizando consultas SQL e permitindo o escalonamento horizontal (NUODB, 2018d).

Quando uma transação precisa de dados que não estão armazenados localmente, o mecanismo de transação consulta seu mapa, localiza um mecanismo de transação que possui as informações, e solicita que sejam enviadas as informações para ele. Se os dados que estão sendo solicitados não forem encontrados em nenhum mecanismo de transação, a solicitação é feita para um gerenciador de armazenamento, que busca os dados a partir de sua memória ou do disco.

Todos os dados são armazenados e gerenciados através de objetos chamados de átomos, que são objetos que representam tipos específicos de informações, como, dados, índices ou esquemas, todos os dados associados a um banco de dados, incluindo metadados internos, são representados também como sendo um átomo (NUODB, 2018c).

Os átomos possuem tipos, que ao contrário dos objetos em uma linguagem orientada a objetos, as instâncias de átomos no NuoDB possuem cópias. Por exemplo, o NuoDB tem um tipo de átomo chamado Tabela. Cada tabela no banco de dados é representada por uma instância do átomo tabela. Cada Mecanismo de transação que usa uma determinada tabela tem uma cópia do átomo de tabela, chamadas de pares. As cópias individuais replicam as alterações entre si, através de mensagens assíncronas, indicando que um mecanismo de transação efetuou uma alteração no átomo (NUODB, 2018b)

O NuoDB implementa o controle de concorrência MVCC, controle de concorrência multiversão, embora as ações ocorram em diferentes Mecanismos de Transação, não possui o bloqueio de duas fases, pois nenhum bloqueio é mantido durante a execução da transação, os conflitos de escrita/escrita são resolvidos através da comparação de versão de linhas. Para manter a consistência de transação, o NuoDB trabalha com duas condições, primeiro, todas as mensagens de átomos em um mecanismo de transação são recebidas por outros mecanismos de transação, e gerenciadores de armazenamento na ordem em que foram enviadas. Em segundos, cada transação é executada em um mecanismo de transação.

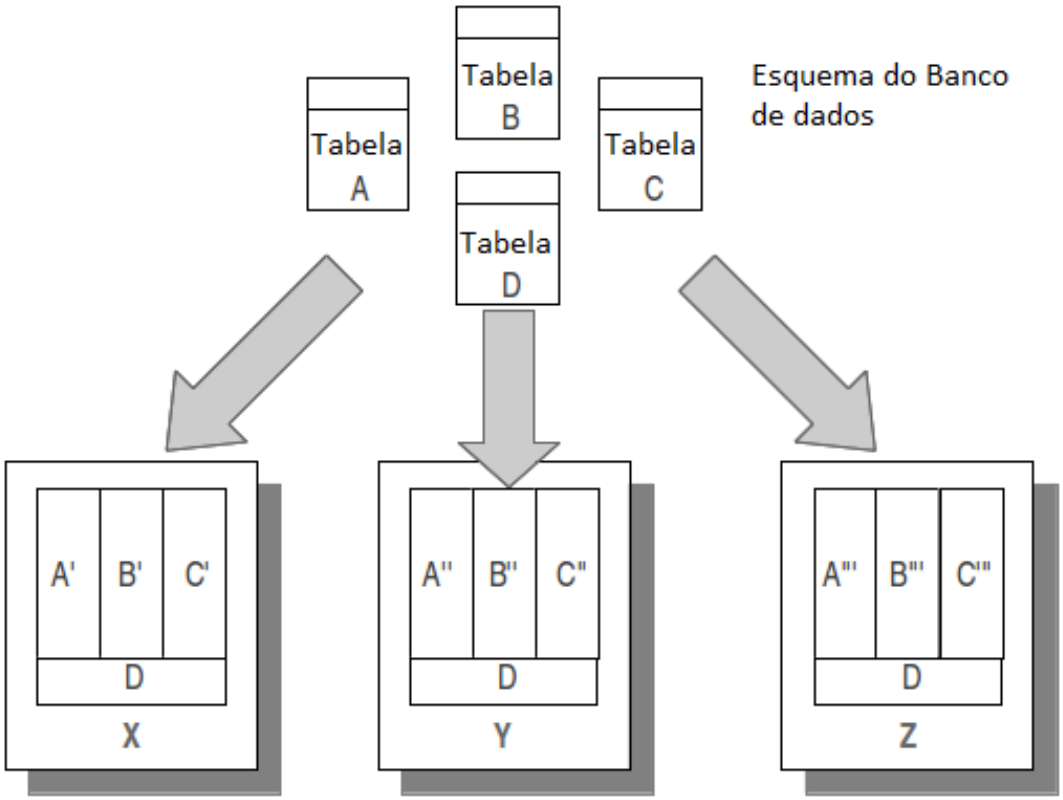
3.2 VOLTDB

O banco de dados VoltDB é capaz de atingir um rendimento 45 vezes maior do que os produtos de banco de dados atuais. A arquitetura também permite que os bancos de dados VoltDB sejam redimensionados com facilidade, particionando as tabelas de banco de dados, e procedimentos armazenados que acessam essa tabelas, em vários sites, ou partições, e uma ou mais máquinas para criar o banco de dados distribuído. Cada site trabalha de forma independente, com isso, cada transação pode ser executada até a conclusão, sem que ocorram bloqueios de registros individuais (VOLTDB, 2018a).

Para garantir a consistência do banco de dados, o VoltDB define cada procedimento armazenado como uma transação, sendo finalizada ou revertida como um todo. Os dados e o processamento podem ser distribuídos a várias partições individuais, que cada uma contém uma parte exclusiva dos dados e do processamento (VOLTDB, 2018c)

O particionamento das tabelas, Figura 15, é feito baseado em coluna especificada, mas para conseguir otimizar ainda mais o desempenho, o VoltDB permite que tabelas pequenas, que sejam grande parte somente leitura, sejam replicadas para todas as partições do cluster, isso contribui para que a execução da transação seja de partição única (VOLTDB, 2018c)

Figura 15 - Representação de particionamento do VoltDB



Partições e replicações

Fonte: Adaptado de https://docs.voltdb.com/UsingVoltDB/IntroHowVoltDBWorks.php#IntroSerialize

O VoltDB ordena transações com base em registros de data e hora lógicos, e as programa para execução em uma partição quando é a vez delas. Quando uma transação é executada em uma partição, ela tem acesso exclusivo a todos os dados nessa partição e, portanto, o sistema não precisa definir bloqueios e travas refinados em suas estruturas de dados. Isso permite que as transações que só precisam acessar uma única partição sejam executadas com eficiência. A desvantagem do controle de simultaneidade baseado em partição é que ele não funciona bem se as transações abrangerem várias partições porque os atrasos de comunicação da rede fazem com que os nós fiquem ociosos enquanto aguardam as mensagens (VOLTDB, 2018c).

Em resumo o processamento serializado é o que garante a consistência transacional sem bloqueios no VoltDB, quando a transação é sobre uma única partição, o processo do servidor executa a transação sozinho, e libera o restante do cluster para tratar de outras solicitações em paralelo, já quando a transação é sobre varias partições, um nó atua como

coordenador e distribuiu o trabalho necessário para os outros nós, coletando os resultados e concluindo a transação (VOLTDB, 2018c).

Para atender o principio de durabilidade o VoltDB possui quatro recursos:

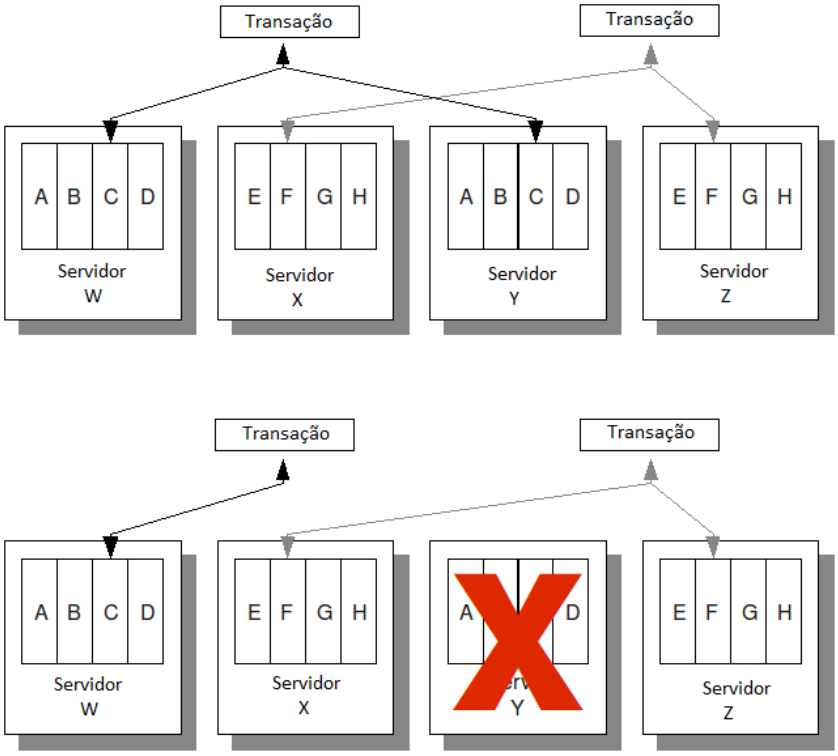
a) *Snapshots* – é a captura instantânea dos dados no banco de dados em um determinado momento gravado no disco;

b) Log de comando - além dos *snapshots* periódicos, o sistema mantém um log de cada transação, que em caso de falha o servidor poderá reproduzir o log para restabelecer o conteúdo dos bancos de dados;

c) *K-Safety*: Figura 16, é a pratica de duplicar partições de banco de dados para que em caso de perda de nós do cluster, não haja interrupção do serviço;

d) Replicação de Banco de Dados: é parecido com o *K-Safety*, porém ao em vez de criar partições redundantes em um único banco de dados, a replicação envolve a criação e a manutenção de uma cópia completa do banco de dados.

Figura 16 - Representação do *K-Safety*



Fonte: Adaptado de https://docs.voltdb.com/UsingVoltDB/ChapKSafety.php

O VoltDB suporta índices de árvore binária, em que cada nó é mantido a contagem de nós da subarvore.

3.3 MEMSQL

O MemSQL é um sistema distribuído em memória altamente escalável, escalonamento horizontal, em que os dados são divididos automaticamente entre os nós. O armazenamento suportado pode ser o de linha, completamente em memória do tipo tabela padrão, ou o repositório de colunas, em disco especificando um tipo de índice (MEMSQL, 2018a).

Os tipos de nós que o MemSQL utiliza são:

a) Nó agregador – manipulam os metadados do sistema distribuído, encaminham as consultas e agregam os resultados, podendo ter mais do que um agregador;
b) Nó folha – armazenam dados e executam as consultas SQL enviadas pelo nó agregador, uma folha consiste em várias partições.

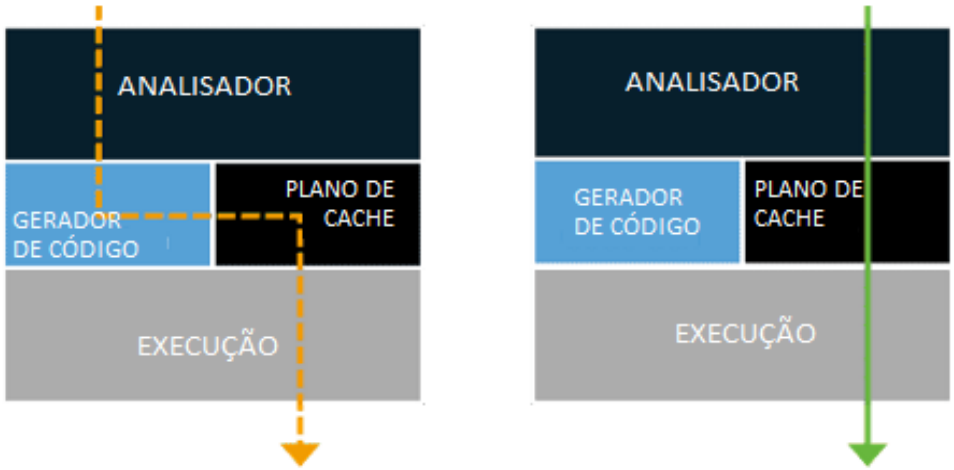
Projetado para ser altamente concorrente, o MemSQL possui um otimizador de consultas distribuídas, que divide igualmente a carga de processamento para maximizar a eficiência do uso dos servidores, e devido ao uso de MVCC, os dados permanecem altamente acessíveis, mesmo em meio a um alto volume de leituras e gravações simultâneas (MEMSQL, 2018a).

Além do MVCC o MemSQL utiliza, uma abordagem diferente para eliminar a necessidade de bloqueios, chamada índices não bloqueantes, de modo em que as gravações nunca bloqueiam as leituras, e vice-versa, utilizando várias estruturas de dados livres de bloqueio, que sempre permitem que uma *thread* progrida, garantindo a taxa de transferência do sistema (MEMSQL, 2018b).

Os dados são distribuídos automaticamente por meio de *sharding*, o particionamento é feito por particionamento de dados horizontal ou em linha, que divide as tabelas em partes menores, *shards* ou partições, em vários servidores físicos. Para conseguir informar onde os dados residem, o MemSQL tem um metadados adicional, usando uma chave de *shard*, para indicar ao otimizador de consultas onde encontrar o dado (MEMSQL, 2018b).

A cada consulta nova executada, o MemSQL remove automaticamente os parâmetros e gera um plano de execução, Figura 17, armazenando posterior em um diretório chamado plano de cache, para que consultas futuras correspondentes a um plano já gerado, sejam executadas imediatamente usando o plano armazenado em cache.

Figura 17 - Representação do processo de geração do plano de execução



Fonte: Adaptado de https://www.memsql.com/content/architecture/

O tipo de índice padrão do MemSQL é uma lista de saltos, alternativa da árvore balanceada, que é otimizada para ser executados na memória, pois são implementadas sem bloqueio e oferecem desempenho de inserção extremamente rápido (PROUT, 2018).

3.4 COMPARATIVO DAS CARACTERÍSTICAS DOS MODELOS RELACIONAL X NOSQL X NEWSQL

Com base nos estudos realizados das características dos sistemas relacional, NoSQL e NewSQL, assim como na análise de suas vantagens e desvantagens de uso, definiu-se para esse trabalho um conjunto de características relevantes que permitem sua comparação:

a) Consistência ACID/BASE;
b) Escalabilidade;
c) Esquema
d) Linguagem padrão
e) Disponibilidade;
f) Controle de concorrência;
g) Método de particionamento;
h) Modelo de dados.

A **consistência** dos dados dos bancos de dados NoSQL, está definido no uso do teorema CAP, consistência, disponibilidade e particionamento, porém como não se pode obter os três princípios do teorema, normalmente a consistência é desprezada, ficando assim com a consistência eventual, baseado no paradigma BASE, já o banco de dados relacional e o banco

de dados NewSQL utilizam a propriedade ACID, que garante a atomicidade, consistência, isolabilidade e durabilidade das transações executadas nos bancos de dados.

A **escalabilidade** nos bancos de dados relacionais ficou estagnada no escalonamento vertical, que está relacionado com o uso de CPU que compartilham memória e discos, com isso somente permite o aumento dos recursos, ou a troca total do equipamento, que normalmente tem um custo elevado. Já os modelos de bancos de dados NoSQL e NewSQL permitem que seja feito o escalonamento horizontal, que se torna mais eficiente, pois pode-se adicionar mais nós, computadores, ao sistema, particionando e replicando as informações, na qual não compartilham memória ou disco. Os computadores adicionados podem ser mais inferiores e por consequência pode se tornar menos custoso o escalonamento.

O **esquema** em um banco de dados NoSQL não é predefinido, o esquema é dinâmico, podendo ser alterado os tipos ou estrutura dos dados a qualquer momento de forma fácil, porém não garante a integridade dos dados, coisa que no banco de dados relacional e NewSQL não ocorre, pois o esquema é predefinido, forçando que os dados inseridos sejam validados no momento da inserção.

A **linguagem** utilizada pelos bancos de dados relacionais e NewSQL, para executar os comandos é a linguagem SQL, baseada em álgebra relacional e cálculo relacional, já os NoSQL, não utilizam o SQL em si, cada modelo de banco de dados NoSQL utiliza uma linguagem própria para realizar os comandos, porém alguns dos bancos estão aceitando uma linguagem semelhante a SQL, desenvolvida para dados não normalizados.

A **disponibilidade** dos bancos de dados está muito relacionado a escalabilidade dos mesmos, pois como os bancos de dados relacionais não conseguem trabalhar de forma eficiente com a distribuição dos dados a disponibilidade do modelo relacional depende muito só de um computador, com isso caso haja interrupção do serviço do mesmo, o sistema ficará indisponível. Os bancos de dados NoSQL e NewSQL já surgiram com o intuito de resolver esse grande problema do modelo relacional, com a facilidade de distribuição dos dados, caso algum dos nós fique indisponível o sistema funcionará normalmente pois os seus dados estarão replicados em algum outro nó.

Como os bancos de dados podem ser de acessos simultâneos, existe o **controle de concorrência,** para que as transações sejam executadas de forma que uma transação não interfira em outra. Os bancos de dados relacionais utilizam o controle de concorrência baseado em bloqueios, que garante que simultâneos usuários não acessem o mesmo item,

utilizando o método 2PL, *Time Stamp* e o otimista, já os bancos de dados NoSQL utilizam de outras formas para ter o controle de concorrência, e uma delas é o MVCC, que cria versões para cada alteração efetuada, sendo que alguns dos bancos de dados NewSQL utilizam também o MVCC para efetuar o controle de concorrência, porém outros utilizam uma combinação do 2PL com o MVCC.

Os bancos de dados NoSQL e NewSQL utilizam como **método de particionamento** o *sharding*, na qual o programador não precisa se preocupar em que servidor está os dados, a solicitação é feita somente para um servidor, já o banco de dados relacional não possui um método de particionamento que auxilie de tal forma, as instruções deveram ser tratadas para cada ocasião.

O **Modelo de dados** dos bancos de dados relacional e NewSQL, é orientado a linha, sendo que o NewSQL pode trabalhar sendo orientado a colunas em alguns casos, já o banco de dados NoSQL disponibiliza quatro formas de armazenar os dados, Orientado a grafos, orientado a documentos, chave-valor e orientado a coluna.

O Quadro 4, mostra o comparativo entre os modelos de bancos de dados.

Quadro 4 - Comparativo entre os modelos de bancos de dados

Característica	**Relacional**	**NoSQL**	**NewSQL**
Consistência ACID/BASE	ACID	BASE	ACID
Escalabilidade	Vertical	Horizontal	Horizontal
Esquema	Predefinido	Flexível	Predefinido
Linguagem Padrão	SQL	Sem padrão	SQL
Disponibilidade	Baixa	Alta	Alta
Controle de concorrência	2PL/Time Stamp/ Otimista/MVCC	MVCC/Otimista	MVCC/2PL
Método de particionamento	-	Sharding	Sharding
Modelo de dados	Orientado a linha	Orientado a grafos / Orientado a documentos / Chave-valor / Orientado a coluna	Orientado a linha / Orientado a coluna

Fonte: Elaborado pelo autor

4 BENCHMARK

Benchmark é a ação de comparar e analisar hardware, software ou processo. Quando utilizado para banco de dados, é utilizado um conjunto de instruções a fim de comparar desempenho de dois ou mais bancos de dados, auxiliando na tomada de decisão de qual banco de dados é o mais adequado.

Para analise de desempenho dos bancos de dados relacional, é comum a utilização do *benchmark* TPC, *Transaction Processing Performance Council*, em que possui diferentes *benchmarks* conforme a necessidade de analise, com o objetivo de definir um conjunto de requisitos funcionais que podem ser executados em qualquer sistema de processamento de transações, que simula um ambiente completo que executa transações em um banco de dados, centrado nas principais atividades de pedidos, pagamentos e estoques, contendo nove tabelas com tamanhos variados de registros, que permite analisar o desempenho do banco de dados (TPC, 2018).

Para a analise de banco de dados NoSQL, o *benchmark* mais utilizado é o YCSB, Yahoo! Cloud Serving *Benchmark,* que tem o objetivo de desenvolver uma estrutura e um conjunto comum de cargas de trabalho, conjunto de dados a serem carregados e conjunto de transações de leitura e escrita no banco de dados, para avaliar o desempenho de diferentes bancos de dados, podendo analisar a quantidade de transações por minuto (YCSB, 2018).

O YCSB é dividido em duas fases, a fase de carregamento e a fase de execução, a fase de carregamento apenas executa a inserção dos dados no banco de dados para a fase de execução. A fase de execução executa leituras, inserções, atualizações e remoções de informações no banco de dados (YCSB, 2018).

Ao final da execução, o sistema apresenta estatísticas de desempenhos, para cada tipo de operação é gerada informações referentes ao total do tempo de execução, taxa média de operações por segundo, entre outras (YCSB, 2018).

O YCSB inclui um conjunto de cargas de trabalhos definidos da seguinte forma:

a) A – combinação de 50% de leitura e 50% de gravação;
b) B – combinação de 95% de leitura e 5% de gravação;
c) C – 100% de leitura;
d) D – novos registros são inseridos, sendo que serão os mais lidos;
e) E – intervalos de registos são lidos;
f) F – cada registro lido será modificado e gravado a alteração.

Embora a cargas de trabalhos apresentam uma visualização completa do desempenho do sistema, o *benchmark* YCSB tem a possibilidade de que sejam definidas cargas de trabalhos novas e diferentes para examinar aspectos ou cenários que mais se adequam a necessidade desejada.

5 TRABALHOS RELACIONADOS

O trabalho de Fatima e Wasnik (2016) efetua um comparativo de desempenho de três sistemas de bancos de dados diferentes, sendo eles MySQL, do modelo relacional, MongoDB, do modelo NoSQL e o VoltDB, do modelo NewSQL. Esse comparativo teve ênfase em grandes quantidades de dados gerados pela Internet das Coisas, conhecido também pela sigla IoT, investigando e comparando as três tecnologias de banco de dados, em relação ao desempenho à medida que a quantidade de dados aumenta. O comparativo efetuado por Fatima e Wasnik (2016) mostrou que o banco de dados VoltDB teve bons resultados em relação aos demais.

Binani et al.(2016) efetuaram um estudo comparativo entre os três sistemas de bancos de dados, relacional, NoSQL e NewSQL, no contexto de Big Data. Para Binani et al. (2016) O NewSQL apresenta o ponto ideal entre consistência, escalabilidade, velocidade e disponibilidade, preenchendo todos os requisitos para ser um banco de dados para aplicativos de Big Data.

6 PROPOSTA DA SOLUÇÃO

Para ser possível realizar a avaliação de desempenho dos sistemas de bancos de dados, será utilizado o *benchmark* YCSB, Yahoo! Cloud Serving Benchmark*,* a fim de realizar a comparação entre os bancos de dados relacional, NoSQL e NewSQL, pois o *benchmark* YCSB fornece testes já prontos para diversos modelos de bancos de dados, ficando mais simples executar os testes, e os valores de retorno serão sob a mesma execução de informações.

Os testes foram realizados em bases de dados com valores gerados automaticamente pela ferramenta de *benchmark,* em uma única tabela*,* com as quantidades de registros e campos definidas a seguir:

a) 100.000 registros com 20 e 50 campos;
b) 500.000 registros com 20 e 50 campos;
c) 1.000.000 registros com 20 e 50 campos;
d) 2.000.000 registros com 20 e 50 campos;
e) 4.000.000 registros com 20 e 50 campos.

As possíveis métricas que serão utilizadas para a comparação serão definidas baseadas no Quadro 5.

Quadro 5 - Métricas escolhidas

Métrica	**Relacional**	**NoSQL**	**NewSQL**
Volume de dados	X	X	X
Particionamento/Replicação		X	X
Controle de Concorrência	X	X	X
Somente Leitura	X	X	X
Linguagem utilizada		X	X

Fonte: Elaborado pelo Autor

Serão selecionadas as métricas que sejam possíveis de ser executadas, e que mostrem valores com alguma diferença significativa, pois as mesmas serão realizadas em etapas, uma métrica por vez ou um conjunto de métricas, podendo também ser comparado somente dois dos três modelos de sistemas de bancos de dados, devido ao outro sistema não fornecer a característica selecionada. Os testes serão executados de forma que simulem acessos como monousuário e multiusuários, para isso serão utilizados *threads*, com quantidades definidas de 1, 50 e 100 *threads*.

Os testes serão realizados nos três modelos de sistemas de bancos de dados apresentados e comparados até o momento, além de serem escolhidos por escolha pessoal do autor para fins de aprendizado, os critérios utilizados para a escolha das ferramentas a serem testadas foram: possuir versão gratuita ou de desenvolvedor, estar no topo do ranking DB-Engines[1]. Para o modelo relacional será utilizado a ferramenta MySQL, para o modelo NoSQL será utilizado o Cassandra, que é orientado a coluna e que mais se parece com o modelo relacional orientado a linhas, e para o modelo NewSQL serão utilizados dois sistemas NuoDB e MemSQL com o intuito de verificar se existe alguma diferença significativa entre o novo modelo.

1 https://db-engines.com/en/ranking

7 AMBIENTE DE TESTES

Os testes foram executados no ambiente de testes com as configurações especificadas no Quadro 6.

Quadro 6 - Configuração do ambiente de teste

	Configuração
Processador	Intel Core™ i5 CPU M 450 2.40GHz
Memória	4 GB
Armazenamento	500 GB
Sistema Operacional	Linux Ubuntu 18.04 LTS 64 bits

Fonte: Elaborado pelo autor

7.1 PREPARAÇÃO DO AMBIENTE DE TESTES

A primeira etapa para realização dos testes, é o download e descompactação do *benchmark* YCSB, com os comandos a seguir:

```
$ curl -O --location https://github.com/brianfrankcooper/YCSB/releases/download/0.14.0/ycsb-0.14.0.tar.gz
$ tar xfvz ycsb-0.14.0.tar.gz
$ cd ycsb-0.14.0
```

O primeiro comando efetua o download do arquivo comprimido no diretório atual, com todos os sistemas de bancos de dados suportados pela ferramenta de *benchmark* YCSB. O segundo comando descompacta os arquivos no diretório atual, e o terceiro e último comando acessa a pasta em que os arquivos foram descompactados. Para verificar se o *benchmark* está configurado corretamente, pode-se executar os comandos a seguir:

```
$ ./bin/ycsb load basic -P workloads/workloada
$ ./bin/ycsb run basic -P workloads/workloada
```

O primeiro comando realiza a inserção dos dados, definido pelo parâmetro "*load*", e o segundo comando executa o *workload* A, definido pelo parâmetro "*run*".

A ferramenta de *benchmark* YCSB também permite que sejam adicionados outros parâmetros ao comando, mas na listagem apresentada a seguir estão somente os parâmetros que foram utilizados:

a) *Fieldcount*: número de campos do registro, sendo o padrão 10 campos;

b) *Recordcount:* número de registros para serem carregados no banco de dados;

c) *Operationcount:* número de operações a serem executadas em cada *workload*;

d) *Threads:* número de *threads* que serão executadas;

e) *Exportfile (>*): caminho para um arquivo em que o resultado será escrito.

7.1.1 **Preparação do ambiente de testes para o Cassandra**

A versão utilizada do banco de dados NoSQL Cassandra foi a 3.11.3, versão mais atualizada e gratuita. Para efetuar a instalação do banco de dados Cassandra, foram executados os seguintes comandos:

```
$ echo "deb http://www.apache.org/dist/cassandra/debian 311x main" | sudo tee -a
/etc/apt/sources.list.d/cassandra.sources.list
$ curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -
$ sudo apt-key adv --keyserver pool.sks-keyservers.net --recv-key A278B781FE4B2BDA
$ sudo apt-get update
$ sudo apt-get install cassandra
```

O primeiro comando adiciona o repositório do Cassandra, e em seguida adiciona e atualiza as chaves do repositório, para assim ser possível instalar a versão mais recente.

O Cassandra fornece o cqlsh, que é um cliente de conexão de linha de comando. Para dar início aos testes deve ser criado a *keyspace* ycsb, e a tabela usertable com a quantidade de campos conforme o teste que será efetuado, de 20 ou 50 campos.

Após a criação do *keyspace* e da tabela, os dados deverão ser carregados para o sistema, executando o comando *load* fornecido pela ferramente YCSB:

```
$./bin/ycsb load cassandra-cql -p hosts=localhost -P workloads/workloadc -s > load.txt
```

Após a realização do carregamento dos dados para o banco de dados, pode ser executada a carga de trabalho com o comando *run* fornecido pela ferramente YCSB:

```
$./bin/ycsb run cassandra-cql -p hosts=localhost -P workloads/workloadc -s > run.txt
```

7.1.2 **Preparação do ambiente de testes do MySQL**

A versão do MySQL utilizada foi a 8.0.12, versão mais recente do MySQL e disponível gratuitamente. Para a instalação do banco de dados MySQL são executados os seguintes comandos:

```
$ sudo apt-get install mysql-server
$ sudo mysql_secure_installation
```

O primeiro comando instala o banco de dados MySQL, e o segundo comando permite definir a senha do usuário *root* para poder ter acesso ao banco de dados. O MySQL

também fornece um cliente para conexão ao banco de dados via linha de comando, que pode ser acessado com o seguinte comando:

```
$ sudo mysql -u root -p
```

Ao acessar o banco de dados pela primeira vez, deve ser criado o *database* ycsb e a tabela usertable, o carregamento dos dados é feito através do seguinte comando *load*:

```
$ ./bin/ycsb load jdbc -P workloads/workloadc -p db.url=jdbc:mysql://127.0.0.1:3306/ycsb -p db.user=root -p db.passwd=senha -p db.batchsize=1000 -p jdbc.autocommit=false -p -cp mysql-connector-java-5.1.47.jar -s > load.txt
```

Para o banco de dados MySQL, a ferramenta YCSB solicita alguns parâmetros que devem ser informados na linha de comando, os valores que utilizados foram sugeridos pela ferramenta:

a) db.url : a url para conexão ao banco de dados MySQL
b) db.user: usuário que será utilizado para conexão;
c) db.passwd: senha do usuário para conexão;
d) db.batchsize: quantidade de linhas que devem ser inseridas antes de efetuar um *commit*;
e) jdbc.autocommit: se deve existir o *commit* automático;
f) -cp: driver utilizado para conexão.

Para realizar a execução da carga de trabalho foi utilizado o seguinte comando:

```
$ ./bin/ycsb run jdbc -P workloads/workloadc -p db.url=jdbc:mysql://127.0.0.1:3306/ycsb -p db.user=root -p db.passwd=senha -p db.batchsize=1000 -p jdbc.autocommit=false -p -cp mysql-connector-java-5.1.47.jar -s > run.txt
```

7.1.3 **Preparação do ambiente de testes do MemSQL**

A versão do MemSQL utilizada foi a 6.5.8, versão mais recente disponível. Primeiramente deve ser efetuado o download dos arquivos de instalação, executando os seguintes comandos:

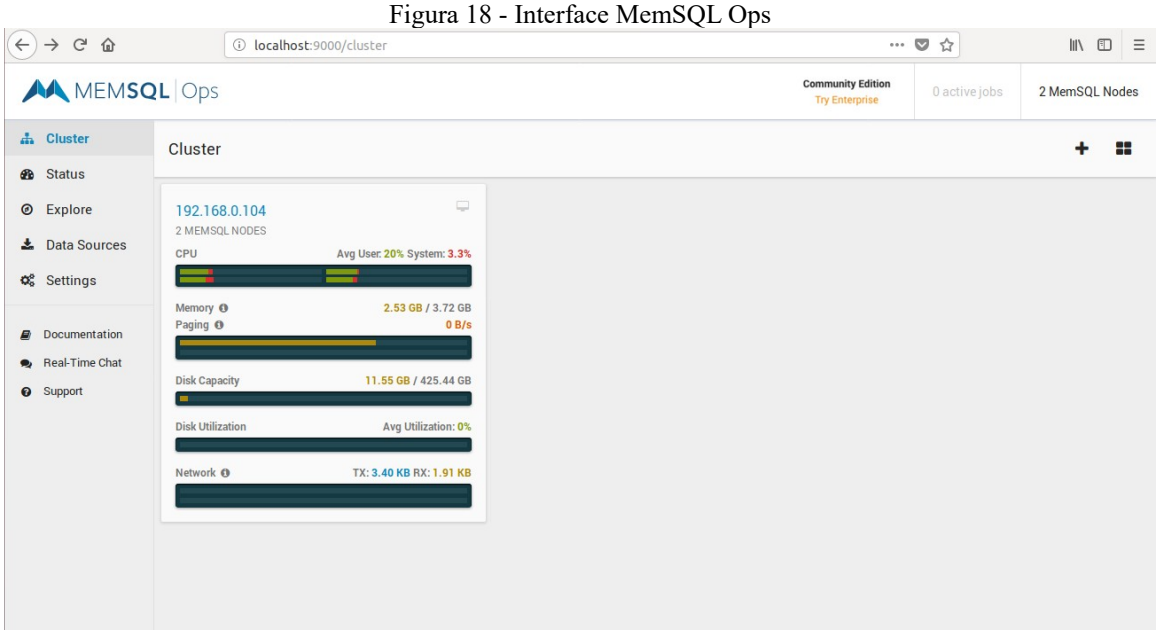
```
$ wget http://download.memsql.com/memsql-ops-6.5.8/memsql-ops-6.5.8.tar.gz
$ tar zxvf memsql-ops-6.5.8.tar.gz
```

Efetuado o download do MemSQL e descompactado os arquivos, deve ser executado o comando a seguir para efetuar a instalação:

```
$ cd memsql-ops-6.5.8
$ sudo ./install.sh –simple-cluster
```

O MemSQL fornece uma interface gráfica, Figura 18, que pode ser acessada pelo navegador utilizando o link **localhost:9000.** Essa interface possibilita monitorar o desempenho do cluster, alterar algumas configurações, expandir o cluster adicionando novos nós e explorar o banco de dados e tabelas.

Figura 18 - Interface MemSQL Ops



Fonte: Elaborado pelo autor.

Para conectar ao banco de dados, o MemSQL fornece um *client* próprio, mas também é compatível com o *client* do MySQL. Para conectar no *client* do próprio MemSQL deve ser utilizado o comando:

```
$ memsql
```

Ao acessar o banco de dados pela primeira vez, deve ser criado o *database* ycsb e a tabela usertable. O carregamento dos dados é feito através do seguinte comando *load*:

```
$ ./bin/ycsb load jdbc -P workloads/workloadc -p db.url=jdbc:mysql://127.0.0.1:3306/ycsb -p db.user=root -p db.passwd= -p db.batchsize=1000 -p jdbc.autocommit=false -p -cp mysql-connector-java-5.1.47.jar -s > load.txt
```

Para o banco de dados MemSQL, o comando é muito parecido com o comando executado para o MySQL. Além da ferramenta YCSB, há alguns outros parâmetros que devem ser informados na linha de comando. O driver para conexão utilizado é o do MySQL.

Para realizar a execução da carga de trabalho foi utilizado o seguinte comando:

```
$ ./bin/ycsb run jdbc -P workloads/workloadc -p db.url=jdbc:mysql://127.0.0.1:3306/ycsb -p db.user=root -p db.passwd= -p db.batchsize=1000 -p jdbc.autocommit=false -p -cp mysql-connector-java-5.1.47.jar -s > run.txt
```

7.1.4 **Preparação do ambiente de testes do NuoDB**

A versão utilizado do banco de dados NuoDB foi a 3.2.2.1, versão mais recente do banco de dados. Para instalação foram executados os seguintes comandos:

```
$ wget https://www.nuodb.com/modals/nojs/nuodb_info_modal_form/file/nuodb-ce_3.2.2.1_amd64.deb/3.2.2.1 Linux.deb
$ sudo dpkg -i nuodb-ce_3.2.2.1_amd64.deb
```

Após a instalação, deve ser acessado o arquivo default.properties e alterada a variável domainPassword e a variável domain, informando uma senha e um usuário. A instalação do NuoDB não cria automaticamente a variável de sistema, para isso devem ser executados os seguintes comandos:

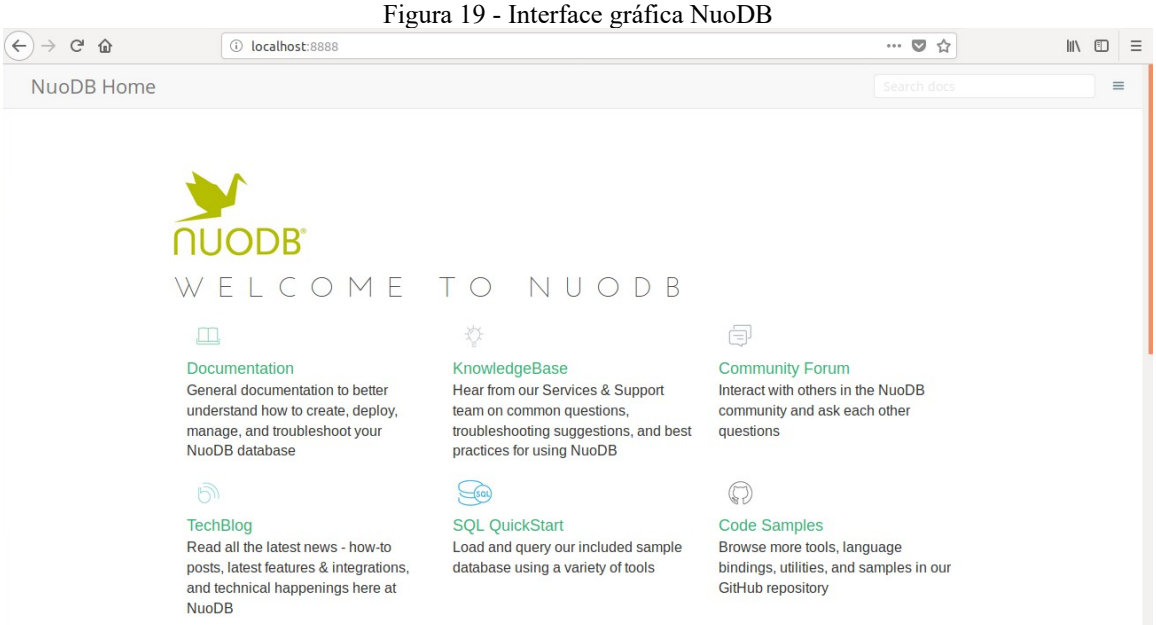
```
$ export NUODB_HOME=/opt/nuodb
$ export PATH=$PATH:$NUODB_HOME/bin
```

Após adicionar as variáveis de sistema, é possível iniciar os serviços do banco de dados NuoDB:

```
sudo service nuoagent start
sudo service nuorestsvc start
```

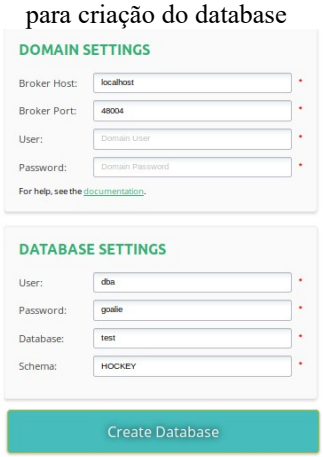
O banco de dados NuoDB fornece uma interface gráfica, Figura 19, acessando pelo navegador o link localhost:8888. No primeiro acesso pode ser criado o *database* desejado, Figura 20.

Figura 19 - Interface gráfica NuoDB



Fonte: Elaborado pelo autor

Figura 20 - Interface gráfica para criação do database



Fonte: Elaborado pelo autor.

Para acessar o banco de dados via terminal, o NuoDB fornece um *client* próprio, que pode ser acessado informando os parâmetros de *schema*, usuário e senha:

```
nuosql ycsb@localhost --user ycsb --password ycsb
```

Ao efetuar o acesso ao banco de dados NuoDB, através do terminal, pode ser criada a tabela usertable, utilizada nos testes do benchmark YCSB. Para efetuar o carregamento dos dados foi utilizado o seguinte comando:

```
sudo ./bin/ycsb load jdbc -P workloads/workloadc -p db.url=jdbc:com.nuodb://localhost/ycsb?schema=ycsb -p db.user=ycsb -p db.passwd=ycsb -p db.batchsize=1000  -p jdbc.autocommit=false -P load.txt -cp nuodbjdbc.jar -s > load.txt
```

Para a execução da carga de trabalho, foi utilizado o comando a seguir:

```
sudo ./bin/ycsb run jdbc -P workloads/workloadc -p db.url=jdbc:com.nuodb://localhost/ycsb?schema=ycsb -p db.user=ycsb -p db.passwd=ycsb -p db.batchsize=1000  -p jdbc.autocommit=false -P load.txt -cp nuodbjdbc.jar -s > run.txt
```

8 RESULTADO DOS TESTES

São apresentados os resultados dos testes realizados, duas vezes cada teste, e apresentando como resultado a média das execuções.

8.1.1 Volume de dados

O volume de dados está relacionado ao tamanho da base de dados, em *Megabytes* e *Gigabytes*, após a inserção dos registros. Essa inserção é feita pelo comando *load,* em cada sistema de dados para cada quantidade de registros e de colunas (campos) definidos.

O tamanho da base de dados pode interferir em diversas situações referentes a desempenho de leitura, escrita e backup. Alguns bancos de dados oferecem a possibilidade de configurar diferentes algoritmos de compactação. Dependendo das características de dados da tabela, a compactação pode resultar em (DATASTAX, 2018b):

a) 25-33% de redução no tamanho dos dados;
b) 25-35% de melhoria de desempenho em leituras;
c) Melhoria de desempenho de 5-10% em gravações.

O Cassandra permite três algoritmos de compactação, *LZ4Compressor*, *SnappyCompressor* ou *DeflateCompressor,* além do *LZ4Compressor, que além de* ser padrão, é o mais rápido na descompactação. A eficácia da compressão está relacionada com a velocidade de descompressão, porém a compactação extra de *DeflateCompressor* ou *SnappyCompressor* não é suficiente para compensar o desempenho reduzido de cargas de trabalho de propósito geral (DATASTAX, 2018c).

O MySQL não tem a possibilidade de escolher o algoritmo para compactar os dados, por padrão é implementado o algoritmo LZ77 para realizar a compactação das tabelas. (MYSQL, 2018b)

Não foi possível realizar todos os testes com o banco de dados MemSQL, devido a ser um banco de dados em memória e a capacidade do ambiente utilizado não suportar mais do que 4 GB.

O Quadro 7 e o Quadro 8, mostram a relação quantidade de registros por espaço ocupado em disco.

Quadro 7 - Tamanho da base de dados com 20 campos

	100.000	500.000	1.000.000	2.000.000	4.000.000
Cassandra	167,39 MB	1012,93 MB	1,97 GB	3,94 GB	7,88 GB
MySQL	360,98 MB	1,68 GB	3,45 GB	7,15 GB	14,12 GB
NuoDB	251, 5 MB	1,28 GB	2,56 GB	4,21 GB	8,44 GB
MemSQL	223,58 MB	1,09 GB	2,18 GB	-	-

Fonte: Elaborado pelo autor.

Quadro 8 - Tamanho da base de dados com 50 campos

	100.000	500.000	1.000.000	2.000.000	4.000.000
Cassandra	482,77 MB	2,41 GB	4,81 GB	9,68 GB	19,35 GB
MySQL	893,73 MB	4,47 GB	8,65 GB	17,97 GB	35,65 GB
NuoDB	628 MB	2,5 GB	5 GB	10 GB	20,02 GB
MemSQL	524,25 MB	2,66 GB	-	-	-

Fonte: Elaborado pelo autor.

Nos testes executados, o banco de dados MySQL é o que apresentou a pior relação quantidade de registros por espaço em disco ocupado. Os demais bancos de dados tiveram diferenças entre eles, mas não sendo muito significativa, sendo o banco de dados Cassandra o que apresentou uma melhor relação quantidade de registros por espaço ocupado.

8.1.2 **Particionamento/Replicação**

O principal benefício do particionamento é que o processamento da carga de trabalho em uma tabela particionada pode ser distribuída em vários servidores. Existem desvantagens ao fragmentar, por exemplo, após o particionamento, em vez de ter um único banco de dados para gerenciar, agora há vários bancos de dados, cada um com seus próprios requisitos de servidor, CPU e memória. Além disso, o particionamento pode afetar negativamente a tolerância a falhas, pois quando um banco de dados ficar indisponível, os dados nesse particionamento não estão acessíveis. É por isso que o particionamento também é frequentemente acompanhado pela replicação, para ter um conjunto de dados duplicados pronto para uso em caso de falha (MULLINS, 2018).

8.1.2.1 Cassandra

No Cassandra a distribuição e a replicação de dados andam juntas, os dados são organizados por tabela e identificados por uma chave primária, que identifica em qual nó os dados estão armazenados. O Cassandra armazena réplicas em vários nós para garantir a confiabilidade e tolerância a falhas. A estratégia de replicação determina de que modo serão armazenadas as réplicas, o número total de réplicas de um cluster é definido como fator de replicação. Um fator de replicação de 1 significa que há apenas uma cópia de cada linha em um nó, já um fator de replicação de 2 significa que existem duas cópias de cada linha, e cada uma delas está armazenada em um nó diferente. Existem duas estratégias de replicação, que são (DATASTAX, 2018d) (DATASTAX, 2018f):

a) *SimpleStrategy*: Usado apenas para um único datacenter e um rack, sendo que a primeira réplica é colocada em um nó determinado pelo particionador, e as réplicas adicionais são colocadas nos próximos nós no sentido horário do anel;
b) *NetworkTopologyStrategy*: Recomendado para a maioria das implementações devido à facilidade de expandir para outros datacenter quando necessário uma expansão futura, a réplica é colocada no mesmo datacenter, percorrendo o anel no sentido horário até atingir o primeiro nó em outro rack. A tentativa de colocar as réplicas em racks distintos se dá pelo motivo dos nós que estão num mesmo rack, tem geralmente a possibilidade de falharem ao mesmo tempo.

Para efetuar o particionamento, o Cassandra utiliza um particionador que determina como os dados são distribuídos entre os nós no cluster, incluindo réplicas. Um particionador é uma função para derivar um token que representa uma linha de sua chave de *parsing*, geralmente por *hashing*, e cada linha de dados é distribuída pelo cluster pelo valor do token. O Cassandra suporta três diferentes particionadores, que são eles:

a) *Murmur3Partitioner*: Particionador padrão utilizado pelo Cassandra, distribui uniformemente os dados pelo cluster com base nos valores de *hash MurmurHash*;
b) *RandomPartitioner*: distribui uniformemente os dados pelo cluster com base nos valores de *hash* MD5;
c) *ByteOrderedPartitioner*: mantém uma distribuição ordenada de dados lexicalmente por bytes-chave.

Para que seja possível utilizar a replicação e ou particionamento, deve ser efetuadas modificações no arquivo de configurações cassandra.yaml, localizado no diretório

/etc/cassandra, o arquivo contém muitas diretivas, com muitos comentários, e somente as diretivas a seguir que devem ser modificadas:

a) cluster_name: será o nome do cluster que será configurado;

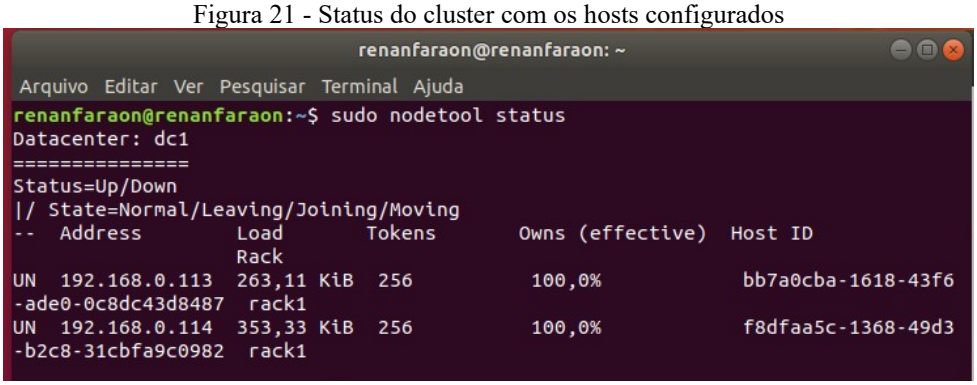
b) seeds: lista separada por vírgula dos endereços IP de cada nó no cluster;

c) listen_address: endereço IP que outros nós no cluster usarão para se conectar;

d) rpc_address: endereço IP para chamadas de procedimentos remotos;

Após ter efetuado as alterações necessárias no arquivo cassandra.yaml, devem ser criadas regras no firewall para liberar as portas 7000 e 9042 utilizadas pelo Cassandra. Com essa configuração foram transformados os nós em um cluster com as configurações padrão de replicação e particionamento. Pode ser verificado se os nós estão se comunicando, verificando o status do cluster conforme apresentado na Figura 21.

Figura 21 - Status do cluster com os hosts configurados



Fonte: Elaborado pelo autor.

8.1.2.2 NuoDB

O NuoDB é projetado desde o início como um sistema distribuído, fornecendo alta disponibilidade e resiliência sem pontos únicos de falha, utilizando uma comunicação ponto a ponto (*peer-to-peer)*, que fornece alta disponibilidade, baixa latência e um modelo de implantação fácil de gerenciar. A divisão dos serviços de processamento transacional e de armazenamento é a base para criar um banco de dados NuoDB distribuído (KYSEL, 2018).

O serviço de transação é responsável por manter a Atomicidade, Consistência e Isolamento na execução de transações, sem saber como os dados estão sendo persistidos. O serviço de gerenciamento de armazenamento é responsável por tornar os dados persistentes, fornecendo acesso aos dados quando há uma falha no cache transacional (KYSEL, 2018).

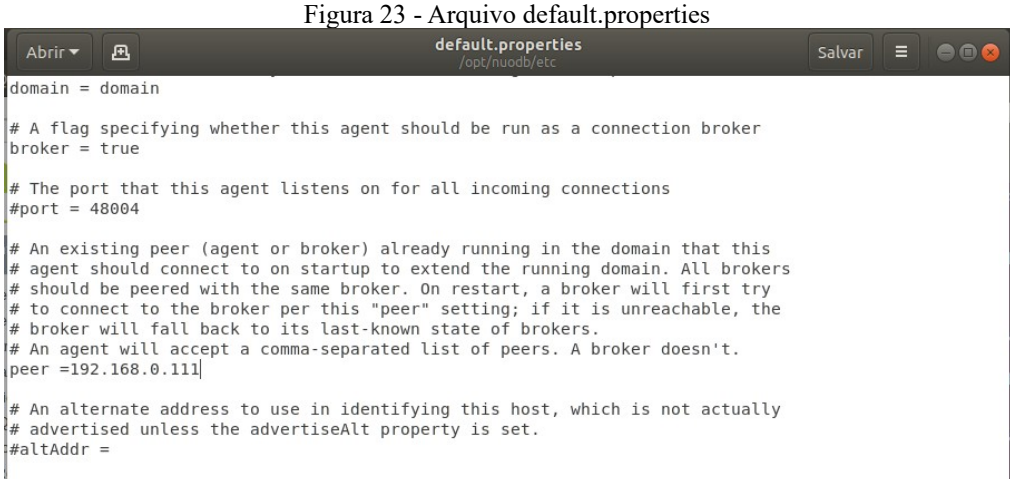
Ao instalar o banco de dados NuoDB, automaticamente são criados dois processos, um TE, mecanismo de transação, e um SM, gerenciador de armazenamento, rodando no mesmo host, que pode ser verificado usando o NuoDB Manager, Figura 22.

Figura 22 - Processos TE e SM rodando no host principal

Fonte: Elaborado pelo autor.

A partir desse banco de dados NuoDB, a execução de uma segunda TE em um segundo host dobra a quantidade de transação e fornece redundância transacional em caso de falha. A execução de mais um SM cria uma cópia consistente do banco de dados completo. Para efetuar a expansão do banco de dados NuoDB para um segundo host, deve ser efetuada a instalação do banco de dados NuoDB no segundo host, e editado o arquivo default.properties, Figura 23, que fica no diretório de instalação do NuoDB: descomentar a variável *peer* e atribuir o endereço IP do primeiro host (KYSEL, 2018).

Figura 23 - Arquivo default.properties



Fonte: Elaborado pelo autor.

Após ter concluído a configuração do segundo host, deve ser reiniciado o serviço do NuoDB no primeiro e no segundo host. Com isso o domínio que havia somente um único host, passou a ter dois hosts disponíveis, Figura 24.

Figura 24 - Hosts que pertencem ao dominio configurado

Fonte: Elaborado pelo autor.

Como o segundo host está provisionado no domínio configurado, agora ele está disponível para escalar o banco de dados. Executando o seguinte comando, será criado o serviço de TE no segundo host do *database* especificado, Figura 25:

```
nuodb [domain]> start process te host 192.168.0.114 database ycsb
```

Figura 25 - Processos que estão rodando nos dois hosts

Fonte: Elaborado pelo autor.

8.1.2.3 MemSQL

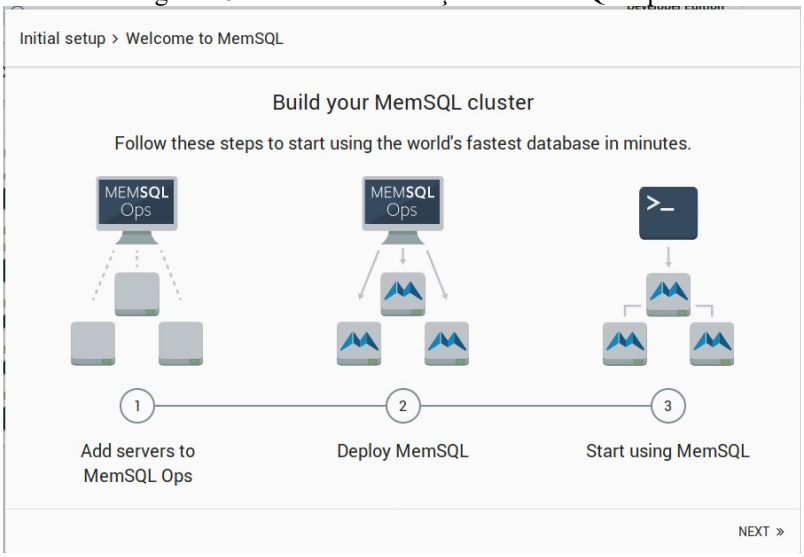
O MemSQL fornece replicação assíncrona e inclui a opção de ter um número diferente de agregadores e folhas. Os nós folha são organizados em grupos de disponibilidade, de modo que cada nó é emparelhado com um nó no outro grupo de disponibilidade. Nós de folha emparelhados replicam dados entre si e podem ser configurados para fazer isso de forma síncrona ou assíncrona. Se um nó folha ficar offline, o MemSQL fará a troca automática para seu parceiro de replicação (PROUT, 2018).

A replicação é gerenciada por partição. Cerca de metade das partições contidas em um nó folha serão mestres e metade serão escravos/réplicas. Isso ajuda o MemSQL a obter um maior paralelismo nas gravações, porque cada nó folha está ativo. Quando ocorrer uma falha, as partições escravas no nó online são promovidas para mestre e podem ser gravadas imediatamente.

O MemSQL permite que toda a configuração possa ser realizada com o cluster online e executando uma carga de trabalho normal. O MemSQL gerencia o particionamento automaticamente, e adicionar novos nós é simples, sendo necessário executar um único comando: *REBALANCE PARTITIONS*, que movimenta as partições para garantir o equilíbrio entre todas as folhas (MEMSQL, 2018b).

Efetuar a adição de mais nós ao cluster do MemSQL pode ser feito tudo através da interface web disponibilizada pela ferramenta, conhecida como MemSQL OPS, Figura 26. No segundo host que será adicionado não precisa ser efetuada nenhuma instalação prévia.

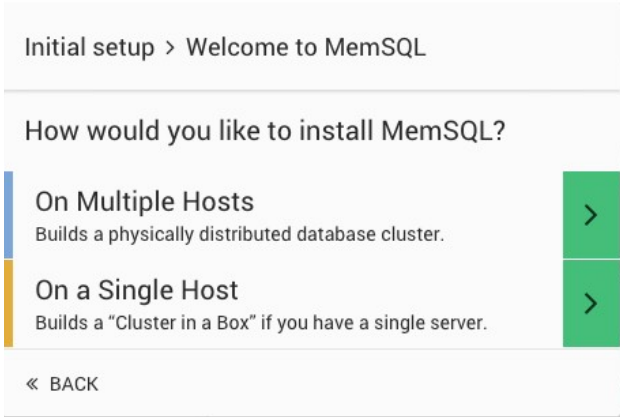
Figura 26 - Tela de inicialização do MemSQL Ops



Fonte: Elaborado pelo autor.

O MemSQL Ops tem a possibilidade de instalar o MemSQL em vários hosts ou em um único host. Uma instalação de vários hosts configura um cluster MemSQL com vários hosts, e uma única instalação de host configura todo um cluster MemSQL com um agregador principal e um nó folha em um único host.

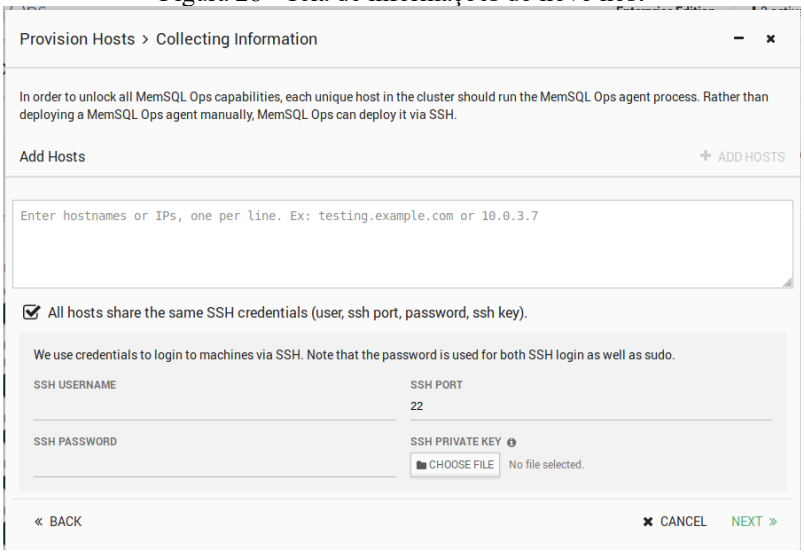
Figura 27 - Tela de escolha do modo de configuração



Fonte: Elaborado pelo autor.

Ao escolher a opção "Em vários hosts", Figura 27, será mostrada a tela de adicionar hosts, através do qual é possível inserir informações sobre seu cluster, Figura 28. Essa tela permite digitar os nomes de host e as credenciais de acesso, nome de usuário, senha e chave privada ssh.

Figura 28 - Tela de informações do novo host



Fonte: Elaborado pelo autor.

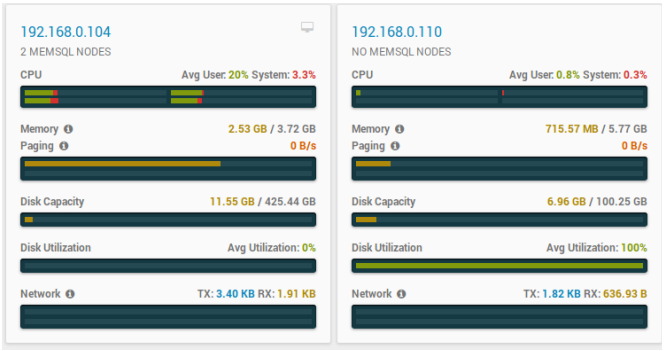
Ao clicar no botão "Provisionar Hosts", Figura 29, será instalado o agente do MemSQL Ops no host adicionado.

Figura 29 - Tela de informação do host que será provisionado

Fonte: Elaborado pelo autor.

Após ter concluído a instalação do agente no segundo host, o mesmo já aparecerá na tala inicial do cluster, Figura 30, porém sem nenhum nó folha ou agregador configurado.

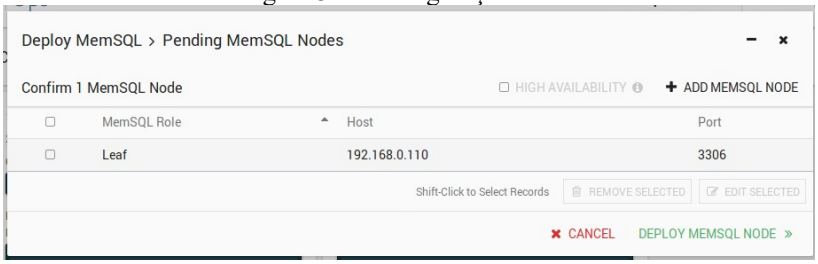
Figura 30 - Exibição dos hosts do cluster



Fonte: Elaborado pelo autor.

Depois que o host estiver configurado, é possível acrescentar os nós do MemSQL, o MemSQL automaticamente mostra uma tela com uma possível configuração do novo nó no segundo host, para seu cluster, Figura 31.

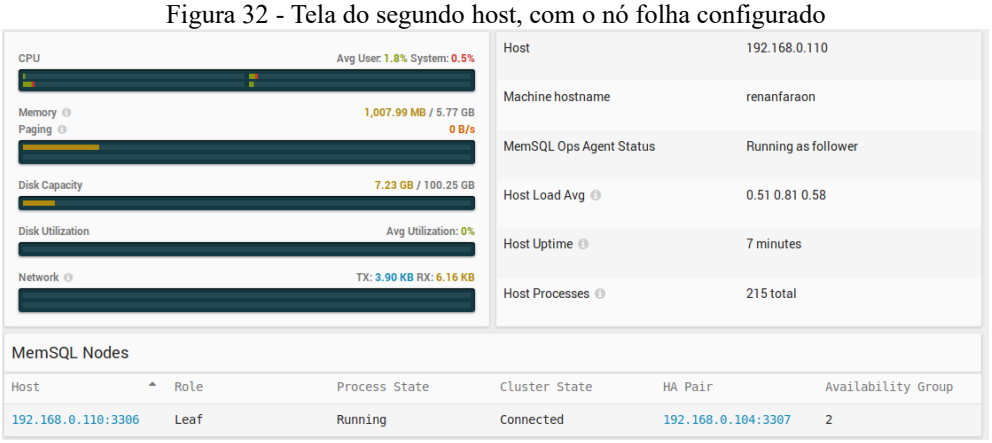
Figura 31 - Configuração do nó



Fonte: Elaborado pelo autor.

Se a configuração foi feita corretamente, ao acessar o segundo host serão exibidas todas as configurações do mesmo com os nós configurados, Figura 32.

Figura 32 - Tela do segundo host, com o nó folha configurado



Fonte: Elaborado pelo autor.

8.1.2.4 Análise

A configuração da replicação ou do particionamento em cada banco de dados testado é simples de ser feita. O banco de dados MemSQL fornece uma interface gráfica para a configuração, ficando muito mais simples de configurar do que os bancos de dados Cassandra e NuoDB que são por arquivos de configurações.

Uma grande diferença encontrada entres os três bancos de dados é a necessidade de reiniciar o serviço do banco de dados, para que sejam adicionadas as novas configurações, isso ocorre com os bancos de dados Cassandra e NuoDB. O banco de dados MemSQL permite que seja adicionado novos nós sem a necessidade de reiniciar o serviço, o banco de dados pode estar executando transações normalmente.

Pode-se perceber também que o banco de dados Cassandra possibilita configurar métodos diferentes de replicação e particionamento. Já os bancos de dados MemSQL e NuoDB não possuem configuração do método utilizado.

8.1.3 **Controle de concorrência**

No contexto do MVCC, quando uma transação atinge um registro que está sendo atualizado por outra transação, a ação executada depende do nível de isolamento. No nível de isolamento, pode-se permitir que a transação leia a atualização antes que a transação de atualização seja confirmada, ou seja, uma leitura suja, ou impedir leituras sujas e apenas ler a versão atualizada, ou seja, uma leitura consistente. Já uma tentativa de atualização que solicita uma versão de registro submetida a uma atualização não confirmada tem duas opções: pode

falhar imediatamente ou pode aguardar a conclusão da transação de atualização e tomar uma decisão com base no estado confirmado (PALMER, 2018b).

O controle de concorrência e o nível de isolamento, podem ser configurados de forma diferente em alguns dos SGBDS.

8.1.3.1 Cassandra

O Cassandra não utiliza transações ACID, com reversão ou bloqueio, porém oferece transações com suporte a atomicidade e isolamento em nível de linha, porém pode ser substituído por alta disponibilidade e desempenho, permitindo configurar a consistência para todas as conexões subsequentes, ou individual para cada transação. A lista a seguir apresenta os níveis de consistência fornecidos pelo Cassandra (DATASTAX, 2018a):

a) *All*: Fornece a maior consistência e a menor disponibilidade de qualquer outro nível;
b) *Each_quorum*: Usado em vários clusters de datacenter para manter a consistência estritamente no mesmo nível em cada datacenter. Por exemplo, escolha este nível se quiser que uma leitura falhe quando um datacenter estiver inoperante e *quorum* não puder ser alcançado nesse datacenter;
c) *Quorum*: Usado em clusters de datacenter único ou múltiplo para manter uma consistência forte em todo o cluster. Use se você puder tolerar algum nível de falha;
d) *Local_quorum*: Usado em vários clusters de datacenter com uma estratégia de posicionamento de réplica ciente de rack. Use para manter a consistência localmente (dentro do único datacenter);
e) *One*: Satisfaz as necessidades da maioria dos usuários porque os requisitos de consistência não são rigorosos. É o padrão utilizado pelo Cassandra.
f) *Two*: Semelhante ao *one*;
g) *Three*: Semelhante ao *one*;
h) *Local_one*: Em vários clusters de datacenter, um nível de consistência *one* geralmente é desejável, mas o tráfego entre datacenter não é. *local_one* realiza isso. Por motivos de segurança e qualidade, você pode usar esse nível de consistência em um datacenter offline para impedir a conexão automática a nós on-line em outros datacenters se um nó offline ficar inativo;

i) *Any*: Fornece baixa latência e uma garantia de que uma gravação nunca falha. Fornece a menor consistência e maior disponibilidade.

8.1.3.2 MySQL

O MySQL fornece tipos diferentes de mecanismos de armazenamentos que tratam de forma diferente o controle de concorrência conforme o Quadro 9.

Quadro 9 - Mecanismos de armazenamento do MySQL

	MyISAM	**Memória**	**InnoDB**	**Arquivo**	**NDB**
MVCC	Não	Não	Sim	Não	Não
Granularidade de Bloqueio	Tabela	Tabela	Linha	Tabela	Tabela

Fonte: https://dev.mysql.com/doc/refman/5.6/en/storage-engines.html

O mecanismo de armazenamento padrão e mais recomendado do MySQL é o InnoDB, único mecanismo que utiliza o MVCC, adicionando três campos a cada linha armazenada no banco de dados, um campo é o DB_TRX_ID que contêm o indicador da última transação que alterou ou inseriu a linha. DB_ROLl_PTR, é um ponteiro para um registro do log undo, segmento de *rollback*. DB_ROW_ID que contém um contador de linhas, conforme novas linhas vão sendo inseridas. O bloqueio é efetuado em nível de linha, podendo ser dos tipos a seguir:

a) Bloqueios Compartilhados e Exclusivos;
b) Bloqueio de Intenção;
c) Bloqueio de registros;
d) Bloqueio de lacuna;
e) Bloqueios de próxima chave;
f) Bloqueios de intenção de inserção;
g) Bloqueios de auto-incremento.

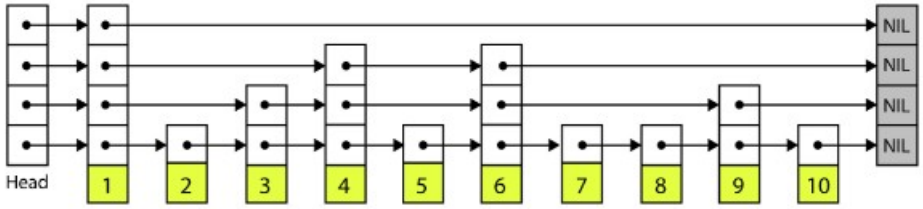
O MySQL fornece quatro níveis diferentes de isolamentos, que podem ser definido para uma única transação ou para todas as conexões subsequentes, o padrão utilizado pelo MySQL é o *repeatable read*, os que podem ser utilizados também são, *read committed*, *read uncommitted*, *serializable (MYSQL, 2018a)*.

8.1.3.3 MemSQL

O MemSQL, além de utilizar o MVCC, utiliza uma abordagem que elimina a necessidade de bloqueios, usando estruturas de dados livres de bloqueios, *skiplist*, onde as gravações nunca bloqueiam as leituras e vice-versa.

O *skiplist,* Figura 33*,* é uma estrutura de dados, parecida com uma árvore B, muito mais simples e intuitiva de ser implementada, fornecendo um nível maior de eficiência, sem a necessidade de balanceamento de árvores ou divisão de página como exigidos por outros tipos de árvores. Fornecendo maior segurança para as *threads*, um melhor paralelismo sob uma carga de leitura/gravação simultânea em que não requerer bloqueio (PROUT, 2018).

Figura 33 - Funcionamento do Skiplist



Fonte: https://www.memsql.com/blog/what-is-skiplist-why-skiplist-index-for-memsql/

Um *skiplist* é composto de elementos ligados a torres, cada torre de um *skiplist* é conectada em cada nível da torre à próxima torre na mesma altura, formando um grupo de listas interligadas, uma para cada nível do *skiplist*. As torres suportam a pesquisa binária, iniciando sempre na torre mais alta e trabalhando na parte inferior, usando os links da torre para verificar quando se deve avançar na lista ou descer a torre para um nível mais baixo (PROUT, 2018).

O nível de isolamento padrão utilizado pelo MemSQL é o *read committed*, suportando também o *repeatable read* e o *snapshot*, que as transações que modificam dados não bloqueiam as transações que leem dados, utilizando o modelo otimista.

8.1.3.4 NuoDB

O NuoDB utiliza o MVCC distribuído, suportando três níveis diferentes de isolamento conforme descritos a seguir (PALMER, 2018a) :

a) *Read_committed*: Sempre retorna a versão mais recente, oferecendo um nível de isolamento mínimo;

b) *Consistent_read*: O mais indicado para um banco distribuído, pois isola completamente as transações de gravações não conflitantes em outros nós;
c) *Write_committed*: É um nível de isolamento apenas do NuoDB. Ele é destinado ao uso em situações em que as leituras devem ser estáveis, mas as atualizações podem ser executadas corretamente com relação à versão confirmada mais recente.

8.1.3.5 Análise

O controle de concorrência fornecido pelos bancos de dados testados fornecem algum tipo de configuração, além de utilizarem o MVCC.

O banco de dados Cassandra, o único que não fornece as propriedades ACID, tem a possibilidade de ser configurado o nível de consistência em que se deseja obter, tendo nove diferentes níveis. O banco de dados MySQL possui o controle de concorrência baseado no mecanismo de armazenamento utilizado, sendo o mais indicado e por padrão, o mecanismo de armazenamento que utiliza o controle de concorrência MVCC, fornecendo sete tipos de configurações para o modelo de bloqueio em nível de linha, e quatro níveis de isolamento diferentes.

O banco de dados MemSQL fornece um controle de concorrência sem bloqueios, possuindo quatro três níveis de isolamento, mesma quantidade oferecida pelo banco de dados NuoDB.

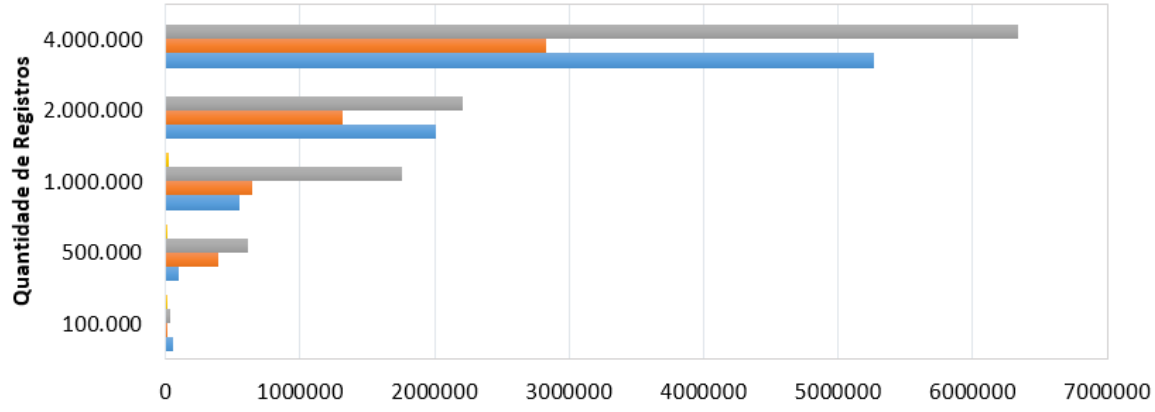
8.1.4 **Somente leituras**

Os testes de somente leitura, foram efetuados com a carga de trabalho C do *benchmark* YCSB, sendo solicitado a leitura de 10% dos registros inseridos. Toda execução era feita após o reinício do sistema, assim não haveria nenhuma informação já carregada na memória, executando duas vezes cada teste e apresentando como resultado a média das execuções no formato de milissegundos(ms).

O resultado apresentado na Figura 34, referente a execução de uma *thread* em registros com 20 campos, é possível observar que existe uma variação de qual sistema é mais eficiente, conforme a quantidade de dados aumenta. Nas execuções que foi possível a realização com o banco de dados MemSQL, o mesmo apresentou o melhor desempenho, tendo uma diferença muito significativa entre os demais bancos de dados, pode-se também observar que o banco de dados Cassandra obteve uma melhor performance até a quantidade de registros atingir 1.000.000, com quantidades de registros superiores a 1.000.000 o banco de

dados MySQL obteve um melhor desempenho entre os demais. O banco de dados NuoDB não obteve um bom desempenho na execução com uma *thread*.

Figura 34 - Tempo de execução(ms) de 1 thread com registros de 20 campos

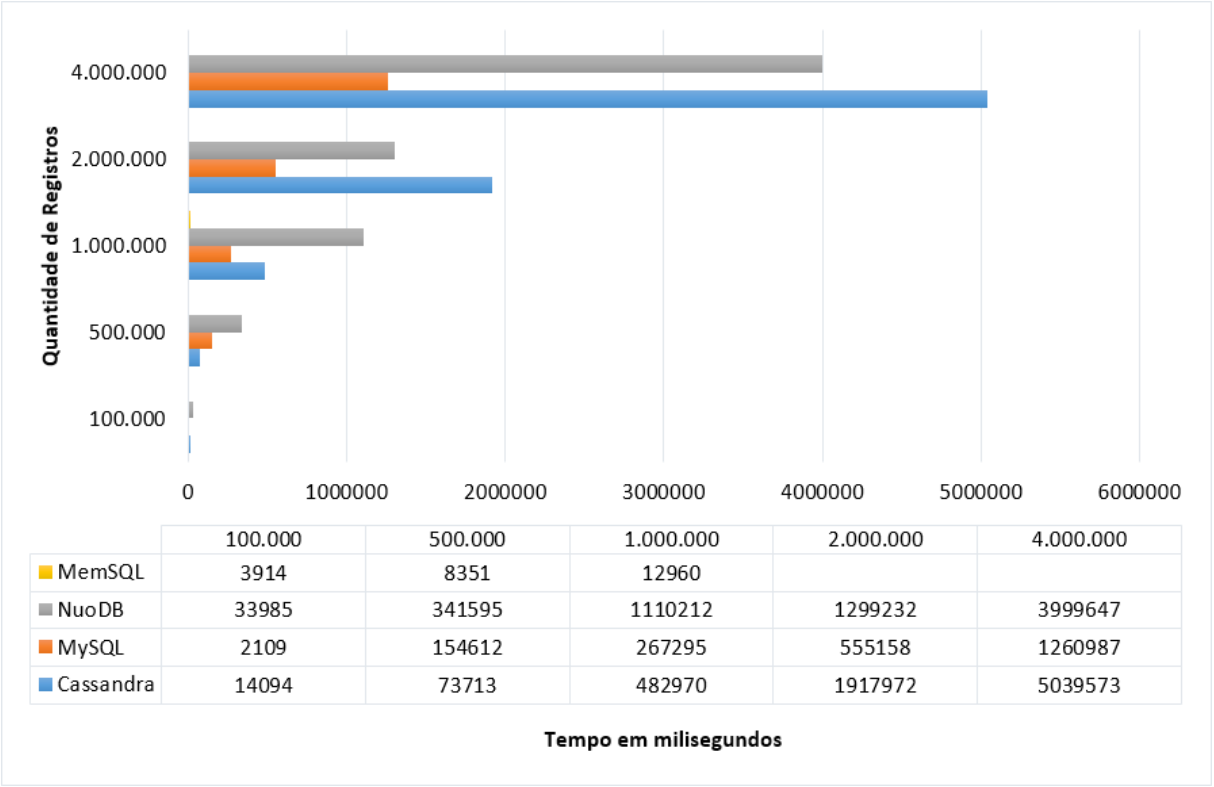


	100.000	500.000	1.000.000	2.000.000	4.000.000
MemSQL	6361	15838	26109		
NuoDB	39585	616975	1761343	2205524	6337151
MySQL	3859	390638	639123	1321053	2824550
Cassandra	51136	97750	547899	2013770	5271231

Fonte: Elaborado pelo autor.

Na Figura 35, referente a execução de 50 *threads* em registros com 20 campos, e na Figura 36, referente a execução de 100 *threads* em registros com 20 campos, foi possível observar, novamente, que o banco de dados MemSQL, nas execuções que foram possíveis de serem executas, obteve um dos melhores desempenhos, tendo uma diferença muito grande. O banco de dados MySQL acabou obtendo o melhor desempenho nas demais execuções. Porém o banco de dados NuoDB que antes com a execução de uma *thread* foi o que obteve o pior desempenho, nos testes com execuções simultâneas obteve um desempenho muito melhor do que o Cassandra quando a quantidade de registros ultrapassou o 1.000.000.

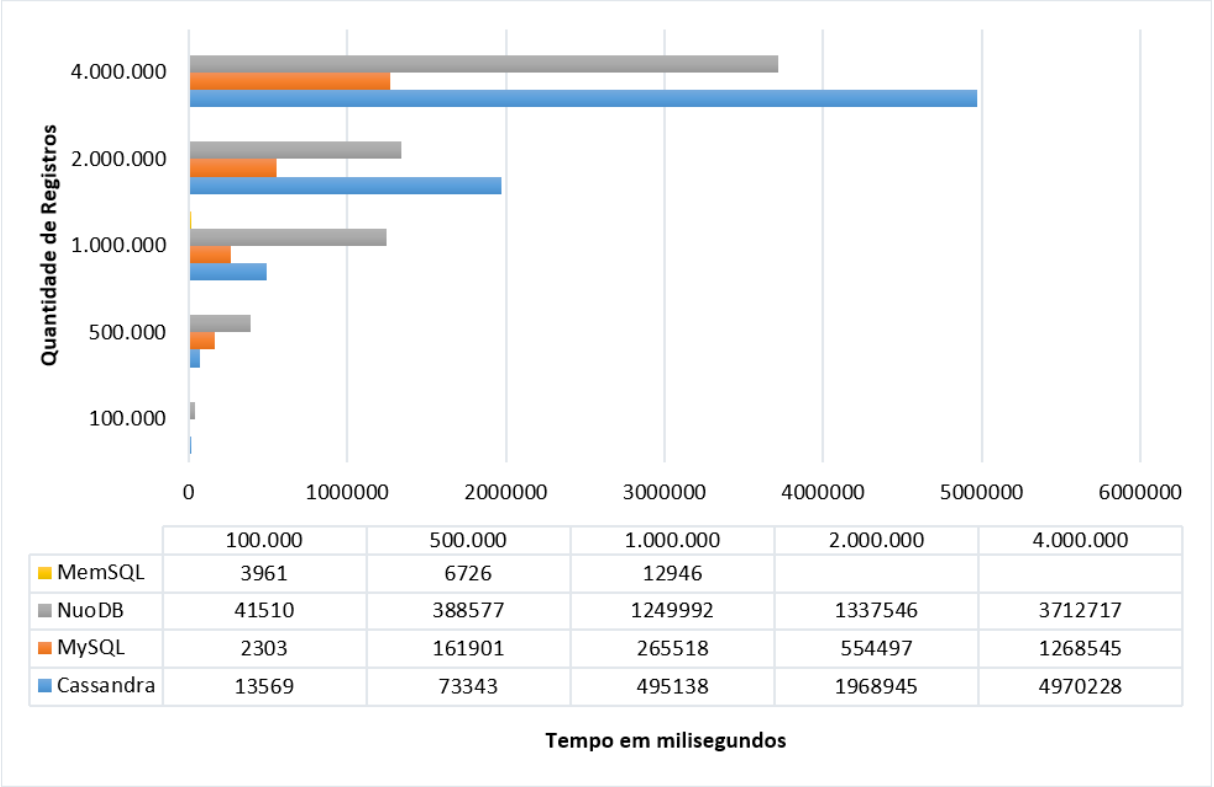
Figura 35 - Tempo de execução(ms) de 50 threads com registros de 20 campos



	100.000	500.000	1.000.000	2.000.000	4.000.000
MemSQL	3914	8351	12960		
NuoDB	33985	341595	1110212	1299232	3999647
MySQL	2109	154612	267295	555158	1260987
Cassandra	14094	73713	482970	1917972	5039573

Fonte: Elaborado pelo autor.

Figura 36 - Tempo de execução(ms) de 100 threads com registros de 20 campos

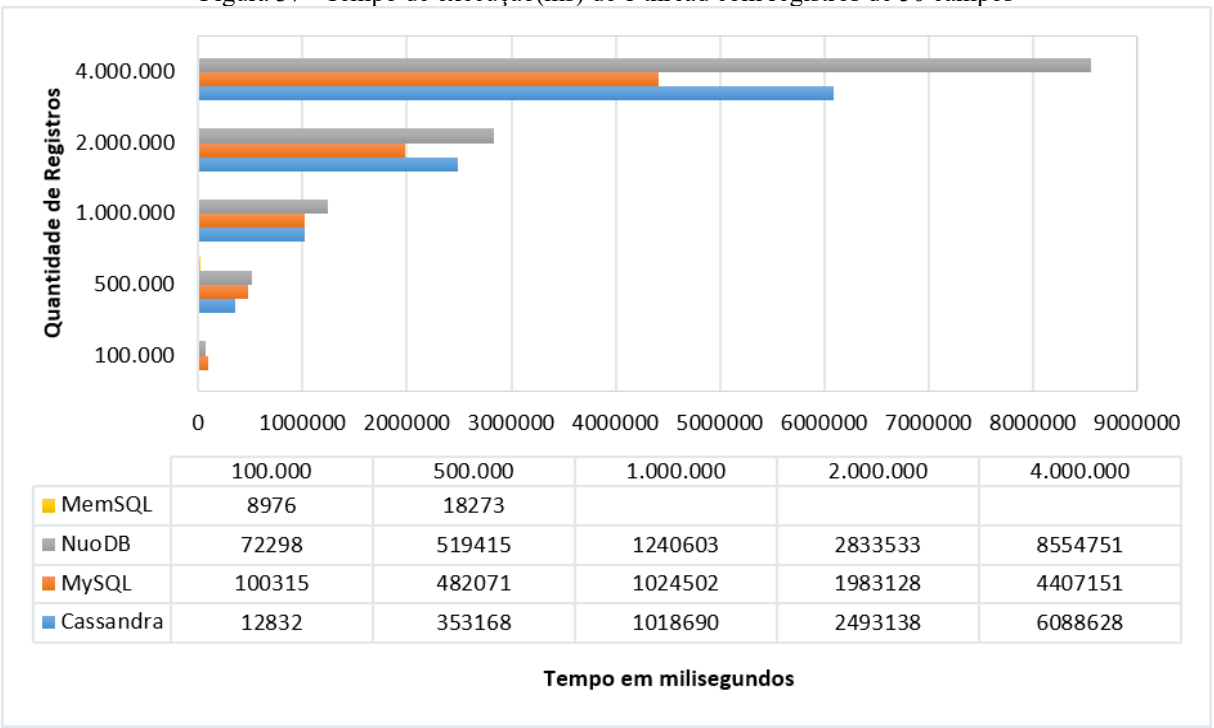


	100.000	500.000	1.000.000	2.000.000	4.000.000
MemSQL	3961	6726	12946		
NuoDB	41510	388577	1249992	1337546	3712717
MySQL	2303	161901	265518	554497	1268545
Cassandra	13569	73343	495138	1968945	4970228

Fonte: Elaborado pelo autor.

A Figura 37, referente a execução de uma *thread* em registros com 50 campos, é possível observar que o banco de dados MemSQL, nas execuções que foram efetuadas, obteve o melhor desempenho entre os demais bancos de dados. Novamente o banco de dados Cassandra teve a melhor performance entre os demais bancos de dados até a quantidade de registros atingir 1.000.000, conforme a quantidade de registros aumenta, o banco de dados Cassandra vai perdendo performance, e o banco de dados MySQL vai sendo o que tem a melhor performance, quando a quantidade de registros for superior a 1.000.000. O banco de dados NuoDB apresenta novamente a pior performance quando executado somente com uma *thread*.

Figura 37 - Tempo de execução(ms) de 1 thread com registros de 50 campos

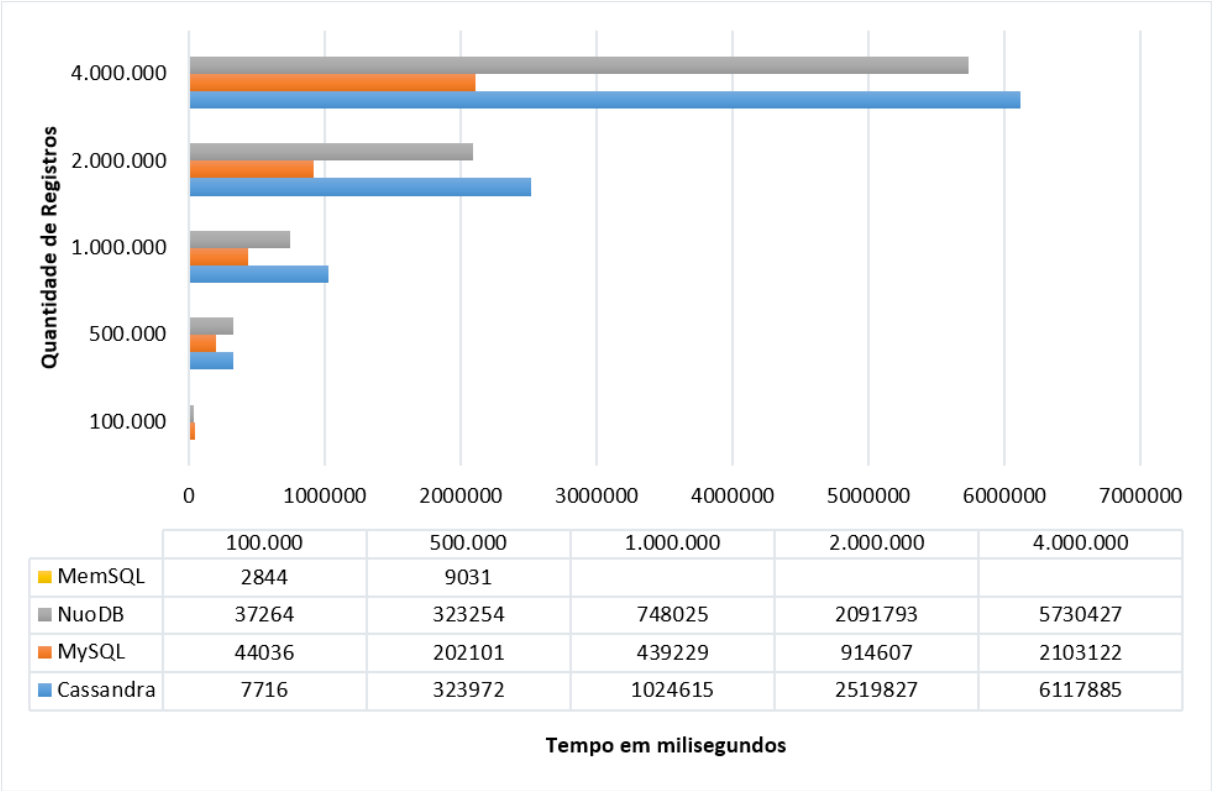


	100.000	500.000	1.000.000	2.000.000	4.000.000
MemSQL	8976	18273			
NuoDB	72298	519415	1240603	2833533	8554751
MySQL	100315	482071	1024502	1983128	4407151
Cassandra	12832	353168	1018690	2493138	6088628

Fonte: Elaborado pelo autor.

Na Figura 38, referente a execução de 50 *threads* em registros com 50 campos, e na Figura 39, referente a execução de 100 *threads* em registros com 50 campos, pode-se observar que o banco de dados MemSQL é o banco de dados que tem uma performance melhor nas execuções possíveis, nas demais execuções o banco de dados MySQL, foi o que apresentou o melhor desempenho. O banco de dados NuoDB novamente apresentou um desempenho melhor quando executado com mais *threads*, ficando assim o banco de dados Cassandra com o pior desempenho entre os bancos de dados.

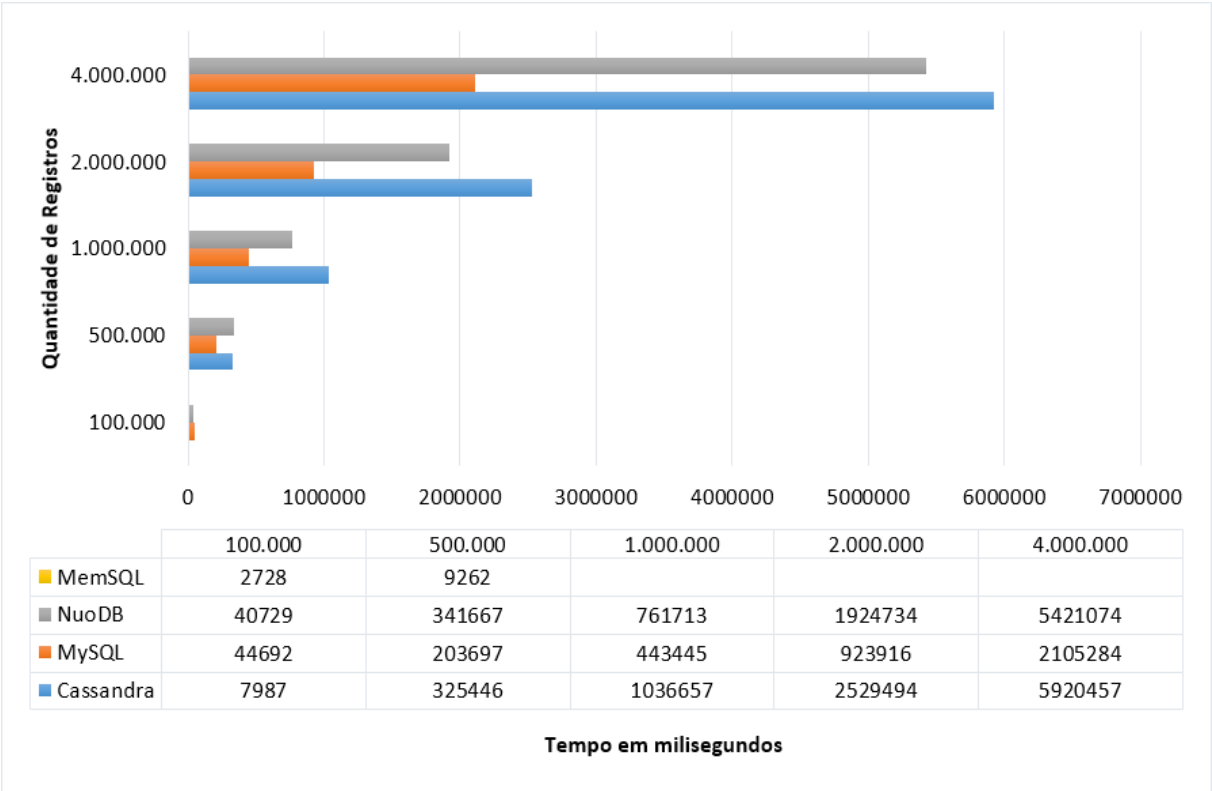
Figura 38 - Tempo de execução(ms) de 50 threads com registros de 50 campos



	100.000	500.000	1.000.000	2.000.000	4.000.000
MemSQL	2844	9031			
NuoDB	37264	323254	748025	2091793	5730427
MySQL	44036	202101	439229	914607	2103122
Cassandra	7716	323972	1024615	2519827	6117885

Fonte: Elaborado pelo autor.

Figura 39 - Tempo de execução(ms) de 100 threads com registros de 50 campos



	100.000	500.000	1.000.000	2.000.000	4.000.000
MemSQL	2728	9262			
NuoDB	40729	341667	761713	1924734	5421074
MySQL	44692	203697	443445	923916	2105284
Cassandra	7987	325446	1036657	2529494	5920457

Fonte: Elaborado pelo autor.

8.1.5 **Linguagem utilizada**

Os sistemas de banco de dados utilizados para os testes, possuem uma linguagem muito semelhantes, os modelos relacional e NewSQL utilizam a linguagem SQL, e o modelo NoSQL utilizado utiliza a linguagem CQL (Cassandra Query Language) que é muito semelhante ao SQL.

Como o modelo NoSQL, Cassandra, é um banco de dados não relacional, em que existe uma diferença no armazenamento e recuperação de dados, para a linguagem SQL o que representa o *database*, no CQL é considerado como um *keyspace,* Figura 40.

Figura 40 - Comando create

```
CREATE KEYSPACE myDatabase WITH replication =
   {'class': 'SimpleStrategy', 'replication_factor': 1};
CREATE DATABASE myDatabase;
```

Fonte: Elaborado pelo autor

Os comandos padrões do SQL *join* e *foreign key,* não são utilizados na linguagem CQL, devido ao cassandra não tratar relacionamentos entre as tabelas.

Os comandos *insert* e *update* são tratados de forma diferente entre as linguagens SQL e CQL. No momento do *insert* no CQL, por padrão, não verifica se a chave primária já existe, criando a linha se nenhuma existir ou atualizando a linha existente, sem retornar a informação do que foi efetuado, porém é possível usar uma condição para inserir somente se a linha não existir, *if not exists*. Com o comando *update*, ocorre a mesma situação, não verificando se a chave primária já existe, mas para a condição pode ser utilizado o *if* para validação da existência de alguma coluna. Para a linguagem SQL, no momento do *insert,* a chave primária não pode existir, caso contrário retornará um erro de chave duplicada, e para o *update*, será alterado o valor somente se a linha existir.

8.2 AVALIAÇÃO DOS SISTEMAS TESTADOS

Comparado os desempenhos dos bancos de dados, pode-se avaliar qual apresentou o melhor desempenho.

Quadro 10 - Resultado em formato de ranking do tamanho da base

Quantidade de campos	Colocação
20 Campos	1º - Cassandra
	2º - MemSQL
	3º - NuoDB
	4º - MySQL
50 Campos	1º - Cassandra
	2º - NuoDB
	3º - MemSQL
	4º - MySQL

Fonte: Elaborado pelo autor.

No Quadro 10, é apresentado a colocação dos bancos de dados testados referente ao tamanho de dados ocupados em *Megabytes* ou *Gigabytes*, sendo no geral, tanto para 20 campos como para a quantidade de 50 campos, em primeiro lugar o banco de dados Cassandra, modelo NoSQL, ocupando menos espaço, em segundo e terceiro lugar ficou o modelo NewSQL, os bancos de dados NuoDB e o MemSQL, alternando as posições quando usado para 20 campos e para 50 campos. Já em última posição com uma relação de quase duas vezes pior do que o primeiro colocado, o banco de dados MySQL, modelo relacional, não apresentou um desempenho muito bom.

Quadro 11- Resultado geral em formato de ranking do tamanho da base

Colocação	Banco de dados
1º	Cassandra
2º	NuoDB
3º	MemSQL
4º	MySQL

Fonte: Elaborado pelo autor

Quadro 12 - Resultado em formato de ranking do teste de somente leitura

Quantidade de campos	Colocação
20 Campos	1º - MemSQL
	2º - MySQL
	3º - Cassandra
	4º - NuoDB
50 Campos	1º - MemSQL
	2º - MySQL
	3º - Cassandra
	4º - NuoDB

Fonte: Elaborado pelo autor

Já no Quadro 11, é apresentado a colocação dos bancos de dados testados referente ao teste de somente leitura, em que foi executado o teste com 10% da base de dados inserida. Pelos poucos testes que pode ser executados com o banco de dados MemSQL, modelo NewSQL, pode-se perceber que possui um desempenho muito melhor do que os demais bancos de dados testados, devido à diferença que houve, de uma média de mais de 20 vezes em relação a maior quantidade de registros que pode ser realizado o teste, foi classificado como primeiro colocado. Em segundo lugar ficou o banco de dados MySQL, modelo relacional, em algumas situações em que a quantidade de registros não era muito grande, acabou ficando em terceiro lugar, porém na grande maioria das vezes obteve um desempenho bom e ficou em segundo lugar. Na terceira colocação ficou o banco de dados Cassandra, modelo NoSQL, mesmo tendo uma perda de desempenho conforme a quantidade de registros e a quantidade de *threads* aumentava, apresentou um desempenho pouco melhor do que o banco de dados NuoDB, modelo NewSQL.

Quadro 13 - Resultado geral em formato de ranking do teste de somente leitura

Colocação	Banco de dados
1º	MemSQL
2º	MySQL
3º	Cassandra
4º	NuoDB

Fonte: Elaborado pelo autor

No estudo mais aprofundado referente ao particionamento e replicação, pode-se perceber que o banco de dados Cassandra permite algumas configurações referentes a forma de particionar e replicar os dados, que podem ser realizadas conforme a necessidade, já os bancos de dados MemSQL e NuoDB, não permite configurar a forma que será feito o particionamento ou a replicação.

Para a análise do controle de concorrência, pode-se perceber que os bancos de dados testados apresentam uma configuração de nível de isolamento, que dependendo do que for utilizado o sistema pode ganhar desempenho e perder em consistência dos dados, ficando a critério do que se faz mais necessário e de maior importância.

Referente a linguagem utilizada, não se tem muito o que falar, pois todos os bancos de dados utilizam uma linguagem muito semelhante.

Comparando aos trabalhos pesquisados, o de Fatima e Wasnik (2016) que utilizou o banco de dados VoltDB, armazenamento em memória, obtivemos um mesmo resultado para o banco de dados NewSQL se comparado o MemSQL que também possui o armazenamento em memória. Já o trabalho de Binani et al.(2016), que efetuou o comparativo e afirmou que o NewSQL preenche todos os requisitos para ser um banco de dados para aplicativos de Big Data, também é um resultado que através do estudo comparativo das características dos modelos avaliados, obteve-se um mesmo resultado.

9 CONCLUSÃO

Com o aumento da quantidade de dados gerados, os sistemas de bancos de dados acabam ficando mais sobrecarregados, exigindo que os mesmos sejam mais eficientes em termos de confiabilidade e disponibilidade. Com isso foram surgindo diversos sistemas de bancos de dados, como os que foram estudados no trabalho apresentando.

O desenvolvimento do estudo proposto nesse trabalho, possibilitou uma análise em alguns dos sistemas de bancos de dados, através de uma comparação qualitativa e quantitativa entre os sistemas de bancos de dados relacional, NoSQL e NewSQL.

No contexto geral os bancos de dados relacionais, mesmo sendo um dos mais antigos e mais utilizados, apresentou um resultado satisfatório nos comparativos quando utilizado com grande quantidade de dados, Os testes efetuados no banco de dados MySQL mostraram que em termos de desempenho ele pode ser considerado o melhor, porém deixa a desejar em aspectos como o tamanho da base de dados, pois ocupa quase o dobro dos demais bancos de dados estudados,

Os testes que foram efetuados com o banco de dados Cassandra, modelo NoSQL, revelaram que, conforme a quantidade de dados aumentava, o desempenho diminuía em comparação aos demais bancos de dados testados.

Os bancos de dados NewSQL, que surgiram com o intuito de unir o melhor dos dois mundos (do relacional, transações ACID e do NoSQL, particionamento horizontal) apresentaram um resultado razoável, mas não tão bom como esperado. Os poucos testes que foram possíveis de serem executados no banco de dados MemSQL, mostraram que o desempenho do banco de dados em memória é muito melhor que os demais, porém, além de o servidor ter que ter uma quantidade considerável de memória, o custo em relação aos discos físicos é muito maior. O banco de dados NuoDB, que armazena os dados em disco da mesma forma que os bancos de dados MySQL e Cassandra, com a quantidade de dados utilizados nos testes não obteve um desempenho razoável para a quantidade de dados testados. Porém, conforme a quantidade de dados e *threads* aumentava, a performance tendia a ser melhor do que o do banco de dados Cassandra.

Esperava-se que o banco de dados NuoDB obtivesse um desempenho igual ou melhor do que o banco de dados Cassandra, porém conforme os testes mostraram, isso não ocorreu na maioria dos casos. Já o banco de dados MemSQL obteve o melhor desempenho de

todos os bancos de dados testados, porém como trabalha com armazenamento em memória principal, diferente dos demais, o custo de armazenamento fica muito significativo em relação aos demais.

9.1 TRABALHOS FUTUROS

Os bancos de dados NewSQL ainda são considerados novos, comparado ao tempo em que os bancos de dados relacional e NoSQL estão no mercado, então há muito a ser estudado e testado ainda.

Para uma análise mais detalhada, pode-se utilizar os mesmos bancos de dados utilizados no trabalho, comparando uma quantidade ainda maior de dados, e efetuando otimizações a fim de comparar se pode haver alguma alteração no desempenho.

Pode-se também aplicar os testes no ambiente distribuído, com os modelos NoSQL e NewSQL, visando avaliar se é possível obter um desempenho melhor, ou até mesmo se ocorre uma perda de desempenho devido aos dados serem distribuídos.

Pelo fato de não ter sido efetuado testes de inserção e atualização de dados, considera-se importante realizar testes que comparem o desempenho dos bancos de dados NewSQL com os demais bancos de dados na realização de operações de inserção, leitura e atualização de dados.

Também pode ser aplicado um comparativo entre o modelo relacional e o NewSQL, utilizando o *benchmark* TPC, ou outro, a fim de avaliar o desempenho dos bancos de dados quando utilizados com um conjunto de tabelas.

REFERÊNCIAS

DATASTAX, Inc. Partitioners. Disponível em:<https://docs.datastax.com/en/archived/ cassandra/2.1/cassandra/architecture/architecturePartitionerAbout_c.html>. Acesso em: 24 out. 2018. f

AMAZON. **O que é banco de dados relacional?**. Disponível em: <https://aws.amazon.com/pt/relational-database/>. Acesso em: 15 abr. 2018. b

AMAZON. **O que é NoSQL?: Bancos de dados não relacionais de alto desempenho**. Disponível em: <https://aws.amazon.com/pt/nosql/>. Acesso em: 18 mar. 2018. a

AMAZON. **O que é um banco de dados de documentos?**. Disponível em: <https://aws.amazon.com/pt/nosql/document/>. Acesso em: 30 abr. 2018. d

AMAZON. **O que é um banco de dados em colunas?**. Disponível em: <https://aws.amazon.com/pt/nosql/columnar/>. Acesso em: 30 abr. 2018. c

ASLETT, Matthew. **NoSQL, NewSQL and Beyond: The answer to SPRAINed relational databases**. Disponível em: <https://blogs.the451group.com/information_management/2011/04/15/nosql-newsql-and-beyond/>. Acesso em: 02 maio 2018.

BINANI, Sneha; GUTTI, Ajinkya; UPADHYAY, Shivam. SQL vs. NoSQL vs. NewSQL- A Comparative Study. **Communications On Applied Electronics.** Nova York, Usa, p. 43-46. ago. 2016.

BRITO, Ricardo J. **Bancos de Dados NoSQL x SGBDs Relacionais: Análise Comparativa**. [S.l.]: Faculdade Farias Brito e Universidade de Fortaleza, 2010. Disponível em: <http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoSQL.pdf>. Acesso em: 15 mar. 2018.

CATTELL, Rick. **Scalable SQL and NoSQL Data Stores**. Disponível em: <http://www.cattell.net/datastores/Datastores.pdf>. Acesso em: 22 abr. 2018.

DARLYMPLE, A. **NoSQL vs. NewSQL: What's the difference?.** 2016. Disponível em: <https://www.voltdb.com/blog/2016/09/22/nosql-vs-newsql-whats-difference/>. Acesso em: 03 mai. 2018.

DATASTAX, Inc. **Configuring data consistency**. Disponível em: <https://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dml_config_consistency_c.html>. Acesso em: 25 out. 2018. a

DATASTAX, Inc. **CREATE TABLE**. Disponível em: <https://docs.datastax.com/en/cql/3.3/cql/cql_reference/cqlCreateTable.html#compressSubprop>. Acesso em: 18 out. 2018. c

DATASTAX, Inc. Data distribution and replication. Disponível em:<http://docs.datastax.com/en/ archived/cassandra/2.1/cassandra/architecture/architectureDataDistributeAbout_c.html>. Acesso em: 24 out. 2018. d

DATASTAX, Inc. **When to compress data**. Disponível em: <https://docs.datastax.com/en/cassandra/3.0/cassandra/operations/opsWhenCompress.html>. Acesso em: 18 out. 2018. b

DB-ENGINES. **DB-Engines Ranking**. Disponível em: <https://db-engines.com/en/ranking>. Acesso em: 28 nov. 2018.

ELMASRI , Ramez; NAVATHE, Shamkant B. **Sistemas de Banco de Dados**. 4. ed. São Paulo: Pearson Education do Brasil Ltda., 2006. 513 p. Disponível em: <http://www.rclick.com.-br/prime/BD/Sistema_de_banco_de_dados_Navathe.pdf>. Acesso em: 15 mar. 2018.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson, 2011. xviii, 788 p. ISBN 9788579360855.

FATIMA, Haleemunnisa; WASNIK, Kumud. **Comparison of SQL, NoSQL and NewSQL databases for internet of things,** 2016 IEEE Bombay Section Symposium (IBSS), Baramati, 2016, pp. 1-6.

GARTNER. **Gartner Says Big Data Creates Big Jobs: 4.4 Million IT Jobs Globally to Support Big Data By 2015.** 2012. Disponível em: <http://www.gartner.com/newsroom/id/2207915>. Acesso em: 29 abr. 2018.

GROLINGER, Katarina et al. **Data management in cloud environments: NoSQL and NewSQL data stores** : Journal Of Cloud Computing: Advances, Systems And Applications, 2013. 22 p. v. 2. Disponível em: <https://link.springer.com/content/pdf/10.1186%2F2192-113X-2-22.pdf>. Acesso em: 02 maio 2018.

GROLINGER, Katarina et al. **Data management in cloud environments: NoSQL and NewSQL data stores**. [S.l.]: Journal Of Cloud Computing: Advances, Systems And Applications, 2013. Disponível em: <https://journalofcloudcomputing.springeropen.com/articles/10.1186/2192-113X-2-22>. Acesso em: 14 mar. 2018.

GUIMARÃES, Célio Cardoso. **Fundamentos de bancos de dados:** modelagem, projeto e linguagem SQL. Campinas, SP: UNICAMP, 2003. 270 p. ISBN 8526806335.

HECHT, Robin; JABLONSKI, Stefan. **NoSQL evaluation**: A use case oriented survey. Hong Kong, China: Cloud And Service Computing (CSC), 2011. Disponível em: <https://ieeexplore-ieee-org.ez314.periodicos.capes.gov.br/document/6138544/>. Acesso em: 04 maio 2018. a

IBM. **Estrutura do Banco de Dados Relacional**. Disponível em: <https://www.ibm.com/support/knowledgecenter/pt-br/SSLKT6_7.6.0/com.ibm.mbs.doc/configur/r_ctr_db_structures.html>. Acesso em: 15 abr. 2018.

KYSEL, Martin. **Quick Dive into NuoDB Architecture**. Disponível em: <https://www.nuodb.com/techblog/quick-dive-nuodb-architecture>. Acesso em: 25 out. 2018.

LENNON, Joe . **Explore o MongoDB**: Saiba por que esse sistema de gerenciamento de bancos de dados é tão popular. 11/Jul/2011. Disponível em: <https://www.ibm.com/developerworks/br/library/os-mongodb4/>. Acesso em: 21 mar. 2018.

LÓSCIO, B. F.; OLIVEIRA, H. R. d.; PONTES, J. C. d. S. **Nosql no desenvolvimento de aplicações web colaborativas**.VIII Simpósio Brasileiro de Sistemas Colaborativos, Brasil, 2011.

MARR, Bernard. **Big Data: The 5 Vs Everyone Must Know.** 2014. Disponível em: <https://www.linkedin.com/pulse/20130827231108-64875646-big-data-the-mega-trend-that-will-impact-all-our-lives>. Acesso em: 29 abr. 2018.

MARR, Bernard. **Big Data: The Mega-Trend That Will Impact All Our Lives**. 2013. Disponível em: <https://www.linkedin.com/pulse/20130827231108-64875646-big-data-the-mega-trend-that-will-impact-all-our-lives>. Acesso em: 29 abr. 2018.

MEMSQL. **Durability: How MemSQL Keeps Your Data Safe**. Disponível em: <https://www.memsql.com/content/durability/>. Acesso em: 25 out. 2018. c

MEMSQL. **How MemSQL Works**. Disponível em: <https://docs.memsql.com/introduction/latest/how-memsql-works/>. Acesso em: 06 maio 2018. a

MEMSQL. **MemSQL Architecture: Technology Innovations Power Convergence of Transactions and Analytics.** Disponivel em: <https://www.memsql.com/content/architecture/>. Acesso em: 06 maio 2018. b

MEMSQL. **Optimizing Table Data Structures**. Disponível em: <https://docs.memsql.com/tutorials/v6.5/optimizing-table-data-structures/>. Acesso em: 18 out. 2018. d

MONGODB. **Sharding**. Disponível em: <https://docs.mongodb.com/manual/sharding/>. Acesso em: 23 abr. 2018.

MULLINS, Craig S. **What are the Database Scalability methods?**. Disponível em: <https://www.nuodb.com/techblog/what-are-database-scalability-methods>. Acesso em: 25 out. 2018.

MYSQL. **How Compression Works for InnoDB Tables**. Disponível em: <https://dev.mysql.com/doc/refman/5.6/en/innodb-compression-internals.html)>. Acesso em: 18 out. 2018. b

MYSQL. **Transaction Isolation Levels**. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html>. Acesso em: 23 out. 2018. a

NEO4J. **What is a Graph Database and the Property Graph?**. Disponível em: <https://neo4j.com/developer/graph-database/>. Acesso em: 1 mai. 2018.

NUODB, **Introduction to Table Partitions and Storage Groups**. Disponível em: <http://doc.nuodb.com/Latest/Content/Introduction-to-Table-Partitions-and-Storage-Groups.htm>. Acesso em: 25 out. 2018. e

NUODB. **NuoDB Architecture.** Disponível em: <http://go.nuodb.com/rs/nuodb/images/Technical-Whitepaper.pdf>. Acesso em: 07 maio 2018. c

NUODB. **NuoDB Emergent Architecture**. Disponível em: <http://go.nuodb.com/rs/nuodb/images/Greenbook_Final.pdf>. Acesso em: 07 maio 2018. b

NUODB. **NuoDB Technical Overview.** Disponível em: <https://www.nuodb.com/resources/technical-overview>. Acesso em: 07 maio 2018. d

NUODB. **Product Overview.** Disponível em: <https://www.nuodb.com/product-overview>. Acesso em: 07 maio 2018. a

OLHAR DIGITAL. Redação. **Afinal, o que é Big Data?**. 2013. Disponível em: <https://olhardigital.com.br/pro/noticia/afinal,-o-que-e-big-data/35366>. Acesso em: 13 jul. 2018.

ORACLE. **A Relational Database Overview**. Disponível em: <https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html>. Acesso em: 15 abr. 2018. b

ORACLE. **Database Concepts - Release 12.2**. Disponível em: <https://docs.oracle.com/en/database/oracle/oracle-database/12.2/cncpt/introduction-to-oracle-database.html#GUID-A42A6EF0-20F8-4F4B-AFF7-09C100AE581E>. Acesso em: 12 mar. 2018. a

ORACLE. **Database Concepts**. Disponível em: <https://docs.oracle.com/cloud/latest/db112/CNCPT/intro.htm#CNCPT001>. Acesso em: 01 jun. 2018. c.

ORACLE. **Oracle NoSQL Database Key-Value Pairs**. Disponível em: <http://www.oracle.-com/technetwork/products/nosqldb/overview/key-value-497224.html>. Acesso em: 1 mai. 2018. c

PALMER, Trek. **Isolation Levels in Terms of MVCC**. Disponível em: <https://www.nuodb.-com/techblog/isolation-levels-terms-mvcc>. Acesso em: 25 out. 2018. a

PALMER, Trek. **MVCC Part 1: An Overview**. Disponível em: <https://www.nuodb.com/techblog/mvcc-part-1-overview>. Acesso em: 25 out. 2018. b

PETTEY, Christy; GOASDUFF, Laurence. **Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data**. Disponível em: <https://www.gartner.-com/newsroom/id/1731916>. Acesso em: 28 abr. 2018.

PRICE, Jason. **Oracle Database 11G SQL: Domine SQL e PL/SQL no banco de dados Oracle**. [S.l.]: Bookman, 2009. 684 p.

PROUT, Adam. **The Story Behind MemSQL's Skiplist Indexes**. Disponível em: <https://www.memsql.com/blog/what-is-skiplist-why-skiplist-index-for-memsql/>. Acesso em: 25 out. 2018.

PUGA, Sandra; FRANÇA, Edson; GOYA, Milton. **Banco de dados: implementação em SQL, PL/SQL e Oracle 11g**. São Paulo: Pearson Education do Brasil, 2013. Disponível em: <https://ucsvirtual.ucs.br/startservico/PEA/>. Acesso em: 15 abr. 2018.

ROB, Peter; CORONEL, Carlos. **Sistemas de banco de dados:** projeto, implementação e gerenciamento. Austrália: Cengage Learning, 2011. xxi, 711 p. ISBN 9788522107865.

RYAN, John. **Big Data: Velocity in Plain English**. Disponível em: <https://www.linkedin.-com/pulse/big-data-velocity-plain-english-john-ryan/>. Acesso em: 12 jun. 2018.

SAS. **O que é Big Data.** Disponível em <https://www.sas.com/pt_br/insights/big-data/what-is-big-data.html/>. Acesso em 28 abr. 2018.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistema de banco de dados**. Rio de Janeiro: Elsevier, c2012. [36], 861 p. ISBN 978853524535

STONEBRAKER, Michael; CATTELL, Rick. **10 Rules For Scalable Performance in Simple Operation' Datastores**. Disponível em: <https://cacm.acm.org/magazines/2011/6/108651-10-rules-for-scalable-performance-in-simple-operation-datastores/fulltext>. Acesso em: 22 abr. 2018.

STONEBRAKER, Michael. **New Sql: An Alternative to Nosql and Old Sql For New Oltp Apps**. Blog@Cacm, 2011. Disponível em: <https://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext>. Acesso em: 14 mar. 2018

TAURION, Cezar. **Big Data**. Rio de Janeiro: Brasport, 2013. 110 p. Disponível em: <https://www.scribd.com/doc/259741402/Big-Data-Cezar-Taurion>. Acesso em: 28 abr. 2018.

TIWARI, Shashank. **Professional NoSQL**. Indianapolis, Indiana: John Wiley & Sons, Inc, 2011. 386 p.

TPC. **Overview of the TPC-C Benchmark.** Disponível em: <http://www.tpc.org/tpcc/detail.asp/>. Acesso em 13 jun. 2018.

VOLTDB. **Availability.** Disponível em: <https://docs.voltdb.com/UsingVoltDB/ChapKSafety.php/>. Acesso em 10 maio 2018. b

VOLTDB. **How VoltDB Works.** Disponível em: <https://docs.voltdb.com/UsingVoltDB/IntroHowVoltDBWorks.php#IntroSerialize/>. Acesso em 10 maio 2018. c

VOLTDB. **NewSQL.** Disponível em: **<**https://www.voltdb.com/blog/category/newsql/**>.** Acesso em: 03 mai. 2018**.**

VOLTDB. **Using VoltDB.** Disponível em: <https://docs.voltdb.com/UsingVoltDB/index.php/>. Acesso em 10 maio 2018. a

YCSB. **Yahoo! Cloud Serving Benchmark (YCSB).** Disponível em: <https://github.com/brianfrankcooper/YCSB/wiki/>. Acesso em 02 jun. 2018.

YEGULALP, Serdar. **What is NoSQL? NoSQL databases explained**. Disponível em: <https://www.infoworld.com/article/3240644/nosql/what-is-nosql-nosql-databases-explained.html>. Acesso em: 21 abr. 2018.