Pós-Graduação em Ciência da Computação

LUCAS FREIRE MELO

Novas Abordagens para Sistemas de Gerenciamento de Grandes Volumes de Dados: Um Estudo Comparativo

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife
2018

Lucas Freire Melo

Novas Abordagens para Sistemas de Gerenciamento de Grandes Volumes de Dados: Um Estudo Comparativo

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Banco de Dados

Orientador(a): Ana Carolina Brandão Salgado

Recife
2018

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

M528n Melo, Lucas Freire
Novas abordagens para sistemas de gerenciamento de grandes volumes de dados: um estudo comparativo / Lucas Freire Melo. – 2018.
87 f.: il., fig., tab.

Orientadora: Ana Carolina Brandão Salgado.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2018.
Inclui referências.

1. Engenharia de software. 2. Gerenciamento de banco de dados. I. Salgado, Ana Carolina Brandão (orientadora). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2019-008

Lucas Freire Melo

Novas Abordagens para Sistemas de Gerenciamento de Grandes Volumes de Dados: Um Estudo Comparativo

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 19/12/2018

BANCA EXAMINADORA

_____

Prof. Dr. Fernando da Fonseca de Souza
Centro de Informática/UFPE

_____

Profa. Dra. Damires Yluska de Souza Fernandes
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba

_____

Profa. Dra. Ana Carolina Brandão Salgado
Centro de Informática/UFPE
(**Orientadora**)

RESUMO

As tecnologias de bancos de dados têm tentado se adaptar à nova realidade de sistemas de informação com grandes volumes, variedade e velocidade de atualização dos dados. No início dos anos 2000, surgiu uma nova geração de sistemas de gerenciamento de bancos de dados voltados para lidar, a um baixo custo, com esses cenários de crescimento no acesso e no volume de informações. Alguns desses sistemas, os chamados NoSQL (*Not Only SQL*), trouxeram soluções que diferem do consagrado modelo relacional, particularmente no que diz respeito a garantir a consistência dos dados. Por outro lado, os sistemas de bancos de dados NewSQL surgiram com o propósito de garantir a escalabilidade horizontal dos sistemas NoSQL, contudo mantendo as garantias do modelo relacional. Os sistemas NoSQL já se encontram bem estabelecidos em termos de participação no mercado, enquanto os sistemas NewSQL ainda buscam construir seu espaço. Chegamos, então, a um cenário atual no qual é vasta a quantidade de opções de SGBD disponíveis na academia e na indústria. Na literatura existem alguns trabalhos que realizaram comparações entre SGBD da nova geração, contudo poucos analisaram a questão do desempenho dos sistemas NoSQL e NewSQL (um critério de grande importância na escolha de um SGBD para uma aplicação). Nesse contexto, este trabalho propõe a realização de um estudo experimental, com o objetivo de comparar o desempenho de sistemas NoSQL e NewSQL em cenários de grandes volumes de dados, de modo a apoiar a escolha de um SGBD para aplicações em diversos cenários de dados. Após a execução dos experimentos, conseguimos confirmar o bom desempenho dos sistemas NoSQL ao lidar com grandes volumes de dados, em especial os sistemas Redis e MongoDB. Também verificamos que o sistema NewSQL VoltDB teve desempenho menor do que os sistemas NoSQL, o que o torna pouco competitivo para ser utilizado por aplicações que necessitem trabalhar com grandes volumes de dados.

Palavras-chave: Big Data. Engenharia de Software Experimental. NewSQL. NoSQL. Sistemas de Gerenciamento de Bancos de Dados.

ABSTRACT

Database technologies have tried to adapt to the new reality of information systems with large volumes, variety and speed of data update. In the early 2000s, a new generation of database management systems emerged to deal, at a low cost, with these scenarios of growth in access and volume of information. Some of these systems, called NoSQL (Not Only SQL), have brought solutions that differ from the established relational model, particularly with respect to ensuring data consistency. On the other hand, the NewSQL database systems came up with the purpose of guaranteeing the horizontal scalability of the NoSQL systems, while maintaining the guarantees of the relational model. NoSQL systems are already well established in terms of market share, while NewSQL systems are still looking for some space in this market. We have now reached a current scenario in which the number of DBMS options available in the academy and industry is vast. In the literature there are some works comparing DBMS of the new generation, however, few studies have analyzed the performance of the NoSQL and NewSQL systems (a criterion of great importance when choosing a DBMS for an application). In this context, this work proposes the realization of an experimental study, aiming at comparing the performance of NoSQL and NewSQL systems in scenarios of large data volumes, in order to support the choice of a DBMS for applications in diverse data scenarios. After performing the experiments, we were able to confirm the good performance of the NoSQL systems when dealing with large volumes of data, especially the Redis and MongoDB systems. We also noticed that the NewSQL VoltDB system had lower performance than NoSQL systems, which makes it uncompetitive to be used by applications that need to work with large volumes of data.

Keywords: Big Data. Software Engineering Experimentation. NewSQL. NoSQL. Database Management Systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

PRINCIPAIS ABREVIAÇÕES

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
ACM	*Association for Computing Machinery*
CPU	*Central Processing Unit*
IDC	*International Data Corporation*
IMDb	*Internet Movie Database*
JMH	*Java Microbenchmark Harness*
JSON	*JavaScript Object Notation*
NoSQL	*Not Only SQL*
OLTP	*Online Transaction Processing*
PB	Petabyte
PDF	*Portable Document Format*
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional
SQL	*Structured Query Language*
XML	*Extensible Markup Language*
ZB	Zettabyte

SUMÁRIO

1

INTRODUÇÃO

Neste capítulo apresentamos uma introdução sobre o problema de gerenciamento de grandes volumes de dados, com foco na comparação de duas abordagens de sistemas desenvolvidos pela academia e pela indústria: NoSQL (*Not Only SQL*) e NewSQL. Nosso trabalho propõe a realização de um estudo experimental, com o objetivo de identificar como o desempenho dos SGBD NewSQL difere do desempenho dos SGBD NoSQL em diferentes cenários de carga de grandes volumes de dados. Inicialmente, na Seção 1.1 apresentamos uma justificativa e motivação para este trabalho. Na Seção 1.2 é descrita a caracterização do problema. A Seção 1.3 apresenta os objetivos deste trabalho. Por fim, a estrutura da dissertação é vista na Seção 1.4.

1.1 Motivação

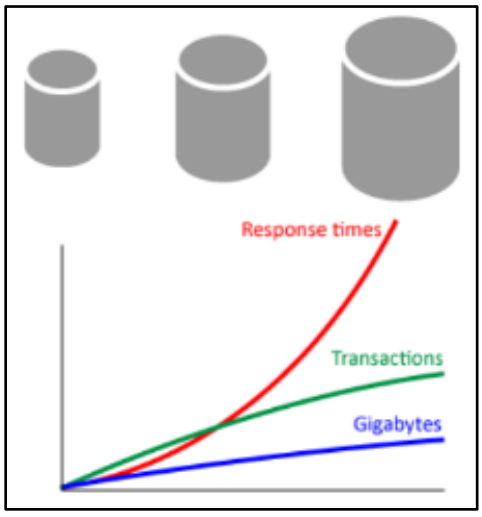
Dentre os mais diversos modelos de dados já criados, o modelo relacional tem sido o principal deles há mais de 30 anos. Criado no início da década de 1970, suas implementações são conhecidas como Sistemas de Gerenciamento de Bancos de Dados Relacionais (ou SGBDR). Suas primeiras implementações foram o System R, da IBM, e o INGRES, desenvolvido na Universidade da Califórnia. Ao longo dos anos,

diversos outros SGBDR surgiram, dentre as quais se destacaram no mercado: Oracle[1], Microsoft SQL Server[2] e MySQL[3] (PAVLO; ASLETT, 2016).

Nos anos 2000, com o surgimento de aplicações de larga escala na Internet, percebeu-se que os sistemas de bancos de dados estavam sendo o gargalo dessas novas aplicações, visto que a quantidade de usuários concorrentes e a de operações executadas eram muito maiores do que os SGBD eram capazes de suportar (PAVLO; ASLETT, 2016).

Segundo AgilData (2016), os crescimentos do volume de dados e do volume de transações seguem uma tendência linear, enquanto o tempo de resposta do banco de dados cresce de forma logarítmica, como pode ser visto na Figura 1. Isso se deve ao fato de que os SGBD tradicionais dependem bastante dos três principais componentes de um computador (CPU, memória e disco) e testes de *benchmark* em um único servidor demonstram que, para obter o melhor desempenho desejado, é preciso que esses três fatores sejam melhorados (a chamada escalabilidade vertical). Além de ser uma estratégia financeiramente cara, com o passar do tempo ela passará a não mais trazer o retorno esperado.

Figura 1 − Tendência de crescimento de dados, transações e tempo de resposta



Fonte: AgilData (2016)

De acordo com o IDC (*International Data Corporation*), o volume de dados criados e consumidos no mundo tem aumentado de maneira exponencial, praticamente dobrando a cada dois anos, indo de 130 exabytes em 2005 para 1.200 exabytes em

[1] http://www.oracle.com
[2] http://www.microsoft.com/sql/
[3] http://www.mysql.com

2010, 8.000 exabytes em 2015 e estimados 40.000 exabytes em 2020 (VENKATESH; NIRMALA, 2012). Esse crescente volume de dados é, em grande parte, não-estruturado e se encontra distribuído por diversas fontes de dados heterogêneas. Nesse mesmo cenário, verificamos também as necessidades de alto desempenho e alta disponibilidade por parte das aplicações de *Big Data* e da Web 2.0 e o alto custo de se implantar a escalabilidade vertical (por meio do acréscimo de hardware). Todos esses fatores levaram ao surgimento e à popularização de uma nova geração de sistemas voltados para o gerenciamento de grandes volumes de dados.

Tais sistemas surgiram com a proposta de rodarem em *clusters* compostos de um grande número de hardware de baixo custo, provendo a escalabilidade horizontal, ou seja, permitindo que máquinas possam ser adicionadas ou removidas do *cluster* sem que haja grande impacto nas aplicações.

Dentro dessa proposta, surgiram diversos sistemas para gerenciamento desses grandes volumes de dados. De início, apareceram os sistemas chamados NoSQL (*Not Only SQL*), trazendo soluções que iam em desacordo com o modelo relacional, sobretudo em relação a garantias quanto à consistência dos dados (STRAUCH; SITES; KRIHA, 2011). Posteriormente, novos sistemas passaram a ser desenvolvidos buscando garantir a escalabilidade horizontal dos NoSQL, porém mantendo as garantias dos sistemas relacionais. Esses foram inicialmente chamados por Aslett (2011) de NewSQL.

1.2 Caracterização do Problema

A escolha de um SGBD a ser utilizado por uma aplicação passa por diferentes critérios. Um deles é o custo financeiro do SGBD. Não apenas quanto ao seu valor de aquisição (proprietário ou *open-source*), mas também quanto a despesas relacionadas a manutenção e suporte do mesmo.

Um segundo critério é o modelo de dados utilizado pelo SGBD. Além do modelo relacional, utilizado pelos tradicionais SGBDR e os NewSQL, existem outros modelos de dados mais flexíveis, utilizados pelos NoSQL, como chave-valor, documentos, família de colunas e grafos (SADALAGE; FOWLER, 2013). O uso desses modelos de dados mais flexíveis auxilia as aplicações que necessitam lidar com dados semi-estruturados, os quais não possuem esquema definido.

Além desses critérios, o desempenho do SGBD também é um fator predominante, visto o crescente volume de dados que precisam ser armazenados e a necessidade de processar esses dados no menor espaço de tempo possível.

Diante desse cenário, neste trabalho foi feito um levantamento das principais características dos sistemas de gerenciamento de grandes volumes de dados da nova geração (NoSQL e NewSQL) e, além disso, foi realizado um estudo experimental de forma a medir o desempenho desses sistemas, no sentido de responder às seguintes questões de pesquisa: (Q1) Como o desempenho dos SGBD NoSQL e NewSQL diferem entre si? (Q2) Como esses sistemas se comportam diante de situações com grandes volumes de dados? (Q3) A partir de certos requisitos de uma aplicação que necessite lidar com grandes volumes de dados, qual sistema/abordagem é a mais adequada? (Q4) Os sistemas NewSQL estão prontos para, de fato, competir com os NoSQL?

1.3 Objetivos

A proposta deste trabalho de pesquisa é realizar uma revisão da literatura, de forma a obtermos um levantamento das características, aspectos positivos e negativos dessa nova geração de sistemas de gerenciamento de dados. Além disso, serão realizados experimentos com o intuito de identificar como o desempenho dos diversos tipos de SGBD NoSQL e NewSQL diferem entre si nos diferentes cenários de carga com grandes volumes de dados. Para isso, foram estabelecidos alguns objetivos específicos. São eles:

- Definir formalmente o estudo experimental, formulando suas hipóteses;
- Planejar o experimento, definindo fatores, variáveis, parâmetros, sujeitos, objetos e desenho;
- Definir a arquitetura do experimento e implementar a solução;
- Executar os experimentos unitários;
- Analisar os dados coletados e validar as hipóteses formuladas.

1.4 Estrutura da Dissertação

A seguir, apresentamos a estrutura deste trabalho, além do presente capítulo:

- O Capítulo 2 apresenta a fundamentação conceitual referente aos conceitos relevantes para o desenvolvimento desta dissertação. Serão

descritos assuntos referentes a Sistemas de Gerenciamento de Banco de Dados, sistemas NoSQL, sistemas NewSQL e Engenharia de Software Experimental;

- O Capítulo 3 apresenta os principais trabalhos relacionados e faz uma análise comparativa dos mesmos, destacando também como o presente trabalho pretende se diferenciar dos demais;
- O Capítulo 4 descreve a caracterização do estudo experimental proposto neste trabalho, apresentando suas definições, objetivos, desenho e arquitetura;
- O Capítulo 5 destaca a execução do estudo experimental, apresentando e analisando os resultados obtidos;
- O Capítulo 6 apresenta algumas considerações sobre a pesquisa, suas conclusões, limitações e trabalhos futuros.

2

FUNDAMENTAÇÃO CONCEITUAL

Neste capítulo, é apresentada uma revisão bibliográfica acerca dos assuntos relacionados ao estudo proposto neste trabalho. Na Seção 2.1 são apresentados brevemente os SGBD e seus problemas em lidar com o crescimento no volume de dados. As Seções 2.2 e 2.3 apresentam, respectivamente, a nova geração de SGBD: os sistemas NoSQL e NewSQL. A Seção 2.4 introduz os principais conceitos relacionados à Engenharia de Software Experimental. Em seguida, a Seção 2.5 apresenta o *framework* JMH (*Java Microbenchmark Harness*), utilizado neste trabalho para auxiliar nas medições dos experimentos. Por fim, na Seção 2.6 são apresentadas as conclusões deste capítulo.

2.1 Sistemas de Gerenciamento de Banco de Dados

Elmasri e Navathe (2010) definem um banco de dados como uma coleção de dados relacionados e que possuem um significado. Essa coleção deve representar algum aspecto do mundo real e ser desenhada, construída e populada com dados para um propósito específico. Um banco de dados pode ser de qualquer tamanho ou complexidade, como uma lista de nomes e endereços ou o catálogo de uma biblioteca com milhares de livros.

Ainda segundo Elmasri e Navathe (2010), um Sistema de Gerenciamento de Banco de Dados (SGBD) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. Dentre as funções de um SGBD destacam-se a definição, construção, manipulação e compartilhamento de um banco de dados entre usuários e aplicações, além de proteção no acesso a esses dados.

Silberschatz et al. (2010) afirmam que o principal objetivo de um SGBD é prover um meio de se armazenar e recuperar informações de um banco de dados de forma prática e eficiente. O gerenciamento dos dados envolve definir as estruturas para armazenar as informações e prover mecanismos para manipulá-las. Além disso, deve-se garantir a segurança dos dados armazenados, mesmo diante de falhas no sistema ou de tentativas de acesso não autorizado.

O primeiro SGBD de propósito geral foi projetado no começo dos anos 1960 por Charles Bachman, na *General Eletric*, chamado *Integrated Data Store* (IDS). Esse SGBD formou a base para os sistemas baseados no modelo de dados em rede. No final dessa mesma década, a IBM desenvolveu o *Information Management System* (IMS), baseado no modelo de dados hierárquico (RAMAKRISHNAN; GEHRKE, 2000).

Na década seguinte, o trabalho de Codd (1970) definiu o modelo relacional, cujo diferencial frente aos modelos anteriores estava na sua simplicidade e na possibilidade de esconder do programador detalhes de baixo nível de programação. Os primeiros SGBD baseados no modelo relacional não obtiveram sucesso, pois apresentavam pior desempenho quando comparados com os sistemas hierárquicos e em rede. Isso mudou quando a IBM desenvolveu técnicas para a construção de um SGBD relacional eficiente, através do seu SGBD System R. Os sistemas comerciais que foram surgindo, tais como DB2[4], Oracle[5], Ingres[6] e DEC Rdb[7], também contribuíram no desenvolvimento de técnicas de processamento eficiente, até que os sistemas relacionais passaram a competir, em termos de desempenho, com os sistemas em rede e hierárquicos (SILBERSCHATZ et al., 2010).

Nos anos 1980, o modelo relacional se consolidou como o principal paradigma para os SGBD. A linguagem de consulta SQL, desenvolvida pela IBM como parte do System R, se tornou a linguagem de consulta padrão para os bancos de dados relacionais desde então (RAMAKRISHNAN; GEHRKE, 2000).

---

[4] https://www.ibm.com/analytics/us/en/db2/

[5] http://www.oracle.com

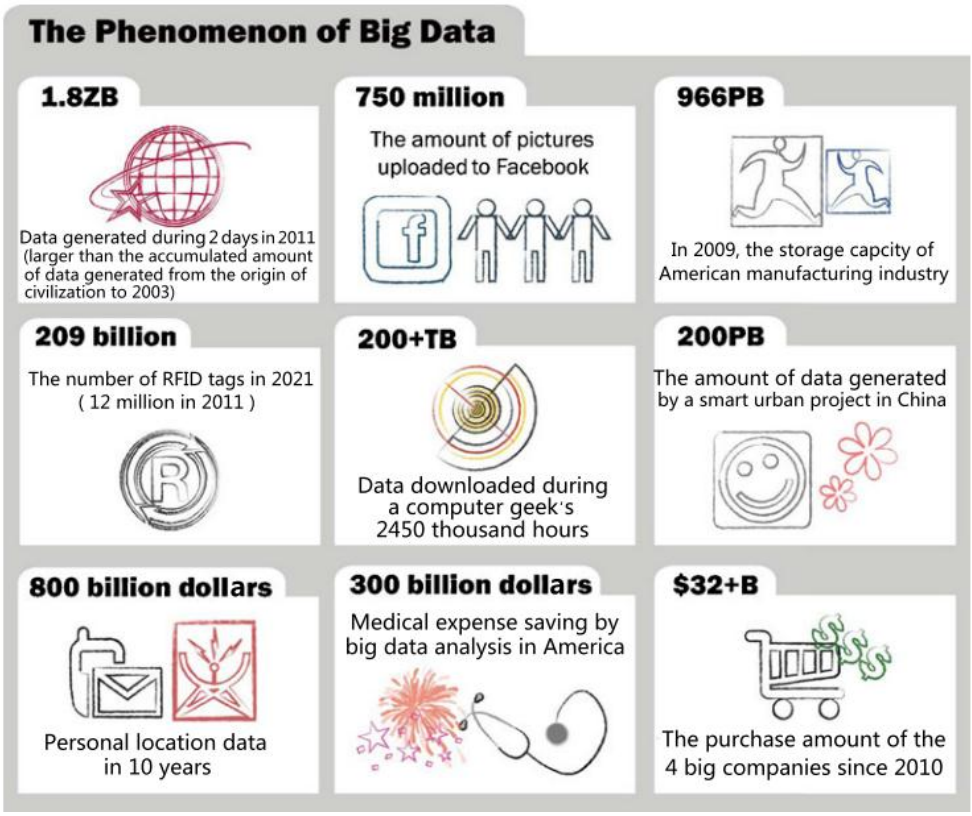
[6] https://www.actian.com/data-management/actian-x-hybrid-rdbms/

[7] https://www.oracle.com/technetwork/database/database-technologies/rdb/overview/index.html

2.1.1 Crescimento no Volume de Dados

De acordo com a *International Data Corporation* (IDC), em 2011 o volume de dados criados e copiados em todo o mundo foi de 1,8 ZB (1 zettabyte equivale a 10^{21} bytes), valor que tem dobrado a cada dois anos. Dentro desse contexto surgiu o termo *Big Data*, normalmente utilizado para descrever enormes massas de dados não estruturados que necessitam de análise em tempo real. Por exemplo, o Google[8] processa centenas de Petabytes (PB) e o Facebook[9] gera mais de 10 PB por mês (CHEN; MAO; LIU, 2014). A Figura 2 ilustra o crescimento do volume de dados global.

Figura 2 − O contínuo aumento do *Big Data*



Fonte: Chen, Mao e Liu (2014)

Esse crescimento no volume de dados tem superado o aumento nas capacidades de armazenamento, fato que tem levado os sistemas de gerenciamento de informações a armazenarem os dados de forma distribuída, porém sendo acessados como se estivessem em uma única máquina (VENKATESH; NIRMALA, 2012).

[8] https://www.google.com/
[9] https://www.facebook.com/

Para Pavlo e Aslett (2016), além do crescimento no volume de dados, outro fator crítico surgiu nos anos 2000: as aplicações na Internet, necessitando estar disponíveis todo o tempo e suportar um enorme número de usuários concorrentes. Nesse cenário, os bancos de dados passaram a ser um gargalo no desempenho dessas aplicações, visto que a demanda por recursos era muito maior do que os SGBD e o hardware da época podiam suportar.

Pavlo e Aslett (2016) argumentam que, inicialmente, o primeiro caminho para tentar superar essas limitações foi mover os bancos de dados para uma máquina com melhor hardware, a chamada escalabilidade vertical. Contudo, o ganho de desempenho decorrente dessa estratégia é limitado e, com o passar do tempo, pouco eficiente (além do seu alto custo financeiro).

Uma outra solução encontrada por algumas empresas foi a criação de um *middleware* para dividir o banco de dados em um *cluster* de máquinas menos caras. Do ponto de vista das aplicações, elas estão interagindo com uma única base de dados lógica, mas que na verdade está armazenada em múltiplos nós físicos. Quando uma aplicação envia uma operação para o banco de dados, o *middleware* redireciona (e eventualmente reescreve) essa operação para os nós do *cluster*, os quais a executam, devolvem seu resultado para o *middleware* e este consolida todos os resultados, devolvendo uma única resposta para a aplicação. Essa estratégia funciona bem para operações simples, como ler ou atualizar um registro único, porém é mais difícil executar uma consulta em múltiplas tabelas ou uma transação que atualize mais de um registro (PAVLO; ASLETT, 2016).

Diante dessas limitações apresentadas pelos SGBD relacionais ao lidar com os crescimentos no volume e no acesso aos dados, no começo dos anos 2000 tem início o surgimento de uma nova geração de sistemas de bancos de dados, voltados para lidar com esses problemas a um baixo custo. Os primeiros sistemas dessa nova geração são os chamados NoSQL (*Not Only SQL*). Posteriormente surgem também os sistemas NewSQL. Essas duas famílias de sistemas da nova geração serão detalhadas nas próximas seções.

2.2 Sistemas NoSQL

São sistemas que surgiram trazendo soluções que rompiam com o consagrado modelo relacional (particularmente no que diz respeito a garantir a consistência dos

dados). De uma forma geral, os sistemas de bancos de dados NoSQL caracterizam-se por (STRAUCH; SITES; KRIHA, 2011):

- Possuírem modelos de dados flexíveis;
- Proverem escalabilidade horizontal;
- Rodarem em um *cluster* de computadores de baixo custo;
- Não darem suporte à linguagem de consulta SQL; e
- Não garantirem conformidade com todas as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), apresentando, em sua maioria, apenas uma consistência eventual.

A principal ferramenta utilizada pelos SGBD relacionais para garantir a consistência dos dados é o gerenciamento de transações. Contudo, o impacto das transações no desempenho pode ser muito alto, especialmente em ambientes lidando com grandes volumes de dados (HARRISON, 2015).

No ano 2000, durante o Simpósio sobre Princípios da Computação Distribuída realizado pela ACM (*Association for Computing Machinery*[10]), Eric Brewer apresentou o Teorema CAP (BREWER, 2000), o qual posteriormente recebeu uma prova formal por Seth Gilbert e Nancy Lynch (GILBERT; LYNCH, 2002). Esse teorema postula que, dadas as três propriedades de Consistência (C), Disponibilidade (A) e Tolerância a Partições (P), somente é possível obter no máximo duas delas ao mesmo tempo em um sistema distribuído. Strauch, Sites e Kriha (2011) resumem essas três propriedades como:

- **Consistência:** tem por objetivo garantir que o sistema esteja em um estado consistente após a execução de uma operação. Em um sistema distribuído, isso significa que, uma vez que algo seja gravado, ele estará disponível para todos que tenham acesso ao sistema;

- **Disponibilidade:** o sistema deve estar sempre disponível para a execução das operações de escrita e leitura. Em um ambiente distribuído, isso significa que o sistema é tolerante a falhas ou atualizações nos nós; e

[10] https://www.acm.org/

- **Tolerância a Partições:** se refere à habilidade do sistema em se manter operacional mesmo na presença de partições na rede, ou seja, de “ilhas” de nós que não conseguem se comunicar entre si.

A maioria dos autores classifica os sistemas NoSQL de acordo com seu modelo de representação de dados, separando-os nas seguintes categorias: chave-valor, documentos, famílias de colunas e grafos.

- **Chave-valor:** Os sistemas NoSQL dessa categoria armazenam os dados em duas partes: uma *string*, que representa a chave, e o dado em si, que é referenciado como o valor, criando o par chave-valor. Os valores são armazenados sem esquema e independentes entre si; portanto, os relacionamentos entre os dados são de responsabilidade da aplicação. O acesso aos dados é sempre feito pela chave, o que provê aos sistemas dessa categoria um ótimo desempenho (NAYAK; PORIYA; POOJARY, 2013). Como exemplos de sistemas NoSQL chave-valor podemos citar Redis[11], DynamoDB[12], Memcached[13], Project Voldemort[14] e Riak[15];

- **Documentos:** Os documentos armazenados nos sistemas NoSQL dessa categoria geralmente estão nos formatos JSON (*JavaScript Object Notation*), XML ou PDF. Os dados em um documento são, de certa forma, similares aos registros em um banco de dados relacional, porém possuem muito mais flexibilidade por não possuírem esquema definido. Bancos de dados de documentos são como bancos de dados chave-valor, onde o documento é armazenado no valor e tanto a chave quanto o valor podem ser pesquisados (NAYAK; PORIYA; POOJARY, 2013). Como exemplos de sistemas dessa categoria podemos citar MongoDB[16] e CouchDB[17];

- **Famílias de colunas:** Os sistemas NoSQL dessa categoria permitem que os dados sejam armazenados com chaves mapeadas para valores e os

[11] https://redis.io/
[12] https://aws.amazon.com/dynamodb/
[13] https://memcached.org/
[14] http://www.project-voldemort.com/
[15] http://basho.com/products/riak-kv/
[16] https://www.mongodb.com/
[17] http://couchdb.apache.org/

valores agrupados em múltiplas famílias de colunas. Famílias de colunas são grupos de dados relacionados que, frequentemente, são acessados juntos (SADALAGE; FOWLER, 2013). Como exemplos de sistemas dessa categoria podemos citar Cassandra[18], Bigtable[19] e HBase[20]; e

- **Grafos:** Bancos de dados de grafos armazenam os dados em forma de um grafo, que consiste de nós e arestas. Podemos pensar em um nó como sendo uma instância de um objeto e nas arestas como os relacionamentos entre os objetos. As arestas possuem uma direção e tanto os nós como as arestas podem conter propriedades. Nesse tipo de banco de dados, a ênfase maior está no relacionamento entre os dados (SADALAGE; FOWLER, 2013). Como exemplos de sistemas NoSQL de grafos podemos citar Neo4J[21], FlockDB[22] e OrientDB[23].

2.3 Sistemas NewSQL

O fato dos SGBD NoSQL não darem suporte à linguagem SQL (utilizada pela grande maioria dos sistemas desenvolvidos no mercado) e não proverem todas as propriedades ACID (especialmente em termos de consistência) é criticado por empresas que necessitam dos benefícios de escalabilidade, alta disponibilidade, alto desempenho e baixo custo, porém não podem abrir mão dessas duas características. Além disso, algumas empresas, dentre elas o Google, avaliam que seus desenvolvedores gastam bastante tempo escrevendo códigos nas aplicações para lidar com problemas de inconsistência de dados nos sistemas NoSQL (PAVLO; ASLETT, 2016).

Para atender a esse cenário, surgiu uma nova geração de bancos de dados alternativos, chamados de NewSQL, prometendo unir os benefícios dos SGBD relacionais e dos sistemas NoSQL.

Pavlo e Aslett (2016) definem os sistemas NewSQL como SGBD relacionais modernos que buscam prover o desempenho escalável dos sistemas NoSQL e, ao

[18] http://cassandra.apache.org/
[19] https://cloud.google.com/bigtable/
[20] https://hbase.apache.org/
[21] https://neo4j.com/
[22] https://github.com/twitter-archive/flockdb
[23] https://orientdb.com/

mesmo tempo, manter o modelo relacional, a linguagem SQL e o suporte às propriedades ACID para as transações, característicos dos SGBD relacionais tradicionais.

Já Venkatesh e Nirmala (2012) definem um SGBD NewSQL, dentro do contexto de processamento de transações, como tendo as seguintes características:

- SQL como principal mecanismo de interação para as aplicações;
- Suporte às propriedades ACID para transações;
- Mecanismo de controle de concorrência sem bloqueio;
- Arquitetura distribuída e que provê um maior desempenho por nó do que os SGBD tradicionais; e
- Arquitetura escalável, *shared-nothing* (ou seja, cujos nós não compartilham CPU, memória ou disco) e capaz de rodar em um grande número de nós sem que haja gargalos.

Pavlo e Aslett (2016) agrupam os SGBD NewSQL nas categorias: novas arquiteturas, *transparent sharding middleware* e banco de dados como um serviço.

- **Novas arquiteturas:** Essa categoria representa os SGBD construídos do zero, ou seja, que não são extensões de sistemas já existentes. A vantagem de um novo SGBD é que ele foi construído pensando no modelo distribuído e, com isso, toda sua arquitetura pode ser otimizada para esse ambiente, como o otimizador de consultas, os protocolos de comunicação entre nós e seu próprio armazenamento primário. Algumas desvantagens dos sistemas dessa categoria é que muitas empresas são cautelosas quanto a adotar tecnologias muito novas e ainda não validadas pelo uso em larga escala, além do número de pessoas com experiência nesses sistemas ser muito inferior quando comparado aos SGBD mais populares. Como exemplos de sistemas dessa categoria podemos citar MemSQL[24], NuoDB[25] e VoltDB[26];

- ***Transparent sharding middleware*:** Essa categoria representa os sistemas que dividem (*sharding*) uma base de dados em múltiplos *shards*

[24] https://www.memsql.com/
[25] https://www.nuodb.com/
[26] https://www.voltdb.com/

(pedaços), os quais são armazenados em *clusters shared-nothing* de nós de um SGBD tradicional, onde cada nó roda o mesmo SGBD e possui apenas parte da base de dados original. Esses sistemas provêem uma solução semelhante aos *middlewares* descritos na Seção 2.1. Nessa arquitetura, existe um componente centralizador (o *middleware*, responsável por coordenar as consultas, transações, replicações e partições) e, em cada nó, uma camada instalada para se comunicar com o *middleware* e executar as consultas na sua base de dados local. A principal vantagem dos sistemas nessa categoria é que a troca do SGBD tradicional pelo *middleware* particionado implica em pouca ou nenhuma alteração nas aplicações. Contudo, o fato de ainda usarem os SGBD tradicionais em cada nó do *cluster* dificulta que esses sistemas consigam escalar de forma a tirar vantagem de todo o potencial do *cluster* (em termos de processamento e memória). Como exemplos de sistemas dessa categoria podemos citar AgilData Scalable Cluster for MySQL[27] e ScaleArc[28]; e

- **Banco de dados como um serviço:** Essa categoria representa os serviços oferecidos na nuvem, nos quais os provedores ficam responsáveis, por exemplo, pela configuração física, *tuning*, replicação e *backups*. Ou seja, as empresas contratantes não precisam ter e manter o SGBD em um hardware próprio e apenas pagam pelo serviço de acordo com a quantidade de recursos utilizados (armazenamento, processamento, memória). A grande maioria desses serviços na nuvem provêem instâncias de SGBD tradicionais e, por isso, não são categorizados como NewSQL. Apenas são considerados parte dessa categoria os serviços que são baseados numa nova arquitetura NewSQL. Como exemplos de sistemas dessa categoria podemos citar Amazon Aurora[29] e ClearDB[30].

[27] http://www.agildata.com/scalable-cluster-for-mysql/
[28] http://www.scalearc.com/
[29] https://aws.amazon.com/pt/rds/aurora/
[30] https://w2.cleardb.net/

2.4 Engenharia de Software Experimental

De acordo com Juristo e Moreno (2013), a Engenharia de Software Experimental objetiva encontrar fatos que correspondam às suposições, especulações e crenças tão presentes na construção de software. Ideias como: “acreditamos que a aplicação de determinadas técnicas dará resultado”, “achamos que um certo número de pessoas será suficiente para concluir o projeto”, “esperamos que o tempo de desenvolvimento seja menor usando uma certa ferramenta” ou “assumimos que a qualidade do produto final será melhor se seguirmos um determinado processo de desenvolvimento”.

Ainda segundo Juristo e Moreno (2013), geralmente essas ideias acabam sendo validadas pelo tempo. Se muitas pessoas adotam aquela ideia, ela deve ser boa. Se poucas pessoas a adotam, ela deve ser falsa e será esquecida com o tempo. Entretanto, qualquer disciplina no ramo da Engenharia precisa de provas de que uma determinada abordagem ou técnica é realmente melhor do que outra. Ou seja, precisa trabalhar mais com fatos do que com suposições.

A definição de Engenharia de Software dada pelo Padrão IEEE 610.12[31] é que, assim como outras disciplinas de Engenharia, ela aplica conhecimentos científicos para o desenvolvimento, operação e manutenção de sistemas de software.

Wohlin et al. (2012) afirmam que a introdução à área de Experimentação se dá através da introdução a um processo de experimentação, com foco em fornecer diretrizes para realizar experimentos na Engenharia de Software. Diante disso, eles definem as seguintes atividades para um processo de experimentação:

- Escopo;
- Planejamento;
- Operação;
- Análise e interpretação; e
- Apresentação e empacotamento.

Por sua vez, Juristo e Moreno (2013) definem que um experimento com certo grau de formalidade deve ter suas atividades divididas nas seguintes fases:

- Definição dos objetivos;
- Design do experimento;
- Execução do experimento; e
- Análise dos dados/resultados coletados.

[31] https://standards.ieee.org/standard/610_12-1990.html

Analisando as metodologias de Wohlin et al. (2012) e de Juristo e Moreno (2013), vemos que seus conceitos, definições e atividades são equivalentes. Portanto, para formalizar o estudo experimental proposto neste trabalho, decidimos seguir o modelo apresentado por Juristo e Moreno (2013), cujas etapas serão detalhadas a seguir.

2.4.1 Definição dos objetivos

Nessa etapa, além de serem definidos os objetivos do experimento, também devem ser formuladas as hipóteses que precisarão ser validadas de acordo com as variáveis a serem medidas e analisadas nas etapas posteriores.

Objetivos:

Os objetivos do experimento são a base na qual o estudo empírico será estabelecido e são formulados a partir do problema que se pretende resolver.

Formulação das Hipóteses:

As hipóteses são formuladas com o intuito de serem rejeitadas (ou refutadas). Por exemplo, se desejamos decidir se uma moeda é falsa, formulamos a hipótese de que ela não é falsa. Essa hipótese é chamada de hipótese nula (simbolizada como H_0). Qualquer hipótese que seja diferente desta é chamada de hipótese alternativa (simbolizada como H_1).

2.4.2 Design do experimento

Nessa etapa são determinadas em que condições o experimento será conduzido, definindo quais variáveis serão examinadas (junto com seus possíveis valores) e quais dados serão coletados. Além disso, deve ser escolhido um desenho de experimento adequado às condições estabelecidas.

Objetos:

Os objetos nos quais um experimento é executado são chamados de unidades experimentais ou objetos experimentais.

Sujeitos:

A pessoa que aplica os métodos ou técnicas nos objetos experimentais é chamada de sujeito experimental.

Variável de Resposta:

A variável de resposta (também chamada de variável dependente) é definida como sendo o resultado de um experimento, ou seja, a característica que é medida para testar os efeitos das variações entre os experimentos unitários. Cada valor obtido da variável de resposta em um experimento unitário é chamado de observação. A análise de todas as observações irá definir se as hipóteses formuladas podem ser validadas.

Parâmetros:

Qualquer característica (qualitativa ou quantitativa) que não deve variar entre os experimentos unitários é chamada de parâmetro. São, portanto, características que não influenciam (ou que não queremos que influenciem) no resultado do experimento ou na variável de resposta.

Fatores e Alternativas:

Toda característica que afeta a variável de resposta é chamada de fator (ou variável independente). Cada fator possui diferentes alternativas (ou tratamentos). Um experimento visa examinar a influência dessas alternativas no valor da variável de resposta.

Variáveis de Bloqueio:

Nem sempre é possível estabelecer um valor fixo para características que não desejamos avaliar em um experimento, ou seja, pode haver variações indesejáveis de um experimento para outro. Essas variações são conhecidas como variáveis de bloqueio e requerem um desenho de experimento chamado Desenho de Bloqueio.

Experimento Unitário:

Cada combinação das possíveis alternativas, executada em um objeto experimental pelo sujeito experimental, é chamada de experimento unitário.

Replicação Interna:

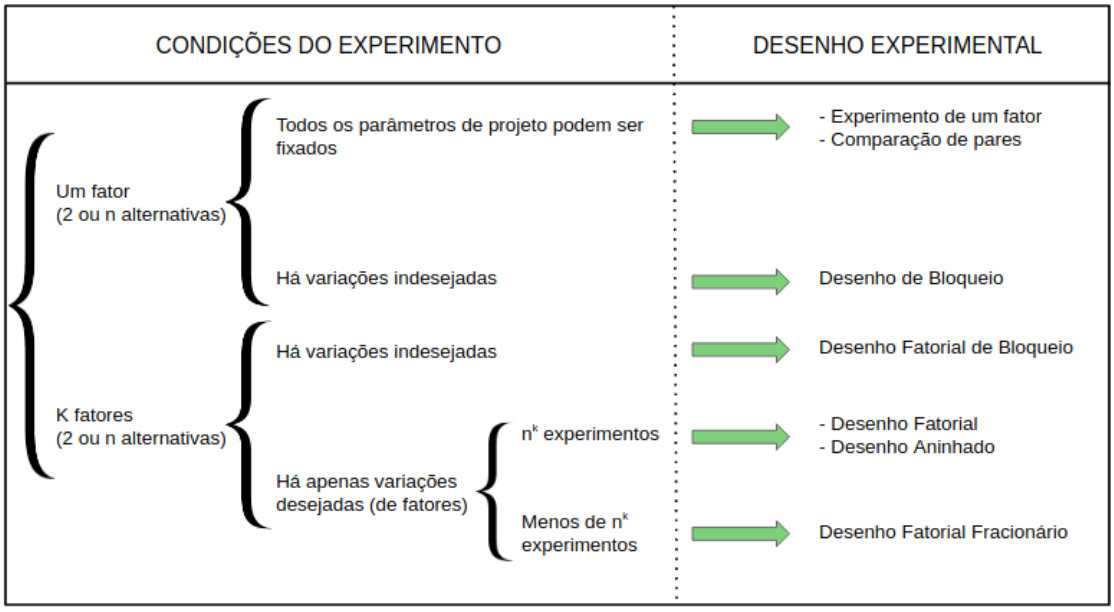
A repetição de alguns ou de todos os experimentos unitários é chamada de replicação interna. Se, por exemplo, os experimentos unitários são repetidos três vezes, esse estudo é dito ser um experimento com três replicações.

A quantidade de repetições do experimento unitário corresponde ao tamanho da amostra estatística. Ter uma amostra de tamanho grande (maior do que 30) significa que podemos utilizar a média e o desvio padrão amostrais como estimativas para a média e o desvio padrão populacionais. Dessa forma, é possível fazer afirmações sobre a população utilizando os dados da amostra.

Desenho do Experimento:

Existem diferentes desenhos experimentais, que variam dependendo do objetivo do experimento, do número de fatores, das alternativas dos fatores, da existência de variações indesejadas, dentre outros. A Figura 3 mostra os desenhos mais comuns.

Figura 3 − Diferentes Desenhos Experimentais



Adaptado de: Juristo e Moreno (2013)

2.4.3 Execução do experimento

Nessa etapa, os experimentos unitários são executados conforme o desenho experimental escolhido.

2.4.4 Análise dos dados/resultados coletados

Após a execução dos experimentos, será feita a análise dos dados coletados durante o experimento. O experimentador examina os dados, fazendo inferências estatísticas e procurando por padrões comportamentais entre as variáveis sendo estudadas, além de validar as hipóteses formuladas.

2.5 JMH (*Java Microbenchmark Harness*)[32]

Java Microbenchmark Harness é um *framework* desenvolvido para auxiliar aplicações Java a realizar medições de *benchmark*, aferindo o tempo de execução de determinados trechos de código (JENKOV, 2015).

Os métodos do código-fonte que serão medidos devem receber a anotação *@Benchmark*. Esses métodos também podem receber a anotação *@Measurement*, que serve para serem especificados parâmetros relativos à execução do método, como a quantidade de iterações que serão rodadas (pelo parâmetro *iterations*). Outra anotação possível é a *@Warmup*, que fará com que o método seja executado uma certa quantidade de vezes (definida pelo parâmetro *iterations*), mas esses resultados sejam desconsiderados (ou seja, não computados), servindo como uma espécie de aquecimento para as execuções que de fato serão computadas e posteriormente analisadas.

JMH possui cinco diferentes modos de execução, os quais devem ser configurados por meio da anotação *@BenchmarkMode*:

- *Throughput*: mede o número de vezes que o método é executado por segundo. É o modo padrão do *framework*;
- *Average Time*: mede o tempo médio de execução do método;
- *Sample Time*: mede o tempo para o método ser executado, incluindo estatísticas como tempo máximo, mínimo e percentis;
- *Single Shot Time*: mede o tempo de uma única execução; e
- *All*: mede todas as informações dos outros modos.

JMH permite também que seja escolhida a unidade de tempo para as medições:

- Nanossegundos;

[32] http://openjdk.java.net/projects/code-tools/jmh/

- Microssegundos;
- Milissegundos;
- Segundos;
- Minutos;
- Horas; e
- Dias.

Variáveis que são necessárias para as medições, mas não para o código-fonte da aplicação em si, são chamadas de variáveis de estado e devem ser declaradas dentro de uma classe de estado. Uma classe de estado deve receber a anotação *@State*. Como um objeto dessa classe pode ser utilizado em várias execuções do método que está sendo medido, JMH permite que seja declarado o escopo da classe de estado. As opções de escopo são:

- *Thread*: cada *thread* vai instanciar seu objeto da classe de estado;
- *Group*: cada grupo de *thread* vai instanciar seu objeto da classe de estado; e
- *Benchmark*: todas as *threads* do *benchmark* irão compartilhar o mesmo objeto.

Os métodos da classe de estado podem receber as anotações *@Setup* e *@TearDown*. A primeira serve para indicar que esse método deve ser executado antes do objeto da classe ser passado para o método do *benchmark*. A segunda indica que o método deve ser executado após a execução do método do *benchmark*. O tempo de execução desses métodos não é computado nas medições do *benchmark*. Essas duas anotações devem receber um parâmetro que indica o momento em que o método deve ser chamado. Esse parâmetro possui as seguintes opções:

- *Level.Trial*: indica que o método deve ser chamado para cada execução de todo o *benchmark*;
- *Level.Iteration*: indica que o método deve ser chamado para cada iteração do *benchmark*; e
- *Level.Invocation*: indica que o método deve ser chamado para cada chamada ao método do *benchmark*.

2.6 Considerações

Neste capítulo, apresentamos os principais assuntos relacionados ao estudo comparativo proposto nesta dissertação. Inicialmente, abordamos os problemas apresentado pelos SGBD tradicionais ao lidar com o recente crescimento no volume de dados das aplicações. Em seguida, focamos nos SGBD da nova geração, que apresentam novas arquiteturas, modelos de dados e estratégias para enfrentar essas dificuldades.

Também neste capítulo, introduzimos os conceitos relacionados à Engenharia de Software Experimental, os quais nos ajudarão na formalização do estudo comparativo foco desta dissertação. Detalhamos também o *framework* JMH (*Java Microbenchmark Harness*), responsável pelas medições de cada experimento executado neste estudo.

No próximo capítulo, os trabalhos relacionados ao estudo comparativo proposto neste trabalho serão analisados.

3

ESTUDOS COMPARATIVOS DE SISTEMAS NOSQL E NEWSQL

Neste capítulo são apresentados alguns trabalhos acadêmicos que realizaram comparações entre sistemas de gerenciamento de dados da nova geração. Na Seção 3.1 esses trabalhos são detalhados, enquanto na Seção 3.2 analisaremos seus pontos em comum, suas diferenças e como o presente trabalho pretende se diferenciar destes apresentados. Na Seção 3.3 são apresentadas as conclusões do capítulo.

3.1 Principais Trabalhos Relacionados

Nas próximas seções serão detalhados os trabalhos mais relevantes que propuseram realizar comparações entre SGBD da nova geração.

3.1.1 Qual Sistema de Banco de Dados NoSQL? Uma Visão Geral de Desempenho

Neste trabalho, Abramova, Bernardino e Furtado (2014) avaliam o tempo de execução em operações de leitura e atualização de dados para cinco SGBD NoSQL:

Cassandra[34], HBase[35], MongoDB[36], OrientDB[37] e Redis[38]. Os dois primeiros são sistemas NoSQL da categoria família de colunas, os dois seguintes são da categoria de documentos e o último é da categoria chave-valor.

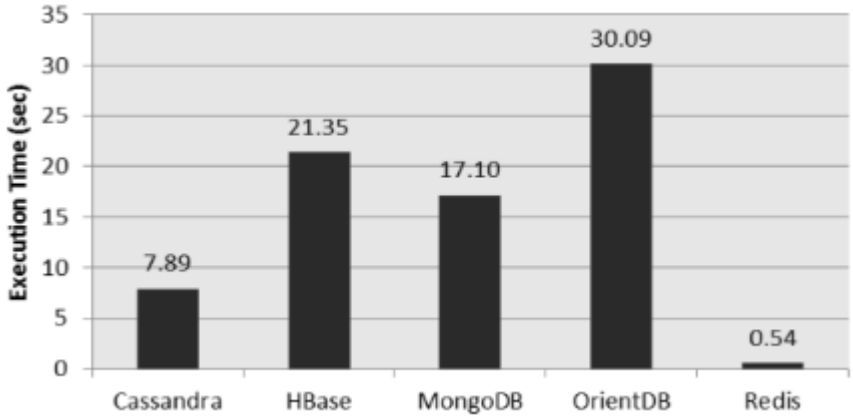
Os autores utilizaram o *Yahoo! Cloud Serving Benchmark*[39] para executar as operações e realizar as medições. Foram gerados 600 mil registros aleatórios para serem utilizados nos testes e as operações foram executadas seguindo a seguinte divisão de cargas:

- Carga A: 50% de leitura e 50% de atualização;
- Carga C: 100% de leitura e 0% de atualização;
- Carga H: 0% de leitura e 100% de atualização.

Os testes foram executados em uma máquina virtual, com 2GB de RAM e sistema operacional Ubuntu Server 32 bits, instalada em um computador com 4GB de RAM e sistema operacional Windows 7.

A Figura 4 apresenta os resultados, em segundos, do tempo de execução da carga A sobre os 600 mil registros.

Figura 4 − Execução da Carga A

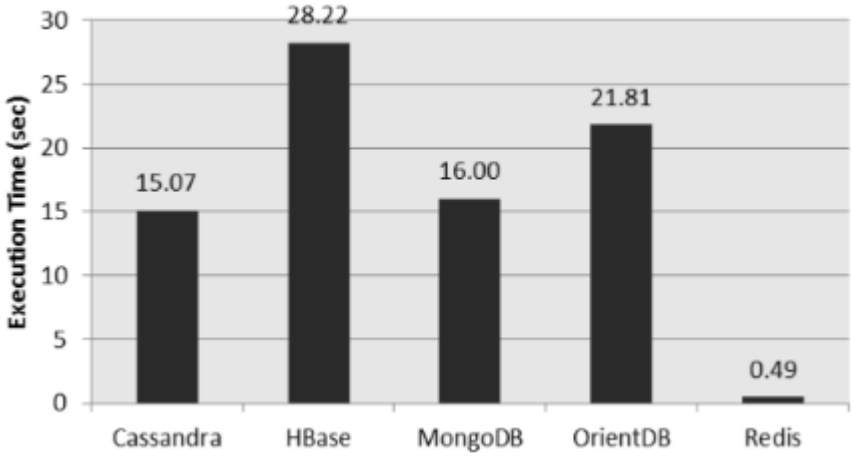


Fonte: Abramova, Bernardino e Furtado (2014)

A Figura 5 apresenta os resultados, em segundos, do tempo de execução da carga C, contendo 1000 operações de leitura sobre os 600 mil registros.

[34] http://cassandra.apache.org/
[35] https://hbase.apache.org/
[36] https://www.mongodb.com/
[37] https://orientdb.com/
[38] https://redis.io/
[39] https://research.yahoo.com/news/yahoo-cloud-serving-benchmark/

Figura 5 − Execução da Carga C

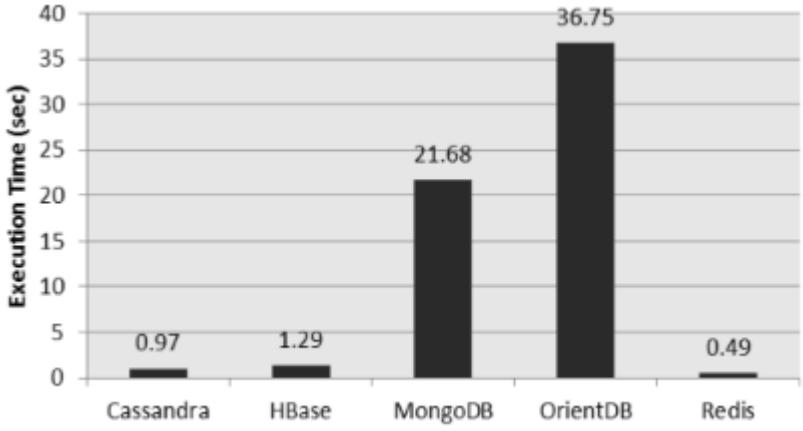


Fonte: Abramova, Bernardino e Furtado (2014)

Analisando os resultados dessas duas execuções, os autores verificaram que o sistema Redis apresentou o melhor desempenho, o que pode ser explicado pelo fato do sistema armazenar os dados em memória volátil. Dentre os sistemas de suas respectivas categorias, o Cassandra e o MongoDB apresentaram os melhores desempenhos.

A Figura 6 apresenta os resultados, em segundos, do tempo de execução da carga H, contendo 1000 operações de atualização sobre os 600 mil registros.

Figura 6 − Execução da Carga H



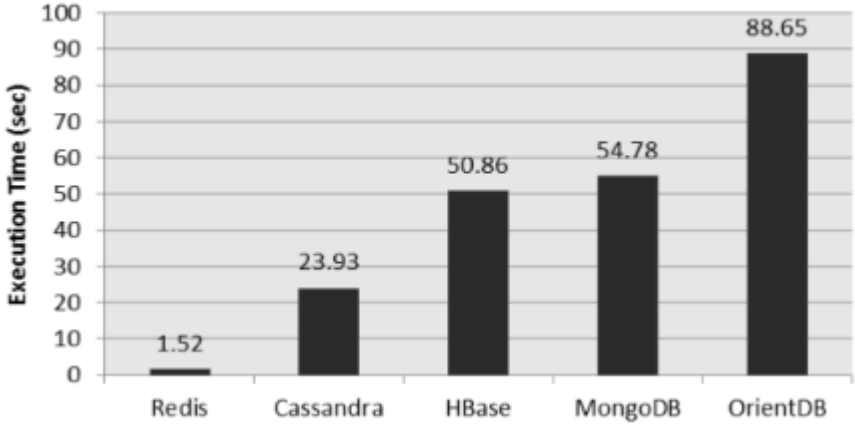
Fonte: Abramova, Bernardino e Furtado (2014)

Analisando os resultados dessa execução, os autores verificaram a grande melhora de desempenho para os dois sistemas da categoria família de colunas. Esse fato é explicado por esses sistemas serem otimizados para a execução de

atualizações. O MongoDB continua sendo o sistema mais rápido de sua categoria e o Redis mantém o desempenho das execuções anteriores.

Na Figura 7 os autores apresentam o tempo total de execução, em segundos, de cada sistema avaliado, somando os tempos das três cargas (A+C+H) e ordenando os sistemas do menor para o maior tempo de execução total.

Figura 7 − Tempo total de execução das cargas A+C+H



Fonte: Abramova, Bernardino e Furtado (2014)

A análise dos autores segue as análises das demais execuções: o sistema Redis, por armazenar os dados em memória volátil, apresenta o melhor desempenho; os sistemas Cassandra e HBase aparecem em seguida, por serem otimizados para a execução de operações de atualização; e, dentro da categoria de documentos, o MongoDB apresentou consistentemente melhor desempenho do que o OrientDB.

3.1.2 Um Estudo Comparativo Avançado dos Principais Sistemas de Bancos de Dados NoSQL e NewSQL com um Método para Análise de Decisões Multicritério

Hajoui et al. (2015) escolheram oito sistemas NoSQL e dois NewSQL para compará-los quanto a uma série de critérios, dando um peso e uma nota para cada um desses critérios e, no final, calculando a pontuação global de cada SGBD.

Os sistemas escolhidos foram: MongoDB, CouchDB[40], Cassandra, HBase, Redis, Riak[41], Neo4j[42], OrientDB, VoltDB[43] e NuoDB[44]. Os critérios avaliados para cada

[40] http://couchdb.apache.org/
[41] http://basho.com/products/riak-kv/
[42] https://neo4j.com/

um desses sistemas foram: desempenho, integridade, confiabilidade, interoperabilidade, suporte à nuvem, complexidade de consultas e segurança.

O peso de cada critério na pontuação global foi definido utilizando o método ROC (*Rank Order Centroid*), um dos vários métodos para análise de decisões multicritério. Após a aplicação do ROC, os autores chegaram à seguinte fórmula para o cálculo da pontuação global:

$$Pontuação\_Global = (0,37*desempenho) + (0,23*integridade) + (0,16*confiabilidade) + (0,11*interoperabilidade) + (0,07*suporte\_à\_nuvem) + (0,04*complexidade\_de\_consultas) + (0,02*segurança)$$

Definidos os pesos de cada critério, os autores aplicaram, para cada sistema, notas de 1 a 9 a cada um desses indicadores de qualidade e calcularam sua pontuação global, conforme pode ser visto na Tabela 1 e ilustrado no gráfico da Figura 8. Os autores afirmam que, apesar desse método de pontuação não ser objetivo, ele "reflete 90% da realidade".

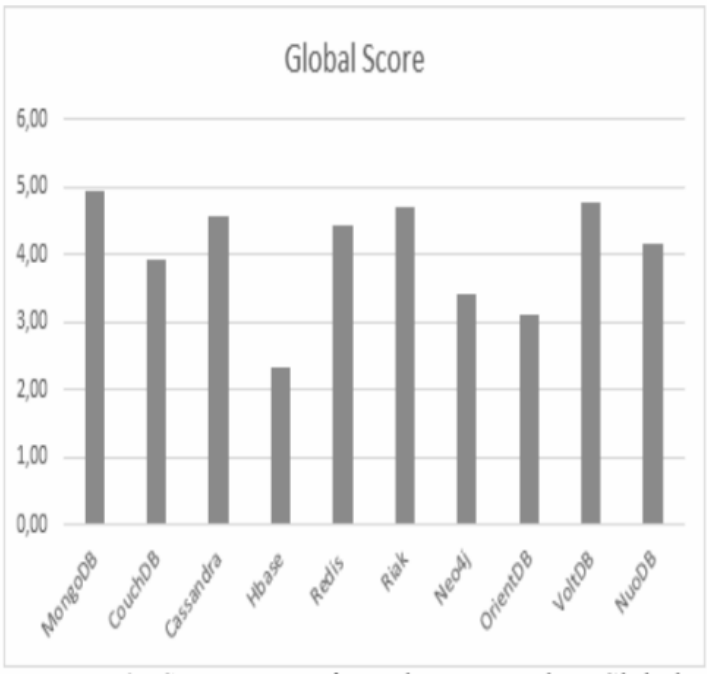
Tabela 1 − Pontuação global para os 10 SGBD

Indicadores de Qualidade	MongoDB	CouchDB	Cassandra	HBase	Redis	Riak	Neo4J	OrientDB	VoltDB	NuoDB
Desempenho	5	3	5	3	7	5	3	1	5	3
Integridade	5	5	5	3	3	5	5	5	5	5
Confiabilidade	5	5	5	1	1	5	3	5	5	5
Interoperabilidade	5	3	3	1	5	3	3	3	3	3
Suporte à Nuvem	5	5	3	1	3	5	3	3	5	7
Complexidade de Consultas	5	3	3	3	5	5	3	5	5	5
Segurança	3	3	5	3	1	1	1	3	5	5
Pontuação Global	**4,96**	**3,92**	**4,56**	**2,32**	**4,42**	**4,70**	**3,42**	**3,12**	**4,78**	**4,18**

Fonte: Adaptado de Hajoui et al. (2015)

[43] https://www.voltdb.com/
[44] https://www.nuodb.com/

Figura 8 − Comparação dos SGBD de acordo com sua pontuação global



Fonte: Hajoui et al. (2015)

A partir desses dados, a análise dos autores é que, apesar do MongoDB ter apresentado a maior pontuação, os sistemas Cassandra, Redis, Riak e VoltDB possuem pontuações próximas a do MongoDB. Portanto, a importância de cada critério deverá ser levada em consideração para a escolha do SGBD.

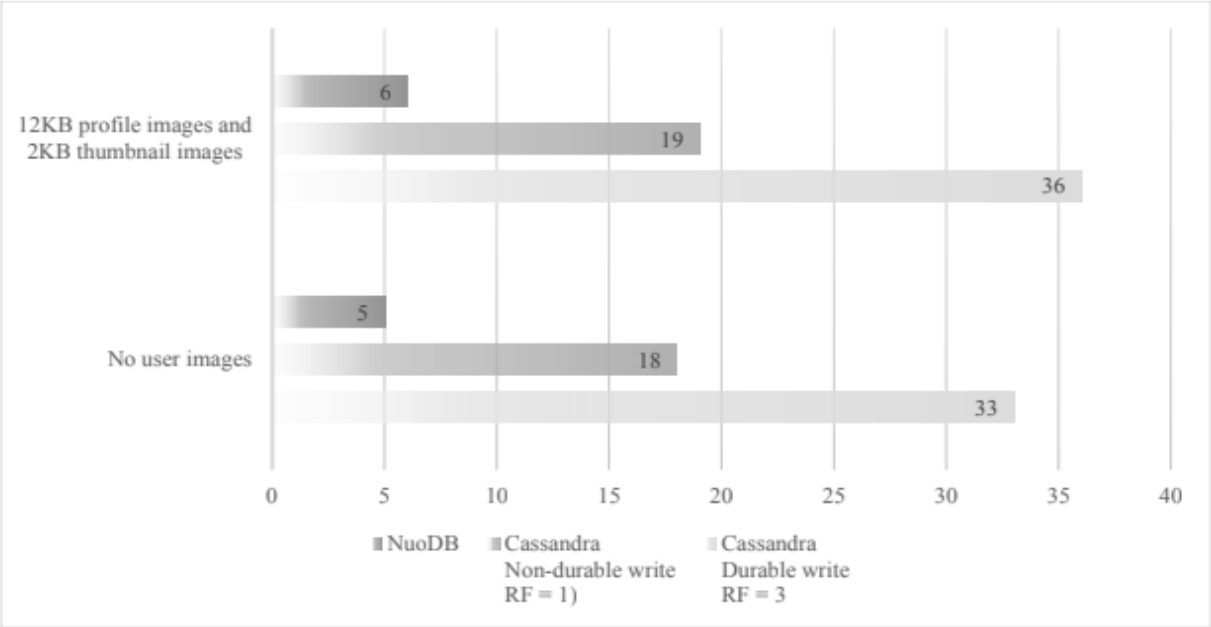
3.1.3 Pesquisa Comparativa de sistemas de banco de dados NoSQL e NewSQL

Neste trabalho, Gurevich (2015) realiza uma série de comparações qualitativas entre sistemas NoSQL e NewSQL, avaliando as seguintes características: possibilidades de consultas, controle de concorrência, replicação, escalabilidade, particionamento, consistência e segurança.

Em seguida, o autor realiza uma comparação de desempenho entre um representante dos SGBD NoSQL (o sistema Cassandra) e um dos NewSQL (o sistema NuoDB), utilizando um sistema de *benchmarking* chamado BG. Esse sistema avalia os SGBD aplicando ações interativas de redes sociais, como convidar um amigo ou ver o perfil de um membro da rede social.

Na Figura 9, o autor apresenta o desempenho dos SGBD quanto ao carregamento de dados (no caso, imagens de um perfil da rede social).

Figura 9 − Carregamento de dados



Fonte: Gurevich (2015)

Neste cenário, o NuoDB claramente apresenta melhor desempenho. Contudo, o autor relata a melhora de desempenho quando o Cassandra está configurado como “escrita não durável” (ou seja, fator de replicação menor que o número de nós) quando comparado com a configuração de “escrita durável” (ou seja, o fator de replicação é igual ao número de nós).

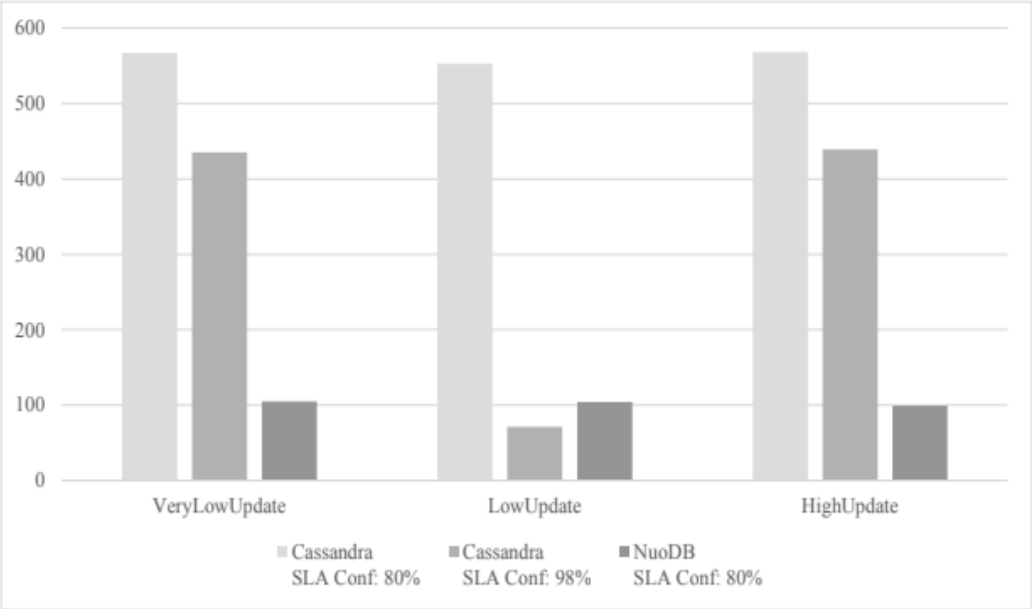
A Figura 10 apresenta o resultado da classificação SoAR (*Social Action Rating*), que computa o número de ações concorrentes realizadas por um sistema, emulando o comportamento de ações interativas em redes sociais.

Neste cenário, o Cassandra apresenta o melhor desempenho, o que o autor explica pelo fato da maior escalabilidade do sistema NoSQL prover melhores resultados em aplicações distribuídas com múltiplos clientes realizando ações concorrentes.

3.1.4 SQL vs. NoSQL vs. NewSQL - Um Estudo Comparativo

Binani, Gutti e Upadhyay (2016) propõem discutir e comparar as soluções de sistemas de bancos de dados relacional (chamado por eles apenas de SQL ou OldSQL), NoSQL e NewSQL, objetivando encontrar a melhor solução para atender a requisitos de *Big Data*.

Figura 10 − Classificação SoAR usando diferentes cargas de dados



Fonte: Gurevich (2015)

Os autores apresentam definições, arquiteturas, modelos de dados, propriedades e outras características desses bancos de dados. O Quadro 1 exibe as informações sumarizadas.

Quadro 1 − Comparação de OldSQL, NoSQL e NewSQL

Características Distintas	**OldSQL**	**NoSQL**	**NewSQL**
Relacional	Sim	Não	Sim
ACID	Sim	Não (provê CAP)	Sim
SQL	Sim	Não	Sim
OLTP	Não totalmente suportado	Suportado	Totalmente suportado
Escalabilidade	Não	Sim	Sim
Complexidade de consultas	Baixa	Alta	Muito alta
Distribuição	Não	Sim	Sim

Fonte: Adaptado de Binani, Gutti e Upadhyay (2016)

A partir desses dados, a conclusão dos autores é que o NewSQL é a melhor escolha para atender às aplicações *Big Data* OLTP (*Online Transaction Processing*), por ser um aprimoramento dos bancos relacionais, provendo escalabilidade horizontal

e mantendo as propriedades ACID. Os autores afirmam que, apesar de ainda estarem em estágio inicial, os bancos NewSQL encontraram o equilíbrio certo entre consistência, escalabilidade, velocidade e disponibilidade.

3.1.5 Uma Avaliação de Desempenho de Bancos de Dados em Memória

Kabakus e Kara (2017) avaliam o desempenho de alguns bancos de dados em memória (NoSQL e relacional) em termos de tempo para realizar uma operação e uso de memória durante a execução da operação. Os cinco SGBD escolhidos pelos autores foram: MongoDB, Redis, Memcached[45], Cassandra e H2[46].

Os resultados do primeiro experimento (operação de escrita) podem ser vistos nas Tabelas 2 e 3.

Tabela 2 − Tempo (em milissegundos) para operação de escrita

SGBD	**Número de registros**			
	1.000	***10.000***	***100.000***	***1.000.000***
Redis	34	214	1.666	14.648
MongoDB	904	3.482	26.030	253.898
Memcached	23	100	276	2.813
Cassandra	1.202	4.487	15.482	140.842
H2	147	475	1.648	7.394

Fonte: Adaptado de Kabakus e Kara (2017)

Tabela 3 − Consumo de memória (em MB) para operação de escrita

SGBD	**Número de registros**			
	1.000	***10.000***	***100.000***	***1.000.000***
Redis	2,5	3,8	4,3	62,7
MongoDB	56,9	263,6	365	155,9
Memcached	5,3	27,2	211	264,9
Cassandra	5,3	7,5	208,1	102,6
H2	2,3	33,2	103,4	540

Fonte: Adaptado de Kabakus e Kara (2017)

[45] https://memcached.org/
[46] http://www.h2database.com/

Analisando os resultados, os autores verificaram a grande diferença de tempo de execução entre os SGBD Memcached e MongoDB, fato que fica mais evidente com o aumento no número de registros. Eles também destacam o Redis como o mais eficiente quanto ao consumo de memória.

As Tabelas 4 e 5 apresentam os resultados do segundo experimento (operação de leitura de um registro).

Tabela 4 − Tempo (em milissegundos) para operação de leitura de um registro

SGBD	**Número de registros**			
	1.000	***10.000***	***100.000***	***1.000.000***
Redis	8	6	8	8
MongoDB	8	10	11	13
Memcached	9	14	14	30
Cassandra	2	2	3	6
H2	25	26	60	171

Fonte: Adaptado de Kabakus e Kara (2017)

Tabela 5 − Consumo de memória (em MB) para operação de leitura de um registro

SGBD	**Número de registros**			
	1.000	***10.000***	***100.000***	***1.000.000***
Redis	1,3	1,3	1,3	1,3
MongoDB	1,3	2,5	2,5	1,3
Memcached	1,3	2,5	1,3	2,5
Cassandra	0	0	0	0
H2	1,2	1,2	9,8	60

Fonte: Adaptado de Kabakus e Kara (2017)

Nesse segundo experimento, os autores destacam o desempenho muito ruim do SGBD H2, tanto em relação ao consumo de memória quanto ao tempo de execução. Além disso, o Cassandra aparece como o sistema com melhor eficiência no consumo de memória.

Os resultados do terceiro experimento (operação de exclusão) podem ser vistos nas Tabelas 6 e 7.

Tabela 6 − Tempo (em milissegundos) para operação de exclusão

SGBD	Número de registros			
	1.000	***10.000***	***100.000***	***1.000.000***
Redis	0	1	0	0
MongoDB	75	88	92	355
Memcached	17	17	16	13
Cassandra	2	2	1	2
H2	10	13	68	174

Fonte: Adaptado de Kabakus e Kara (2017)

Tabela 7 − Consumo de memória (em MB) para operação de exclusão

SGBD	Número de registros			
	1.000	***10.000***	***100.000***	***1.000.000***
Redis	0	0	0	0
MongoDB	10	10	10	11,3
Memcached	2,2	2,1	2,2	2,2
Cassandra	0	0	0	0
H2	1,2	1,8	4,9	62,9

Fonte: Adaptado de Kabakus e Kara (2017)

Analisando os resultados, os autores apontam o Redis com o melhor desempenho, completando as operações em menos de 1 milissegundo, e o MongoDB com o pior desempenho. Quanto ao consumo de memória, Cassandra e Redis são os mais eficientes, enquanto o H2 apresenta o pior resultado.

As Tabelas 8 e 9 apresentam os resultados do último experimento (operação de leitura de toda a base de dados).

Nesse último experimento, os autores destacam o MongoDB, com o menor tempo de execução e o baixo consumo de memória. O sistema Memcached foi excluído deste experimento por não possuir a funcionalidade de leitura de toda a base de dados.

Tabela 8 − Tempo (em milissegundos) para operação de leitura de toda a base de dados

SGBD	Número de registros			
	1.000	***10.000***	***100.000***	***1.000.000***
Redis	10	9	11	11
MongoDB	9	15	9	8
Cassandra	9	32	24	54
H2	9	14	18	20

Fonte: Adaptado de Kabakus e Kara (2017)

Tabela 9 − Consumo de memória (em MB) para operação de leitura de toda a base de dados

SGBD	Número de registros			
	1.000	***10.000***	***100.000***	***1.000.000***
Redis	2,1	2,1	2,2	2,2
MongoDB	1,3	1,3	1,1	0,8
Cassandra	0,6	0,6	1,3	1,3
H2	1,2	1,1	9,7	10,6

Fonte: Adaptado de Kabakus e Kara (2017)

3.1.6 Bancos de Dados NewSQL - Avaliação Experimental do MemSQL e VoltDB

Nesse trabalho, Oliveira e Bernardino (2017) avaliam o desempenho de dois SGBD NewSQL (os sistemas MemSQL e VoltDB) em diferentes tipos de consulta de dados, utilizando o sistema de *benchmarking* TPC-H[47].

Inicialmente os autores realizaram a inserção dos registros nas bases de dados. Foram inseridos 1GB de dados, divididos em 8 tabelas do TPC-H. Os tempos que os SGBD levaram para carregar esses dados são apresentados na Tabela 10.

Analisando os resultados, os autores verificaram a grande diferença de tempo de carregamento entre os dois SGBD, no qual o tempo total do VoltDB chega a ser 11 vezes maior do que o tempo do MemSQL.

Em seguida, os autores realizaram os testes dos diferentes tipos de consulta de dados do TPC-H. Foram executados 20 tipos de consulta em uma base com 1GB de dados. Os resultados podem ser vistos na Figura 11.

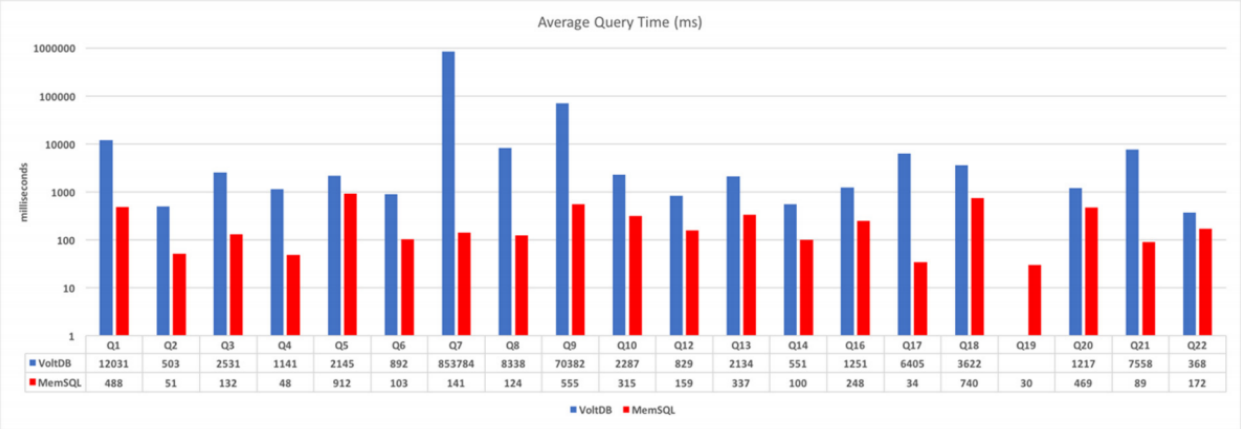
[47] http://www.tpc.org/tpch/

Tabela 10 − Tempo (em segundos) para carregamento dos dados

Tabelas	VoltDB	MemSQL
Nation	0,32	0,17
Region	0,48	0,18
Part	9,64	0,65
Supplier	0,89	0,35
Partsupp	32,56	0,86
Customer	7,14	0,49
Orders	68,74	2,90
Lineitem	339,71	33,58
Tempo Total	**459,48**	**39,18**

Fonte: Adaptado de Oliveira e Bernardino (2017)

Figura 11 − Tempo médio (em milissegundos) de execução das consultas do TPC-H



Fonte: Oliveira e Bernardino (2017)

As consultas executadas diferenciam-se entre si por buscarem informações em diferentes números de tabelas, além do fato das tabelas possuírem diferentes quantidades de registros. Mais uma vez os autores verificaram o melhor desempenho do MemSQL, como pode ser visto em todas as consultas realizadas com o *benchmark*.

3.2 Discussões

Dada a visão geral dos trabalhos relacionados, podemos realizar uma comparação e elencar as principais diferenças entre os mesmos.

O trabalho de Abramova, Bernardino e Furtado (2014) realiza um experimento de forma a medir o desempenho de alguns SGBD NoSQL para três tipos de cargas de dados. São medidos os tempos de execução em operações de leitura e atualização de dados.

Enquanto isso, Hajoui et al. (2015) comparam alguns SGBD NoSQL e NewSQL de forma subjetiva, aplicando pesos e notas a cada um deles (a partir de indicadores de qualidade) e calculando sua pontuação global.

Gurevich (2015), por sua vez, realiza comparações qualitativas sobre sistemas NoSQL e NewSQL, avaliando uma série de características técnicas dos mesmos. Ele também realiza uma comparação de desempenho entre um SGBD NoSQL e um NewSQL, no contexto de aplicações de redes sociais.

O trabalho de Binani, Gutti e Upadhyay (2016) compara qualitativamente sistemas de bancos de dados Relacionais, NoSQL e NewSQL — em termos de arquiteturas, modelos de dados, propriedades e demais características — no contexto de requisitos para aplicações *Big Data*.

Kabakus e Kara (2017) realizam um experimento para medir o desempenho de alguns SGBD NoSQL e um SGBD Relacional. São medidos os tempos de execução para operações de escrita, leitura e exclusão de dados e também o consumo de memória dessas operações.

Por fim, Oliveira e Bernardino (2017) comparam o desempenho de dois SGBD NewSQL quanto ao tempo de execução em diferentes tipos de operações de consulta de dados.

O resumo das principais características dos trabalhos relacionados é apresentado no Quadro 2.

Analisando os trabalhos apresentados, percebemos que algumas comparações realizadas não levaram em consideração o desempenho dos sistemas de banco de dados, algo essencial na escolha de um SGBD para uma aplicação. Dentre os trabalhos que realizaram as medições de desempenho, apenas um deles comparou sistemas NoSQL e NewSQL, porém essa comparação se deu apenas no contexto de aplicações de redes sociais.

Esta dissertação pretende se diferenciar dos trabalhos acima, inicialmente, por realizar um estudo experimental formalizado, com todas as definições apresentadas na Seção 2.4. Essa formalização é importante por possibilitar que outros pesquisadores consigam reproduzir este trabalho e, com isso, validar seus resultados. Além disso, este estudo irá comparar os representantes da nova geração de sistemas de banco de

dados (NoSQL e NewSQL) em cenários abrangentes, com grandes volumes de dados e diferentes tipos de cargas de dados, de forma a subsidiar a escolha do SGBD para aplicações dos mais diversos contextos.

Quadro 2 − Resumo das principais características dos trabalhos relacionados

Trabalhos	Tipo de comparação	Inclui sistemas NoSQL?	Inclui sistemas NewSQL?	Tipo de análise?	Houve medição de desempenho?
Abramova, Bernardino e Furtado	Quantitativa	Sim	Não	Objetiva	Sim
Hajoui et al.	Quantitativa	Sim	Sim	Subjetiva	Não
Gurevich	Qualitativa e Quantitativa	Sim	Sim	Objetiva	Sim
Binani, Gutti e Upadhyay	Qualitativa	Sim	Sim	Objetiva	Não
Kabakus e Kara	Quantitativa	Sim	Não	Objetiva	Sim
Oliveira e Bernardino	Quantitativa	Não	Sim	Objetiva	Sim

Fonte: O Autor (2018)

3.3 Considerações

Este capítulo apresentou, de forma detalhada, alguns trabalhos que realizaram comparações entre sistemas de gerenciamento de dados.

Uma análise comparativa entre esses trabalhos foi realizada, enfatizando as principais características e diferenças dos mesmos, além de discutirmos como o presente trabalho pretende se diferenciar dos que foram apresentados.

O próximo capítulo descreve detalhadamente a proposta de estudo comparativo deste trabalho, o qual objetiva comparar o desempenho de alguns SGBD em diferentes contextos de carga de dados.

4

DESCRIÇÃO DO ESTUDO COMPARATIVO

Neste capítulo propomos um estudo experimental que vai comparar o desempenho de diferentes tipos de SGBD NoSQL e NewSQL em diferentes cenários de carga de grandes volumes de dados. Para isso, o capítulo está estruturado da seguinte maneira: inicialmente, na Seção 4.1, apresentamos a caracterização do trabalho, com seus objetivos e as contribuições que ele irá trazer. Na Seção 4.2 é descrita a formalização do estudo experimental. A Seção 4.3 detalha a arquitetura definida e implementada para a execução dos experimentos. Por fim, as conclusões do capítulo são apresentadas na Seção 4.4.

4.1 Caracterização do Trabalho

As tecnologias de bancos de dados têm se adaptado rapidamente à nova realidade de grandes volumes, variedades e velocidade de crescimento e atualização dos dados e, por isso, a quantidade de opções de SGBD disponíveis na academia e na indústria é vasta. Contudo, como apontam Stonebraker e Cetintemel (2005), "*one size does not fit all*": a ideia de que uma única solução poderá satisfazer a todas as situações e necessidades não é válida atualmente. Consequentemente, para cada

problema existente, devemos analisar seus requisitos e compará-los com as particularidades de cada solução disponível.

Os requisitos de um banco financeiro são bastante diferentes dos de uma aplicação de rede social. Por exemplo, não é essencial garantir as propriedades ACID para atualizações no status do Facebook ou comentários no Twitter. Claramente, não é ideal que exista perda de dados, inconsistência ou interrupção do serviço. Contudo, aceitar a eventualidade desses problemas acontecerem (mesmo que de forma temporária) pode trazer uma grande flexibilidade para essas aplicações, dado que as torna livres do alto custo de manter essa integridade (STRAUCH; SITES; KRIHA, 2011).

Diante do exposto, este trabalho de pesquisa realiza um estudo comparativo que permita analisar o desempenho de alguns dos principais representantes da nova geração de SGBD, em situações de grandes volumes de dados, de forma a indicar qual tipo de sistema dessa nova geração melhor se adequa a cada um dos cenários que serão analisados.

Existem, por exemplo, aplicações que realizam mais operações de escrita do que de leitura de dados. Existem também aquelas que fazem predominantemente operações de leitura de dados. E ainda aquelas em que existe um equilíbrio na realização dessas operações. Buscando abranger diferentes cenários de interação entre uma aplicação e sua base de dados (no que se refere a operações de escrita e leitura de dados), definimos cinco categorias de aplicações:

- **Categoria A:**

Dentro desse grupo estão aplicações nas quais predominam as operações de escrita de dados, ou seja, aplicações cuja escrita representa 95% do total de operações de banco de dados, enquanto a leitura representa apenas 5% desse total.

- **Categoria B:**

Dentro desse grupo estão aplicações com mais escrita do que leitura de dados, ou seja, aplicações cuja escrita representa 70% do total de operações de banco de dados, enquanto a leitura representa 30% desse total.

- **Categoria C:**

Dentro desse grupo estão aplicações balanceadas entre escrita e leitura de dados, ou seja, aplicações cuja escrita representa 50% do total de operações de banco de dados, enquanto a leitura representa também 50% desse total.

- **Categoria D:**

Dentro desse grupo estão aplicações com mais leitura do que escrita de dados, ou seja, aplicações cuja escrita representa 30% do total de operações de banco de dados, enquanto a leitura representa 70% desse total.

- **Categoria E:**

Dentro desse grupo estão aplicações nas quais predominam as operações de leitura de dados, ou seja, aplicações cuja escrita de dados representa apenas 5% do total de operações de banco de dados, enquanto a leitura de dados representa 95% desse total.

A escolha dessas categorias e de seus percentuais de operações de escrita e leitura foi feita pelo próprio pesquisador, de forma que os experimentos consigam abranger os mais diversos cenários de aplicações em diferentes contextos de leitura e escrita de dados.

Usamos os modelos e diretrizes propostos por Juristo e Moreno (2013), conforme descritos na Seção 2.4, para definir um estudo experimental que irá avaliar o desempenho de diferentes tipos de SGBD NoSQL e NewSQL para cada uma das categorias listadas acima.

4.2 Definições do Estudo Comparativo

A formalização deste estudo experimental é importante pois permite que outros pesquisadores possam reproduzir esta pesquisa da forma mais aproximada possível. Se os resultados da replicação externa forem consistentes com os desta pesquisa, isso demonstra um aumento de confiança nas hipóteses sustentadas pelo estudo original (JUDD; SMITH; KIDDER, 1991).

Este estudo experimental foi dividido nas seguintes etapas:

- Definição dos objetivos;
- Design do experimento;

- Execução do experimento; e
- Análise dos dados/resultados coletados.

As duas primeiras etapas serão detalhadas a seguir, enquanto as demais serão tratadas no capítulo seguinte.

Objetivo Geral:

Os experimentos realizados nesta pesquisa visam prover embasamento para a escolha de um SGBD de acordo com os requisitos de uma aplicação que necessite lidar com grandes volumes de dados.

Para atingir este objetivo, realizamos cinco experimentos, de acordo com os diferentes tipos de carga de grandes volumes de dados que queremos analisar:

- **1º Experimento:** Categoria A;
- **2º Experimento:** Categoria B;
- **3º Experimento:** Categoria C;
- **4º Experimento:** Categoria D;
- **5º Experimento:** Categoria E.

Formulação das Hipóteses:

A partir do objetivo acima, conseguimos formular as seguintes hipóteses a serem validadas pelos experimentos:

a) 1º Experimento - Categoria A (95% de escrita e 5% de leitura):

- Hipótese nula (H_{10}): não há diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para cargas da categoria A;
- Hipótese alternativa (H_{11}): existe diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para cargas da categoria A.

b) 2º Experimento - Categoria B (70% de escrita e 30% de leitura):

- Hipótese nula (H_{20}): não há diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para cargas da categoria B;
- Hipótese alternativa (H_{21}): existe diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para cargas da categoria B.

c) 3º Experimento - Categoria C (50% de escrita e 50% de leitura):

- Hipótese nula (H_{30}): não há diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para cargas da categoria C;
- Hipótese alternativa (H_{31}): existe diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para cargas da categoria C.

d) 4º Experimento - Categoria D (30% de escrita e 70% de leitura):

- Hipótese nula (H_{40}): não há diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para cargas da categoria D;
- Hipótese alternativa (H_{41}): existe diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para cargas da categoria D.

e) 5º Experimento - Categoria E (5% de escrita e 95% de leitura):

- Hipótese nula (H_{50}): não há diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para cargas da categoria E;
- Hipótese alternativa (H_{51}): existe diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para cargas da categoria E.

Objetos:

Os objetos deste estudo experimental são os *scripts* das cargas de dados que serão executadas nos SGBD. Ao longo da pesquisa, foi escolhida a base de dados pública IMDb[48] (*Internet Movie Database*), a partir da qual foram definidos os esquemas de dados e os tamanhos dos *scripts*.

A base de dados do IMDb possui informações relacionadas a filmes e programas de TV, como elenco, equipe de produção, diretores, roteiristas, avaliação, dentre outras. Seus dados são atualizados diariamente e estão separados em diferentes arquivos. Para este estudo experimental, foram utilizados os dados contidos no arquivo *title.basics.tsv.gz*, que contém mais de 1 milhão de registros. Nesse arquivo estão informações básicas, tais como: título do filme/programa de TV, título na língua original, se é adulto, ano de lançamento, ano de término (para programas de TV), duração e gênero.

[48] Disponível em: https://www.imdb.com/interfaces/

Sujeito:

Como os objetos serão aplicados diretamente nos computadores do experimento, o sujeito experimental será o próprio pesquisador, que será o responsável pela execução de todos os experimentos.

Variável de Resposta:

A variável de resposta será o tempo de execução dos *scripts* de carga de grandes volumes de dados, em cada SGBD sendo avaliado.

Parâmetros:

Um importante parâmetro que precisa ser definido neste estudo experimental é a configuração da máquina na qual os experimentos serão executados.

Neste estudo, os experimentos foram executados em uma máquina (rodando no *data center* do Centro de Informática da UFPE), com a seguinte configuração: 4 núcleos de processamento Intel Xeon X3363 2.83GHz e 16GB de RAM, no qual estão instalados o sistema operacional Ubuntu 16.04 LTS de 64 bits e o pacote OpenJDK 8.

Fatores e Alternativas:

Definimos dois fatores que serão variados nos experimentos. São eles:

1. Os SGBD NoSQL e NewSQL a serem comparados; e
2. Os tamanhos das diferentes cargas de dados.

Vimos que os sistemas NoSQL são classificados, de acordo com seu modelo de representação, em: chave-valor, documentos, famílias de colunas e grafos. Diante do fato de que os sistemas de bancos de dados em grafos utilizam um paradigma distante do modelo relacional dos NewSQL que queremos comparar, decidimos deixá-los fora do escopo deste trabalho.

Por sua vez, os sistemas NewSQL costumam ser classificados nas categorias: novas arquiteturas, *transparent sharding middleware* e banco de dados como um serviço. Baseados no fato de que a categoria *transparent sharding middleware* utiliza como base os SGBD relacionais tradicionais e a categoria banco de dados como um serviço é voltada para sistemas na nuvem, decidimos deixar os NewSQL dessas duas categorias fora do escopo deste trabalho.

Portanto, as categorias de SGBD NoSQL e NewSQL que serão comparadas neste trabalho são:

- NoSQL chave-valor;
- NoSQL de documentos;
- NoSQL de família de colunas; e
- NewSQL de nova arquitetura.

Quanto ao primeiro fator a ser variado nos experimentos, decidimos por escolher um SGBD de cada categoria comparada, para ser o representante da mesma. A escolha de cada SGBD se deu em função de sua relevância no mercado e de serem soluções *open-source*. Quanto à relevância, foram avaliados rankings de SGBD produzidos pelas seguintes fontes:

- *DB-Engines*[49]: mede a popularidade dos SGBD, a partir de parâmetros como número de menções ao sistema na Web, frequência de discussões técnicas, número de vagas de emprego oferecidas (nas quais o sistema é mencionado), relevância nas redes sociais, dentre outros. Quando da definição dos experimentos, os SGBD *open-source* melhores colocados para cada categoria eram:
  - NoSQL chave-valor: Redis[50];
  - NoSQL de documentos: MongoDB[51];
  - NoSQL de família de colunas: Cassandra[52]; e
  - NewSQL de nova arquitetura: HyperSQL[53].
- *G2 Crowd*[54]: ordena os SGBD de acordo com as avaliações e críticas feitas pelos usuários do próprio site. Quando da definição dos experimentos, os SGBD *open-source* melhores colocados para cada categoria eram:
  - NoSQL chave-valor: Redis;
  - NoSQL de documentos: MongoDB;
  - NoSQL de família de colunas: Cassandra; e
  - NewSQL de nova arquitetura: VoltDB[55].

[49] https://db-engines.com/en/ranking_definition
[50] https://redis.io/
[51] https://www.mongodb.com/
[52] http://cassandra.apache.org/
[53] http://hsqldb.org/
[54] https://www.g2crowd.com/categories/database-management
[55] https://www.voltdb.com/

- *Gartner*[56]: elabora pesquisas de mercado para fazer seu ranking. Quando da definição dos experimentos, os SGBD *open-source* melhores colocados para cada categoria eram:
  - NoSQL chave-valor: Redis;
  - NoSQL de documentos: MongoDB;
  - NoSQL de família de colunas: não havia sistemas dessa categoria; e
  - NewSQL de nova arquitetura: nenhum sistema era *open-source*;
- *Stack Overflow*[57]: produz seu ranking a partir de pesquisas com desenvolvedores que são usuários do site. Quando da definição dos experimentos, os SGBD *open-source* melhores colocados para cada categoria eram:
  - NoSQL chave-valor: Redis;
  - NoSQL de documentos: MongoDB;
  - NoSQL de família de colunas: Cassandra; e
  - NewSQL de nova arquitetura: não havia sistemas dessa categoria.

A partir desta análise, definimos que as alternativas para o primeiro fator, ou seja, os SGBD a serem comparados, são:

- Redis (NoSQL chave-valor), versão 4.0.8;
- MongoDB (NoSQL de documentos), versão 3.6.3;
- Cassandra (NoSQL de família de colunas), versão 3.11.3; e
- VoltDB (NewSQL de nova arquitetura), versão 8.1 Community Edition.

As versões dos SGBD selecionados são as versões mais recentes no momento da execução dos experimentos.

Para o segundo fator deste estudo, no qual queremos avaliar o desempenho de sistemas de gerenciamento de grandes volumes de dados, definimos que as alternativas para os tamanhos das diferentes cargas de dados são:

- 100 mil registros;
- 500 mil registros; e
- 1 milhão de registros.

[56] http://www.gartner.com/technology/research/methodologies/research_mq.jsp
[57] https://insights.stackoverflow.com/survey/2018#technology-databases

Variáveis de Bloqueio:

Visto que as únicas variações existentes nos experimentos são as alternativas dos fatores sendo analisados, sabemos que não haverá variações indesejadas e, portanto, não utilizaremos variáveis de bloqueio.

Experimento Unitário:

Em cada experimento, um experimento unitário será responsável por executar uma combinação de uma alternativa do fator SGBD com uma alternativa do fator tamanho da carga de dados. Por termos 4 alternativas para o primeiro fator e 3 alternativas para o segundo, dizemos que cada experimento executará 12 experimentos unitários.

Replicação Interna:

Em cada experimento unitário serão realizadas 50 iterações (ou repetições). Este é o tamanho da amostra estatística. Com este tamanho de amostra, buscamos garantir que os valores da média e desvio padrão amostrais estejam próximos o suficiente da média e desvio padrão populacionais. Assim, conseguimos fazer afirmações sobre cada SGBD com base nas informações dadas por suas amostras, aumentando, assim, a confiabilidade nos resultados do experimento.

Desenho do Experimento:

Para a escolha do desenho experimental, primeiramente verificamos que serão analisados os efeitos de dois fatores na variável de resposta. Além disso, já verificamos que não utilizaremos variáveis de bloqueio. Por esses motivos e por dispormos da possibilidade de executar todas as combinações possíveis entre os tamanhos de carga de dados e os SGBD analisados, os experimentos deste estudo irão utilizar o Desenho Fatorial, ou seja, em cada experimento unitário serão combinadas entre si cada uma das possíveis alternativas de todos os fatores. A Tabela 11 mostra o desenho dos 12 experimentos unitários que serão executados para cada um dos cinco experimentos.

4.3 Arquitetura do Experimento

Este estudo experimental foi implementado na linguagem de programação Java, acessando os SGBD com o auxílio de suas respectivas bibliotecas e utilizando o

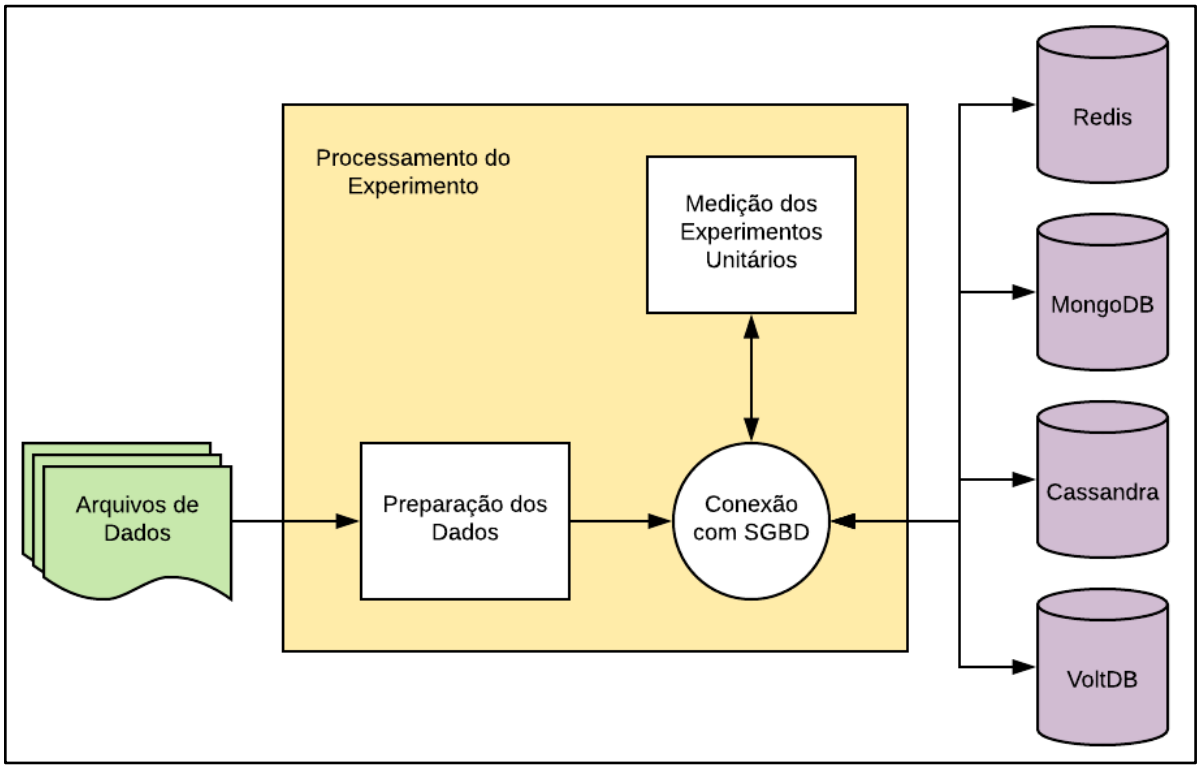
framework JMH (*Java Microbenchmark Harness*[58]) (apresentado na Seção 2.5), versão 1.19, para realizar a medição dos experimentos unitários. A arquitetura do experimento é apresentada na Figura 12.

Tabela 11 − Desenho do experimento

		SGBD			
		Redis	*MongoDB*	*Cassandra*	*VoltDB*
Tamanho da carga de dados	*100 mil*	E_{11}	E_{21}	E_{31}	E_{41}
	500 mil	E_{12}	E_{22}	E_{32}	E_{42}
	1 milhão	E_{13}	E_{23}	E_{33}	E_{43}

Fonte: O Autor (2018)

Figura 12 − Arquitetura do Experimento



Fonte: O Autor (2018)

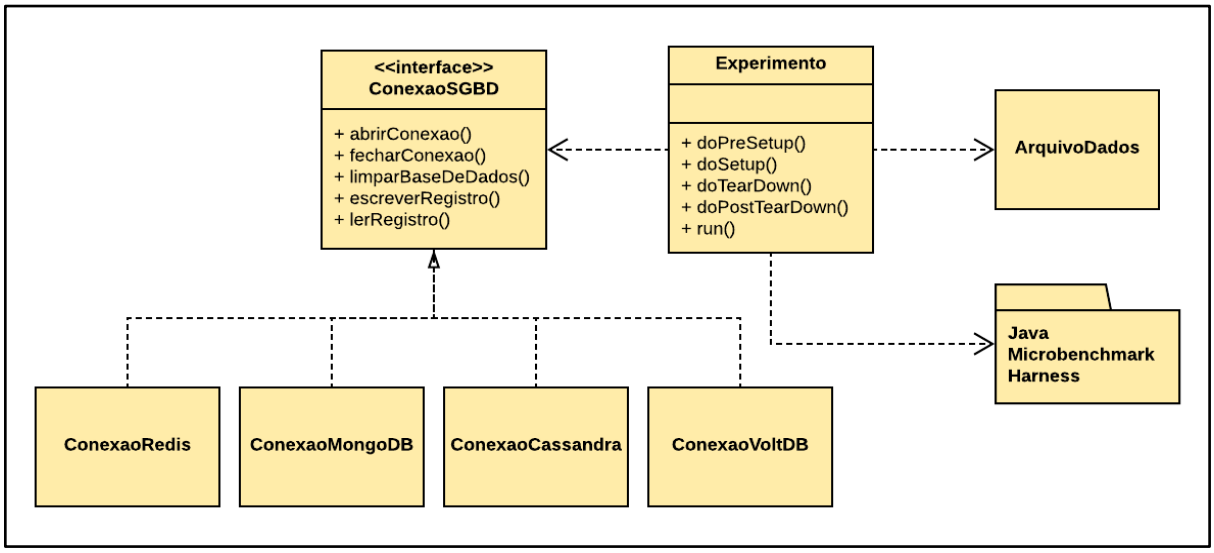
Nesta arquitetura, o módulo de Processamento do Experimento lê as informações contidas nos arquivos de dados (provenientes do IMDb), faz a preparação

58 http://openjdk.java.net/projects/code-tools/jmh/

dos dados (modelando-os de acordo com cada SGBD do experimento) e se conecta com cada SGBD para realizar as operações de escrita e leitura. Em paralelo a essas operações, são feitas as medições de desempenho dos experimentos unitários.

A Figura 13 exibe o diagrama de classes do processamento do experimento, refletindo a arquitetura apresentada. As classes do diagrama serão detalhadas a seguir.

Figura 13 − Diagrama de classes



Fonte: O Autor (2018)

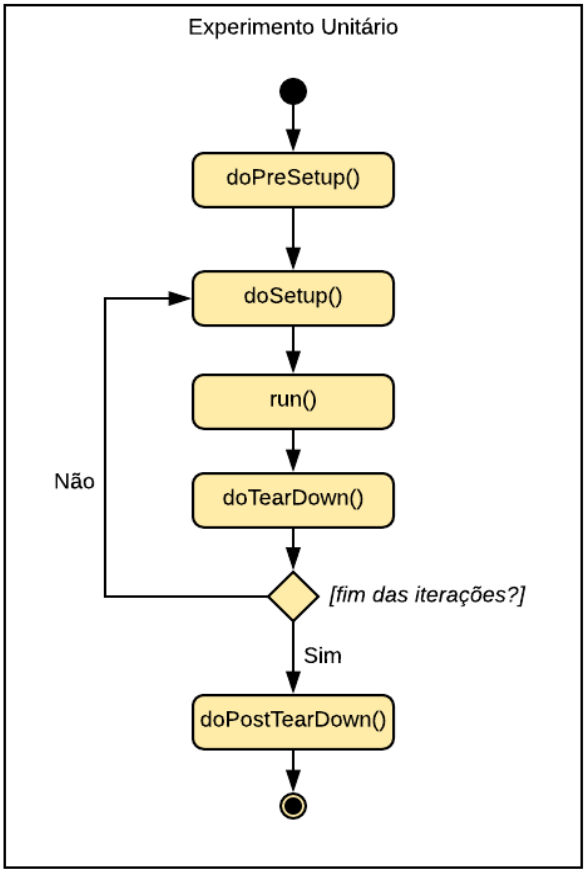
Classe *Experimento*:

Esta é a classe principal do projeto, na qual os experimentos são configurados, executados e medidos. A Figura 14 contém o diagrama de atividades dessa classe, exibindo o fluxo dos métodos durante a execução de um experimento unitário.

O método *doPreSetup()* é responsável por configurar os arquivos de dados (instanciando a classe *ArquivoDados*) e realizar a conexão com o servidor de banco de dados (instanciando uma classe que implemente a interface *ConexaoSGBD*). Como este método é executado apenas uma vez antes do início do experimento unitário, ele recebe a anotação *@Setup* do JMH com valor *Level.Trial*.

O método *doSetup()* limpa a base de dados e abre o arquivo de dados que será lido. Esse método é executado antes de cada iteração do experimento unitário e, por isso, recebe a anotação *@Setup* do JMH com valor *Level.Invocation*.

Figura 14 − Diagrama de atividades da classe *Experimento*



Fonte: O Autor (2018)

O método *run()* é onde o experimento realmente acontece. Na fase de escrita dos dados, ele lê cada linha do arquivo de dados, prepara esse registro de acordo com o modelo de dados do SGBD e escreve o registro no banco (através da interface *ConexaoSGBD*). Na fase seguinte, é feita a leitura da base de dados, de acordo com a quantidade de registros especificada para cada experimento unitário. Este método recebe as seguintes anotações:

- *@Benchmark*, para indicar que esse é o método no qual as medições serão realizadas pelo JMH;
- @Warmup, especificando o parâmetro *iterations* com valor 5;
- *@Measurement*, especificando o parâmetro *iterations* com valor 50;
- *@BenchmarkMode* com valor *Mode.AverageTime*; e
- *@OutputTimeUnit*, com valor *TimeUnit.SECONDS*.

O método *doTearDown()* fecha o arquivo de dados que foi lido. Ele é executado após cada iteração do experimento unitário e, por isso, recebe a anotação *@TearDown* do JMH com valor *Level.Invocation*.

O método *doPostTearDown()* é responsável por fechar a conexão com o servidor de banco de dados (através da interface *ConexaoSGBD*). Como esse método é executado apenas uma vez após o término do experimento unitário, ele recebe a anotação @*TearDown* do JMH com valor *Level.Trial*.

Classe *ArquivoDados*:

Esta é uma classe de apoio à classe principal *Experimento*. Nela são definidos o local onde os arquivos de dados estão armazenados no computador, a quantidade de registros em cada arquivo e qual arquivo deve ser utilizado para cada experimento unitário (considerando a categoria do experimento e o tamanho da carga de dados).

As informações contidas nos arquivos de dados, referentes aos títulos cadastrados no IMDb, são:

- *tconst*: identificador alfanumérico;
- *titleType*: tipo do título (se é filme, série de TV, programa de TV, curta metragem, dentre outros);
- *primaryTitle*: nome do título usado em materiais de divulgação;
- *originalTitle*: nome do título na língua original;
- *isAdult*: se é adulto;
- *startYear*: ano de lançamento;
- *endYear*: ano de término (para programas de TV);
- *runtimeMinutes*: duração em minutos;
- *genres*: gêneros.

Interface *ConexaoSGBD*:

Esta interface será instanciada por cada classe responsável por gerenciar a conexão e o acesso aos dados dos SGBD. Seus métodos são:

- *abrirConexao()*;
- *fecharConexao()*;
- *limparBaseDeDados()*;
- *escreverRegistro()*; e
- *lerRegistro()*.

Consequentemente, as classes que instanciam essa interface, e serão detalhadas a seguir, são:

- *ConexaoRedis*;

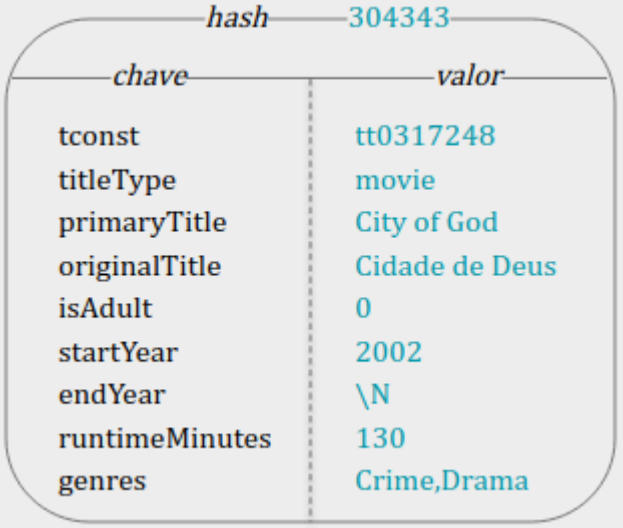
- *ConexaoMongoDB*;
- *ConexaoCassandra*; e
- *ConexaoVoltDB*.

Classe *ConexaoRedis*:

O SGBD Redis possui inúmeras bibliotecas[60] desenvolvidas para as mais diversas linguagens de programação. Como este estudo experimental foi implementado na linguagem de programação Java, escolhemos a biblioteca *Jedis*[61] para gerenciar o acesso ao SGBD.

Os dados são escritos e lidos do Redis como uma coleção de pares chave-valor para cada linha contida no arquivo de dados (ou seja, para cada título do IMDb). Essa coleção de pares é associada a um *hash*, que serve como uma chave para essa coleção. A Figura 15 apresenta um exemplo da modelagem dos dados para o Redis.

Figura 15 − Exemplo de modelagem de dados para o sistema Redis



Fonte: O Autor (2018)

Classe *ConexaoMongoDB*:

Esta classe gerencia o acesso ao SGBD MongoDB por meio de sua biblioteca oficial *MongoDB Java Driver*[62].

Os dados são armazenados no MongoDB como um objeto no formato JSON (*JavaScript Object Notation*), inserido em uma coleção de documentos chamada

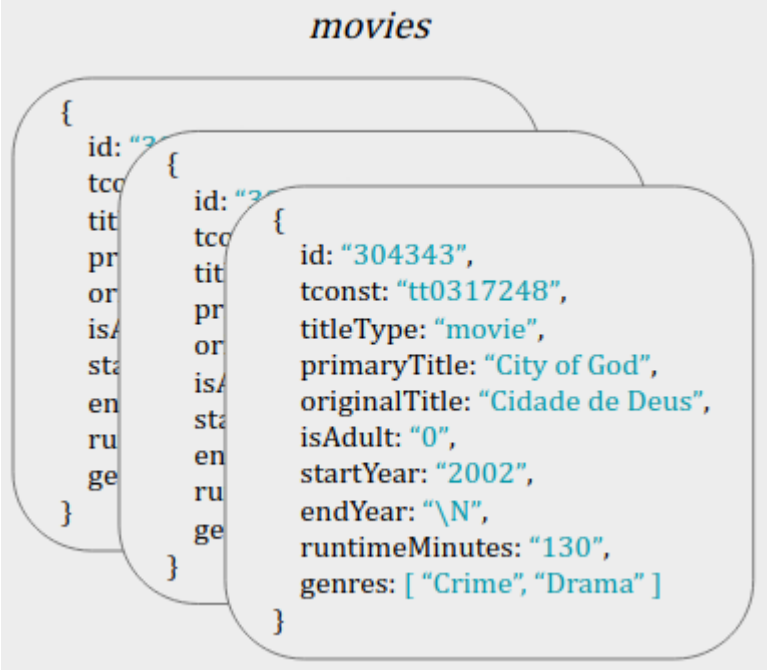
[60] https://redis.io/clients

[61] https://github.com/xetorthio/jedis

[62] https://mongodb.github.io/mongo-java-driver/

‘*movies*’. A Figura 16 apresenta um exemplo da modelagem dos dados para o MongoDB.

Figura 16 − Exemplo de modelagem de dados para o sistema MongoDB



Fonte: O Autor (2018)

Classe *ConexaoCassandra*:

O SGBD Cassandra também possui diversas bibliotecas[65] para várias linguagens de programação. Como este estudo experimental foi implementado na linguagem de programação Java, escolhemos a biblioteca *Datastax Java Driver*[66].

Os dados são armazenados no Cassandra na família de colunas ‘*movies*’, a qual é criada dentro de uma *keyspace*, chamada ‘*moviesKeyspace*’. A Figura 17 apresenta um exemplo da modelagem dos dados para o Cassandra.

Classe *ConexaoVoltDB*:

Esta classe gerencia o acesso ao SGBD VoltDB por meio de sua biblioteca oficial *VoltDB Java Client*[67].

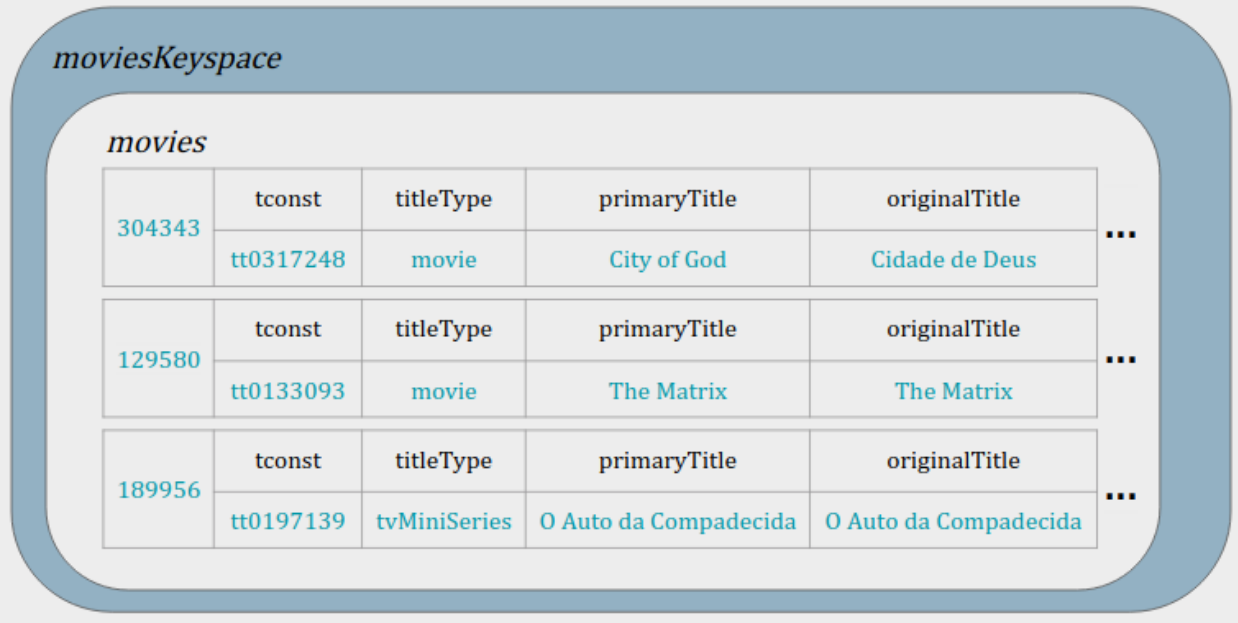
Sendo um SGBD relacional, seus dados são armazenados nas colunas da tabela ‘*movies*’, como pode ser visto na Figura 18.

[65] http://cassandra.apache.org/doc/latest/getting_started/drivers.html

[66] https://github.com/datastax/java-driver

[67] https://docs.voltdb.com/javadoc/java-client-api/

Figura 17 − Exemplo de modelagem de dados para o sistema Cassandra



Fonte: O Autor (2018)

Figura 18 − Exemplo de modelagem de dados para o sistema VoltDB

Fonte: O Autor (2018)

4.4 Considerações

Neste capítulo apresentamos a descrição do estudo comparativo que definimos para comparar o desempenho de diferentes tipos de SGBD NoSQL e NewSQL, de acordo com os requisitos de uma aplicação que necessite lidar com grandes volumes de dados.

O estudo experimental foi formalizado: seus objetivos foram definidos, suas hipóteses foram formuladas e suas condições foram estabelecidas, juntamente com seu desenho.

Ainda neste capítulo, detalhamos a arquitetura do experimento, com as particularidades de sua implementação, as modelagens dos dados para cada SGBD e as configurações do *framework JMH* para a medição dos experimentos unitários.

No próximo capítulo apresentaremos a execução dos experimentos e os resultados coletados. Além disso, analisaremos os resultados obtidos e faremos discussões acerca dos mesmos.

5

EXPERIMENTOS E ANÁLISE DOS RESULTADOS

Neste capítulo, são apresentados aspectos relacionados à execução dos experimentos propostos neste trabalho e são feitas análises acerca dos resultados coletados, a fim de verificar se a hipótese nula de cada um dos experimentos pode ser rejeitada. O capítulo está estruturado da seguinte maneira: a Seção 5.1 apresenta informações sobre como os experimentos-piloto e os experimentos principais foram executados; além disso, os resultados dos experimentos principais são apresentados e as hipóteses formuladas são validadas de acordo com os objetivos propostos para esse trabalho de pesquisa. A Seção 5.2 analisa os resultados obtidos, extraindo conclusões acerca dos mesmos. Por fim, na Seção 5.3 apresentamos as considerações finais do capítulo.

5.1 Definição dos Experimentos

Antes da execução dos experimentos principais (a partir dos quais serão feitas as análises dos resultados), decidimos executar alguns experimentos preliminares, aqui

chamados de experimentos-piloto, com o objetivo de validar as definições apresentadas nas Seções 4.2 e 4.3, quanto à formalização do estudo e sua arquitetura.

Nos experimentos-piloto foram mantidas a maior parte das características dos experimentos principais, à exceção da alternativa referente aos tamanhos das diferentes cargas de dados e da configuração da máquina onde os experimentos são executados. Como esses experimentos foram executados com pequenos volumes de dados e seu objetivo foi apenas validar as definições do estudo experimental, não faremos análises em cima de seus resultados.

Após a execução dos experimentos-piloto e certos de que as definições do estudo experimental e sua arquitetura estavam validadas, foram executados os experimentos principais, seguindo todas as definições, formalizações e arquitetura apresentadas no Capítulo 4. A partir dos resultados destes experimentos principais fizemos as análises dos resultados coletados, a validação das hipóteses formuladas e as conclusões acerca deste trabalho de pesquisa.

5.1.1 Execução dos Experimentos-Piloto

Nos experimentos-piloto, os tamanhos das cargas de dados utilizadas foram 1.000, 10.000 e 50.000 registros. Quanto à configuração da máquina, estes experimentos foram executados em um computador com 4 núcleos de processamento Intel Core i7-4500U 1.80GHz e 8GB de memória RAM, no qual estão instalados o sistema operacional Ubuntu 16.04 LTS de 64 bits e o pacote OpenJDK 8.

As Tabelas 12 a 16 mostram os resultados da execução dos cinco experimentos-piloto nos SGBD Redis, MongoDB, Cassandra e VoltDB. Nestas tabelas são exibidos o tempo médio de execução e o desvio padrão (destacado entre parênteses) obtido em cada experimento unitário.

Tabela 12 − Resultados do 1º Experimento-Piloto - Categoria A (95% de escrita e 5% de leitura)

Tamanho da Carga de Dados	**Tempo Médio de Execução (em segundos)**			
	Redis	**MongoDB**	**Cassandra**	**VoltDB**
1 mil	0,044 (0,001)	0,341 (0,023)	0,374 (0,081)	0,534 (0,043)
10 mil	0,430 (0,013)	1,273 (0,061)	3,335 (0,047)	4,893 (0,208)
50 mil	2,103 (0,068)	5,355 (0,152)	16,603 (0,164)	23,735 (0,110)

Fonte: O Autor (2018)

Tabela 13 − Resultados do 2º Experimento-Piloto - Categoria B (70% de escrita e 30% de leitura)

Tamanho da Carga de Dados	Tempo Médio de Execução (em segundos)			
	Redis	MongoDB	Cassandra	VoltDB
1 mil	0,040 (0,001)	0,307 (0,034)	0,346 (0,011)	0,436 (0,008)
10 mil	0,395 (0,013)	0,978 (0,052)	3,268 (0,032)	4,342 (0,082)
50 mil	1,990 (0,064)	3,966 (0,098)	16,431 (0,150)	21,559 (0,226)

Fonte: O Autor (2018)

Tabela 14 − Resultados do 3º Experimento-Piloto - Categoria C (50% de escrita e 50% de leitura)

Tamanho da Carga de Dados	Tempo Médio de Execução (em segundos)			
	Redis	MongoDB	Cassandra	VoltDB
1 mil	0,039 (0,001)	0,285 (0,028)	0,336 (0,019)	0,374 (0,009)
10 mil	0,376 (0,014)	0,768 (0,054)	3,242 (0,037)	3,836 (0,087)
50 mil	1,881 (0,057)	2,914 (0,081)	16,292 (0,111)	19,623 (0,201)

Fonte: O Autor (2018)

Tabela 15 − Resultados do 4º Experimento-Piloto - Categoria D (30% de escrita e 70% de leitura)

Tamanho da Carga de Dados	Tempo Médio de Execução (em segundos)			
	Redis	MongoDB	Cassandra	VoltDB
1 mil	0,036 (0,001)	0,274 (0,046)	0,332 (0,014)	0,236 (0,005)
10 mil	0,358 (0,008)	0,593 (0,035)	3,194 (0,019)	3,291 (0,216)
50 mil	1,753 (0,052)	1,912 (0,077)	15,958 (0,138)	17,786 (0,478)

Fonte: O Autor (2018)

Tabela 16 − Resultados do 5º Experimento-Piloto - Categoria E (5% de escrita e 95% de leitura)

Tamanho da Carga de Dados	Tempo Médio de Execução (em segundos)			
	Redis	MongoDB	Cassandra	VoltDB
1 mil	0,032 (0,001)	0,266 (0,024)	0,321 (0,015)	0,163 (0,005)
10 mil	0,319 (0,010)	0,315 (0,035)	3,123 (0,014)	1,716 (0,034)
50 mil	1,582 (0,060)	0,549 (0,037)	15,569 (0,047)	15,844 (1,217)

Fonte: O Autor (2018)

Analisando as tabelas acima, apesar de representarem resultados com poucos volumes de dados, já podemos perceber algumas tendências que foram confirmadas

com os resultados dos experimentos principais, como o desempenho superior dos SGBD Redis e MongoDB.

5.1.2 Execução dos Experimentos Principais e Validação das Hipóteses

Nesta seção serão apresentados os resultados e validadas as hipóteses dos cinco experimentos principais, executados nos SGBD Redis, MongoDB, Cassandra e VoltDB, com os tamanhos das cargas de dados e a configuração da máquina conforme descritos na formalização do experimento, apresentada no Capítulo 4.

O objetivo final de nosso experimento é produzir afirmações sobre dada característica de uma população (os SGBD NoSQL e NewSQL) a partir de informações colhidas de uma amostra dessa população (os resultados dos experimentos). A isso é dado o nome de Inferência Estatística. Para atingir este objetivo, precisamos realizar testes estatísticos de hipóteses, que fornecem uma metodologia que nos permite verificar se determinados dados amostrais trazem evidências que apoiem ou não uma hipótese formulada (MORETTIN; BUSSAB, 2017).

De fato, precisamos validar as hipóteses nulas de cada um dos cinco experimentos realizados. Para isso, iremos testar a diferença entre as médias de tempo de execução de cada SGBD, de forma a verificar se uma determinada alternativa (no nosso caso, um sistema NoSQL ou NewSQL) é melhor do que outra, nos cenários de grandes volumes de dados.

Como os tamanhos das amostras dos experimentos (no caso, a quantidade de repetições do experimento unitário) são grandes (ou seja, maiores do que 30), podemos assumir a normalidade na distribuição das amostras. Apesar das variâncias das populações serem desconhecidas, Montgomery, Runger e Calado (2000) afirmam que, se os tamanhos das amostras excederem a 40, é possível utilizar testes para variância conhecida. Logo, iremos aplicar nos resultados coletados nos experimentos os testes de hipóteses para a diferença de médias com variâncias conhecidas.

O objetivo de um teste de hipóteses é dizer, usando uma dada estatística, se a hipótese nula é ou não aceitável. Essa decisão é tomada considerando uma região chamada de região crítica ou região de rejeição do teste. Caso o valor observado na estatística pertença a essa região, rejeitamos a hipótese nula; caso contrário, não a rejeitamos. Essa região é sempre construída sob a suposição da hipótese nula ser verdadeira (MORETTIN; BUSSAB, 2017).

De forma geral, as hipóteses nulas definidas na Seção 4.2 afirmam não haver diferença no tempo de execução entre os SGBD em determinado cenário de carga de dados. Isso significa que precisamos averiguar, em cada cenário, se é significativa a diferença entre o tempo de execução médio observado em um SGBD e o tempo de execução médio observado em outro SGBD. Como estamos testando a igualdade de duas médias e partindo do pressuposto que a hipótese nula é verdadeira, especificamos que:

$$H_0: \mu_1 - \mu_2 = 0$$

onde μ é a média populacional.

Portanto, a estatística do teste de hipóteses para a diferença de médias com variâncias conhecidas será calculada por meio da variável aleatória Z, que tem distribuição normal padrão e é obtida pela fórmula (MONTGOMERY; RUNGER; CALADO, 2000):

$$Z_0 = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

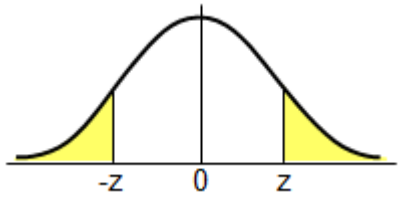
onde $\bar{X}$ corresponde à média amostral, σ é o desvio padrão amostral e n é o tamanho da amostra.

Para a aplicação do teste de hipóteses, precisamos definir o seu nível de significância (denotado por α), que corresponde à probabilidade de se rejeitar a hipótese nula quando essa for verdadeira (conhecido como erro de tipo I). Quanto menor for esse nível α, mais significativo será o resultado da amostra para rejeitar a hipótese nula. Isso quer dizer que, quanto menor o valor de α, menor a probabilidade de se obter uma amostra com estatística pertencente à região crítica (MORETTIN; BUSSAB, 2017).

Para nossos testes de hipóteses, fixamos o valor de α em 0,001 (0,1%). Isso significa que existe 1 chance em 1000 de rejeitarmos a hipótese nula quando ela deveria ter sido aceita. Em outras palavras, temos uma confiança de 99,9% de termos tomado a decisão correta.

Para esse nível de significância, os valores da estatística do teste de hipóteses correspondentes à região crítica (de acordo com a Tabela 17) são os valores menores que -3,29 e maiores que 3,29. Portanto, calculado o valor da estatística Z, se ele pertencer ao intervalo de -3,29 a 3,29, a hipótese nula não pode ser rejeitada para o nível de significância de 0,1%. Caso o valor de Z esteja fora desse intervalo, podemos rejeitar a hipótese nula para esse mesmo nível de significância.

Tabela 17 − Valores críticos da distribuição normal para testes bilaterais

Nível de significância α	**0,10**	**0,05**	**0,01**	**0,005**	**0,002**	**0,001**
Valores críticos de Z	-1,645 ou 1,645	-1,96 ou 1,96	-2,58 ou 2,58	-2,81 ou 2,81	-3,08 ou 3,08	-3,29 ou 3,29
						

Fonte: Adaptado de Juristo e Moreno (2013)

Como esse teste de hipóteses é aplicado para verificar a diferença nas médias de duas populações, inicialmente ordenaremos os SGBD do menor tempo médio de execução para o maior. Em seguida, o teste será aplicado aos pares de SGBD de acordo com essa ordenação. Ou seja, serão comparados o 1º SGBD com o 2º, o 2º com o 3º e o 3º com o 4º, dentro de cada cenário de carga de dados.

A seguir, apresentaremos os resultados obtidos em cada um dos cinco experimentos executados e os seus respectivos testes de hipóteses. Como o leitor poderá verificar, os resultados obtidos nos testes de hipótese são bastante semelhantes. Por esse motivo, decidimos apresentar inicialmente os resultados dos cinco experimentos para, em seguida, apresentar as considerações acerca da validação das cinco hipóteses nulas.

1º Experimento - Categoria A (95% de escrita e 5% de leitura):

Na Tabela 18 são exibidos o tempo médio de execução e o desvio padrão (destacado entre parênteses) obtidos nos experimentos unitários da Categoria A.

Aplicados os testes de hipóteses para a hipótese nula H_{10} nos resultados dos experimentos unitários, a Tabela 19 apresenta os pares de SGBD comparados e os valores da estatística do teste Z.

Tabela 18 − Resultados do 1º Experimento - Categoria A

Tamanho da Carga de Dados	Tempo Médio de Execução (em segundos)			
	Redis	MongoDB	Cassandra	VoltDB
100 mil	4,510 (0,010)	10,507 (0,080)	31,577 (0,247)	72,403 (0,679)
500 mil	22,449 (0,030)	52,040 (0,525)	158,403 (0,803)	363,834 (1,988)
1 milhão	45,276 (0,048)	104,142 (0,163)	322,334 (1,793)	727,352 (4,707)

Fonte: O Autor (2018)

Tabela 19 − Testes de Hipótese do 1º Experimento - Categoria A

Tamanho da Carga de Dados	SGBD 1	SGBD 2	Z
100 mil	Redis	MongoDB	-525,972
	MongoDB	Cassandra	-573,840
	Cassandra	VoltDB	-399,545
500 mil	Redis	MongoDB	-397,903
	MongoDB	Cassandra	-783,934
	Cassandra	VoltDB	-677,510
1 milhão	Redis	MongoDB	-2449,647
	MongoDB	Cassandra	-856,952
	Cassandra	VoltDB	-568,582

Fonte: O Autor (2018)

2º Experimento - Categoria B (70% de escrita e 30% de leitura):

A Tabela 20 apresenta o tempo médio de execução e o desvio padrão (destacado entre parênteses) obtidos nos experimentos unitários da Categoria B.

Tabela 20 − Resultados do 2º Experimento - Categoria B

Tamanho da Carga de Dados	Tempo Médio de Execução (em segundos)			
	Redis	MongoDB	Cassandra	VoltDB
100 mil	4,185 (0,004)	7,775 (0,045)	30,943 (0,256)	60,522 (0,601)
500 mil	20,897 (0,015)	38,157 (0,121)	158,304 (0,862)	303,738 (1,994)
1 milhão	41,709 (0,036)	76,756 (0,184)	317,825 (1,289)	607,047 (3,316)

Fonte: O Autor (2018)

Após a aplicação dos testes de hipóteses para a hipótese nula H_{20} nos resultados dos experimentos unitários, a Tabela 21 apresenta os pares de SGBD comparados e os valores da estatística do teste Z.

Tabela 21 − Testes de Hipótese do 2º Experimento - Categoria B

Tamanho da Carga de Dados	SGBD 1	SGBD 2	Z
100 mil	Redis	MongoDB	-561,899
	MongoDB	Cassandra	-630,268
	Cassandra	VoltDB	-320,176
500 mil	Redis	MongoDB	-1000,988
	MongoDB	Cassandra	-976,008
	Cassandra	VoltDB	-473,393
1 milhão	Redis	MongoDB	-1321,785
	MongoDB	Cassandra	-1309,161
	Cassandra	VoltDB	-574,837

Fonte: O Autor (2018)

3º Experimento - Categoria C (50% de escrita e 50% de leitura):

Na Tabela 22 são exibidos o tempo médio de execução e o desvio padrão (destacado entre parênteses) obtidos nos experimentos unitários da Categoria C.

Tabela 22 − Resultados do 3º Experimento - Categoria C

Tamanho da Carga de Dados	Tempo Médio de Execução (em segundos)			
	Redis	MongoDB	Cassandra	VoltDB
100 mil	3,915 (0,005)	5,658 (0,035)	30,494 (0,241)	51,016 (0,448)
500 mil	19,504 (0,033)	27,489 (0,047)	153,241 (0,829)	256,993 (1,745)
1 milhão	39,038 (0,031)	54,808 (0,141)	314,559 (1,267)	512,469 (2,440)

Fonte: O Autor (2018)

Aplicados os testes de hipóteses para a hipótese nula H_{30} nos resultados dos experimentos unitários, a Tabela 23 apresenta os pares de SGBD comparados e os valores da estatística do teste Z.

4º Experimento - Categoria D (30% de escrita e 70% de leitura):

A Tabela 24 apresenta o tempo médio de execução e o desvio padrão (destacado entre parênteses) obtidos nos experimentos unitários da Categoria D.

Tabela 23 − Testes de Hipótese do 3º Experimento - Categoria C

Tamanho da Carga de Dados	SGBD 1	SGBD 2	Z
100 mil	Redis	MongoDB	-348,600
	MongoDB	Cassandra	-721,136
	Cassandra	VoltDB	-285,256
500 mil	Redis	MongoDB	-983,184
	MongoDB	Cassandra	-1070,899
	Cassandra	VoltDB	-379,748
1 milhão	Redis	MongoDB	-772,408
	MongoDB	Cassandra	-1440,764
	Cassandra	VoltDB	-509,007

Fonte: O Autor (2018)

Tabela 24 − Resultados do 4º Experimento - Categoria D

Tamanho da Carga de Dados	Tempo Médio de Execução (em segundos)			
	Redis	**MongoDB**	**Cassandra**	**VoltDB**
100 mil	3,629 (0,004)	3,477 (0,033)	29,933 (0,394)	41,515 (0,472)
500 mil	18,200 (0,035)	16,765 (0,035)	150,248 (0,790)	208,341 (1,060)
1 milhão	36,472 (0,051)	32,829 (0,072)	301,202 (1,416)	415,641 (1,408)

Fonte: O Autor (2018)

Após a aplicação dos testes de hipóteses para a hipótese nula H_{40} nos resultados dos experimentos unitários, a Tabela 25 exibe os pares de SGBD comparados e os valores da estatística do teste Z.

Tabela 25 − Testes de Hipótese do 4º Experimento - Categoria D

Tamanho da Carga de Dados	SGBD 1	SGBD 2	Z
100 mil	MongoDB	Redis	-32,333
	Redis	Cassandra	-472,050
	Cassandra	VoltDB	-133,202
500 mil	MongoDB	Redis	-205,000
	Redis	Cassandra	-1180,766
	Cassandra	VoltDB	-310,724
1 milhão	MongoDB	Redis	-291,954
	Redis	Cassandra	-1321,123
	Cassandra	VoltDB	-405,236

Fonte: O Autor (2018)

5º Experimento - Categoria E (5% de escrita e 95% de leitura):

Na Tabela 26 são exibidos o tempo médio de execução e o desvio padrão (destacado entre parênteses) obtidos nos experimentos unitários da Categoria E.

Tabela 26 − Resultados do 5º Experimento - Categoria E

Tamanho da Carga de Dados	Tempo Médio de Execução (em segundos)			
	Redis	MongoDB	Cassandra	VoltDB
100 mil	3,297 (0,018)	0,749 (0,022)	28,928 (0,284)	29,493 (0,255)
500 mil	16,393 (0,020)	3,014 (0,042)	145,389 (0,834)	148,290 (0,735)
1 milhão	32,687 (0,064)	5,921 (0,054)	291,291 (1,594)	296,535 (0,956)

Fonte: O Autor (2018)

Aplicados os testes de hipóteses para a hipótese nula H_{50} nos resultados dos experimentos unitários, a Tabela 27 apresenta os pares de SGBD comparados e os valores da estatística do teste Z.

Tabela 27 − Testes de Hipótese do 5º Experimento - Categoria E

Tamanho da Carga de Dados	SGBD 1	SGBD 2	Z
100 mil	MongoDB	Redis	-633,839
	Redis	Cassandra	-636,886
	Cassandra	VoltDB	-10,467
500 mil	MongoDB	Redis	-2033,668
	Redis	Cassandra	-1093,378
	Cassandra	VoltDB	-18,453
1 milhão	MongoDB	Redis	-2260,203
	Redis	VoltDB	-1146,257
	VoltDB	Cassandra	-19,950

Fonte: O Autor (2018)

Validação das Hipóteses:

A partir dos resultados dos testes de hipótese aplicados nos cinco experimentos, verificamos que, em todos os casos, o valor da estatística Z está fora do intervalo definido para a região crítica (de -3,29 a 3,29). Portanto, podemos rejeitar as hipóteses nulas H_{10}, H_{20}, H_{30}, H_{40} e H_{50} com nível de significância de 0,1%. Concluímos, então, que existe diferença nos tempos de execução entre os tipos de SGBD NoSQL e NewSQL para grandes volumes de dados nas cinco categorias avaliadas.

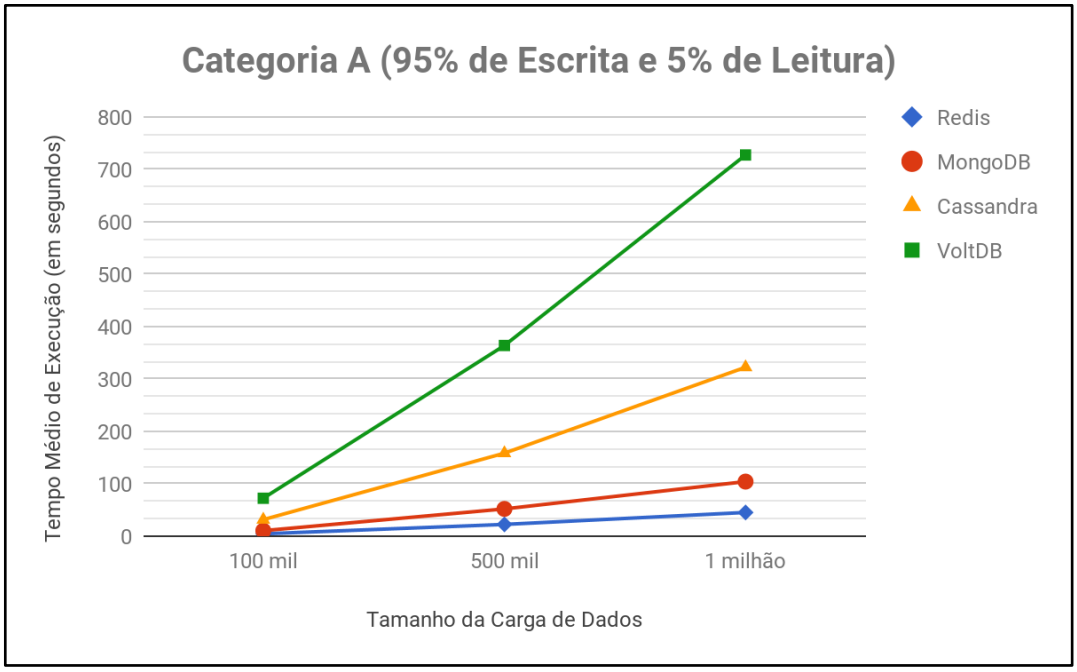
5.2 Resultados dos Experimentos

Rejeitadas todas as hipóteses nulas, nas seções seguintes será exibido um conjunto de gráficos, comparando os resultados coletados nos experimentos, e, por fim, apresentaremos análises e conclusões acerca desses gráficos e dos resultados observados.

5.2.1 Tamanho da Carga de Dados X Tempo Médio de Execução do SGBD

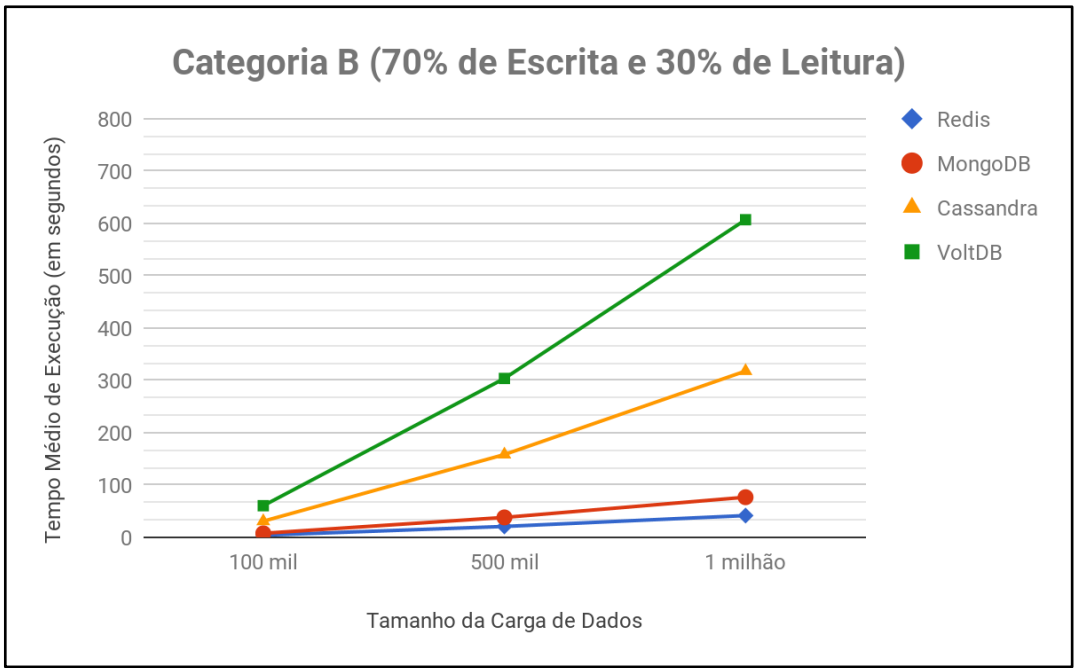
As Figuras 19 a 23 apresentam gráficos contendo os resultados obtidos em cada um dos cinco experimentos executados, a partir da perspectiva de cada categoria avaliada.

Figura 19 – Gráfico do 1º Experimento - Categoria A



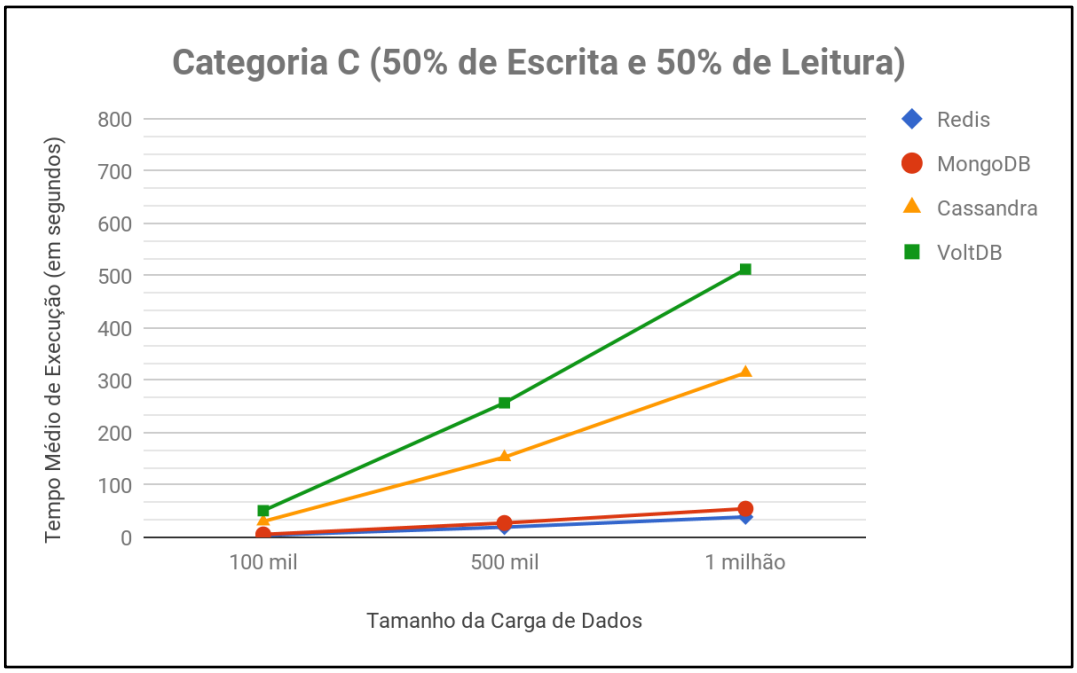
Fonte: O Autor (2018)

Figura 20 – Gráfico do 2º Experimento - Categoria B



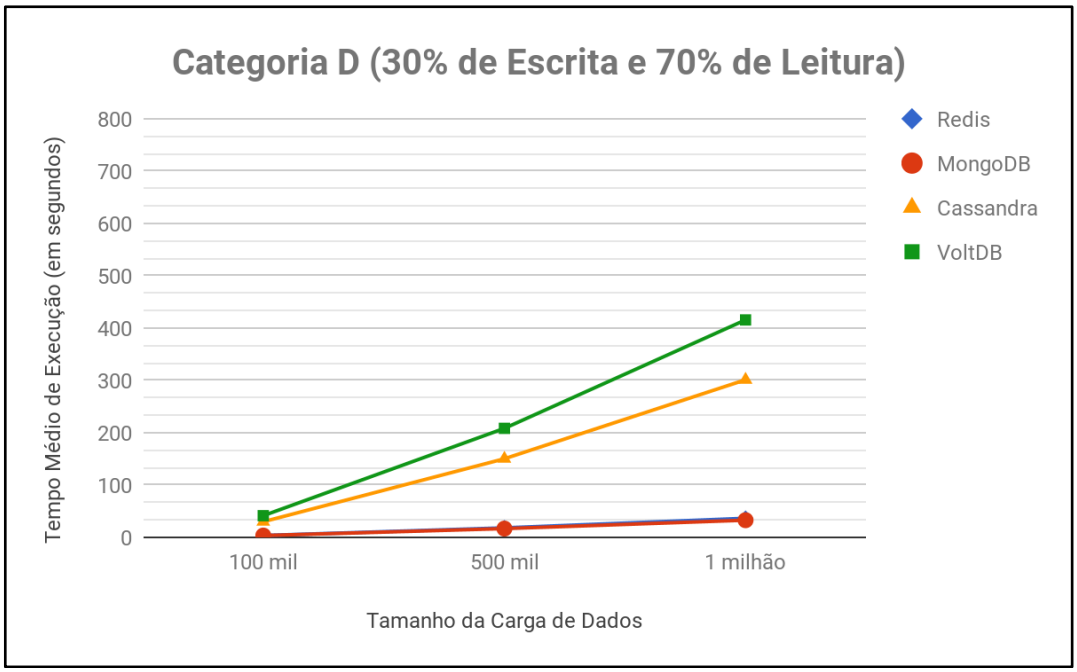
Fonte: O Autor (2018)

Figura 21 – Gráfico do 3º Experimento - Categoria C



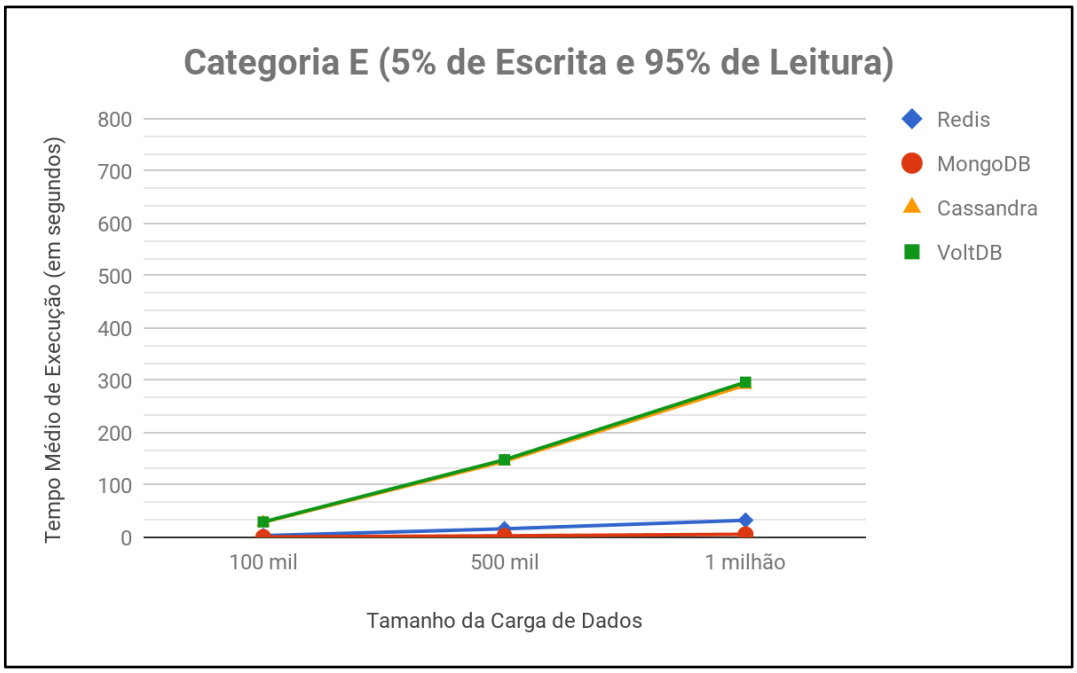
Fonte: O Autor (2018)

Figura 22 – Gráfico do 4º Experimento - Categoria D



Fonte: O Autor (2018)

Figura 23 – Gráfico do 5º Experimento - Categoria E



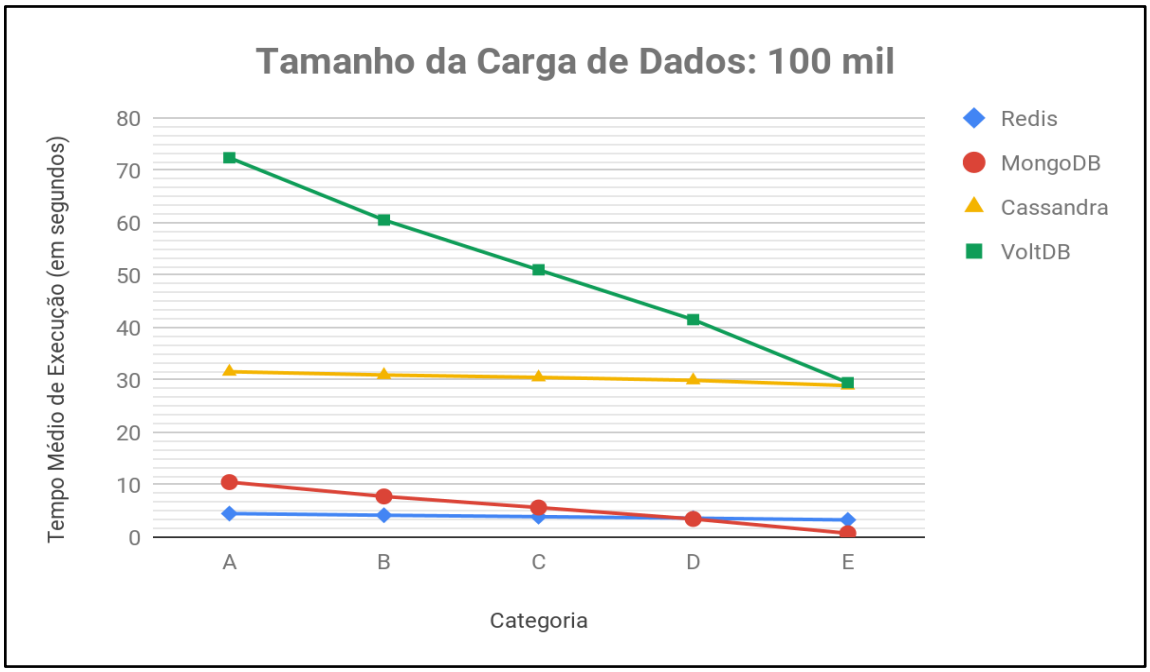
Fonte: O Autor (2018)

5.2.2 Categoria X Tempo Médio de Execução do SGBD

Além de analisar os resultados a partir da perspectiva de cada categoria, também podemos analisar os resultados do ponto de vista dos tamanhos das cargas de dados.

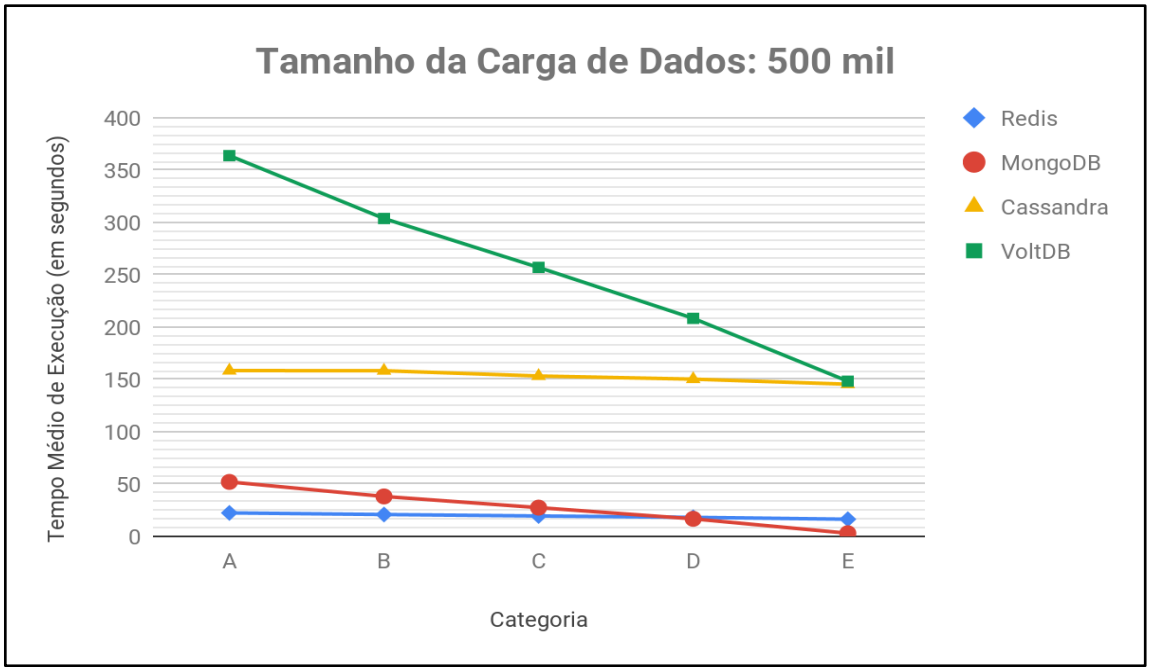
Para isso, a Figura 24 apresenta um gráfico com os resultados obtidos com o tamanho da carga de dados igual a 100 mil registros, a Figura 25 apresenta os resultados para 500 mil registros e a Figura 26 para 1 milhão de registros.

Figura 24 − Gráfico de Tamanho da Carga Igual a 100 mil registros



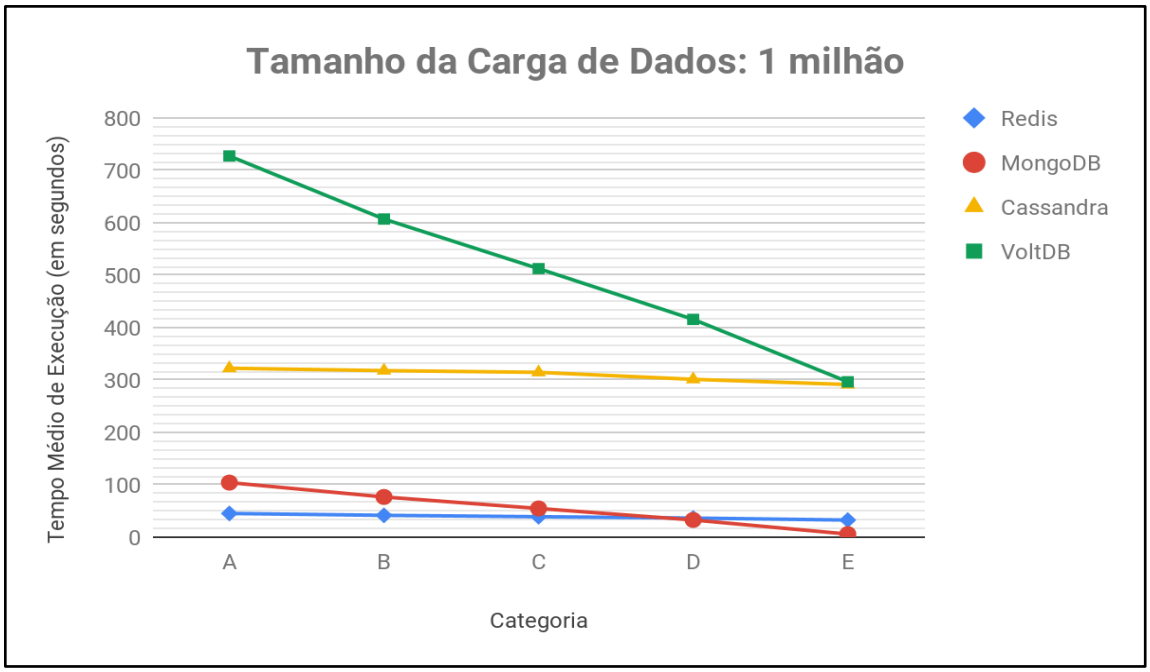
Fonte: O Autor (2018)

Figura 25 − Gráfico de Tamanho da Carga Igual a 500 mil registros



Fonte: O Autor (2018)

Figura 26 – Gráfico de Tamanho da Carga Igual a 1 milhão de registros



Fonte: O Autor (2018)

5.2.3 Análise dos Resultados

Analisando todos os gráficos e resultados já apresentados, podemos verificar que, de forma geral, existe uma clara diferença de desempenho entre os SGBD Redis e MongoDB em comparação com os SGBD Cassandra e VoltDB. Em todas as situações, os dois primeiros são pelo menos três vezes mais rápidos do que os dois últimos, chegando essa diferença a ser de até nove vezes mais rápidos.

Observando os resultados dos sistemas Redis e MongoDB, percebemos que, à medida em que as operações de leitura se tornam mais predominantes, o desempenho do MongoDB frente ao Redis melhora consideravelmente. Nas situações em que acontecem mais escritas de dados do que leituras, o desempenho do Redis se sobressai. Contudo, nos experimentos das Categorias D e E observamos que o MongoDB consegue ser o mais rápido, independentemente do tamanho da carga de dados.

Considerando os outros dois sistemas, Cassandra e VoltDB, constatamos o melhor desempenho do primeiro em todas as situações de categorias e tamanhos. O VoltDB consegue obter resultados semelhantes ao Cassandra apenas nos experimentos da Categoria E, na qual predominam as operações de leitura de dados.

Outras conclusões que conseguimos extrair dos resultados é quanto à melhora de desempenho à medida em que as operações de escrita diminuem e aumentam as

operações de leitura de dados. Como esperado, todos os SGBD apresentam melhores desempenhos nesse sentido. Entretanto, nos SGBD Redis e Cassandra essa melhora é menos perceptível do que nos SGBD MongoDB e VoltDB. Comparando os resultados da Categoria A (mais lenta) com os da Categoria E (mais rápida), vemos que o Cassandra e o Redis apresentam melhoras apenas de 9% e 27%, respectivamente, enquanto no VoltDB e no MongoDB esses percentuais chegam a 59% e 94%, respectivamente.

Diante dos resultados e análises apresentados, podemos concluir que aplicações cuja interação com o banco de dados se assemelhe à distribuição de operações das Categorias A, B ou C deveriam dar preferência ao SGBD Redis. Já as aplicações cuja interação se aproxime mais às Categorias D e E deveriam optar pelo SGBD MongoDB.

Quando analisamos o comportamento dos sistemas NoSQL ao lidar com grandes volumes de dados, podemos verificar o excelente desempenho dos sistemas Redis e MongoDB em todas as categorias e tamanhos de carga de dados. Esse fato pode ser explicado, além de outros fatores, pelos seus modelos de dados, respectivamente chave-valor e documentos. Ambos são modelos flexíveis, cuja estrutura de dados é simplificada e, portanto, exige menor custo computacional. Pudemos confirmar, assim, o bom comportamento dos sistemas NoSQL ao lidar com grandes volumes de dados.

Em contrapartida, retomemos a definição de sistemas NewSQL dada por Pavlo e Aslett (2016), anteriormente apresentada no Capítulo 2. Nela os autores afirmam que os sistemas NewSQL são “SGBD relacionais modernos que buscam prover o desempenho escalável dos sistemas NoSQL”. Observando os resultados obtidos neste estudo experimental, constatamos que em nenhum dos critérios analisados o NewSQL VoltDB apresentou um desempenho satisfatório, chegando a apresentar tempos de execução dezenas de vezes piores do que o mais rápido dos sistemas NoSQL. Concluímos dessa forma que, em cenários de grandes volumes de dados e com os SGBD que foram comparados, não se confirmou a promessa dos sistemas NewSQL proverem um desempenho semelhante ao que encontramos nos sistemas NoSQL. Mesmo que os sistemas NewSQL apresentem as demais funcionalidades prometidas (modelo relacional, linguagem SQL, escalabilidade horizontal, suporte às propriedades ACID), o fraco desempenho frente aos sistemas NoSQL os tornam pouco competitivos e praticamente inviabiliza sua utilização por aplicações que necessitam trabalhar com grandes volumes de dados.

5.3 Considerações

Neste capítulo apresentamos inicialmente os resultados obtidos com a execução dos experimentos-piloto, realizados com o objetivo de validar as definições e a arquitetura do estudo experimental.

Os resultados coletados nos experimentos principais foram apresentados e pudemos analisar e rejeitar as hipóteses nulas que haviam sido definidas na formalização dos experimentos.

Por fim, realizamos análises sobre os resultados encontrados e extraímos conclusões acerca dos mesmos, a partir dos objetivos que haviam sido propostos para este trabalho.

6

CONCLUSÕES

Neste trabalho, foi realizado um estudo experimental comparativo que permitiu analisar o desempenho de alguns dos principais representantes das novas gerações de SGBD (sistemas NoSQL e NewSQL) em situações de grandes volumes de dados, de forma a indicar qual tipo de sistema dessas novas gerações melhor se adequam a cada um dos cenários que foram analisados.

Os SGBD avaliados no estudo foram: Redis (NoSQL da categoria chave-valor), MongoDB (NoSQL de documentos), Cassandra (NoSQL de família de colunas) e VoltDB (NewSQL da categoria nova arquitetura). Foram definidas cinco categorias de aplicações, com relação aos tipos de operações de banco de dados que são predominantes (dentre escrita e leitura de dados). Para cada uma dessas categorias foi definido um experimento, o qual foi executado em uma máquina rodando nos servidores do Centro de Informática da UFPE.

Os resultados de cada experimento foram avaliados, suas hipóteses nulas foram rejeitadas e conseguimos extrair conclusões acerca desses resultados, baseadas nas perguntas de pesquisa elencadas no início deste trabalho: (Q1) Como o desempenho dos SGBD NoSQL e NewSQL diferem entre si? (Q2) Como esses sistemas se comportam diante de situações com grandes volumes de dados? (Q3) A partir de certos requisitos de uma aplicação que necessite lidar com grandes volumes de dados, qual sistema/abordagem é a mais adequada? (Q4) Os sistemas NewSQL estão prontos para, de fato, competir com os NoSQL?

Verificamos que todas as perguntas de pesquisa elencadas foram respondidas por meio dos resultados obtidos nos experimentos realizados. Particularmente no que diz respeito à pergunta Q4, concluímos que os sistemas NewSQL ainda não estão prontos para competir com os sistemas NoSQL, dado o seu baixo desempenho ao lidar com grandes volumes de dados.

6.1 Principais Contribuições

Retomando o quadro resumo apresentado no Capítulo 3, contendo as principais características dos trabalhos relacionados, incluímos o presente trabalho, como pode ser visto no Quadro 3.

Quadro 3 − Resumo das principais características dos trabalhos relacionados

Trabalhos	**Tipo de comparação**	**Inclui sistemas NoSQL?**	**Inclui sistemas NewSQL?**	**Tipo de análise?**	**Houve medição de desempenho?**
Abramova, Bernardino e Furtado	Quantitativa	Sim	Não	Objetiva	Sim
Hajoui et al.	Quantitativa	Sim	Sim	Subjetiva	Não
Gurevich	Qualitativa e Quantitativa	Sim	Sim	Objetiva	Sim
Binani, Gutti e Upadhyay	Qualitativa	Sim	Sim	Objetiva	Não
Kabakus e Kara	Quantitativa	Sim	Não	Objetiva	Sim
Oliveira e Bernardino	Quantitativa	Não	Sim	Objetiva	Sim
Melo	**Quantitativa**	**Sim**	**Sim**	**Objetiva**	**Sim**

Fonte: O Autor (2018)

Dentre os principais diferenciais deste trabalho, é importante apontar que ele se destaca dos outros trabalhos de pesquisa comparativos por, além de formalizar seu estudo experimental, ter buscado subsidiar a escolha de um SGBD da nova geração para aplicações em diferentes contextos de interação com o banco de dados. Isso foi possível por meio da comparação de SGBD NoSQL e NewSQL em cenários diversos, com diferentes tipos de cargas de dados, e voltados para grandes volumes de dados.

Além disso, destacamos outras contribuições importantes deste trabalho:

- **Formalização do estudo experimental.** Essa formalização é essencial para viabilizar a possibilidade de reprodução deste trabalho por outros pesquisadores, de forma a validar seus resultados. Havendo consistência entre os resultados destas possíveis replicações e os obtidos em nossa pesquisa, isso reforçará sua validade e aumentará a confiança nos nossos resultados e nas conclusões extraídas destes.
- **Definição de uma arquitetura para a execução dos experimentos.** Para a execução dos experimentos, foi definida uma arquitetura de software utilizando o *framework Java Microbenchmark Harness*, que auxiliou na medição dos experimentos unitários.
- **Análise estatística dos resultados.** A aplicação de testes estatísticos de hipóteses nos resultados coletados a partir dos experimentos permitiu que pudéssemos rejeitar as hipóteses nulas formuladas no estudo experimental. Isso nos possibilitou concluir, com 99,9% de confiança, que existe diferença nos tempos de execução entre os SGBD NoSQL e NewSQL para grandes volumes de dados nos diferentes cenários que foram estudados.

É importante destacar também que a definição dos SGBD comparados foi feita a partir de métricas de mercado. Foram avaliados rankings sobre a relevância de SGBD no mercado, produzidos por fontes de grande importância, como o site *Stack Overflow* (uma das maiores comunidades de desenvolvedores de software) e a empresa *Gartner* (uma das principais empresas de pesquisa e consultoria do mundo), dentre outros.

6.2 Limitações e Ameaças à Validade

Uma limitação apresentada neste trabalho se dá em relação aos SGBD comparados no estudo experimental. Durante a pesquisa, foram estabelecidos critérios bem definidos para a escolha dos SGBD, dentre os quais ser uma solução *open-source*. Contudo, existem outros sistemas disponíveis no mercado (SGBD comerciais e com licença paga), os quais poderiam apresentar desempenhos superiores aos que foram comparados, mas que não foram escolhidos por não se adequarem aos requisitos estabelecidos neste trabalho.

Além disso, uma possível ameaça à validade deste trabalho seria a possibilidade de haver interações entre os experimentos unitários, ou seja, que a execução de um

experimento unitário influenciasse na execução de outro experimento unitário, distorcendo o resultado deste último. Essa ameaça foi mitigada pois, antes da execução de cada experimento unitário, são apagados todos os registros da base de dados do SGBD, fazendo com que todos os experimentos unitários sejam executados a partir do mesmo estado inicial.

6.3 Trabalhos Futuros

A seguir, apontamos alguns caminhos que podem ser explorados em trabalhos futuros:

- **Incluir outros SGBD NoSQL e NewSQL no estudo experimental.** Para nosso estudo experimental, foram escolhidos um SGBD de cada tipo a ser comparado (os NoSQL chave-valor, de documentos e família de colunas e o NewSQL de nova arquitetura). Um trabalho futuro poderia selecionar mais de um SGBD de cada tipo, possibilitando comparar o desempenho de sistemas de mesmo modelos de dados.
- **Executar os experimentos em um *cluster* de máquinas.** O foco dos experimentos realizados neste trabalho foi em avaliar o desempenho dos SGBD em uma única máquina. Como trabalho futuro, é relevante comparar os sistemas em um ambiente de *cluster* de máquinas, no qual seria possível, por exemplo, examinar o comportamento dos sistemas em cenários de escalabilidade horizontal.

REFERÊNCIAS

ABRAMOVA, Veronika; BERNARDINO, Jorge; FURTADO, Pedro. Which nosql database? A performance overview. Open Journal of Databases (OJDB), v. 1, n. 2, p. 17-24, 2014.

AGILDATA. AgilData Scalable Cluster for MySQL Whitepaper. 2016. Disponível em: <http://www.agildata.com/scalable-cluster-for-mysql-whitepaper/>. Acesso em: 18 nov. 2018.

ASLETT, Matthew. What we talk about when we talk about NewSQL. 2011. Disponível em: <https://blogs.the451group.com/information_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsql/>. Acesso em: 18 nov. 2018.

BINANI, Sneha; GUTTI, Ajinkya; UPADHYAY, Shivam. SQL vs. NoSQL vs. NewSQL-A Comparative Study. database, v. 6, n. 1, 2016.

BREWER, Eric A. Towards robust distributed systems. In: PODC. 2000.

CHEN, Min; MAO, Shiwen; LIU, Yunhao. Big data: A survey. Mobile networks and applications, v. 19, n. 2, p. 171-209, 2014.

CODD, Edgar F. A relational model of data for large shared data banks. Communications of the ACM, v. 13, n. 6, p. 377-387, 1970.

ELMASRI, Ramez; NAVATHE, Shamkant. Fundamentals of database systems. Addison-Wesley Publishing Company, 2010.

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

GUREVICH, Yuri. Comparative Survey of NoSQL/NewSQL DB Systems. 2015. Tese de Doutorado. The Open University.

HAJOUI, Omar et al. An Advanced Comparative Study of the Most Promising NoSQL and NewSQL Databases with a Multi-Criteria Analysis Method. Journal of Theoretical & Applied Information Technology, v. 81, n. 3, 2015.

HARRISON, Guy. Next Generation Databases: NoSQL, NewSQL, and Big Data. Apress, 2015.

JENKOV, Jakob. JMH - Java Microbenchmark Harness. 2015. Disponível em: <http://tutorials.jenkov.com/java-performance/jmh.html>. Acesso em: 18 nov. 2018.

JUDD, Charles M.; SMITH, Eliot R.; KIDDER, Louise H. Research Methods in Social Relations, Fort Worth: Holt, Rinehart and Winston. 1991.

JURISTO, Natalia; MORENO, Ana M. Basics of software engineering experimentation. Springer Science & Business Media, 2013.

KABAKUS, Abdullah Talha; KARA, Resul. A performance evaluation of in-memory databases. Journal of King Saud University-Computer and Information Sciences, v. 29, n. 4, p. 520-525, 2017.

MONTGOMERY, Douglas C.; RUNGER, George C.; CALADO, Verônica. Estatística Aplicada e Probabilidade para Engenheiros. Grupo Gen-LTC, 2000.

MORETTIN, Pedro Alberto; BUSSAB, Wilton Oliveira. Estatística Básica. Editora Saraiva, 2017.

NAYAK, Ameya; PORIYA, Anil; POOJARY, Dikshay. Type of NOSQL databases and its comparison with relational databases. International Journal of Applied Information Systems, v. 5, n. 4, p. 16-19, 2013.

OLIVEIRA, João; BERNARDINO, Jorge. NewSQL Databases-MemSQL and VoltDB Experimental Evaluation. In: KEOD. 2017. p. 276-281.

PAVLO, Andrew; ASLETT, Matthew. What's really new with NewSQL?. ACM Sigmod Record, v. 45, n. 2, p. 45-55, 2016.

RAMAKRISHNAN, Raghu; GEHRKE, Johannes. Database management systems. McGraw Hill, 2000.

SADALAGE, Pramod J.; FOWLER, Martin. NoSQL Essencial: Um guia conciso para o mundo emergente da persistência poliglota. Novatec Editora, 2013.

SILBERSCHATZ, Abraham et al. Database system concepts. New York: McGraw-Hill, 2010.

STONEBRAKER, Michael; CETINTEMEL, Ugur. "One size fits all": an idea whose time has come and gone. In: Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on. IEEE, 2005. p. 2-11.

STRAUCH, Christof; SITES, Ultra-Large Scale; KRIHA, Walter. NoSQL databases. Lecture Notes, Stuttgart Media University, v. 20, 2011.

VENKATESH, Prasanna; NIRMALA, S. NewSQL-The New Way to Handle Big Data. 2012. Disponível em: <http://opensourceforu.com/2012/01/newsql-handle-big-data/>. Acesso em: 18 nov. 2018.

WOHLIN, Claes et al. Experimentation in Software Engineering. Springer Science & Business Media, 2012.