UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

MATHEUS DORNELAS RODRIGUES

ANÁLISE SOBRE BASES DE DADOS PARA ARMAZENAMENTO E CONSULTA DE DADOS NÃO ESTRUTURADOS E SEMIESTRUTURADOS.

TRABALHO DE GRADUAÇÃO

RECIFE
2016

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

ANÁLISE SOBRE BASES DE DADOS PARA ARMAZENAMENTO E CONSULTA DE DADOS NÃO ESTRUTURADOS E SEMIESTRUTURADOS.

Trabalho apresentado como requisito parcial para obtenção do grau de bacharel em Ciência da Computação pelo Centro de Informática da Universidade Federal de Pernambuco.

Aluno: Matheus Dornelas Rodrigues

Orientador: Fernando da Fonseca de Souza

AGRADECIMENTOS

Primeiramente, eu gostaria de agradecer a meus pais e meus irmãos que me deram todo o apoio e suporte desde o momento da escolha do curso até o presente momento, em que o período da graduação está perto do fim.

Também agradeço a meus grandes amigos que me acompanham desde a época da escola: Diógenes Henrique, Givanildo Secundes, Guaraci Tavares, Helton Barbosa, Ronan Gomes e Rodrigo Viana. Mesmo seguindo caminhos diferentes não deixamos a amizade acabar.

Além disso, também quero lembrar dos amigos que fiz nessa etapa da minha vida e que sem dúvida me ajudaram a concluir a graduação:, Vinícius Folha, Victor Vernilli, Túlio Lages, Carlos Rafael, Marcelo Ferrão, Vinícius Lira, Igor Pinheiro, Germano Zaicaner, Felipe Farias, Mariana Macedo, Arthur Padilha, David Benko, Lucas Arruda e tantos outros que gostaria de citar.

Fui felizardo em ter escolhido o Centro de Informática para estudar, local onde a excelência está diretamente relacionada ao fato de que os professores conseguem inspirar os alunos a sempre explorarem novos caminhos. Agradeço ao professor Fernando Fonseca que me orientou durante o desenvolvimento desse trabalho e ao professor Francisco Airton que me orientou durante minha iniciação científica e me apresentou ao mundo acadêmico.

RESUMO

Este trabalho de graduação visa analisar e comparar o desempenho de sistemas de bases de dados relacionais e NoSQL (Not only SQL) para o armazenamento e consulta de dados semiestruturados e não estruturados.

Devido ao fato que esses dados não têm uma estrutura bem definida e por geralmente estarem relacionados a um grande volume, as bases de dados relacionais tradicionais vêm sendo preteridas e a utilização de bases de dados NoSQL é mais adequada em cenários desse tipo. Para justificar a opção por NoSQL é necessário uma análise das características desses dois tipos de sistemas de armazenamento. Além disso, devido às várias maneiras de como se pode manipular dados em sistemas NoSQL, é importante mostrar como as diferentes categorias de sistemas de banco de dados desse paradigma funcionam, pois alguns desses sistemas são melhores que outros em determinados cenários.

Uma situação que une dados não estruturados e a movimentação de uma grande massa de dados é na aquisição de informações dos usuários por partes de *sites*, aplicativos móveis ou outros dispositivos eletrônicos. Um estudo de caso será analisado e por meio desse estudo e de experimentos complementares será avaliada qual a melhor alternativa para o armazenamento remoto das atividades dos usuários de um aplicativo móvel.

Palavras Chave: NoSQL, Base de Dados Relacionais, Aplicativos Móveis, Armazenamento Remoto

ÍNDICE

ÍNDICE DE FIGURAS

1 INTRODUÇÃO

O problema de armazenar e organizar uma quantidade crescente de informação é enfrentado desde quando bibliotecas eram a maior fonte de conhecimento. Com o desenvolvimento da tecnologia, a informação também passou a ser representada na forma virtual, mas com o crescimento na produção de dados os mesmos problemas apareceram. De acordo com um infográfico feito pela CSC (*Computer Sciences Corporation*) a produção de dados em 2020 vai ser 44 vezes maior do que foi em 2009, o estudo está falando na produção de cerca de 35 zetabytes (10^{21}) de dados, onde 70% dessa quantia será gerada por pessoas físicas (CSC, 2010). Além disso, a maioria dos dados produzidos não segue uma estrutura bem definida sendo chamados de dados não estruturados ou semiestruturados (GANDOMI e HAIDER, 2015). A explosão de dados pode ser atribuída principalmente ao surgimento da Internet, mas especificamente a Web 2.0, quando qualquer pessoa pôde produzir conteúdo digital (O'REILLY, 2005).

1.1 Armazenamento e produção de dados não estruturados

Com o início da Web 2.0, foi intensificado o desenvolvimento de dispositivos eletrônicos móveis como: smartphones, câmeras digitais, sensores, *smartwatches*, entre outros. Enfrenta-se um cenário no qual a informação é produzida, captada e transmitida em alta velocidade, em grandes volumes e sem uma estrutura bem definida. Foi nesse contexto que surgiu o que chamaram de *Big Data*. Pode-se definir *Big Data* como um grande conjunto de dados com tipos distintos, sendo produzido em alta velocidade e que requer formas de processamento e armazenamento não triviais. O motivo pelo qual o fenômeno tem tomado importância é por causa do conhecimento que pode ser extraído a partir do processamento desses dados. Empresas de várias áreas tem investido na extração de conhecimento para tomar decisões estratégicas, descobrir detalhes sobre seus clientes e otimizar seus processos internos (CHEN e ZHANG, 2014).

Um dos fatores que caracterizam *Big Data* é a variedade dos dados, o que significa que os dados podem ser encontrados de maneira estruturada ou não. Segundo Cukier (2010), os dados estruturados, presentes em tabelas ou bancos de dados relacionais, constituem apenas 5% de todos os dados existentes. Os outros 95% são encontrados na forma não estruturada em vídeos, imagens, áudios, textos, além dos

tipos semiestruturados como por exemplo arquivo JSON e XML (GANDOMI e HAIDER, 2015).

Para que haja o processamento desses dados é preciso considerar o modo como estão armazenados. Este trabalho vai dar ênfase nesta etapa. Aplicações que utilizam um grande tráfego de dados requerem uma taxa de leitura e escrita muito alta, além de requerer disponibilidade e particionamento para que possam ser processados por sistemas distribuídos e em tempo real. Tais requisitos são comuns em aplicações de *Big Data*. Até pouco tempo atrás os bancos de dados relacionais eram utilizados para todos os tipos de aplicações, mas os fatores citados anteriormente fazem com que esse modelo de base de dados não seja o mais adequado. Alguns fatores como o esquema rígido como os dados são organizados, o *overhead* nas operações de escrita e leitura, bem como não oferecer escalabilidade horizontal não contribuem para o uso de bases de dados relacionais para desenvolvimento de soluções que manipulem um grande volume de dados. Em vez delas é mais comum o uso de bases de dados NoSQL (*Not only SQL*) (GUDIVADA , RAO e RAGHAVAN, 2014).

Mas com o passar dos anos vendo a necessidade do mercado e das aplicações de *BigData*, as empresas responsáveis pelos principais bancos de dados relacionais começaram a incrementar seus sistemas para fornecer novas funcionalidades que se assemelham as características comum aos sistemas do paradigma NoSQL. Essas funcionalidades procuram manter a eficiência quando se trabalha com dados não estruturados, fornecer a capacidade de funcionar como um sistemas distribuído, inclusive possibilitando realizar consultas sem utilizar a linguagem SQL.

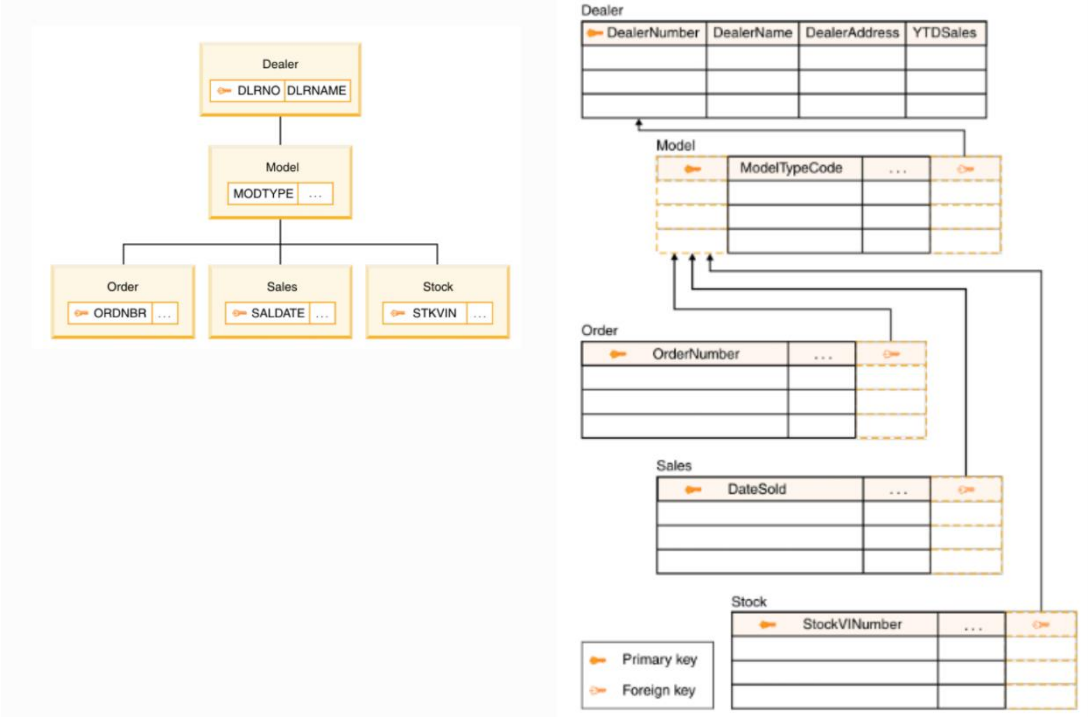
1.2 Sistemas de Gerenciamento de Bancos de Dados

A necessidade de guardar e organizar a informação acompanha o ser humano desde os tempos primordiais e com o desenvolvimento da tecnologia o surgimento de sistemas de armazenamento de dados foi natural. Já por volta da década de 60 existiu a necessidade de gerenciar estruturas de dados complexas. Assim surgiram os primeiros Sistemas de Gerenciamento de Banco de Dados (SGBD). O IBM IMS[1] (*Information Management System*) e o IDMS[2] são dois dos primeiros sistemas desenvolvidos. Esses sistemas seguem o modelo navegacional e organizam os dados de forma hierárquica, na

[1] https://www-01.ibm.com/software/data/ims/
[2] http://www.ca.com/us/products/ca-idms.html

qual a informação contida no nível inferior dependia da informação do nível superior usando uma estrutura semelhantes a árvores. A Figura 1 ilustra o modelo como os dados são organizados nesses sistemas em comparação com o modelo relacional.

Figura 1 - Modelo hierárquico e relacional para a mesma base de dados.



Fonte:www.ibm.com/support/knowledgecenter/SSEPH2_13.1.0/com.ibm.ims13.doc.apg/ims_comparehierandreldbs.htm

Em 1969, Edgar Codd um pesquisador da IBM publicou um artigo chamado "*A Relational Model of Data for Large Shared Data Banks*" (CODD, 1970) no qual propunha o que chamou de bancos de dados relacionais, que se diferenciavam dos bancos de dados hierárquicos porque utilizavam um modelo de organização em tabelas que contêm linhas e colunas ao invés de seguir uma rígida estrutura hierárquica ou métodos complexos para navegar entre os dados. Em seu trabalho, Codd define o que ficou conhecido como As doze regras de Codd (*Codd's Twelve Rules*), listando os requisitos que um banco de dados deve obedecer para ser considerado relacional (BROWN, 2002).

Segundo artigo publicado pela IBM em 2003 (IBM, 2003) Don Chamberlin, co-inventor do SQL, afirmou que a ideia de Edgar Frank Codd era que o relacionamento

dos dados deveria ocorrer em função dos seus valores e não apenas por referências de ponteiros. Essa forma de pensar simplificou o acesso à informação e mudou o modo de como as base de dados poderiam ser usadas. Manipular bancos de dados ficou mais próximo da linguagem natural e abstraiu detalhes nas operações sobre os dados fazendo com que pessoas que não possuíam grande conhecimento técnico pudessem operar sistemas de armazenamento.

A IBM foi pioneira em pesquisa e desenvolvimento dos SGBD relacionais. Durante o desenvolvimento do SGBD relacional chamado Sistema R[3] (*IBM System R*) foi criada uma linguagem para manipulação desse sistema batizada com o nome de SQL (*Structured Query Language*), que algum tempo depois iria virar a linguagem padrão para todas bases de dados relacionais (IBM, 2003). Com a grande aceitação do modelo relacional pelo mercado, foram lançados diversos outros sistemas como o IBM DB2[4], Oracle Database[5], Microsoft SQLServer[6], entre outros.

De acordo com o site DB-Engines.com[7], que constrói *rankings* mensais com as bases de dados de acordo com sua popularidade na Internet, os sistemas de gerenciamento de banco de dados relacionais são os mais utilizados. Na construção dos rankings alguns dos critérios analisados são: menções em fóruns técnicos, quantidade de buscas no Google[8], ofertas de empregos, presença no currículo de profissionais, entre outros. Pela Figura 2 percebe-se que dos dez SGBD mais populares até abril de 2016 sua maioria segue o modelo relacional.

Figura 2 - Ranking feito pelo website DB-Engines listando SGBD mais populares.

Rank			DBMS	Database Model	Score		
Apr 2016	Mar 2016	Apr 2015			Apr 2016	Mar 2016	Apr 2015
1.	1.	1.	Oracle	Relational DBMS	1467.53	-4.48	+21.40
2.	2.	2.	MySQL	Relational DBMS	1370.11	+22.39	+85.53
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1135.05	-1.45	-14.07
4.	4.	4.	MongoDB	Document store	312.44	+7.11	+33.85
5.	5.	5.	PostgreSQL	Relational DBMS	303.73	+4.10	+35.41
6.	6.	6.	DB2	Relational DBMS	184.08	-3.85	-13.56
7.	7.	7.	Microsoft Access	Relational DBMS	131.97	-3.06	-10.22
8.	8.	8.	Cassandra	Wide column store	129.67	-0.66	+24.78
9.	9.	↑ 10.	Redis	Key-value store	111.24	+5.02	+16.69
10.	10.	↓ 9.	SQLite	Relational DBMS	107.96	+2.19	+5.67

[3] https://en.wikipedia.org/wiki/IBM_System_R
[4] http://www.ibm.com/analytics/us/en/technology/db2/
[5] https://www.oracle.com/database/index.html
[6] https://www.microsoft.com/en-us/server-cloud/products/sql-server/
[7] http://db-engines.com/en/
[8] http://www.google.com

Fonte: http://db-engines.com/en/ranking

1.3 **Motivação**

Nos últimos anos devido ao fácil acesso à Internet, evolução e barateamento no preço dos sensores e evolução dos dispositivos móveis ocorreu uma explosão na produção de informação, e isso evidenciou alguns problemas. Consultar dados em bases de dados SQL que contêm muito conteúdo pode se tornar uma operação lenta. Para tentar solucionar esses problemas surgiram outros tipos de bancos de dados, sendo que um desses tipos seguiam o paradigma que foi chamado de NoSQL, pois não utiliza SQL para realizar manipulação dos dados (ABRAMOVA e BERNARDINO, 2013).

Por volta de 1998, Strozzi (1998) lançou o seu SGBD chamado Strozzi NoSQL, que na verdade era um bancos de dados relacional mas que não utilizava SQL para fazer consultas. O conceito de NoSQL que é mais difundido atualmente começou a ganhar força por volta dos anos 2000 com o desenvolvimento de SGBD não relacionais que manipulavam os dados de forma diferente dos sistemas relacionais. A premissa desses sistemas era suprir as dificuldades enfrentadas pelos sistemas de armazenamento mais comuns no mercado atual, como por exemplo: ser facilmente escalável para trabalhar como um sistemas distribuídos, ter alto desempenho na manipulação de grandes volumes de dados e alta disponibilidade (ABRAMOVA e BERNARDINO, 2013), (PARKER, POE e VRBSKY, 2013), (CATTELL, 2010).

Os SGBD DynamoDB[9] e Big Table[10] foram grandes contribuintes no desenvolvimento e afirmação dos bancos de dados NoSQL (GANDOMI e HAIDER, 2015), desenvolvidos pela Amazon[11] e pela Google[12], empresas líderes em oferecer softwares como serviços na nuvem. Esses sistemas começaram a ser utilizados e estudados por muitos usuários.

Com o passar dos anos, mais sistemas que não utilizam modelo relacional e a linguagem SQL foram lançados, e cada sistema explora a manipulação de dados de forma diferente, focando em garantir aspectos específicos como: consistência, escalabilidade ou disponibilidade. Baseado nisso é comum dividir os bancos de dados NoSQL em quatro categorias (CUKIER, 2010): orientado a documentos, orientado a

[9] https://aws.amazon.com/pt/dynamodb/
[10] https://cloud.google.com/bigtable/
[11] http://www.amazon.com
[12] http://www.google.com

colunas, chave-valor e orientado a grafos. Dependendo do tipo de aplicação na qual vão ser usados, algum tipo pode ter vantagem sobre outro.

A tendência na adoção do modelo NoSQL já é perceptível, o que pôde ser visto na Figura 2 que mostra a presença de 3 dessas bases de dados em meio às dez mais populares. Em uma pesquisa realizada pela empresa Gartner[13] que anualmente publica o Quadrante Mágico para sistemas de gerenciamento de base de dados operacionais (*Magic Quadrant for Operational Database Management Systems*), que pode ser visualizado na Figura 3, apontou a Microsoft como primeira colocada no quadrante de líderes no ano de 2015 (FEINBERG, 2015). Para especialistas isso se deve aos investimentos em tecnologia de armazenamento NoSQL para computação nas nuvens e a evolução do SQL Server, que passa a ser uma base de dados híbrida com capacidades de implementação tanto para aplicações na nuvem e localmente (JANAKIRAM, 2015). Além disso, dois sistemas NoSQL, MongoDB[14] e Redis Labs[15], também são classificadas como soluções líderes.

Figura 3 - Quadrante Mágico para sistemas de gerenciamento de base de dados operacionais

Fonte: https://www.gartner.com/doc/reprints?id=1-2PO8Z2O&ct=151013&st=sb

[13] http://www.gartner.com/technology/home.jsp
[14] https://www.mongodb.com/
[15] https://redislabs.com/

Além da Microsoft, a Oracle também incrementou seus sistemas de armazenamento Oracle Database[16] e o MySQL[17] com mecanismos para lidar com dados não estruturados de maneira mais eficiente. Isso aponta como as características antes comum apenas às tecnologias NoSQL estão ganhando espaço perante tecnologias tão consolidadas.

1.4 Objetivos

Este trabalho de graduação tem como objetivo geral a análise dos paradigmas de armazenamento SQL e NoSQL avaliando fatores fundamentais no emprego das tecnologias para sistemas que trabalham com dados não estruturados ou semiestruturados.

Como objetivos específicos deste trabalho, destacam-se as seguintes atividades:

- Explicação das características dos paradigmas SQL e NoSQL;
- Estudo sobre os sistemas de armazenamento SQL e NoSQL focando em como atuam com dados não estruturados ou semiestruturados. As afirmações serão feitas com base em outros trabalhos acadêmicos e na documentação das bases de dados; e
- O desenvolvimento de um estudo de caso, no qual serão avaliados três sistemas de gerenciamento de bancos de dados que armazenam o log de eventos dos usuários de um aplicativo móvel.

Em suma, o trabalho visa mostrar como os sistemas NoSQL lidam com dados não estruturados e como os sistemas que seguem o paradigma relacional se moldaram para fornecer mais funcionalidades para superar alguns fatores que são apontados como limitações quando os paradigmas de armazenamento são comparados.

1.5 Estrutura da monografia

Além deste capítulo, este trabalho está estruturado como segue.

O Capítulo 2 apresenta uma visão geral sobre os sistemas de gerenciamento de dados relacionais tradicionais e NoSQL; como surgiu a necessidade pela mudança de paradigma de armazenamento; e quais são as principais diferenças e características de cada um dos tipos. No Capítulo 3 será aprofundado o estudo em alguns sistemas

[16] https://www.oracle.com/database/index.html
[17] https://www.mysql.com

relacionais e não relacionais, mostrando como realizam a manipulação dos dados quando estes não tem estrutura bem estabelecida.

O Capítulo 4 descreverá a execução de um estudo de caso, partindo da descrição do ambiente de execução, tipo dos dados utilizados, sistemas de armazenamento utilizados e os dados coletados por meio de testes que verificam o desempenho de cada uma das soluções utilizadas. O Capítulo 5 apresenta as conclusões do trabalho, suas contribuições, suas limitações e possíveis trabalhos de extensão futura.

2 PARADIGMAS DE ARMAZENAMENTO RELACIONAL E NOSQL

Atualmente existem dois paradigmas principais para sistemas de armazenamento de dados, um deles é o modelo relacional que como foi falado anteriormente, existe desde a década de 70. O outro modelo é o não relacional ou NoSQL, que só veio a ser mencionado em publicações cientificas em 1998. Neste capítulo, será mostrado uma visão geral de características de ambos paradigmas, procurando mostrar como funcionam e suas diferenças.

Outro fator importante, é mostrar como esses conceitos de banco de dados se comportam diante de dados que não possuem uma estrutura bem definida. Para os sistemas que obedecem ao modelo relacional esses desafios são maiores, visto que seguem um modelo rígido composto por tabelas com colunas com tipos pré-definidos.

2.1 Conceitos do Modelo Relacional

O modelo relacional (ELMASRI e NAVATHE, 2011) organiza os dados em tabelas, e estas têm como objetivo representar as entidades no mundo real. As tabelas são compostas de colunas e linhas, onde cada coluna representa atributos ou campos da entidade, e cada campo possui um tipo previamente definido (texto, número inteiro, número real, booleano, entre outros). Os dados contidos em um conjunto de colunas e que se relacionam, formam uma linha da tabela. Dados inexistentes são representados pelo valor nulo. A Figura 3 apresenta uma tabela *Dealer* que teve o esquema apresentado na imagem anterior de preenchida.

Figura 4 - Tabela *Dealer* povoada.

Dealer

DealerNumber	DealerName	DealerAddress	YTDSales
1	Mariana	Rua A, nº23	2014-03-24 11:04:23
2	David	Rua B, nº43	2015-09-27 14:14:23
3	Túlio	Rua T, nº44	2015-07-09 11:04:23
4	Sarah	Rua R, nº234	2013-11-17 16:14:23
5	Vinícius	Rua E, nº112	2015-04-22 14:12:23
6	Leandro	Rua A, nº34	2015-12-01 15:24:43

Fonte: O Autor (2016).

Essa forma de armazenamento foi criada pensando em dados bem estruturados, então para garantir consistência das informações armazenadas várias regras são definidas por meio de restrições que podem ser estabelecidas no uso dos bancos de dados desse tipo (ELMASRI e NAVATHE, 2011). Abaixo é apresentada de forma resumida uma lista com as principais restrições:

1. Restrições de domínio - especificam que os tipos de dados definidos para cada coluna devem ser respeitados;
2. Restrições de chave - alguns atributos podem ser denominados de atributos chave e servem para identificar unicamente cada linha da tabela, no caso das chaves primárias; ou para representar a relação entre duas tabelas, no caso das chaves estrangeiras. Dentre outras limitações que podem ser aplicadas em atributos está a especificação se é permitido ter valor nulo ou se devem ter valor único;
3. Restrição de integridade referencial - afirma que um campo de uma tabela pode referenciar um registro de outra tabela através de uma chave primária existente. É utilizado o conceito de chave estrangeira, o qual representa o relacionamento e dependência entre duas tabelas.

Para manipulação de informação que está contida ou que será inserida na base de dados existem três operações básicas: inserção, atualização e remoção. A inserção adiciona novos dados nas tabelas, a atualização modifica valores de dados já existentes no banco de dados e a remoção retira valores do bancos de dados. Durante a inserção de dados todas as restrições descritas anteriormente correm o risco de serem violadas, mas

por padrão os sistemas de gerenciamento de banco de dados monitoram e rejeitam modificações assim que detectam que algum erro ocorreu. Da mesma forma, qualquer violação que ameace a integridade do banco de dados devido a alguma outra operação é rejeitada (ELMASRI e NAVATHE, 2011).

Garantir a integridade do banco de dados em sistemas nos quais vários usuários podem estar acessando e manipulando ao mesmo tempo pode ser um problema. Para isso foi criado o conceito de transações. Transação é um processo de software sendo executado sobre um banco de dados que pode conter várias instruções para leitura e modificação dos dados e ao final da execução o banco de dados deve ter um estado válido (ELMASRI e NAVATHE, 2011). Um conjunto de operações executadas por uma transação tem o caráter atômico pois todas devem ser executadas com êxito. Caso ocorra alguma falha, todas operações que foram executadas são revertidas.

É necessário que as transações possuam algumas propriedades que permitam controle de concorrência e ajudem na recuperação em caso do banco de dados em caso de falhas (GRAY e REUTER , 1992):

- Atomicidade - A execução de uma transação pode ser abortada a qualquer momento e as modificações revertidas caso haja algum erro;
- Consistência - Uma transação deve executar sem falhas. Desta forma garantindo que o banco não terá um estado inválido ao fim das modificações;
- Isolamento - Certifica que todos os recursos utilizados pela transação estão sendo acessados apenas por ela;
- Persistência - Uma vez finalizada a execução, as modificações são efetuadas e podem ser acessadas por outras transações ou operações realizadas pelo banco.

2.2 **Conceito do Modelo NoSQL**

O paradigma para armazenamento chamado de NoSQL surgiu como uma alternativa ao modelo relacional pelo de fato de manter um bom desempenho com a manipulação de bases de dados grandes, prover facilidade para implementação de sistemas distribuídos e manipular dados não-estruturados de maneira mais eficiente. Em Cattell (2010), as principais características dos sistemas NoSQL são resumidas em seis pontos:

1. Ser facilmente escalável horizontalmente, ou seja, operar de forma sincronizada em vários servidores;
2. Ter a capacidade de replicar e distribuir os dados entre servidores, característica ligada ao item anterior. Essa distribuição serve para garantir que o sistema não tenha um único ponto de falha e que a carga de leitura e escrita seja dividida de forma que aumente o desempenho do sistema;
3. Apresentar uma interface simples para manipulação da base de dados, geralmente abstraindo as instruções de inserção, atualização e remoção para comandos mais simples e amigáveis ao usuário. Muitos sistemas de banco de dados NoSQL oferecem uma interface REST (*REpresentational State Transfer*) que lida com os dados através de requisições HTTP[18] diretamente nos servidores;
4. Possuir modelo de controle de concorrência fraco quando comparado com as propriedades ACID (*Atomicity, Consistency, Isolation, Durability*) (ELMASRI e NAVATHE, 2011) das transações dos modelos relacionais. Também é comum ver o termo BASE (*Basically Available, Soft State, Eventual Consistent*) para representar as propriedades de concorrência (MCCREARY e KELLY, 2013):
    - Essencialmente disponível (*Basically Available*) - isso indica que o sistema permite um certo grau de inconsistência para que o sistema fique sempre disponível. Requisições podem ser recebidas e executadas parcialmente antes que outras transações prévias tenham sido executadas completamente;
    - Estado impreciso (*Soft State*) - essa limitação informa que o sistema como um todo nunca estará 100% consistente;
    - Eventualmente consistente (*Eventual Consistent*) - esse ponto afirma que eventualmente, quando todos os processos forem executados, o sistema estará em um estado consistente. Mas o sistema sempre está apto a receber novas transações e não verifica se as transações anteriores foram bem sucedidas, portanto o sistema por si só não sabe o seu estado.

    Em resumo, as transações são tratadas de maneira diferente. Com modo BASE, a prioridade é fazer com que o sistema esteja sempre disponível pra

18

receber novas transações. A inconsistência do que está armazenado por um curto período de tempo não é um problema;

5. Uso eficiente de índices e memória RAM para armazenar os dados;
6. Capacidade de armazenar dados com estruturas mutáveis, atributos podem ser removidos ou inseridos. Pode-se tomar como exemplo o armazenamento de objetos no formato JSON[19], que pode ter diferentes estruturas dependendo da informação que está contida.

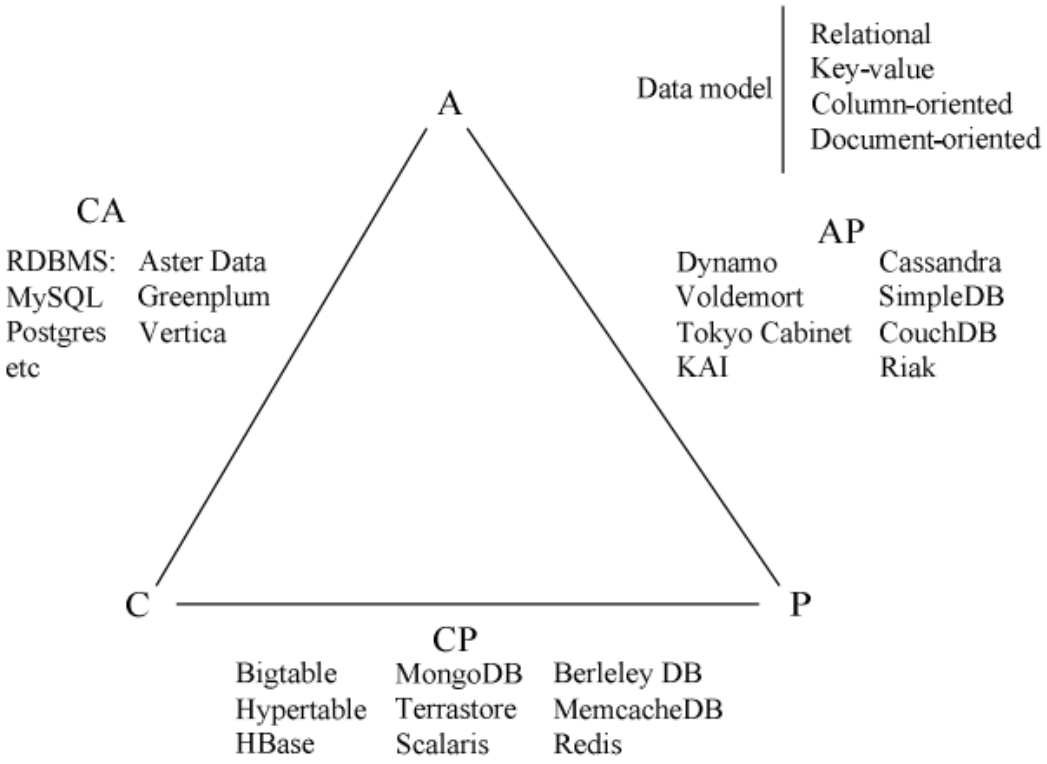
Um tipo de categorização aplicada às tecnologias NoSQL é em relação a um conceito criado por Eric Brewer[20] chamado de teorema CAP, o qual afirma que um sistema distribuído não consegue ter consistência, estar sempre disponível e ser distribuído ao mesmo tempo, apenas dois desses requisitos podem ser atendidos (MCCREARY e KELLY, 2013). O teorema ajuda a entender que quando os dados estão particionados deve-se considerar o que é mais importante para o sistema: consistência ou disponibilidade.

A Figura 4, mostra uma classificação de sistemas bancos de dados, incluindo relacionais, pelo teorema CAP:

- Consistência e Disponibilidade - o sistema não está preocupado com o fator de dividir os dados entre servidores. O foco é em replicar os dados entre servidores para garantir consistência e disponibilidade;
- Consistência e Particionamento - os dados estão consistentes entre todos os nós, e mantem a tolerância a partições tornando-se indisponível quando um nó é desligado;
- Disponibilidade e Particionamento - os nós mantem-se disponíveis mesmos quando não podem se comunicar uns com os outros e irão sincronizar uma vez que a comunicação for restaurada, mas não dá para garantir que os dados em todos os nós serão os mesmos.

[19] http://www.w3schools.com/json/
[20] https://en.wikipedia.org/wiki/Eric_Brewer_(scientist)

Figura 5 - Sistemas de Banco de dados classificados conforme o Teorema CAP.



Fonte: (CAI, HUANG, *et al.*, 2013)

Os sistemas de banco de dados NoSQL podem ser classificados em quatro categorias principais: chave-valor, orientado a colunas, orientado a documentos e orientado a grafos (CATTELL, 2010). De maneira geral, os sistemas de cada uma dessas categorias procuram oferecer uma solução para o armazenamento de grande volume de dados não-estruturados e de maneira distribuída.

O que caracteriza um sistema para ser classificado em uma das categorias citadas anteriormente é o padrão estrutural com o qual os dados são armazenados (MCCREARY, 2014). Sendo assim, cada sistema de banco de dados tem autonomia sobre outros aspectos importantes como o controle de concorrência, o mecanismo de divisão dos dados entre servidores, replicação dos dados para garantir a recuperação do sistema em caso de falha e o modo como realiza as consultas para acessar os dados armazenadas. Antes de optar por um sistema, é ideal que o desenvolver avalie todos esses fatores e tente achar o que mais se adequa às suas necessidades.

2.2.1 Chave-Valor

É o modelo mais simples de armazenamento NoSQL, pois sua estrutura é composta de duas partes: a chave, que é uma cadeia de caracteres; e o valor, que pode ser um dígito, uma cadeira de caracteres ou qualquer valor binário. Essas duas partes são associadas e funcionam de forma semelhante a um dicionário, no qual o campo chave é a palavra e o campo valor é o significado daquela palavra. A informação no campo chave deve ser única, mas o formato de seu conteúdo pode variar podendo ser o diretório para o arquivo, uma valor randômico, um rota de requisição para uma interface REST ou até uma consulta SQL (MCCREARY e KELLY, 2013).

O uso desse tipo de sistema é indicado quando se lida com dados sem uma estrutura bem definida, arquivos grandes ou informações que não se relacionam (PENCHIKALA, 2014). Geralmente a manipulação dos dados nos sistemas chave-valor é simples: não existe linguagem de consulta e toda a manipulação das base de dados ocorre em torno de três funções: uma para inserção, uma para remoção e outra para acesso. Isso mostra como essas bases de dados são simples quando comparadas aos sistemas que seguem o modelo relacional. Por não se preocuparem em representar o relacionamento entre as informações, torna-se muito mais fácil de implementar o sistema de forma distribuída.

De maneira geral o acesso à informação ocorre exclusivamente buscando pelo conteúdo do campo chave. Qualquer outro tipo de consulta mais sofisticada deve ser implementado na camada de aplicação (MCCREARY e KELLY, 2013); (HECHT e JABLONSKI, 2011). É comum que sistemas distintos implementem certos mecanismos de modos diferentes, como o controle de concorrência, interface de acesso, modo de armazenamento, método para divisão dos arquivos em sistemas distribuídos, entre outros.

2.2.2 Orientado a colunas

Esse modo de armazenamento utiliza os conceitos de linhas e colunas semelhantes ao modelo relacional, mas não existe o conceito de tabela. As colunas podem ser criadas à medida que os dados são inseridos e tipos não são pré-estabelecidos. A representação visual dos bancos de dados orientados a colunas é semelhante a uma tabela de banco de dados relacional. A maioria das implementações dos sistemas que seguem esse padrão estrutural foi baseado no sistema BigTable

(CHANG, DEAN, *et al.*, 2008) desenvolvido pela Google[21] no ano de 2004 e que funciona com o esquema chave-valor, só que aqui uma chave pode estar associada com um número arbitrário de valores em diferentes colunas. Esses valores agrupados formam uma linha.

O principal objetivo do BigTable, e consequentemente dos sistemas inspirados por ele, é ter um alto grau de escalabilidade possibilitando o particionamento horizontal (linhas) e vertical (colunas). De forma semelhante ao modelo estrutural anterior, o modelo orientado a colunas não se preocupa em representar o relacionamento entre as informações, mas é possível agrupar as colunas em grupos pré-definidos pelo usuário, tentando fazer com que valores que possam ter relacionamentos fiquem próximos quando armazenados. Para obter um modo mais eficiente de associar valores, deve ser implementado na camada de aplicação (HECHT e JABLONSKI, 2011; CATTELL, 2010).

O sistema Cassandra[22] introduziu um nível de agrupamento de colunas chamado super-colunas que tornou possível a manipulação de dados com estruturas mais complexas e expressivas (CATTELL, 2010).

2.2.3 Orientado a documento

Considerado o padrão de armazenamento que lida melhor com dados de estrutura complexa, pois consegue representar os relacionamentos entre os dados de forma eficiente, organiza a informação em uma estrutura chamada "documento" que não tem esquema definido e é composto por vários campos chave-valor, além de permite criar índices secundários. Os campos de valor podem possuir valores textuais, numéricos, booleanos, arquivos, entre outros e não seguem padrões de tipo ou tamanho. Esses sistemas agrupam os dados em um conceito chamado de coleção. Podem ser criadas várias coleções para agrupar informações que possam ter alguma relação entre si, mas isso é opcional, pois todas as informações podem estar agrupadas na mesma coleção (CATTELL, 2010).

Também é comum nos sistemas de banco de dados orientados a objeto haver um mecanismo que torna possível relacionar um documento a outro, geralmente isso ocorre fazendo referência a algum dos campos chave do documento que se quer relacionar,

[21] https://www.google.com
[22] http://cassandra.apache.org/

como uma chave estrangeira do modelo relacional, chamado de índice secundário. Outra característica bastante importante é o motor de buscas que permite realizar consultas complexas, geralmente a partir de objetos no formato JSON.

Como mostrado na Figura 1, o MongoDB é o sistema de armazenamento NoSQL mais popular atualmente, ficando na quarta colocação mesmo disputando com bancos de dados relacionais já consolidados e líderes no mercado de tecnologia. O principal objetivo desse sistema é preencher a lacuna entre o modelo chave-valor e os modelos relacionais tradicionais, unindo o melhor de ambas as partes (GU e ENG, 2015).

2.2.4 Orientado a Grafos

Como o próprio nome já indica, os bancos de dados dessa categoria organizam os dados em um modelo de grafo, introduzindo os conceitos de nós e arestas para representar a informação e os relacionamentos, respectivamente. As informações contidas nos nós são representadas em um conjunto de pares chave-valor e as aresta ligam os nós que possuem informações relacionadas. Bancos de dados baseados em grafos são indicados para casos de uso nos quais os dados estão muito relacionados, pois esses sistema conseguem ótimo desempenho. Representam relações de muitos-para-muitos (JAYATHILAKE, SOORIAARACHCHI, *et al.*, 2012).

A busca por dados nesses bancos pode ser feita por meio de dois modos. Um deles é a comparação de padrões, o qual tenta achar por um padrão de nós e arestas mais parecido com ele que está sendo buscado. O outro método é percorrendo o grafo a partir de um nó inicial escolhendo entre duas estratégias: busca por largura ou busca em profundidade (HECHT e JABLONSKI, 2011).

De maneira similar às outras categorias, os sistemas dessa categoria variam na implementação de certos mecanismo como controle de concorrência, particionamento e replicação. Os sistemas Neo4J[23] e RDF4J[24] (anteriormente chamado de Sesame) tratam a concorrência com *locks* e transações, enquanto o FlockDB[25] desenvolvido pelo Twitter[26], funciona com a estratégia de que a última atualização é a correta (*last-update-wins*). O Neo4J e FlockDB garantem apenas consistência parcial dos dados, pois

[23] http://neo4j.com/
[24] http://rdf4j.org/
[25] https://en.wikipedia.org/wiki/FlockDB
[26] https://twitter.com/

replicam seus dados em diferentes computadores para fornecer disponibilidade, enquanto no RDF4J não há replicação, existindo apenas um ponto de armazenamento, dessa forma assegurando consistência (HECHT e JABLONSKI, 2011).

2.3 SQL e NoSQL na manipulação de dados semiestruturados e não estruturados

Agrupar dados semanticamente similares é a uma prática comum independente do sistemas de armazenamento que está em uso. Mas, em alguns casos a informação pode ter características próprias que não são comuns a outras informações ou podem ser formadas por informações que não seguem nenhum de tipo de estrutura.

O método mais comum para representar dados semiestruturados é com objetos no formato JSON ou XML[27]. Esses formatos possuem uma sintaxe bem definida, mas possibilitam a inserção de novos campos além de permitir a adoção de tipos de dados diferentes para representar a mesma informação. Já dados não estruturados podem ser considerados como arquivos binários sem nenhum tipo de organização interna, dessa forma não podem ser representados por esquemas (PARKER, POE e VRBSKY, 2013). Como mostrado anteriormente, dados não estruturados estão presentes em vários tipos de aplicações e sua necessidade e utilidade são indiscutíveis. E isso colaborou com a popularização dos sistemas de armazenamento NoSQL.

É possível armazenar qualquer tipo de dados em sistemas de bancos de dados relacionais, mas muitas vezes, principalmente quando a base de dados tem tendência a aumentar drasticamente de tamanho, não é a melhor alternativa (JIANG , LUO, *et al.*, 2013).

Quando se tenta armazenar arquivos textuais que não possuem estrutura fixa em tabelas com colunas de tipos primitivos encontram-se dificuldades de manter a otimização do espaço, a simplicidade do esquema, a normalização do esquema e a facilidade de consulta à informação. Imagine-se que surja a necessidade de armazenar novas informações em uma entidade que já está presente no banco de dados, algumas alternativas são:

[27] http://www.w3schools.com/xml/xml_whatis.asp

- Alterar a tabela incluindo novos campos, e deixando o preenchimento opcional. Os registros antigos ou os novos que não possuírem as informações novas terá o campo preenchido com o valor NULL; ou
- Criar novas tabelas para as entidades com as novas modificações.

Já com os bancos de dados que seguem o paradigma NoSQL esse tipo de adaptação não é necessária, pois os sistemas de armazenamento não funcionam com o conceito de tabelas e colunas com tipos pré-definidos, a organização e valores dos dados são abstraídos em conceitos mais abrangentes como nos casos dos bancos chave-valor e orientados a documentos.

Em sistemas de banco de dados relacionais o armazenamento de dados multimídia (áudio, vídeo e imagens) é tradicionalmente feito em colunas do tipo BLOB. Esse tipo representa uma coleção de dados binários que são armazenados como uma única entidade, ou seja, numa coluna do tipo BLOB se pode armazenar qualquer informação. Mas sua utilização muitas vezes não é aconselhada, pois pode acarretar numa queda de desempenho, já que pode guardar arquivos tão grandes quanto o sistema de armazenamento permita.

Uma alternativa para lidar com dados não estruturados ou grandes em sistemas de armazenamento relacionais é armazenar uma referência para um diretório do sistema de arquivos. Dessa forma, é preciso apenas de uma coluna para armazenar a localização onde o arquivo se encontra, prevenindo que haja uma queda no desempenho de consulta e inserção. Mas adotar essa estratégia dificulta a manutenção, pois caso o arquivo que está sendo referenciado seja apagado ou mudado de lugar é necessário uma atualização manual na coluna que referencia o arquivo e em outras tabelas caso a coluna funcione como chave estrangeira.

Devido ao aumento da produção de dados não estruturados os bancos de dados relacionais viram a necessidade de adaptar seu funcionamento para dar suporte a esse tipo de informação. Aos poucos cada um dos bancos de dados relacionais incrementaram seus sistemas com funcionalidades que tornassem possíveis o manuseio de arquivos multimídia e textos semiestruturados.

Sistemas de bancos de dados consagrados como Oracle Database, SQL Server, PostGres e MySQL evoluíram e desenvolveram maneiras eficientes de lidar com dados não estruturados, além de integração com computação na nuvem tornando possível a implementação desses sistemas de maneira distribuída, escalável, com particionamento,

replicação e processamento de dados entres servidores, tentando se tornarem alternativas adequadas para aplicações de *BigData*.

2.4 Considerações Finais

Neste capítulo foram mostradas as características gerais dos sistemas SQL e NoSQL. A preferência por uma das abordagens depende da situação na qual serão aplicadas, pois cada modelo tem suas vantagens e desvantagens, não existindo situação ótima para todos os cenários. Por meio de uma abordagem mais tradicional, pode-se afirmar que os sistemas relacionais oferecem consistência em vez de ter um esquema mais maleável, fornecer particionamento e por vezes um ganho no desempenho. Já os sistemas não relacionais, não podem garantir consistência, oferecer disponibilidade e particionamento ao mesmo tempo.

Por ser uma parte fundamental na solução de problemas relacionados a *BigData*, os sistemas de armazenamento procuraram evoluir e superar suas limitações, principalmente os sistemas de gerenciamento de banco de dados relacionais, que como dito na seção anterior é preterido quando o problema envolve dados não estruturados e grandes volumes de dados.

No próximo capítulo será abordado com mais detalhes como quatro sistemas de armazenamento fazem para armazenar, organizar, particionar e distribuir seus dados. Serão focados dados semiestruturados ou não estruturados.

3 DADOS NÃO ESTRUTURADOS EM SISTEMAS DE BANCOS DE DADOS

Atualmente pode-se afirmar que o uso de sistemas de bancos de dados NoSQL é mais adequado para lidar com os desafios apresentados por problemas de BigData (JIANG , LUO, *et al.*, 2013); (LOMOTEY e DETERS, 2013). Mas sistemas de armazenamento relacionais procuram unir sua robustez à escalabilidade. Para conseguir manipular grandes conjuntos de maneira eficiente, esse modelo enfrenta alguns desafios (JIANG , LUO, *et al.*, 2013):

- O modelo relacional é inconveniente para representar dados não estruturados;
- A relação entre as informações é feita por chaves; e
- A informação sem estrutura bem definida pode acarretar o aparecimento de muitos campos NULL, degradando o desempenho do sistema.

Para manter os usuários que presam pela consistência das operações transacionais os sistemas relacionais se moldaram para se tornarem uma alternativa aos sistemas não relacionais. Nas seções a seguir será mostrado como alguns sistemas de bancos de dados fazem para lidar com grandes volumes de dados não estruturados de maneira eficiente. Partindo do armazenamento até as possibilidades de processamento.

A importância de fornecer um cenário propício ao processamento desses dados deve-se a quantidade de conhecimento específico que pode ser extraída aplicando algoritmos de processamento, inteligência artificial, entre outros. Será abordado como os sistemas de bancos de dados dão suporte, por exemplo, para o modelo de programação MapReduce (DEAN e GHEMAWAT, 2008). Tal modelo foi desenvolvido pela Google e surgiu com o propósito de processar grandes volumes de dados de forma distribuída em um agrupamento de computadores, comumente chamado *cluster*.

Nesse modelo de programação os usuários definem seu método de processamento por meio de funções *map* e *reduce* e o sistema automaticamente paraleliza a computação pelas máquinas do cluster. A função *map* recebe as entradas no formato chave-valor e a transforma em um conjunto de chaves intermediárias que são passadas para a função *reduce*, que processa os dados e entrega como resultado final um conjunto menor de valores, geralmente nenhum ou apenas um valor (DEAN e GHEMAWAT, 2008).

Os aspectos que serão analisados nas próximas são:

- Existência do suporte ao armazenamento de dados semi e não estruturados e como estes dados ficam organizados de maneira a facilitar a recuperação através de consultas;
- Capacidade de particionamento da base de dados para que o sistema funcione de maneira distribuída visando o ganho de desempenho e um mecanismo recuperação de falhas;
- Suporte ao *framework* Apache Hadoop, solução bastante empregada para o pós-processamento dos dados por fornecer um ambiente de código aberto, modular com diversos *frameworks* processamento.

3.1 Oracle Database

O Oracle Database é um dos sistemas de armazenamento relacional mais utilizados no mundo. Com a crescente produção de dados e o advento do *BigData* evoluiu sua tecnologia para garantir que seus usuários pudessem armazenar e processar seus dados de maneira eficiente de acordo a com demanda dos seus usuários. Com o passar dos anos foram inseridas novas funcionalidades para lidar com grandes arquivos tentando manter um bom desempenho, além de funcionar como um sistema distribuído fornecendo escalabilidade horizontal.

Para armazenar dados não estruturados é comum para à maioria dos sistemas de bancos de dados relacionais o uso do tipo BLOB, que funciona como um container de dados binários. Para mudar esse cenário e melhorar o desempenho a Oracle desenvolveu tecnologias específicas para aumentar a capacidade do sistema na manipulação de documentos XML (Oracle XML DB), arquivos multimídia (Oracle Multimedia), texto (Oracle Text) e informação geo espacial (ORACLE, 2014).

Para ser uma alternativa ao tipo BLOB, foi desenvolvido o tipo SecureFile que é um novo formato para armazenar grandes arquivos binários do sistema, maneira que a Oracle encontrou para resolver problemas com desempenho na manipulação de arquivos volumosos. Trabalhar com SecureFile garante que o processo de manipulação dos arquivos fará parte de uma transação, dessa forma garantindo a atomicidade e consistência para o momento da leitura (ORACLE, 2014).

Os fatores que fazem com que seja vantajoso trabalhar com SecureFile ao invés do sistema de arquivos padrão do sistema operacional são dois:

- Um mecanismo que evita arquivos duplicados sejam armazenados; e
- Um algoritmo que avalia o ganho de desempenho antes de realizar o processo de compressão, podendo realizá-lo ou não.

O Oracle XML DB é uma tecnologia que proporciona a manipulação de arquivos no formato XML de maneira nativa, está presente em todas as versões do sistema e traz total suporte para armazenamento e consulta para todos os padrões da linguagem XML. Essa foi a primeira plataforma a trazer uma abordagem híbrida, tornando o conteúdo em documentos XML acessíveis na linguagem SQL e vice-versa (DRAKE, 2013).

O Oracle Text utiliza SQL para armazenamento, indexação e análise de documentos de texto. Essa tecnologia permite análise linguística, e várias estratégias de buscas incluindo buscas por palavras chave, contexto, operações booleanas, casamento de padrões, XML, entre outros (FORD, 2013).

Para dar suporte a arquivos de vídeo, áudio e imagens, além do armazenamento tradicional no tipo BLOB, foi desenvolvido o Oracle Multimedia. Essa funcionalidade também permite que além dos arquivos externos ao sistema, também sejam armazenados arquivos no formato SecureFile, que aumenta drasticamente o desempenho e a capacidade de manipulação do conteúdo pelo banco de dados. O limite máximo de tamanho para os arquivos é de 128 TB (terabytes).

A tecnologia Oracle Multimedia dispõe de tipos de objetos especiais para cada tipo de arquivo: ORDAudio para aúdio, ORDDoc para qualquer tipo de arquivo, ORDImage e SI_StillImage para imagens, ORDVideo para vídeos. O Oracle Multimedia ainda possui funções capazes de extrair metadados e atributos dos arquivos (extensão, localização, nome, data da última atualização, entre outros), além de inserir novas informações que possam ser criadas pela aplicação que estará usando o banco de dados (ANNAMALAI, PALMER e MAVRIS, 2009).

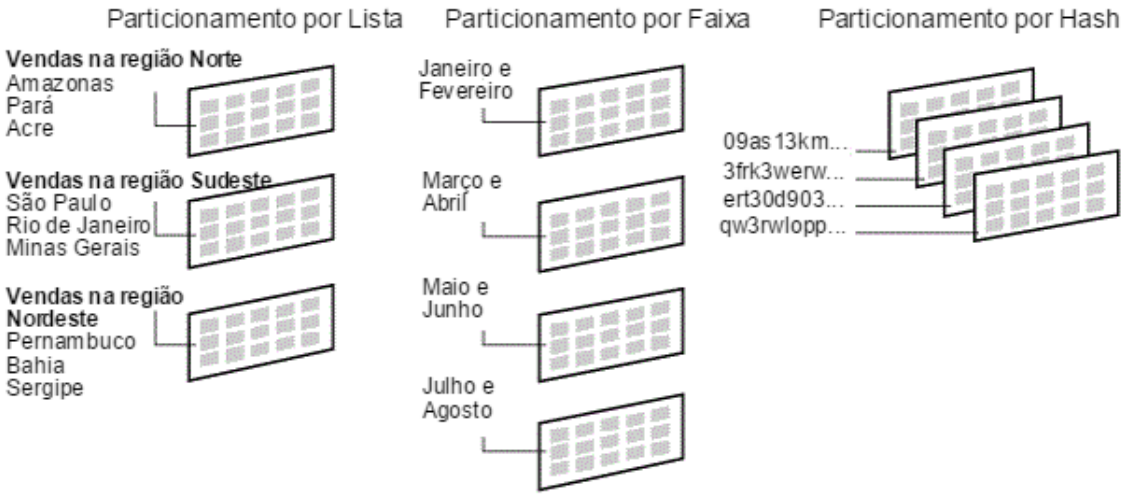
Para ajudar nos problemas enfrentados quando se tem que lidar com tabelas ou índices muitos grandes é possível a divisão em pedaços menores chamados de partições. Cada partição de uma tabela ou índice deve ter a mesma estrutura lógica, colunas com os mesmos nomes, tipos e restrições. Vale citar que o particionamento ocorre de maneira transparente para a camada de aplicação e não há a necessidade de modificação das consultas feitas em SQL.

Oracle oferece três estratégias básicas que controlam como os dados são colocados em partições individuais:

- Faixa – a tabela é particionada de acordo com uma faixa de valor de uma determinada coluna. Uma coluna que armazena datas é adequada para esse método de particionamento;
- *Hash* – o particionamento por *hash* mapeia os dados baseado no algoritmo de *hashing* aplicado ao valor de uma chave escolhida pelo usuário. Esse modo de particionamento faz com que as partições geralmente fiquem com uma quantidade de dados equilibrada; e
- Lista – permite que o desenvolvedor controle explicitamente como o dados serão mapeados para as partições definindo uma lista de valores discretos para um valor escolhido como chave.

A partir dessas três estratégias é possível dividir uma tabela no nível simples, aplicando uma das estratégias; ou de maneira composta mesclando duas estratégias. A Figura 5 ilustra como as tabelas podem ser divididas entre partições.

Figura 6 - Modos de particionamento no Oracle Database



Fonte: http://docs.oracle.com/database/121/VLDBG/GUID-1A6B6658-BCBE-438C-AF5A-C0AAEB08FE87.htm#VLDBG1059 (Imagem adaptada).

Para o processamento dos dados o *framework* Apache Hadoop[28] é uma ótima alternativa, mas o Oracle Database também oferece a opção para o programador desenvolver seu próprio algoritmo de processamento já que dispõe de estruturas como *Pipelined Table Functions* que podem ser utilizadas para construir um modelo MapReduce na própria base de dados (SHANK e DIJCKS, 2009).

[28] http://hadoop.apache.org/

Para utilizar o Apache Hadoop os dados contidos na base de dados devem ser passados para o sistema de arquivos HDFS (*Hadoop Data File System*) através do Apache Sqoop, uma ferramenta que foi desenvolvida para transporte de dados entre o Apache Hadoop e sistemas de armazenamento estruturados. A ferramenta importa dados do Oracle Database diretamente no sistema de arquivos do *framework*. Com os dados transferidos para lá ele fica totalmente disponível a todo o ecossistema que o Apache Hadoop pode oferecer. Os dados podem ser processados por meio de MapReduce, Spark, Mahout, entre outros.

3.2 MySQL

MySQL é o sistema de bancos de dados de código aberto mais popular do mundo e o líder entre os sistemas de armazenamento relacionais de código aberto para soluções Web e na nuvem. O MySQL é desenvolvido, distribuído e apoiado pela Oracle.

O sistema utiliza o InnoDB[29] como motor de armazenamento padrão desde de sua versão número 5.5. Ele funciona como um sistema de banco de dados relacional tradicional com transações ACID e restrições de integridade por meio de chaves primárias e estrangeiras. Através da sua interface de comunicação SQL o motor de armazenamento oferece apenas suporte nativo a JSON e os demais tipos de dados não estruturados são armazenados como BLOB.

Documentos JSON são inseridos em colunas JSON como textos (tipo *string*) e no momento da recuperação eles são convertidos em um formato interno que permite rápido acesso ao campos do documento. Quando é solicitada a leitura de um documento JSON este documento não precisa ser convertido para texto, a informação é lida na forma binária acessando apenas o valor que foi solicitado.

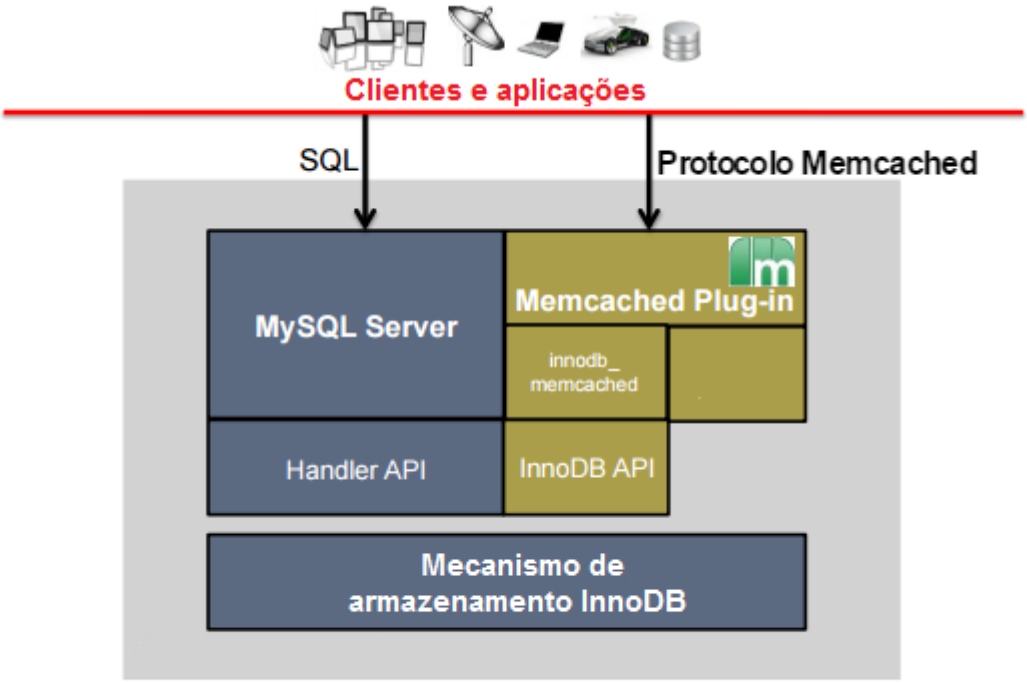
Para os demais arquivos binários é utilizado o tipo BLOB que possui quatro variações de acordo com o tamanho máximo de bytes que podem possuir: TINYBLOB, BLOB, MEDIUMBLOB e LONGBLOB. O formato BLOB é semelhante ao formato VARBINARY, mas a diferença é que um BLOB pode ter o tamanho que o usuário desejar enquanto o VARBINARY é limitado a 255 *bytes*.

[29] https://pt.wikipedia.org/wiki/InnoDB

Para conseguir uma ingestão de grande volume de dados com grande rapidez a Oracle desenvolveu interfaces NoSQL que se comunicam diretamente para o InnoDB através de *plugin* do Memcached evitando a camada SQL. Com o uso desse *plugin* é possível utilizar operações simples escrevendo diretamente nas tabelas (disco rígido) evitando o overhead relacionado com a camada SQL. Mas ainda é possível o acesso aos dados através de consultas escritas na linguagem SQL. Dessa forma dados podem ser escritos no formato chave-valor em tabelas e com um desempenho 9 vezes melhor que seria por meio da inserção tradicional. Mesmo sem utilizar operações SQL o sistema ainda trabalha com transações e garante as propriedades ACID (ORACLE, 2015).

A Figura 6 mostra a arquitetura do banco de dados MySQL quando utilizado com o *plugin* Memcached.

Figura 7 - Arquitetura do MySQL com o plugin Memcached



Fonte: (ORACLE, 2015).

O MySQL não fornece escalabilidade horizontal dessa forma não é possível dividir uma base de dados por de vários servidores. O mecanismo de particionamento do sistema funciona apenas para separar os dados em diferentes diretórios em uma mesma máquina. As regras de particionamento são semelhantes às do Oracle Database,

podendo ser modular, por faixa ou lista de valores, ou por funções que calculam um valor *hash* sobre a chave de particionamento. Dependendo do método de particionamento que for escolhido, podem ser utilizados como chave de particionamento: uma coluna, o retorno de uma função que utiliza uma ou mais colunas, ou um conjunto de valores de uma ou mais colunas.

Para o processamento dos dados o Apache Hadoop é uma ótima alternativa, já que o MySQL não oferece suas próprias ferramentas para processamento de grandes volumes de dados. Assim como no Oracle Database é necessário o auxílio do Apache Sqoop para fazer o transporte dos dados do MySQL para o sistema de arquivos do *framework* (HDFS).

3.3 MongoDB

MongoDB é o sistema de bancos de dados não relacional mais popular segundo o ranking feito pelo site DBEngine.com, mostrado na Figura 2. Esse sistema organiza os dados no modelo de documento, dá suporte a arquivos não estruturados de maneira nativa e não requer nenhum tipo de migração ou adaptação para armazenar a informação de maneira mais otimizada. Um documento é um conjunto de pares chave-valor. Nesses campos qualquer informação pode ser armazenada facilmente, e qualquer modificação no esquema do banco de dados é desnecessária já que os documentos não se importam em como a informação está organizada.

No MongoDB não há tabelas ou esquema, em vez disso, os documentos podem ser agrupados em coleções que podem ser comparadas com as tabelas. Não há necessidade de realizar operações de *join.* O relacionamento entre dados no banco de dados pode ser feito de duas formas. A primeira maneira é encadeando documentos, colocando um dentro do outro. Isso pode ser usado para representar relacionamento um-para-um e um-para-muitos. A segunda opção é armazenar uma referência para o outro documento, de maneira semelhante a uma chave estrangeira do paradigma relacional (PARKER, POE e VRBSKY, 2013).

Os documentos no MongoDB são codificados em objetos semelhantes a JSON, chamados de BSON[30], o que faz com que o armazenamento seja leve e rápido. Esses objetos dão suporte a booleanos, inteiros, *float*, datas, *strings* e outros tipos de dados

[30] http://bsonspec.org/

binários com um máximo de 16MB. Para arquivos com tamanhos maiores é utilizada a especificação GridFS[31].

Em vez de armazenar arquivos em um único documento, a especificação GridFS divide o arquivo em partes e guarda cada parte como um arquivo diferente, indicado para armazenar áudio e vídeos (CATTELL, 2010). Por padrão, cada pedaço do arquivo tem o tamanho de 255kB, podendo ser configurado, com a exceção do último pedaço que só tem o tamanho necessário. O GridFS utiliza duas coleções para armazenar arquivos: em uma ficam os dados e em outra apenas informações sobre os dados (*metadata*).

Para ter um ganho de desempenho o MongoDB mapeia na memória os dados armazenados recentemente. O sistema utiliza um sistema de indexação análogo a dos sistemas relacionais: cada documentos é identificado por um identificador único atribuído e indexado automaticamente. Mas, além da indexação desse identificador único, podem ser criados índices para outros campos à escolha do desenvolvedor. Isso é chamado de índice composto (ABRAMOVA e BERNARDINO, 2013).

O MongoDB é escalável horizontalmente, ou seja para aumentar sua capacidade ele divide sua carga de trabalho por diversas máquinas. Para isso realiza o particionamento automático (*autosharding*) no qual uma base de dados é dividida entre diversos servidores independentes. É dessa forma que o sistema lida com demanda crescente de dados e operações de leitura e escrita. A divisão da base de dados ocorre no nível das coleções, e para que uma coleção seja dividida é preciso escolher uma chave de particionamento. De maneira semelhante ao Oracle Database, essa chave é algum campo utilizado como índice ou um índice composto presente em todos os documentos.

Os valores chaves de particionamento são divididos em pedaços e distribuídos nos *shards*. A divisão da chave pode ser feita por meio de duas formas por faixa ou *hash*. Esses dois métodos funcionam de maneira semelhante aos métodos de mesmo nome do sistema Oracle Database.

Comparando o desempenho dos dois métodos dá pra observar que a divisão por faixa fornece um ótimo desempenho em consultas realizadas com base nas chaves, mas pode ocasionar um desbalanceamento na quantidade de dados em cada *shard*. Já no método *hash* a distribuição de dados nos *shards* é feita de maneira aleatória, o que

[31] https://docs.mongodb.com/manual/core/gridfs/

dificulta uma sobrecarga em algum dos *shards* mas no pior caso uma consulta precisará verificar todos os *shards* para retornar o resultado.

Para complementar a indexação escalável de documentos do MongoDB, o processamento também pode ser feito de maneira escalável e distribuída utilizando o modelo de programação MapReduce. Esse modelo de programação e sistemas de banco de dados NoSQL são facilmente relacionados devido às grandes quantidades de dados que são produzidas atualmente (DEDE, GOVINDARAJU, *et al.*, 2013).

O MongoDB possui de forma nativa sua própria implementação do MapReduce, mas recentemente alguns sistemas NoSQL forneceram extensões que permitem usuários utilizar a implementação MapReduce do framework Apache Hadoop em suas bases de dados (DEDE, GOVINDARAJU, *et al.*, 2013).

3.4 ElasticSearch

O ElasticSearch[32], diferentemente dos outros sistemas vistos até agora, é na sua essência um motor de buscas, mas que pode ser utilizado como ferramenta de armazenamento NoSQL orientada a documentos. Assim como a maioria dos outros sistemas de bancos de dados NoSQL, o sistema foi modelado para ser distribuído, escalável, além de realizar buscas em tempo real (KONONENKO, BAYSAL, *et al.*, 2014). A capacidade de busca e armazenamento é devida ao sistema ser construído sobre a biblioteca Lucene[33] (VANDIKAS, 2014).

Lucene é uma das bibliotecas mais avançadas da atualidade quando se fala em recuperação de informação: ela fornece às aplicações um motor de busca de alto desempenho. O ElasticSearch utiliza as funcionalidades da biblioteca tornando o processo de busca e indexação de textos em operações simples por meio de uma RESTful API (GORMLEY e TONG, 2015).

Assim como no MongoDB os documentos são as principais entidades no ElasticSearch, e da mesma maneira que nos sistemas NoSQL consistem de conjunto de campos chave-valor seguindo o formato JSON. Mas uma limitação desse motor de busca é o fato de só trabalhar com documentos puramente textuais, não dando suporte para outros tipos de arquivos.

[32] https://www.elastic.co/
[33] https://lucene.apache.org/

O que faz com que o ElasticSearch se diferencie das outras bases de dados que também têm a capacidade de manipular documentos textuais é sua forma de indexá-los na base de dados. O sistema pode indexar todo conteúdo de cada documento de forma a torná-los buscáveis. Além de buscar, é possível ordenar e filtrar documentos. Essa maneira diferente em como os dados são pensados é o que torna possível que buscas complexas sejam realizadas (GORMLEY e TONG, 2015).

Para explorar todo o potencial da recuperação de informação fornecida pelo Lucene, o ElasticSearch possui várias possibilidades de consultas (GORMLEY e TONG, 2015):

- Busca estruturada - consultas realizadas em campos com dados precisos, geralmente números, como por exemplo, datas. O resultado dessas buscas é sim ou não, retorna resultado ou não;
- Busca em texto - nesse tipo de busca dois fatores são avaliados:
    - Relevância – a habilidade de classificar resultados pela sua relevância ao trecho de texto dado como entrada. A relevância pode ser calculada usando TF/IDF, que é a frequência do termo no documento dividido pela quantidade de vezes que o termo aparece em todos os documentos. Ainda podem ser utilizados algoritmos que consideram geolocalização, similaridade *fuzzy*, entre outros;
    - Análise - o processo de quebrar e converter um bloco de texto em pedaços para criar um índice invertido e possibilitar buscas por índice invertido;
- Busca em multi-campos - consultas que combinam dois ou mais campos;
- Busca por correspondência aproximada - também chamada de busca por frase, ao invés de considerar os documentos como um "saco de palavras" e apenas verificar se o trecho dado como entrada está contida ali, esse tipo de busca verifica se todos os termos da entrada estão contidos e se aparecem na mesma ordem; e
- Busca por correspondência parcial - busca por trecho parcial de palavra.

Os documentos são organizados em *types* e armazenados em estruturas chamadas *index.* Não confundir com o verbo indexar: neste caso é um substantivo que representa um conceito semelhante a uma base de dados. Cada *type* possui seu próprio

esquema, chamado de *mapping*, o qual define como o documento está estruturado assim como as colunas de uma tabela. O *mapping* também tem a função de informar ao ElasticSearch como os dados em cada documento devem ser indexados.

Quando um documento possui uma estrutura diferente e é inserido em um *type* já existente, este tem seu *mapping* atualizado. Antes de indexar um documento o ElasticSearch faz uma análise da estrutura do objeto, é assim que os *mappings* são definidos, além de serem configuráveis de acordo com a vontade do programador (KUC e ROGOZINSKI, 2013). A Quadro 1 mostra uma comparação entre as nomenclaturas de estruturas com papéis semelhantes entres as bases de dados abordadas.

Quadro 1 - Comparação da nomenclatura de estrutura entre os sistemas

ElasticSearch	**MongoDB**	**SQL**
Index	Base de dados	Base de dados
Shard	Shard	Partição
Mapping/Type	Coleção	Coelção
Campo chave	Campo chave	Coluna
Objeto JSON	Objeto JSON	Tuplas

Fonte:

Para manipular grandes volumes de dados o ElasticSearch utiliza uma abordagem de particionamento bastante semelhante com o MongoDB e o Oracle Database, que é a divisão de um index em múltiplos *shards*. Dessa forma podem ser alocados em diversas máquinas. A manutenção desses *shards* é transparente para os usuários, pois todas as operações feitas na camada de aplicação abstraem como o sistema de armazenamento está estruturado. Cada documento é armazenado em apenas um *shard* e o mecanismo de particionamento dos dados entre os nós se baseia no identificado único (ID) de cada documentos. A documentação indica que a criação de campo com o ID fique a cargo do próprio sistema, pois ele irá fazer de uma maneira a garantir o melhor desempenho nas buscas.

O ElasticSearch pode ser utilizado de maneira combinada com outros dois sistemas: o Logstash[34] e o Kibana[35]. O Logstash tem como principal função recolher, agregar, processar e uniformizar dados provenientes de sistemas distintos para análise. O Kibana é uma ferramenta para visualização de dados do ElasticSearch. As três

[34] https://www.elastic.co/products/logstash
[35] https://www.elastic.co/products/kibana

ferramentas que formam a pilha ELK funcionam da seguinte forma: o Logstash coleta dados a partir dos logs gerados pelo ElasticSearch ou por sistemas que interagem com o motor de busca, então ocorre um processamento desses arquivos para indexá-los no ElasticSearch. A partir daí os dados estão disponíveis pra criação de gráficos em tempo real no Kibana e permitem ao usuário monitorar seus sistemas em tempo real.

Para otimizar o processamento em tempo real por meio do *framework* Apache Hadoop foi desenvolvido o ElasticSearch para Hadoop (*ElasticSearch for Hadoop*) (SHUKLA, 2015). Um componente que funciona como um conector de mão dupla entre o *framework* e o motor de buscas, de forma que cada um possa utilizar aquilo que o outro oferece de melhor. Esse componente permite o tráfego de dados entre as duas partes, além de permitir que o ElasticSearch realize busca em tempo real nos arquivos contidos no Hadoop.

3.5 Análise dos bancos de dados

O Quadro 2 mostra um resumo dos pontos que foram abordados nas seções anteriores para quatro sistemas de armazenamento de banco de dados. Os pontos são o suporte a dados semiestruturados, suporte a dados não estruturados, a capacidade de funcionar como sistema distribuído de maneira nativamente e a possibilidade de dar suporte ao framework Apache Hadoop para realizar o pós processamento.

Quadro 2 - Análise dos sistemas de armazenamento de dados.

	Arquivos semiestruturados (XML, JSON).	**Arquivos não estruturados.**	**Escalabilidade horizontal**	**Particionamento**	**Processamento**
Oracle Database	Oracle XML DB, Oracle Text	BLOB, SecureFiles, Oracle Multimedia	Sim	Sim	Apache Hadoop, *Pipelined Table Functions*
MySQL	JSON, XML	BLOB	Não	Sim	Apache Hadoop
MongoDB	JSON(BSON)	BSON, GridFS	Sim	Sim	Apache Hadoop, MongoDB MapReduce
ElasticSearch	JSON	Não	Sim	Sim	Pilha ELK, ElasticSearch para Hadoop

Fonte: O Autor (2016)

3.6 Considerações finais

Nas seções anteriores foram analisados diversos aspectos de como quatro sistemas de gerenciamento de banco de dados lidam com dados não estruturados. É importante frisar que os aspectos abordados se relacionam, pois além de fornecer a capacidade de armazenar é essencial faze-lo de maneira eficiente, preocupando-se com a organização interna e em ser capaz de fornecer mecanismos que garantam o bom funcionamento para diversas situações, desde consultas simples a métodos de pós-processamento.

Dar suporte de maneira nativa aos tipos de dados semiestruturados XML e JSON é aparentemente comum, pois além desses tipos possuírem uma sintaxe bem definida, elas são puramente textuais. Somando o fato do surgimento da Web 2.0 e que boa parte das informações na Internet é trafegads utilizando esses dois formatos de arquivos, evidencia-se ainda mais a importância da manipulação desses formatos.

Um desafio é maior quando o assunto envolve arquivos não estruturados, arquivos binários sem nenhuma estrutura interna e que podem ser de diversos tipos, arquivos de texto, de áudio, de vídeo, imagens, entre outros. Além disso, outro desafio é lidar com o tamanho desses arquivos, que no caso de vídeos, por exemplo, podem facilmente conter centenas de megabytes.

Os sistemas de banco dados que seguem o modelo relacional oferecem a opção de armazenar esses arquivos em colunas do tipo BLOB que são *containers* de bits. Mas o desempenho em quando se lida com colunas desse tipo nem sempre é satisfatório. Uma alternativa criada pelo Oracle Database e pelo MongoDB foi criar novas alternativas, SecureFile e o GridFS respectivamente.

Devido à importância já citada anteriormente dos problemas relacionadoss a *BigData*, um fator como a capacidade de funcionar como um sistema distribuído para aumentar a disponibilidade, dividir a carga de trabalho e fornecer mecanismo de recuperação é essencial. Todos, com a exceção do MySQL, fornecem mecanismos para escalar horizontalmente sua base de dados.

O último aspecto que evidencia a importância de extrair conhecimento dos volumes de dados que são produzidos atualmente é o fato de todos os sistemas abordados fornecerem módulos, conectores ou algum tipo de ferramenta que auxilie na integração com o *framework* Apache Hadoop.

No próximo capítulo será proposto um estudo de caso simples, no qual será avaliada a velocidade de inserção e de consulta nos quatros sistemas abordados neste capítulo, quando estes funcionam como um repositório remoto de documentos JSON que contém as ações dos usuários ao longo da utilização de um aplicativo móvel. Para realização dos experimentos será utilizado um ambiente com configurações simples, mas que pode ser expandido para dimensões bem maiores do que a simulada.

4 ESTUDO DE CASO

Atualmente é comum que aplicativos móveis, serviços de software online (SaaS) e redes sociais capturem e guardem informações sobre as ações dos seus usuários com o objetivo de melhorar a experiência desses usuários nas próximas utilizações. A aplicação de técnicas de mineração de dados é bastante utilizada para encontrar padrões de comportamento e extrair informações úteis. Realizar mineração de dados para extração de conhecimento em dados provenientes de grandes redes sociais e demais *softwares*, abre espaço para que pesquisadores e desenvolvedores testem seus algoritmos e cheguem a novas conclusões que não seriam descobertas de maneira trivial (ZHANG, KREITZ, *et al.*, 2013, MARKOVIKJ, GIEVSKA, *et al.*, 2012).

Oferecer um serviço personalizado ao cliente não é privilégio das grandes empresas que contam com milhões de usuários que movimentam milhares de gigabytes de informação diariamente. Devido à grande quantidade de sistemas de código aberto e o baixo custo para locação de infraestrutura para computação na nuvem faz com que empresas que ainda estão no início se desenvolvam sobre uma base sólida e que ofereça capacidade de se expandir conforme as dimensões dos seus projetos cresçam.

É comum que sistemas registrem informações sobre seu funcionamento em um sistema em arquivos de *log*, esses arquivos podem armazenar todos os eventos relacionados ao funcionamento do software para análise posterior (ANDREWS , 1998). Esses arquivos são muitos utilizados para a descoberta de erros, ajudando os programadores a encontrarem problemas, no estudo de caso proposto neste trabalho também serão armazenadas informações que refletem sobre as ações realizadas pelos usuários durante o uso do aplicativo. O objetivo é saber quais botões foram acionados, qual tela foi aberta, qual ação foi feita em determinada tela, entre outros.

O estudo de caso quer definir qual o melhor banco de dados para armazenar essas informações e fornecer acesso eficiente aos dados, tanto para consultas quanto para outros tipos de pós-processamento. Sendo assim, dois fatores principais serão analisados: a capacidade de lidar com requisições de inserção e a rapidez na consulta dos dados após o armazenamento. O ambiente de teste e a descrição como os testes serão realizados estão descritos nas próximas seções.

4.1 Ambiente de simulação

Simular um cenário real com vários dispositivos enviando requisições para inserção e consulta é inviável para as dimensões desse trabalho. Além dos custos envolvidos para garantir a aquisição ou locação de todos os componentes físicos necessários ainda teria que ser considerada a dificuldade em configurar e controlar os ambientes de testes para garantir igualdade de condições para todos os cenários que serão testados.

Serão preparados três cenários para simulação com sistemas de banco de dados distintos: MongoDB, MySQL e ElasticSearch. Apesar do MongoDB e ElasticSearch serem considerados sistemas da mesma categoria, banco de dados NoSQL orientado a documentos, o ElasticSearch tem como principal características ser um motor de busca.

Os ambientes de simulação desenvolvidos neste trabalho são formados por três computadores, como é mostrado na Figura 7. As máquinas foram hospedadas no serviço Amazon EC2 e possuem as mesmas configurações: processador Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz, 15GB de memória RAM, unidade de armazenamento SDD e sistema operacional Ubuntu 14.04. As três máquinas foram configuradas para estar na mesma sub-rede de modo que possam se comunicar sem interferência externa.

Um computador terá o papel de emissor de requisições, algo equivalente ao dispositivo móvel. Outro computador será um servidor que receberá as requisições e repassará ao terceiro computador, o qual terá um dos sistemas de armazenamento instalados.

Figura 8 - Organização dos computadores

Fonte: O Autor (2016).

Para garantir igualdade das situações os sistemas de armazenamento funcionarão de maneira *standalone* e nas configurações padrão. Abordar esses sistemas e suas capacidades de funcionar como sistemas distribuídos abre um leque de configurações e

topologias nas quais os computadores estão organizados que dificultam garantir a igualdade de condições para avaliar de maneira justa. O exemplo mais claro é o fato de os sistemas NoSQL fornecerem mecanismos para se escalar horizontalmente de maneira nativa, enquanto o MySQL não o faz.

4.2 Configuração dos testes

Para avaliar a capacidade de inserção e de consulta de dados, os testes foram divididos em duas etapas. Na primeira etapa será avaliada a capacidade de inserir os dados vindos nas requisições. Serão enviados três (3) lotes de requisições com informações para serem inseridas no banco de dados. Os lotes conterão 1.000, 10.000 e 100.000 requisições respectivamente e será enviado um de cada vez. A partir daí, serão coletadas métricas como o tempo de resposta e a quantidade de requisições por segundo que o sistema foi capaz de lidar.

Na segunda etapa do procedimento de simulação, serão feitas cinco consultas ao banco de dados para cada uma das situações anteriores, além de um cenário extra quando a base de dados contém 1.000.000 de registros, e será avaliado o tempo que cada sistema leva para recuperar informações.

4.3 Modelo dos dados

O aplicativo do qual os dados serão coletados é para a área de educação, então as informações são sobre questões respondidas e suas respostas, simulados realizados, material de estudo visto, entre outros. Os dados que serão enviados e armazenados são objetos JSON, que podem variar na quantidade de campos e nos tipos desses. Na Figura 9 estão três (3) exemplos de objetos JSON que ilustram o formato e as possibilidades de estrutura que sua sintaxe permite combinar.

Figura 9 - Exemplo de dados utilizados


```
{
    "device_id": 28,
    "event": "open_question",
    "params": {
        "questionsCodes": ['4a321','sfr24',
    },
    "date": 1393648751000
}
{
    "device_id": 331,
    "event": "question_correction",
    "params": {
        "questionCode": '4a321',
        "userRespose": 'A',
        "correctResponse": 'C'
    },
    "date": 1393649044000
}
{
    "device_id": 57,
    "event": "select_disciplines",
    "params": {
        "dados": [
            {
                "id": 1,
                "text": "Mathematic",
                "state": 0
            },{
                "id": 2,
                "text": "History",
                "state": 0
            },{
                "id": 3,
                "text": "Geography",
                "state": 0
            },{
                "id": 5,
                "text": "Chemistry",
                "state": 1
            }
        ]
    },
    "date": 1393883268000
}
```


Fonte: O Autor (2016)

O campo chave ‘device_id’ representa um identificador único para cada dispositivo cadastrado. Em ‘event’ fica registrada a ação registrada pelo usuário, em ‘params’ estão os valores que representam as ações em mais detalhes com informações de acordo com o que foi feito pelo usuário, e o campo ‘date’ informa o momento da ação. De acordo com a imagem é possível ver que o campo ‘param’ do objeto pode variar bastante de formato e complexidade.

4.4 Ferramentas utilizadas

Descrição dos aplicativos utilizados para realização dos testes.

4.4.1 JMeter

O JMeter[36] é uma ferramenta *open source* para desktop, desenvolvida pela Apache na linguagem Java e pode ser usada para analisar o comportamento de testes funcionais e medir o desempenho. Para a realização de testes, a ferramenta dá suporte a vários tipos de requisições. Além disso, existem controladores lógicos e controles condicionais que são usados para construir planos de teste. O JMeter fornece um controle de *threads* que simulam os usuários do cenário real. É possível configurar a quantidade de *threads*, quantas vezes cada uma será executada e, também, o intervalo de tempo entre cada execução. Além disso, existem diversos tipos de estruturas que utilizam os resultados obtidos das requisições para poderem gerar gráficos e tabelas.

4.4.2 NodeJS

Para fornecer uma interface de comunicação transparente entre o emissor e o repositório foi desenvolvida uma RESTful API na linguagem JavaScript utilizando a plataforma NodeJS[37] e o framework Express[38] para criar um servidor simples capaz de lidar com requisições enviadas pelo emissor e repassá-las ao sistema de banco de dados.

As aplicações desenvolvidas sobre essa plataforma são executadas sobre o motor de execução JavaScript V8 desenvolvido pela Google que garante um bom desempenho até mesmo para aplicações simples[39]. Mas para garantir que esta interface RESTful não irá se tornar um gargalo para o funcionamento do sistema, as requisições serão enviadas de forma sequencial por apenas uma (1) *thread*.

4.5 Resultados

A seguir, os resultados dos experimentos de inserção e consultas serão apresentados e analisados. Tal análise se dá tanto pela avaliação dos resultados de

[36] http://jmeter.apache.org/
[37] https://nodejs.org/en/
[38] http://expressjs.com/pt-br/
[39] https://developers.google.com/v8/

experimentos individualmente, bem como na comparação entre os resultados obtidos a partir dos diferentes estados de povoamento nos quais os bancos de dados se encontrarão.

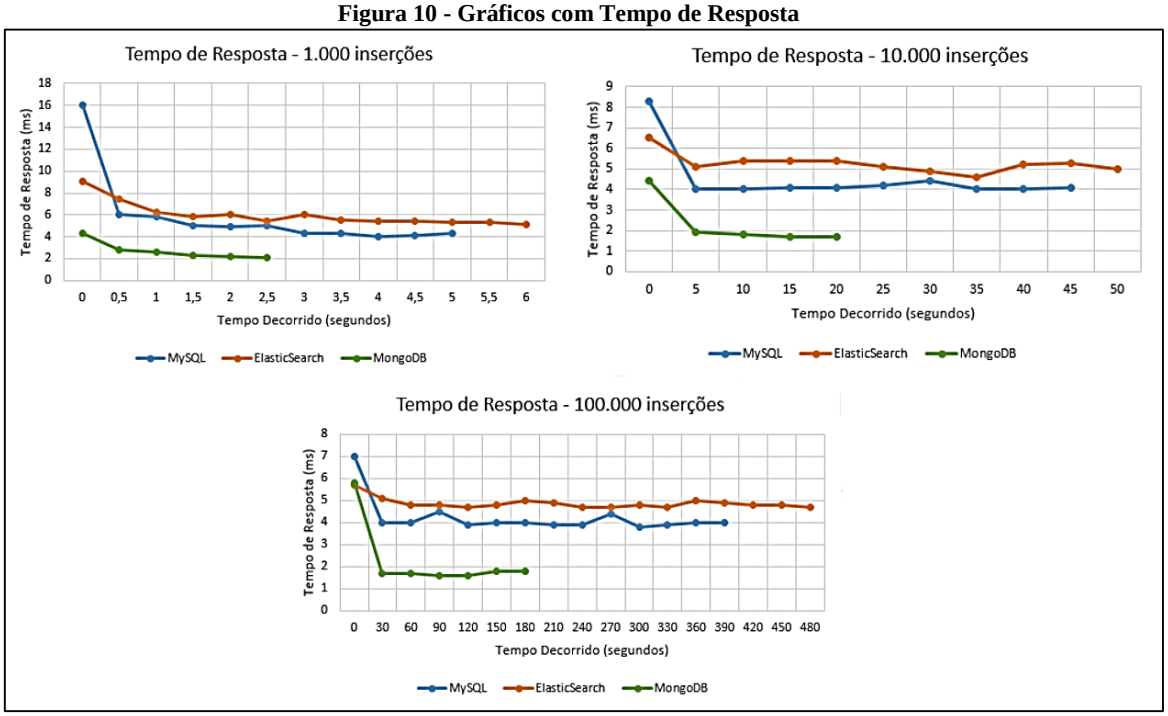
4.5.1 Primeira etapa – Inserções

Para avaliar o desempenho no momento de inserir novos documentos no banco de dados foram definidos como métricas: o tempo de resposta e quantas instruções de inserção o sistema consegue lidar por segundo. A primeira métrica informa quanto tempo um banco de dados leva para informar ao emissor que o documento foi inserido com sucesso; já a segunda métrica procura avaliar a quantidade de instruções que o sistema de armazenamento é capaz de lidar por unidade de tempo.

Os primeiros três gráficos na Figura 10 mostram o valor do tempo de resposta ao longo do tempo. É possível perceber para cada um dos sistemas um comportamento padrão: nos primeiros instantes o tempo de resposta é consideravelmente maior, isso se deve ao fato de que o servidor que recebe as conexões ainda não tem uma conexão aberta com o sistema de armazenamento. Após abrir conexão e mantê-la estabelecida, o que se vê é uma queda no tempo de resposta e uma estabilidade até o fim do processo.

Nessa etapa dos experimentos o MongoDB se destaca apresentando resultados duas vezes superiores quando comparado aos outros sistemas. Com esse resultado vê-se a superioridade no desempenho das operações não transacionais. O ElasticSearch apresentou os piores resultados, mas não é de se estranhar, visto que devido ao seu objetivo de ser uma ferramenta de busca de alto desempenho ele presa pela indexação completa dos documentos antes de realizar as inserções. Ainda assim obteve resultados bem próximos ao MySQL.

Figura 10 - Gráficos com Tempo de Resposta

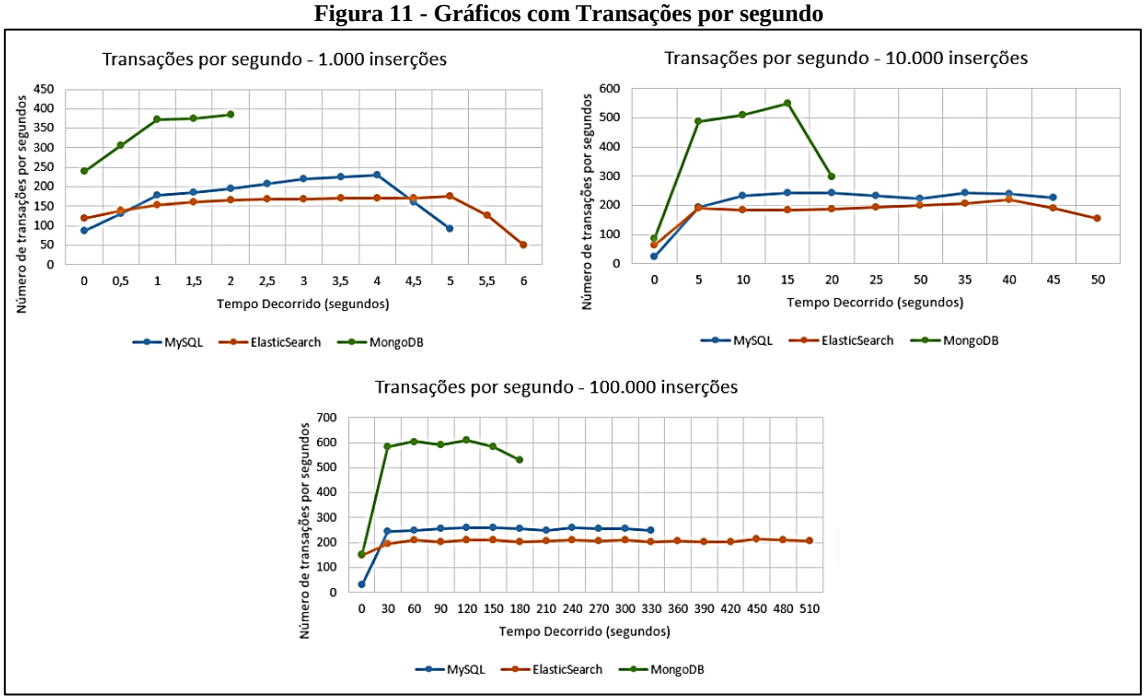


Fonte: O Autor (2016)

Outro fator que também evidencia a eficiência do MongoDB e que está diretamente relacionado com a métrica vista anteriormente é a vazão ou *throughput* do sistema de armazenamento, que indica a quantidade de instruções que está sendo processada por unidade de tempo. Os gráficos mostrados na Figura 11 são praticamente um espelho dos gráficos com o tempo de resposta e mostram que tanto o ElasticSearch quanto o MySQL possuem uma limitação de operações por segundo sempre executando valores próximos a 200 requisições por segundo, enquanto MongoDB aumenta sua capacidade de processamento de acordo com a demanda exigida pelo servidor.

Ainda sobre a curva que representa o MongoDB, o trecho final dos gráficos para 10.000 e 100.000 requisições apresenta uma queda no número de transações por segundo, o que representa que ocorre um acúmulo de transações por parte do sistema e quando o emissor pára de enviar as requisições o sistema de banco de dados ainda está resolvendo as instruções restantes.

Figura 11 - Gráficos com Transações por segundo



Fonte: O Autor (2016)

4.5.2 Segunda etapa – Consultas

Para avaliar o desempenho e a capacidade de realizar consultas em documentos JSON foram elaboradas cinco (5) consultas que foram executadas em cada um dos sistemas analisados. O script das consultas se encontra no Anexo 1. Como cada sistema tem sua própria linguagem para manipulação dos dados, foi necessária a elaboração de quinze (15) consultas distintas, mas todas com o mesmo valor semântico.

Tanto o ElasticSearch quanto o MongoDB utilizam objetos JSON para realizar as consultas, enquanto o MySQL utiliza SQL com funções para acessar os valores dos documentos na coluna JSON. Para a obtenção dos resultados cada consulta foi executada dez (10) vezes para cada cenário e foi adotado como valor resultante o tempo médio para as dez execuções.

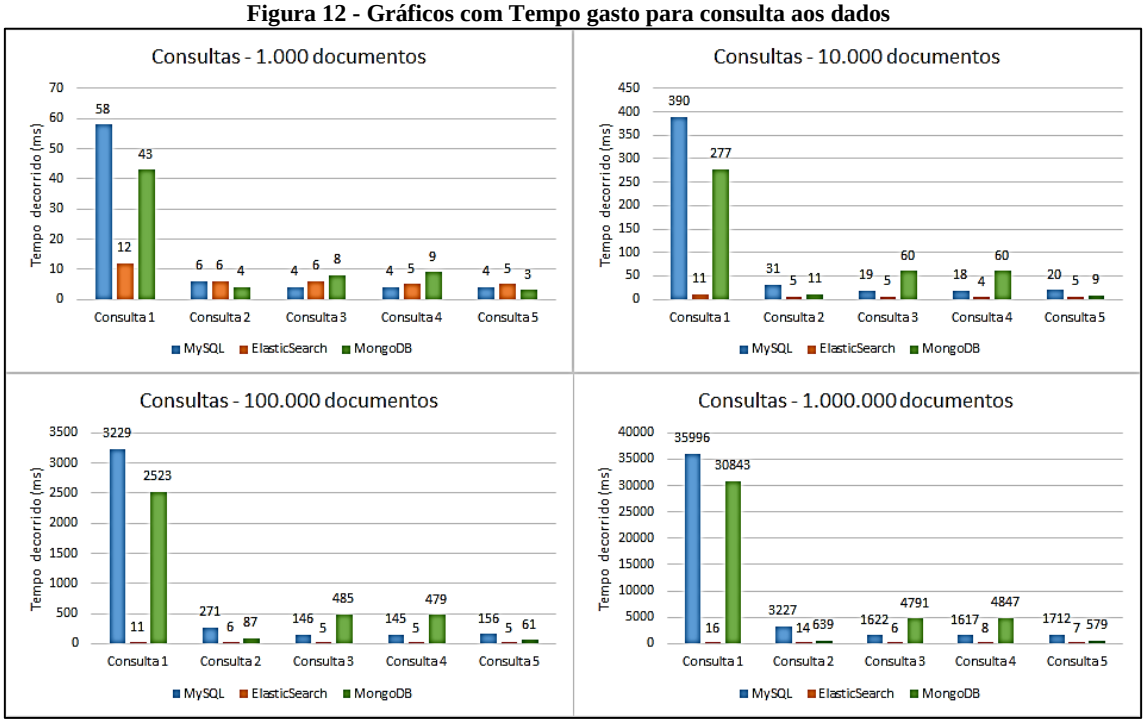
Em todos os casos é perceptível que a Consulta 1 leva mais tempo do que as demais para ser executada, devido ao fato de solicitar todos os documentos da base de dados sem nenhum tipo de restrição. As quatro consultas acessam e avaliam valores contidos nos documentos fazendo comparações e agrupando para fornecer resultados mais refinados e que façam sentido ao usuário.

Apesar de ter apresentado o pior desempenho para inserir os documentos na base de dados, o ElasticSearch se sobressai diante dos demais sistemas nessa etapa. O motor

de buscas justifica o fato de indexar todos os campos do documento e apresentar resultados muito superiores, tendo uma variação muito pequena nos valores de resposta quando o volume da base de dados aumenta 1.000 vezes.

O primeiro gráfico da Figura 11, o qual mostra os resultados para a base de dados com 1.000 documentos inseridos difere dos outros, pois, mostra o MySQL como sendo o mais eficiente para quatro (4) das cinco (5) consultas. Mas, à medida que a quantidade de documentos inseridos aumenta, o MySQL começa a obter resultados superiores apenas ao MongoDB e apenas nas consultas 3 e 4. Curiosamente, são essas consultas que requerem mais trabalho dos motores de busca dos sistemas, pois envolvem mais campos para comparação. No caso da Consulta 4 ocorre a avaliação de um período temporal por meio da comparação com um campo *timestamp*.

Figura 12 - Gráficos com Tempo gasto para consulta aos dados



Fonte: O Autor (2016)

4.6 Considerações Finais

Analisando o cenário proposto pode-se observar que dois pontos são cruciais para o bom funcionamento do sistema como um todo. O primeiro fator é a construção da RESTful API que deve ser capaz de lidar com o máximo de requisições simultâneas possíveis, de modo que garanta que todas as requisições sejam tratadas corretamente,

em caso de sucesso ou erro. Neste projeto não foi investigada a capacidade da RESTful API que foi desenvolvida, a única preocupação foi que ela não se tornasse um gargalo entre o emissor de requisições e o banco de dados.

O outro fator fundamental, e que foi analisado mais profundamente, tem a ver com os sistemas de banco de dados utilizados. O propósito dos experimentos foi avaliar dois aspectos: a velocidade de inserção e a velocidade de acesso e recuperação da informação. O que foi percebido é que nenhum dos sistemas de armazenamento utilizados se sobressai perante os outros nos dois aspectos analisados, mostrando que o desenvolvedor e sua equipe devem analisar o que for mais importante para sua aplicação.

Caso a velocidade de resposta das requisições ou fluxo de entrada de dados não seja um fator de extrema importância, o ElasticSearch se mostra a melhor alternativa devido à velocidade com a qual os dados podem ser acessados. Já o MongoDB se mostra como uma escolha mais equilibrada, possuindo ótimos resultados na etapa de inserção e resultados que podem ser considerados satisfatórios na realização das consultas. O MySQL se mostra uma alternativa também equilibrada em alguns cenários, mas mostra um desempenho inferior aos outros sistemas quando precisa responder a requisições com um número grande de informações.

5 CONCLUSÃO

Este trabalho apresentou uma análise de como quatro sistemas de gerenciamento de banco de dados oferecem suporte a dados semiestruturados e não estruturados. Foram abordados como os sistemas MySQL, MongoDB e ElasticSearch fazem para armazenar dados que não são estruturados, e inserindo o contexo de BigData, foi mostrado quais as possibilidades de escalabilidade e de processamento para quando houver a necessidade de lidar com grandes volumes de informação.

Também foi abordada a importância do BigData na atualidade e como ele influencia de maneira direta nas tendências das características fundamentais para que uma base de dados seja considerada robusta para lidar com grandes volumes de dados. Isso evidenciou algumas dificuldades enfrentadas por sistemas relacionais e o porquê de sistemas NoSQL serem considerados como mais adequados para problemas de BigData.

A partir daí foi visto como os dois paradigmas de armazenamento se diferenciam, destacando-se características gerais do método de armazenamento de cada um dos paradigmas, começando dos conceitos mais básicos definidos no momento da sua criação até as soluções atuais que foram desenvolvidas pela necessidade de evolução e adaptação.

Finalmente, foram apresentados experimentos para simulação de um estudo de caso no qual o principal objetivo foi avaliar o desempenho dos sistemas de armazenamento na manipulação de dados do tipo JSON.

5.1 Contribuições

A principal contribuição foi a execução dos experimentos para avaliar qual tipo de sistema possui melhor desempenho para funcionar como repositório para um aplicativo móvel que utiliza JSON como formato para trocar de informações. A configuração dos cenários reproduzidos durante os experimentos é simples, mas representa a estrutura básica de como grandes sistemas podem funcionar, bastando escalar qualquer uma das três partes definidas. Para tanto, basta aumentar o número de emissores de requisição, melhorar a capacidade da RESTful API para lidar com muitas requisições simultâneas e expandir o banco de dados para que funcione como um sistema distribuído.

Os experimentos executados procuraram avaliar a capacidade de ingestão e rapidez na consulta aos dados. Por meio dos resultados pode-se observar que o MongoDB conseguiu desempenhos bem melhores do que os outros dois sistemas, tanto no tempo de resposta quanto na taxa de inserções por segundo. Já na etapa dos experimentos que envolveram a execução de consultas aos sistemas, o ElasticSearch fez jus ao nome de motor de buscas e obteve resultados bem superiores aos outros sistemas à medida que a quantidade de documentos no banco aumentava.

É importante falar que as afirmações feitas a partir dos resultados dos experimentos levam em consideração o cenário e a configuração do ambiente no qual foram realizados. Apesar de que, aumentando a complexidade da configuração do ambiente de testes, principalmente para os repositórios quando os sistemas são capazes de formar *clusters,* só tende a aumentar ainda mais suas capacidades.

5.2 Principais Limitações

A principal dificuldade foi na etapa dos experimentos. Especificamente, a decisão de quais testes realizar e como preparar os ambientes para tentar garantir a igualdade de condições. Na escolha dos testes, a principal preocupação era que eles avaliassem o sistema sem sofrer interferência com limitações referentes às etapas de envio e no repasse das requisições. A dificuldade em tentar manter a igualdade de condições é por causa das diferenças entre os sistemas que permitem configurações de otimização bastante distintas. Então foi preferível usar as configurações padrão. Além disso, cada sistema tem uma interface de comunicação diferente e por isso a necessidade de desenvolver uma RESTful API como meio de uniformizar a comunicação entre cliente e servidor.

5.3 Contribuições Futuras

Para possíveis contribuições futuras, se aumentaria a topologia do ambiente de testes, ou seja, seria aumentada a quantidade de máquinas enviando requisições simultâneas, o servidor seria otimizado e os sistemas de banco de dados seriam configurados para funcionarem como sistemas distribuídos para que consigam lidar com uma grande demanda de dados. Além disso, poder-se-ia substituir o sistemas MySQL

por uma variação do próprio sistema, chamada MySQL Cluster[40] que consegue tornar o sistema relacional escalável horizontalmente, permitindo que não saísse em desvantagem na análise desse aspecto perante os outros dois sistemas analisados.

[40] https://www.mysql.com/products/cluster/

6 REFERÊNCIAS BIBLIOGRÁFICAS

ABRAMOVA, V.; BERNARDINO, J. **NoSQL databases:** MongoDB vs Cassandra. Proceedings of the International C* Conference on Computer Science and Software Engineering. New York, NY: ACM. 2013. p. 14-22.

ANDREWS , J. H. **Testing using log file analysis:** tools, methods, and issues. Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on. Honolulu: IEEE. 1998. p. 157 - 166.

ANNAMALAI, ; PALMER, C.; MAVRIS, S. **Oracle White Paper: Oracle Multimedia: Managing Multimedia**. [S.l.]. 2009.

BROWN, P.R.F. **A Brief History of Modern RDBMS IT Management: Emergence of the RDBMS.** Disponível em: http://www.mountainman.com.au/software/history/it3.html. Acesso em: 12 abr. 2016.

CAI, L. et al. **Performance analysis and testing of HBase based on its architecture**. Computer and Information Science (ICIS), 2013 IEEE/ACIS 12th International Conference on. Niigata: IEEE. 2013. p. 353 - 358.

CATTELL, R. Scalable SQL and NoSQL Data Stores. **ACM SIGMOD Record**, December 2010. 12-27.

CHANG, F. et al. Bigtable: A Distributed Storage System for Structured Data. **ACM Transactions on Computer Systems (TOCS)**, Nova Iorque, v. 26, n. 2, Junho 2008.

CHEN, C. L. P.; ZHANG, C.-Y. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. **Information Sciences**, v. 275, p. 314-347, 10 ago. 2014.

CHODOROW, K. **MongoDB The Definite Guide**. 2nd. ed. [S.l.]: O'Reilly Media, 2013.

CODD, E. F. A Relational Model of Data for Large Shared Data Banks. **Communications of the ACM**, San Jose, CA, v. 13, n. 6, p. 377-387, June 1970.

CSC, COMPUTER SCIENCES CORPORATION. **Big Data Just Beginning to Explode - Interactive Infographic**. Disponível em: http://www.csc.com/big_data/flxwd/83638-big_data_just_beginning_to_explode_interactive_infographic. Acesso em: 02 abr. 2016.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Communications of the ACM**, v. 51, n. 1, p. 107-113, January 2008.

DEDE, E. et al. **Performance evaluation of a MongoDB and hadoop platform for scientific data analysis**. Science Cloud '13 Proceedings of the 4th ACM workshop on Scientific cloud computing. [S.l.]: [s.n.]. 2013. p. 13-20.

DRAKE,. **Oracle XML DB in Oracle Database 12c**. Oracle. [S.l.]. 2013.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 6. ed. [S.l.]: [s.n.], 2011.

FEINBERG, D., ADRIAN, M., HEUDECKER, N., RONTHAL, A.M., PALANCA ,T. **Magic Quadrant for Operational Database Management Systems.** Disponível em: https://www.gartner.com/doc/reprints?id=1-2PMFPEN&ct=151013. Acesso em: 02/05/2016.

FORD, R. **Oracle Text**. Oracle. [S.l.]. 2013.

GANDOMI, A.; HAIDER, M. Beyond the hype: Big data concepts, methods, and analytics. **International Journal of Information Management**, v. 35, n. 2, p. 137–144, April 2015.

GORMLEY, C.; TONG, Z. **Elasticsearch:** The Definitive Guide. 1st. ed. [S.l.]: O'Reilly Media, 2015.

GRAY , J.; REUTER , A. **Transaction Processing:** Concepts and Techniques. 1. ed. [S.l.]: Morgan Kaufmann, 1992.

GU, Y.; ENG, J. **Analysis of data storage mechanism in NoSQL database MongoDB**. Consumer Electronics - Taiwan (ICCE-TW), 2015 IEEE International Conference. Taipei: IEEE. 2015. p. 70 - 71.

GUDIVADA , V. N.; RAO, D.; RAGHAVAN, V. V. **NoSQL Systems for Big Data Management**. 2014 IEEE World Congress on Services. Anchorage: IEEE. 2014. p. 190-197.

HECHT, R.; JABLONSKI, S. **NoSQL Evaluation:** A Use Case Oriented Survey. Cloud and Service Computing (CSC), 2011 International Conference on. Hong Kong: IEEE. 2011. p. 336 - 341.

IBM. **Texto sobre Edgar F. Codd**. Disponível em: http://www-03.ibm.com/ibm/history/exhibits/builders/builders_codd.html. Acesso em: 12 abr. 2016.

JANAKIRAM, M.S.V. **Microsoft, Oracle, AWS are the top database leaders, according to Gartner.** Disponível em: http://www.techrepublic.com/article/microsoft-oracle-aws-are-the-top-database-leaders-according-to-gartner/. Acesso em: 04 abr. 2016

JAYATHILAKE, D. et al. **A study into the capabilities of NoSQL databases in handling a highly heterogeneous tree**. 2012 IEEE 6th International Conference on Information and Automation for Sustainability. Beijing: IEEE. 2012. p. 106 - 111.

JIANG , Z. et al. **Managing Large Scale Unstructured Data with RDBMS**. Dependable, Autonomic and Secure Computing (DASC), 2013 IEEE 11th International Conference. Chengdu: IEEE. 2013. p. 613 - 620.

KONONENKO, O. et al. **Mining Modern Repositories with Elasticsearch**. 11th Working Conference on Mining Software Repositories. [S.l.]: ACM. 2014. p. 328-331.

KUC, R.; ROGOZINSKI, M. **Mastering ElasticSearch**. [S.l.]: Packt Publishing, 2013.

LOMOTEY, R. K.; DETERS, R. **Unstructured data extraction in distributed NoSQL**. 2013 7th IEEE International Conference on Digital Ecosystems and Technologies (DEST). Menlo Park: IEEE. 2013. p. 160 - 165.

MARKOVIKJ, D. et al. **Mining Facebook data for predictive personality modeling**. Conference: Proceedings Of AAAI International Conference on Weblogs and Social Media. [S.l.]: [s.n.]. 2012.

MCCREARY, D. **Making Sense of NoSQL**. 1. ed. [S.l.]: Manning, 2014.

MCCREARY, D.; KELLY, A. **Making Sense of NoSQL:** A guide for managers and the rest of us. [S.l.]: [s.n.], 2013.

O'REILLY,T. **Design Patterns and Business Models for the Next Generation of Software.** Disponível em: http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html. Acesso em: 03 abr. 2016.

ORACLE. **Oracle White Paper: Unstructured Data Management with**. Oracle. [S.l.]. 2014.

ORACLE, **Unstructured Data Management with Oracle Database 12c**, Disponível em: http://www.oracle.com/technetwork/database/information-

management/unstructured-data-management-wp-12c-1896121.pdf. Acesso em: 05 jun. 2016.

ORACLE. **MySQL Whitepaper: Unlocking New Big Data Insights with MySQL & Hadoop**. Oracle. [S.l.]. 2015.

PARKER, Z.; POE, S.; VRBSKY, S. V. **Comparing NoSQL MongoDB to an SQL DB**. Proceedings of the 51st ACM Southeast Conference. New York, NY: ACM. 2013.

SHANK, S.; DIJCKS, J.-P. **Oracle White Paper: In-Database Map-Reduce**. Oracle. [S.l.]. 2009.

SHUKLA, V. **Elasticsearch for Hadoop**. [S.l.]: Packt Publishing, 2015.

VANDIKAS, K. **Performance Evaluation of an IoT Platform**. 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies. Oxford: IEEE. 2014. p. 141 - 146.

ZHANG, B. et al. **Understanding User Behavior in Spotify**. IEEE INFOCOM. Turin: IEEE. 2013. p. 220 - 224.

ANEXO 1 - CONSULTAS

Consulta 1

ElasticSearch

```
{
  "query": {
    "bool": {
      "must": [
        {
          "match_all": {}
        }
      ],
      "must_not": [],
      "should": []
    }
  }
}
```

MySQL

```
SELECT * FROM stats;
```

MongoDB

```
d.getCollection('stats').find({});
```

Consulta 2

ElasticSearch

```
{
  "query": {
    "term": {
      "event": "questoes.corrigiu"
    }
  },
  "aggs": {
```

```
    "estudioso": {
      "terms": {
        "field": "DeviceId"
      }
    }
  }
}
```

MySQL

```
SELECT JSON_EXTRACT(event, '$.device_id'),COUNT(*) FROM stats WHERE JSON_EXTRACT(event, '$.event') = 'questoes.corrigiu' GROUP BY JSON_EXTRACT(event, '$.device_id');
```

MongoDB

```
db.getCollection("stats").aggregate([
        {$match:{"event": {"$eq":'questoes.corrigiu'}}},
        {$group:{_id:
{"device_id":"$device_id","event":"$event"},"questoes_device":{$sum: 1}}},
        {$group:{_id:{"device_id":"$_id.device_id"},"total_sum":{"$sum":"$qu
estoes_device"}}}
        ]);
```

Consulta 3

ElasticSearch

```
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "DeviceId": 61
          }
        },
        {
```

```
            "match": {
              "event": "questoes.corrigiu"
            }
           }
         ]
        }
      },
      "filter": {
        "script": {
          "script": "doc[\"params.correta\"].value == doc[\"params.resposta\"].value"
        }
      }
    }
```

MySQL

```
SELECT * FROM stats WHERE JSON_EXTRACT(event, '$.event') = 'questoes.corrigiu' AND JSON_EXTRACT(event, '$.device_id') = 61 AND JSON_EXTRACT(event, '$.params.correta') = JSON_EXTRACT(event, '$.params.resposta');
```

MongoDB

```
db.getCollection("stats").aggregate([
        {"$project":{device_id:1, event:1, params:1, date:1,questoes_device:1,"same":{"$eq":["$params.correta","$params.resposta"]}}},
        {$match:{"same":true}},
        {$match:{"device_id": 61}},
        {$match:{"event":'questoes.corrigiu'}}
        ]);
```

Consulta 4

ElasticSearch

```
{
  "query": {
    "bool": {
```

```
      "must": [
        {
          "match": {
            "DeviceId": 1962
          }
        },
        {
          "match": {
            "event": "questoes.corrigiu"
          }
        },
        {
          "match": {
            "params.correta": 1
          }
        }
      ],
      "filter": [
        {
          "range": {
            "date": {
              "gte": "1394727424000"
            }
          }
        }
      ]
    }
  }
}
```

MySQL

SELECT * FROM stats WHERE JSON_EXTRACT(event, '$.event') = 'questoes.corrigiu' AND JSON_EXTRACT(event, '$.device_id') = 61 AND

JSON_EXTRACT(event, '$.params.correta') = JSON_EXTRACT(event, '$.params.resposta') AND JSON_EXTRACT(event, '$.date') >= 1394727424000;

MongoDB

```
db.getCollection("stats").aggregate([
        {"$project":{device_id:1, event:1, params:1, date:1,questoes_device:1,"same":{"$eq":["$params.correta","$params.resposta"]}}},
        {$match:{"same":true}},
        {$match:{"device_id": 61}},
        {$match:{"event":'questoes.corrigiu'}},
        {$match:{"date": {$gte:1394042656000}}},
        ]);
```

Consulta 5

ElasticSearch

```
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "event": "material.vademecum"
          }
        },
        {
          "match": {
            "params.titulo": "Leis"
          }
        }
      ]
    }
  }
}
```

MySQL

```
SELECT * FROM stats WHERE JSON_EXTRACT(event, '$.event') = 'material.vademecum' AND JSON_EXTRACT(event, '$.params.titulo') = 'Leis';
```

MongoDB

```
db.getCollection("stats").find({ $and: [ { 'event': 'material.vademecum' }, { 'params.titulo': 'Leis' } ]});
```