

UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE COMPUTAÇÃO E TECNOLOGIA DA INFORMAÇÃO
BACHARELADO EM SISTEMAS DA INFORMAÇÃO

AVALIAÇÃO NOSQL PARA INDEXAÇÃO DE DADOS

CALIEL LIMA DA COSTA

CAXIAS DO SUL

2012

CALIEL LIMA DA COSTA

AVALIAÇÃO NOSQL PARA INDEXAÇÃO DE DADOS

TRABALHO DE CONCLUSÃO DE CURSO
DE SISTEMAS DE INFORMAÇÃO, COMO
REQUISITO PARCIAL PARA OBTENÇÃO
DO TÍTULO DE BACHAREL.

ORIENTADORA: PROF^a. DR^a. HELENA
GRAZIOTTIN RIBEIRO

CAXIAS DO SUL

2012

“It is a very sad thing that nowadays there is so little useless information”

Oscar Wilde (1854-1900)

“A ideia de que algo tão organicamente diverso e dinamicamente produtivo quanto o conhecimento humano possa ser gerenciado, no sentido comum da palavra, traz em seu bojo um erro fatal. A forma mais segura de inibir o desenvolvimento de fenômenos espontâneos é tentar gerenciá-los. Gestão pressupõe impor algum tipo de ordem sobre o conhecimento – exatamente o que não se deve fazer. O que podemos, e devemos, fazer é gerenciar as circunstâncias em que o conhecimento pode prosperar. Em outras palavras, a ideia seria gerenciar culturas de conhecimento.”

Karl Albrecht, 2004

RESUMO

A Gestão do Conhecimento, através de suas técnicas e ferramentas, busca ajudar as empresas a aprender e inovar para atingir melhores resultados e serem mais eficientes no mercado. No entanto, essas mesmas empresas possuem cada vez mais informações internas e externas disponíveis e, muitas vezes, essas diversas informações (dados relacionais, e-mails, vídeos, imagens, etc.) precisam ser analisadas em um intervalo curto de tempo. Esse cenário é denominado Big Data.

Os bancos de dados relacionais podem não estar preparados para suprir as necessidades desse cenário. Conforme o volume de dados aumenta, a capacidade de processamento do servidor também deve aumentar para continuar atendendo aplicações e usuários, necessitando de investimentos crescentes em infraestrutura. Para atender a essas necessidades surgiu um novo paradigma de banco de dados chamado NoSQL. O modelo de dados deixa de ser relacional e passa a ser representado como objetos, documentos ou grafos, e a arquitetura do sistema é voltada para múltiplos servidores, propiciando fácil escalabilidade e alta disponibilidade.

O trabalho apresenta todo este contexto e caracteriza os principais sistemas de bancos de dados existentes no mercado. Ele tem como objetivo avaliar a utilização de um banco de dados NoSQL para a indexação de mensagens do Twitter selecionadas através de uma Taxonomia.

Palavras-Chave: Gestão do Conhecimento, Taxonomia, Redes Sociais, Twitter, SGBD, Banco de dados, NoSQL, Modelo Relacional, Big Data, Amazon, DynamoDB, AWS.

ABSTRACT

Knowledge Management, through its tools and techniques, seeks to help companies into learn and innovate to achieve better results be more efficient on the market. Yet, these same companies have increasingly internal and external information available and often these various information (relational databases, emails, videos, images, etc.) Need to be analyzed in a short interval. This scenario is called Big Data.

The relational database may not be prepared to meet the needs of this scenario. As the volume of data increases, the processing capacity of the server must also increase to continue to meet users and applications, requiring increased investment in infrastructure. To meet these needs emerged a new paradigm of database called NoSQL. The data model ceases to be relational and shall be represented as objects, documents or graphs, and the system architecture is focused on multiple servers, providing easy scalability and high availability.

The aims to evaluate the use of a NoSQL database for indexing Twitter messages selected through a taxonomy.

Keywords: Knowledge Management, Taxonomy, Social Network, Twitter, SGBD, Data Base, NoSQL, Relational Model, Amazon, DynamoDB, AWS.

Titulo em Inglês: NoSQL evaluation to index data.

LISTA DE FIGURAS

Figura 1: Classificação Brasileira de Ocupações	21
Figura 2: Escalonamento	29
Figura 6: Particionamento e Replicação de dados	47
Figura 7: Versionamento de registros	48
Figura 8: Leitura e gravação no BD	51
Figura 3: Taxonomia de Redes Sociais	54
Figura 4: Taxonomia de Gestão do Conhecimento	54
Figura 5: Taxonomia Técnica	55
Figura 9: REST API do Twitter	56
Figura 10: Streaming API do Twitter	57
Figura 11: Modelo Conceitual	59
Figura 12: Primeira versão das tabelas	59
Figura 13: Segunda versão das tabelas	60
Figura 14: Terceira versão das tabelas	60
Figura 15: Ciclo de vida SCRUM	63
Figura 16: Acunote	64
Figura 17: Interface do site de consulta	68
Figura 18: Camadas do projeto	69

LISTA DE GRÁFICOS

Gráfico 1: Distribuição da importação	72
Gráfico 2: Unidades de gravação em Message e Message_by_Created	80
Gráfico 3: Unidades de gravação em Message_By_Tag e Message_by_TagCreated	80
Gráfico 4: Tempo máximo de resposta da gravação na tabela Message	81
Gráfico 5: Tempo mínimo de reposta da gravação na tabela Message	81

LISTA DE TABELAS

Tabela 1: Resumo dos bancos de dados	43
Tabela 2: Site oficial dos produtos	44
Tabela 3: Técnicas de implementação	45
Tabela 4: Gravação de registros	62
Tabela 5: Análise de Mensagens	73

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
ANSI	American National Standards Institute
API	Application Programming Interface
BASE	Disponibilidade básica, Estado leve e Consistência eventual
BD	Banco de Dados
BI	Bussiness Intelligence
BSON	Binary JSON
CAP	Consistência, Disponibilidade e Tolerância à partição
CBO	Classificação Brasileira de Ocupações
DBA	Database administrator
DBMS	Database Management System
DBRef	Database Reference
DER	Diagrama Entidade-Relacionamento
DSK	Software development kit
FK	Foreign Key
GC	Gestão do Conhecimento
GFS	Google File System
HDFS	Hadoop Distributed File System
HK	Hash Key
HQL	Hypertable Query Langue
ISO	International Standards Organization
JSON	JavaScript Object Notationt
MVCC	Multiversion Concurrency Control

NoSQL	Não Apenas SQL
OLAP	Sistemas de Processamento Analítico
OLTP	Sistemas de Processamento Transacionais
PK	Primary Key
RDBMS	Relational Database Management Systems
REST	Representation State Transfer
RK	Range Key
SGBD	Sistemas de Gerência de Banco de Dados
SNS	Social Network Site
SQL	Linguagem de Consulta Estruturada
SSD	Solid State Disks
TDD	Test Driven Development
UTC	Universal Time Coordinated
XP	Extreme Programming

SUMÁRIO

1	INTRODUÇÃO	14
1.1	Objetivo	15
1.2	Organização do Documento	15
2	REDES SOCIAIS E GESTÃO DO CONHECIMENTO.....	16
2.1	Redes Sociais	16
2.2	Twitter	17
2.3	Gestão do Conhecimento	18
2.4	Taxonomia e Folksonomia	19
3	BANCO DE DADOS	22
3.1	Modelo Relacional	23
3.2	NoSQL	24
<i>3.2.1</i>	<i>Big Data</i>	<i>25</i>
<i>3.2.2</i>	<i>O Movimento NoSQL</i>	<i>27</i>
<i>3.2.3</i>	<i>Técnicas de Implementação</i>	<i>29</i>
3.3	Categorias de NoSQL.....	30
<i>3.3.1</i>	<i>Chave-valor (key-value).....</i>	<i>30</i>
<i>3.3.2</i>	<i>Orientado a Colunas</i>	<i>34</i>
<i>3.3.3</i>	<i>Orientado a Documentos.....</i>	<i>38</i>
<i>3.3.4</i>	<i>Orientado a Grafos</i>	<i>41</i>
<i>3.3.5</i>	<i>Comparativo entre os Bancos</i>	<i>42</i>
4	BANCO DE DADOS DYNAMO.....	45
4.1	Desenho da Solução	45
4.2	Particionamento e Replicação	46

4.3	Versionamento dos Dados.....	47
4.4	O produto Amazon DynamoDB.....	49
4.4.1	<i>Operações de Banco de Dados</i>	50
4.4.2	<i>Consultas.....</i>	52
5	DESENVOLVIMENTO	53
5.1	Taxonomia.....	53
5.2	Integração com o Twitter.....	56
5.3	Modelo de Dados	58
5.4	Metodologia de Desenvolvimento.....	63
5.5	Projeto de Desenvolvimento	65
5.5.1	<i>Marvin</i>	65
5.5.2	<i>Telescreen.....</i>	66
5.5.3	<i>Camadas.....</i>	69
6	CONCLUSÃO	71
6.1	Análise das Informações Encontradas no Twitter	71
6.2	Performance do Amazon DynamoDB	79
6.3	Desenvolvimento com Amazon DynamoDB.....	82
6.4	Trabalhos Futuros.....	83
	REFERÊNCIAS BIBLIOGRÁFICAS	85

1 INTRODUÇÃO

Todos os dias são produzidos 2,5 quintilhões de bytes de informações (IBM, [21--]), sendo assim 90% dos dados no mundo foram criados nos últimos dois anos e, segundo a revista *The Economist* (THE ECONOMIST, 2010), somente 5% da informação criada é estruturada, constituída de números e palavras organizadas que podem ser lidas pelos computadores.

Essa situação está mudando com aumento das marcas (*tags* do inglês) em textos, vídeos e imagens.

Em 1998 foi utilizado o termo NoSQL para um banco de dados relacional que omitiu o uso da linguagem SQL. Posteriormente em 2009 foi utilizado novamente em uma conferência em São Francisco por defensores de um modelo não relacional (STRAUCH, 2011). Atualmente, o termo NoSQL, que significa “Not only SQL” ou “Não apenas SQL” em português (LÓSCIO, DE OLIVEIRA e PONTES, 2011), é utilizado para solucionar as questões de escalabilidade no armazenamento e processamento de grandes volumes de dados (DE DIANA e GEROSA, 2010).

Após a adoção com sucesso pelo Google, Amazon, Facebook e Twitter de uma infraestrutura NoSQL, outras empresas também vem migrando para a plataforma. No início do ano de 2012, um site de *streaming* de vídeo com mais de 100 milhões de visualizações de páginas diárias e picos de mais de 300 mil consultas por segundo, anunciou a utilização de NoSQL como a principal tecnologia de banco de dados (BD) nesse contexto.

Em uma pesquisa realizada com 325 gerentes de TI aproximadamente 100 deles assumiram que estão realizando análises avançadas para inteligência de negócios, análise preditiva, prospecção de dados e tarefas de análise estatística (VIJAYAN, 2011). Em outra pesquisa realizada pela Couchbase com 1300 desenvolvedores, arquitetos de software e administradores de TI, 50% deles iniciaram projetos com NoSQL em 2011 e 70% das empresas com mais de 250 desenvolvedores planejam iniciar um projeto em 2012 (ABEL, 2012).

O mercado NoSQL está em evolução e as plataformas não estão consolidadas, mas a sua utilização tem reduzido o custo de armazenamento e processamento, podendo ser uma

grande oportunidade para as corporações obterem conhecimentos estratégicos para a tomada de decisão.

1.1 Objetivo

O objetivo é avaliar a utilização de um banco de dados NoSQL para a indexação de mensagens do Twitter selecionadas através de uma Taxonomia.

1.2 Organização do Documento

Este trabalho está estruturado em capítulos conforme segue:

- a) Redes Sociais e Gestão do Conhecimento: Apresenta-se a definição de redes sociais, o contexto do Twitter e o conceito de Gestão do Conhecimento.
- b) Banco de Dados: Neste capítulo os bancos de dados em especial o modelo relacional é conceituado, apresentação sobre as origens do NoSQL e os principais bancos disponíveis no mercado segundo suas categorias.

2 BANCO DE DADOS DYNAMO

O Dynamo foi apresentado em 2007 baseado nos seguintes requisitos e pressupostos :

- a) Modelo de consulta: operações de leitura e escrita de dados identificados por uma chave, não permitindo operações em múltiplos registros e nem esquema relacional.
- b) Propriedade ACID: experiências na Amazon mostraram que para prover as características ACID é necessário ter uma baixa disponibilidade, no entanto a disponibilidade é um dos requisitos principais do SGBD. Sendo assim, o objetivo era diminuir a consistência se isso trouxesse maior disponibilidade.
- c) Eficiência: o sistema deve funcionar em uma infraestrutura formada por hardware de baixo custo.
- d) Uso Interno: somente utilizado na infraestrutura interna da Amazon, onde o ambiente é considerado seguro não necessitando de autenticação e autorização.
- e) Escalabilidade Incremental: o sistema deve ser capaz de aumentar o processamento horizontalmente no menor tempo possível na operação e nos sistemas adjacentes.
- f) Simétrico: cada nodo da infraestrutura tem as mesmas responsabilidades, não existindo nodos com funções ou responsabilidades distintas.
- g) Descentralização: além de simétrico, a descentralização *peer-to-peer* deve ser utilizada ao invés de controle centralizado.
- h) Heterogenia: o sistema deve ser capaz de explorar melhor a heterogenia da infraestrutura, ou seja, o processamento do nodo deve ser proporcional às suas capacidades.

2.1 Desenho da Solução

Para que o Dynamo possa ser particionado, replicado, versionado e escalável, foram utilizadas algumas técnicas que são listadas na Tabela 3: Técnicas de implementação.

Tabela 3: Técnicas de implementação

Problema	Técnica	Vantagens
Particionamento	Consistent Hashing	Escalabilidade incremental

Alta disponibilidade na gravação	Vector clocks com reconciliação na leitura	O tamanho da versão está dissociado da taxa de utilização
Recuperação de falhas permanentes	Antientropia utilizando árvores Merkle	Sincronização de réplicas em <i>background</i>
Sócios e detecção de falhas	Protocolo Gossip-based membership	Mantém a simetria e evita um registro centralizado de informações

Fonte: Tabela baseada em resumo disponível em DeCandia, *et al.* .

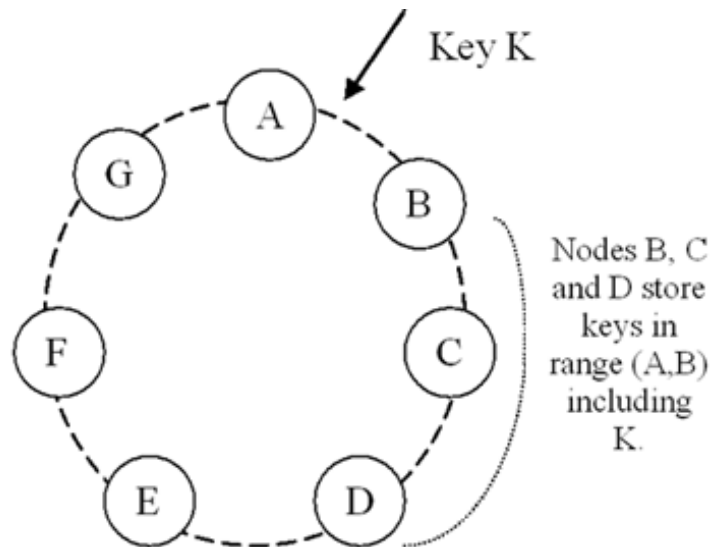
2.2 Particionamento e Replicação

Para realizar o particionamento dos dados é utilizada a técnica de *consistent hashing* onde a função *hash* é tratada como um espaço circular fechado ou anel. A cada nodo é atribuído um valor aleatório que representa a sua posição no anel. Para cada chave de valor é calculado o valor do *hash* que corresponde a uma posição no anel. Com base nesta posição, o anel é percorrido no sentido horário até encontrar a primeira posição maior que a calculada, sendo assim cada nó é responsável pela região no anel entre o seu predecessor e ele mesmo (o processo pode ser visto na Figura 3).

A versão padrão de *consistent hashing* apresenta dois problemas:

- a) A distribuição aleatória de nós gera uma distribuição não uniforme de dados e processamento.
- b) O algoritmo é feito para um ambiente homogêneo de nós.

Para resolver esses problemas foi utilizada uma variação do algoritmo que utiliza o conceito de nós virtuais. Cada nó físico pode gerar múltiplos nós virtuais (de acordo com a capacidade do servidor), que são mapeados no anel.



Fonte: <http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>

Figura 3: Particionamento e Replicação de dados

O processo seguinte é a replicação dos dados em múltiplos servidores, parte fundamental para atingir alta disponibilidade e durabilidade. Após identificar o nó de armazenamento (conforme visto na etapa de particionamento) o dado é armazenado localmente e a operação é confirmada ao programa chamador. Assincronamente este servidor realiza a atualização do dado em nós que estão ao seu alcance, no exemplo da Figura 3 estes nós são o C e D.

Para evitar falhas de hardware, a replicação não pode ocorrer em nós virtuais pertencentes ao mesmo servidor físico.

2.3 Versionamento dos Dados

O Dynamo é um sistema de consistência eventual, ou seja, a operação é dada como terminada após atualizar o dado no nó coordenador. As atualizações das réplicas são feitas de forma assíncrona. Neste cenário pode acontecer de uma nova requisição de leitura retornar o valor anterior a última atualização.

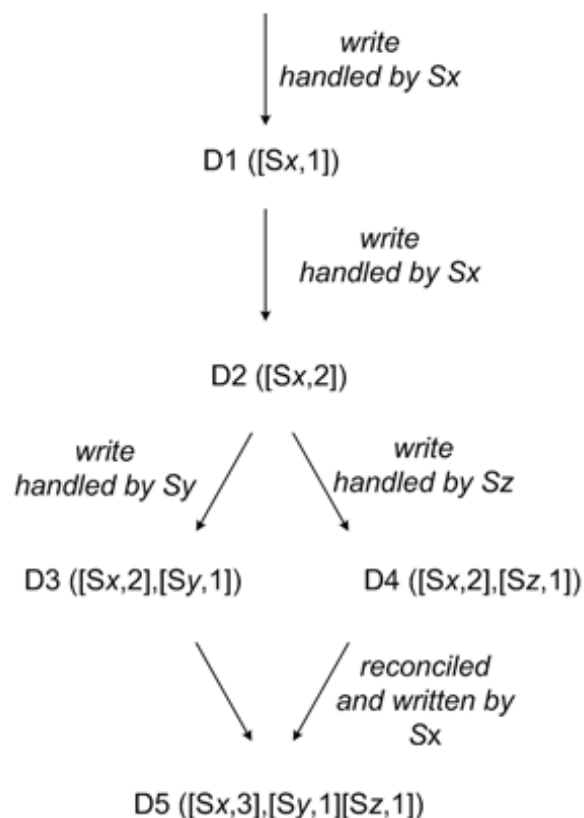
Para prevenir problemas de divergência de cada alteração num registro, gera-se uma nova versão imutável do dado, permitindo que múltiplas versões do dado existam ao mesmo tempo. Na maioria das vezes, as novas versões se sobrepõem automaticamente a versões

anteriores, porém, quando existem múltiplas versões paralelas do registro, o sistema não consegue resolver o conflito, sendo necessário realizar a união entre os dois registros.

Para o controle do versionamento de cada registro é utilizada a técnica de *vector clocks*. Cada versão do registro recebe uma lista de tuplas (*vector clocks*), através da comparação de dois *clocks* é possível determinar a ordem entre os registros ou se houve uma ramificação entre eles.

A Figura 4 apresenta um exemplo onde pode ser analisada a ramificação de um registro. Nela um determinado registro é alterado pelo o usuário S_x gerando a versão D1, em seguida o mesmo usuário altera o registro criando a versão D2. A ramificação ocorre quando os usuários S_y e S_z realizam a alteração no mesmo momento, criando duas versões distintas D3 e D4, respectivamente. No passo final, quando o usuário S_x vai realizar uma nova alteração, precisa realizar a reconciliação entre as versões criando a versão D5.

Os valores apresentados entre parâmetros representam o *vector clock*, apresentando qual foi o caminho que a versão percorreu.



Fonte:
<http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>

Figura 4: Versionamento de registros

2.4 O produto Amazon DynamoDB

O Amazon DynamoDB foi lançado como produto em 18 de janeiro de 2012 e é o resultado de 15 anos de aprendizado nas áreas de banco de dados não relacionais em larga escala e serviços em nuvem. O Amazon DynamoDB foi baseado no Dynamo que também serviu de base para outros bancos de dados. O Dynamo foi originalmente desenhado para ser utilizado nas operações chaves de e-commerce da Amazon, como o carrinho de compras e o serviço de seção.

Com base nisso, o Amazon DynamoDB é um banco de dados NoSQL que fornece rapidez e performance, além de ser totalmente gerenciado pelo administrador. Ele automaticamente distribui os dados e o tráfego das tabelas a um número de servidores que seja capaz de lidar com as requisições dos clientes. Os dados são armazenados em discos *Solid State Disks* (SSD) e são automaticamente replicados através de múltiplas regiões para prover alta disponibilidade e durabilidade .

O DynamoDB possui as seguintes características:

- a) Provisionamento de transferência: Durante a criação ou edição da tabela, a capacidade de requisição desejada pode ser especificada. O SGBD se encarrega de alocar os recursos necessários para garantir a capacidade especificada. O provisionamento é feito através da quantidade de unidades de 1KB que se deseja gravar ou ler.
- b) Escalabilidade de armazenamento: Não existe um limite de quantidade de dados armazenados. O serviço automaticamente aloca mais espaço quando necessário.
- c) Distribuição Total: O escalonamento horizontal e automatizado replica a mesma tabela em vários servidores.
- d) Construído à prova de falhas: O processo automático e assíncrono de replicação de dados através de múltiplos servidores espalhados em diferentes regiões garante, além de alta disponibilidade, também a proteção dos dados contra falhas individuais ou coletivas de hardware.
- e) Consistência forte e contadores atômicos: Diferente de outros BD não relacionais, o DynamoDB facilita o desenvolvimento para garantir a consistência forte durante a leitura, retornando sempre o último valor do registro. Além disso, a API fornece chamadas para incremento e decremento de contadores de forma atômica.

- f) Flexibilidade: O BD não possui um esquema fixo de dados, o que significa que cada registro de uma tabela pode ter um número diferente de atributos e de tipos de dados.

2.4.1 Operações de Banco de Dados

A API fornece as operações de criação, atualização e exclusão de tabelas. A atualização da tabela permite o aumento ou diminuição do provisionamento de transferência. Cada tabela é formada por uma chave única (campo *Hash Key*) e pode ou não ter uma variação (campo *Range Key*). O conteúdo de um atributo pode ser: número, literal, conjunto de números ou conjunto de literais. O tamanho em bytes de cada registro é definido pelo somatório do tamanho dos nomes dos campos mais o tamanho binário dos dados.

Também são fornecidos métodos para adicionar, atualizar e excluir registros das tabelas. Durante a atualização de itens é possível modificar valores e adicionar ou remover colunas. Para otimizar as buscas, pode ser utilizada uma operação de retorno de um ou múltiplos itens através de sua chave primária, inclusive em múltiplas tabelas.

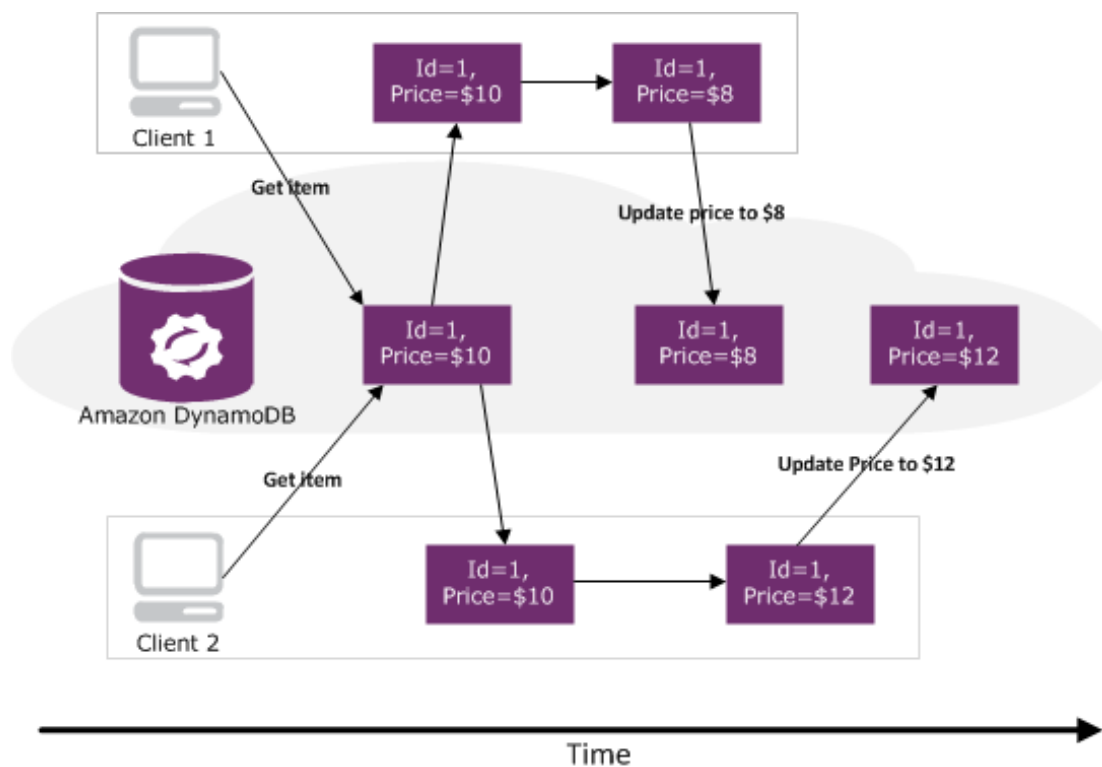
2.4.1.1 Leitura e consistência

O SGBD mantém múltiplas cópias de cada item para garantir durabilidade, para que isso aconteça após uma operação de alteração de dados é necessário que o dado seja gravado em múltiplos servidores, o que demora algum tempo. Essa demora faz com que o dado fique temporariamente inconsistente, ou seja, caso uma leitura seja feita imediatamente o valor antigo pode ser retornado. Essa é a forma padrão de leitura. Mas em alguns casos, é necessário utilizar uma leitura consistente, para isso o DynamoDB retorna o mais recente que reflita todas as operações de escrita. Essa forma de leitura é mais suscetível à lentidão da rede.

2.4.1.2 Controle de Concorrência

Em um ambiente multiusuário, em alguns casos, é necessário garantir que a atualização de um usuário não afete a gravação de outro. Para isso, o DynamoDB suporta a escrita condicional, que nada mais é do que a verificação de valor gravado antes de realizar a gravação de um novo valor. Para contadores de valor, é possível utilizar operações atômicas que incrementam ou decrementam valores sem sofrerem interferências de outras operações de gravação.

A Figura 5 apresenta uma simulação de como os dados são solicitados e gravados, Nela o registro é lido simultaneamente pelos usuários “Client 1” e “Client 2”, enquanto o “Client 1” realiza a alteração do valor para 8, o “Client 2” realiza a alteração para 10 e ambos são atualizados no BD, gerando duas versões do registro.



Fonte: <http://aws.amazon.com/pt/documentation/dynamodb/>

Figura 5: Leitura e gravação no BD

2.4.2 Consultas

Para consultas, existem dois mecanismos: *Query* e *Scan*. A *Query* permite a consulta na tabela utilizando o campo *Hash* e, opcionalmente, um filtro de *Range*. Esse mecanismo é o mais eficiente para buscar itens na tabela.

A operação de *Scan* é a forma menos eficiente, pois realiza uma varredura em todos os dados da tabela. Nessa operação é possível realizar pesquisa por valores que não são chave, porém isso implica em uma busca comparativa registro-a-registro.

Para uma melhor performance, o *Scan* deve somente ser utilizado quando a *Query* não for possível. Para diminuir o tempo de resposta do *Scan* e *Query*, o retorno das requisições são páginas com tamanho máximo de 1MB e a quantidade de registros é delimitada através de parâmetros.

- c) Desenvolvimento: definição da taxonomia, funcionamento do Amazon DynamoDB, integração com o Twitter, modelagem de dados, metodologia de desenvolvimento e o desenvolvimento do sistema
- d) Conclusão: análise da existência de informações relevantes no Twitter, da utilização do DynamoDB e do modelo NoSQL e, por fim, algumas sugestões de trabalhos futuros.

3 REDES SOCIAIS E GESTÃO DO CONHECIMENTO

Este capítulo apresenta o referencial teórico com a revisão de literatura enfocando as características das Redes Sociais, o surgimento do Twitter e Gestão do Conhecimento (GC) nas empresas.

3.1 Redes Sociais

O dicionário Oxford (OXFORD DICTIONARIES ONLINE, [21--]) define rede social como “uma rede de interações sociais e relações pessoais” que “permite os usuários de se comunicarem com os outros”. Já o site G1 (2008) define como “relação estabelecida entre indivíduos com interesses em comum em um mesmo ambiente” e o site Dictionary.com (DICTIONARY.COM, [21--]) complementa com “um site web ou serviço online que facilita a comunicação”. Segundo Rodrigo (2010), as redes sociais são utilizadas para o compartilhamento de interesses e objetos em comum. Para Powell (2010), o termo refere-se a uma comunidade onde pessoas estão conectadas de alguma forma.

Com o crescimento da Web 2.0 (NATIONS, [21--]), os sites de redes sociais começaram a se popularizar. Para Boyd e Ellison (2007), um site de rede social (*Social Network Site* ou SNS em inglês) permite que os usuários tenham um perfil (público ou semi público) e mantenham uma lista de usuários com quem compartilham conteúdo. Após a entrada em uma rede social, o usuário necessita informar outros usuários com os quais mantém relações. Essas relações normalmente são bidirecionais, mas alguns sites possuem relações unidirecionais.

Como através da web qualquer um pode escrever um artigo e publicá-lo em tempo real, surge o termo mídia social. Segundo Comm (2009), esse termo é uma forma de publicação em que as histórias são trocadas e os profissionais de marketing colocam suas mensagens diante das pessoas que criam uma rede de contatos.

Após analisar uma centena de sites, Jain (2010) listou os 40 SNSs mais populares na internet, nos primeiros quatro lugares estão:

- a) Facebook: criada por Mark Zuckerberg com o objetivo de ser uma rede social para os alunos de Harvard. Tornou-se de uso público em 2006 e passou da marca de 500 milhões de usuários.
- b) MySpace: iniciado em 2003 e ainda mantém uma larga base de usuários.
- c) Twitter: começou como um microblog, mas se transformou numa plataforma de mensagens fornecendo notícias e tendências.
- d) LinkedIn: rede social para uso profissional, ajudando na busca de empregos e conexão com parceiros de negócios.

O termo microblog é um novo estilo mídia social no qual os membros possuem limites estritos para o conteúdo, incentivando a criatividade das postagens. Mesmo existindo uma série de sites na internet, entre eles Spink e Yammer, o Twitter estabeleceu um padrão para essa categoria.

3.2 Twitter

O Twitter foi criado em julho 2006 pelos programadores Evan Williams, Jack Dorsey e Biz Stone. Segundo o próprio site, o Twitter é uma rede de informações em tempo real que conecta o usuário às últimas histórias, ideias, opiniões e notícias sobre o que ele acha interessante (TWITTER, [21--]).

Após a abertura da conta no Twitter, os usuários podem postar mensagens curtas de até 140 caracteres chamadas de *Tweets*, que podem ser lidas por qualquer outro usuário do serviço (HUBERMAN, ROMERO e WU, 2008). Cada usuário declara seus interesses seguindo outros usuários, e é notificado quando os seus seguidos postam novas mensagens. Kwak, *et al.* (2010) explica que os relacionamentos não necessitam ser recíprocos, o fato de seguir um usuário não implica em que esse usuário siga de volta. As mensagens podem ser reenviadas para fora da lista de seguidores do remetente através do mecanismo de *retweet*.

Segundo Comm (2009), um dos motivos da popularidade da ferramenta foi permitir que os usuários enviassem mensagens através de mensagens SMS diretamente de aparelhos celulares sem conexão com a internet. Segundo Carlson (2011), em março de 2011, o site possuía 175 milhões de usuários registrados, deste total somente 29 milhões eram usuários ativos (com ao menos um relacionamento). Em outubro de 2011, o diretor executivo Dick

Costolo, comentou em um evento em São Francisco que são enviados diariamente 250 milhões de Tweets, um crescimento de 175% comparado com setembro de 2010 (TSOTSIS, 2011).

3.3 Gestão do Conhecimento

O conhecimento, segundo Tuomi (*apud* Silva, 2004), está no topo da hierarquia formada por dados, informação e conhecimento. Os dados são fatos sem interpretação, as informações são dados estruturados de forma compreensível e o conhecimento é a informação utilizada em um contexto. Nonaka e Takeuchi (1997) complementam que a informação é um meio para construir o conhecimento. Dessa forma, Gestão do Conhecimento (GC) é um assunto amplo e, segundo Terra (2005), pode ser resumida como:

A Gestão do Conhecimento centra-se em três aspectos principais: foco nos ativos intangíveis (principalmente o fator humano), tornar a gestão do conhecimento algo explícito, incentivar e criar mecanismos que facilitem aos empregados o compartilhamento de seus conhecimentos.

Segundo Monteiro (2008), o conhecimento está dividido em duas categorias:

- a) Conhecimento Explícito: é independente do contexto e está ligado aos procedimentos, marcas e patentes e, segundo Choo (2006), como é expresso através de símbolos é mais facilmente difundido.
- b) Conhecimento Tácito: é o conhecimento prático relacionado a um contexto e está ligado à experiência das pessoas, para Choo (2006), este conhecimento é “usado pelos membros da organização para realizar seu trabalho e dar sentido ao seu mundo”.

Como o conhecimento explícito é mais facilmente documentado e reproduzido, a maior vantagem competitiva das empresas está em seu capital humano. Para Terra (1999), o conhecimento tácito não pode ser copiado e leva tempo para ser formado, sendo assim, a GC está ligada a como as empresas utilizam e combinam os conhecimentos de cada funcionário. Para Santiago Jr. (2004), cada vez que um conhecimento é reutilizável ele aumenta de valor, tornando-se mais valioso e poderoso que os ativos físicos ou financeiros.

A GC nas empresas serve para criar e transferir o conhecimento em benefício da empresa. Segundo Moura (*apud* BENEDETTI, 2008):

“Gestão do Conhecimento é um conjunto de estratégias que visa colocar o conhecimento em ação, através de sistemas e processos que possibilitem as pessoas contribuírem para o conhecimento coletivo da empresa e dele retirarem o que necessitam para o seu desenvolvimento e, ao mesmo tempo, para o aperfeiçoamento das operações organizacionais”.

Segundo Benedetti (2008), o fluxo da gestão é dividido em seis fases:

- a) Mapeamento: consiste do levantamento das informações úteis.
- b) Aquisição, captura e criação: apropriação ou combinação das informações mapeadas.
- c) Empacotamento: é a incorporação da informação em meio físico (papel ou eletrônico).
- d) Armazenamento: processo de identificar e recuperar dados associados com o armazenamento da informação.
- e) Comunicação: é a etapa de compartilhar a informação através de grupos de trabalho.
- f) Inovação: surge a modificação do ambiente gerando novos produtos, e modificando o processo de produção.

Segundo Nonaka e Takeuchi (1997), a criação e expansão do conhecimento são feitas através da conversão do conhecimento tácito em conhecimento explícito, em consequência das interações sociais entre as pessoas. Existem diversas práticas de GC, entre elas: Análise de Redes Sociais, Taxonomia, Mapeamento de Processos, Comunidade de Prática, Ferramentas de Colaboração e Bases de Conhecimento.

3.4 Taxonomia e Folksonomia

O principal exemplo de Taxonomia é a Classificação dos Seres Vivos de 1735, de Karl von Linné, que organizou os seres vivos de forma hierárquica de acordo com características em comum. Segundo Terra, *et al.* (1998), taxonomia “é um sistema para classificar e facilitar o acesso à informação”. No ambiente corporativo é utilizada para organizar informações não estruturadas.

Para Benedetti (2008), o uso permite que as pessoas utilizem os mesmos termos para um fato ou situação, diminuindo ambiguidades nas informações da organização. Timon, *et al.* (2009) complementa afirmando que uma classificação hierárquica favorece o agrupamento e categorização do conhecimento explícito da organização. Outra vantagem, segundo Vogel (2009), é a rapidez para localização das informações desejadas.

Para a construção de uma Taxonomia, deve existir colaboração entre especialistas e arquitetos da informação, respeitando os seguintes critérios (TERRA, *et al.*, 1998):

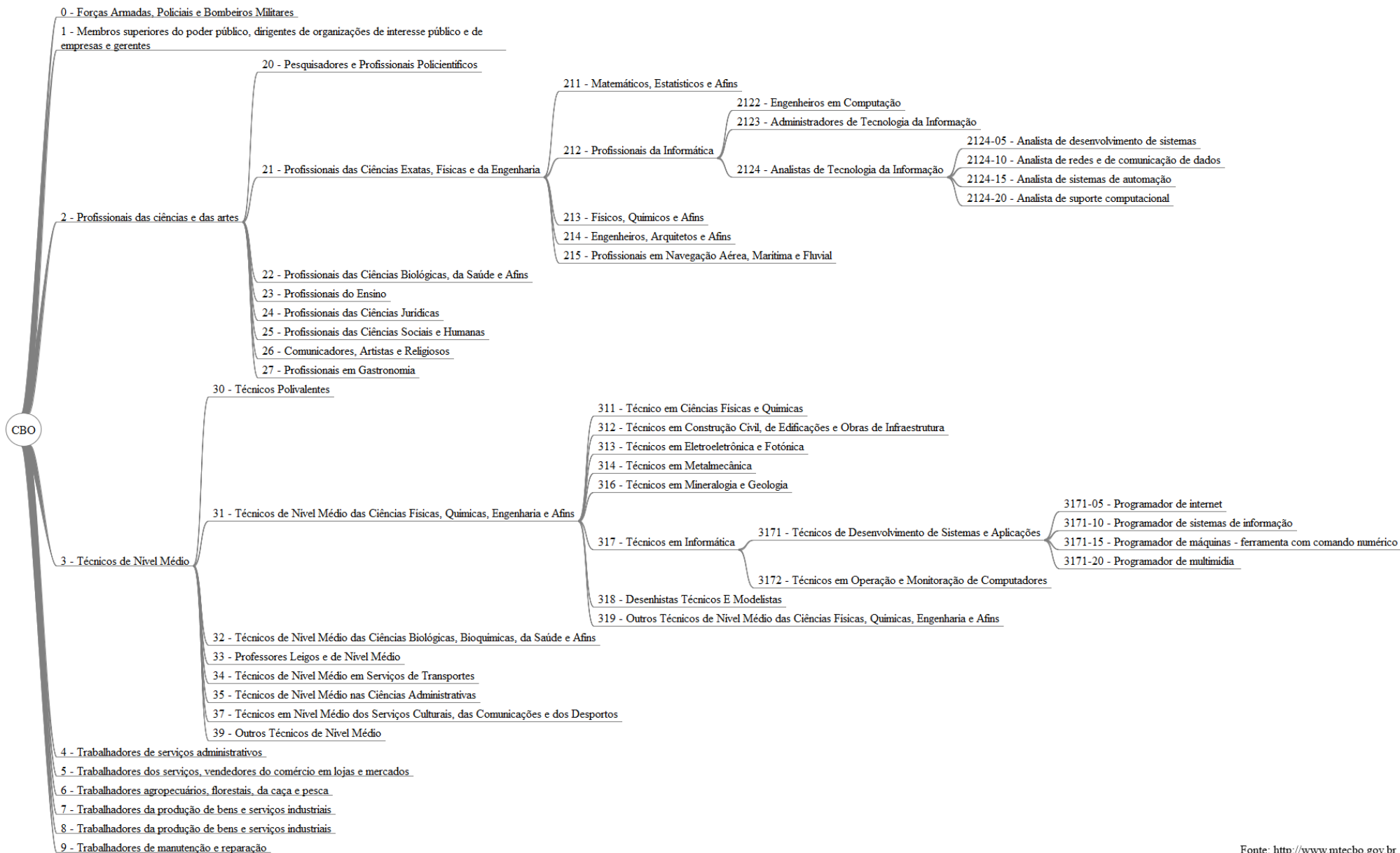
- a) Comunicabilidade: os termos devem estar claros e no contexto dos usuários.
- b) Utilidade: possuir somente os termos necessários.
- c) Estimulação: apresentar termos que induzem o usuário a continuar utilizando o sistema.
- d) Compatibilidade: ela deve conter somente termos do contexto dos usuários.

Um exemplo de taxonomia é a Classificação Brasileira de Ocupações (CBO), apresentada de forma reduzida na Figura 1.

O termo Folksonomia foi criado por Thomas Vander Wal em 2004, com a junção da palavra *folk* (pessoas em inglês) e da palavra taxonomia. Nesta forma de organização cada usuário organiza o conhecimento da sua maneira, não sendo necessário um especialista e um vocabulário controlado. Para Kato e Gledson (2009), a Folksonomia apresenta algumas vantagens, entre elas:

- a) Flexibilidade: um vocabulário rígido não consegue ser ágil para lidar com bases de informação que crescem muito rápido.
- b) Identificação de padrões: em conjunto com a mineração de dados, permite que os padrões de organização da informação possam ser compartilhados entre os usuários.
- c) Colaboração: existe um “filtro social colaborativo”, se muitas pessoas classificam a informação com o mesmo termo, a informação pode ser encontrada mais facilmente.

Ainda para Kato e Gledson (2009), a Folksonomia apresenta alguns pontos negativos, todos em virtude da falta de vocabulário estruturado, entre eles estão: uso de termos em singular e plural, utilização de sinônimos, erros de grafia e utilização de palavras com significados diferentes de acordo com o contexto. Por causa do dinamismo dos termos, as buscas são contextualizadas e personalizadas, princípios da Web 3.0 (ALVES, 2007).



Fonte: <http://www.mtecbo.gov.br>

Figura 1: Classificação Brasileira de Ocupações

4 BANCO DE DADOS

Segundo Date (DATE, 2004, p. 10), “um banco de dados é uma coleção de dados persistentes”, onde persistentes significa que não são transitórios, ficando armazenados enquanto uma instrução não os remova.

Os primeiros programas eram responsáveis por fazer a interface com o usuário, processamento e armazenamento físico dos dados (HEUSER, 1998). Para facilitar o desenvolvimento e a manutenção de aplicações, surgiram no início da década de 1970 os Sistemas de Gerência de Banco de Dados (SGBD¹). No final desta década e início da seguinte, começaram a parecer no mercado os SGDB relacionais². Mesmo com a entrada dos sistemas orientados a objeto nas décadas de 1980 e 1990, o modelo relacional continua sendo o mais utilizado (DATE, 2004, p. 24-25).

Para Elmasri e Navathe (2006), os bancos de dados relacionais possuem as seguintes características básicas:

- a) Natureza autodescritiva do sistema de BD: além dos dados armazenados, o sistema possui a definição da estrutura de dados e suas restrições.
- b) Isolamento entre os Programas e Dados e Abstração de Dados: a estrutura dos dados está armazenada e separada do programa que o acessa. O SGBD fornece uma representação conceitual de dados e não a forma como estão armazenados e o processo necessário para recuperá-los.
- c) Suporte para as Múltiplas Visões dos Dados: cada usuário pode visualizar os dados armazenados de maneira diferente.
- d) Compartilhamento de Dados e o Processamento de Transação Multiusuário: visto que múltiplos usuários podem acessar o mesmo dado simultaneamente, o SGBD possui o conceito de transação, que garante o acesso logicamente correto ao dado sem a interferência de outras transações.

¹ Em inglês, Database Management System (DBMS).

² Em inglês, Relational Database Management Systems (RDBMS).

As transações, segundo Orend (2010), possuem as propriedades de: Atomicidade, Consistência, Isolamento e Durabilidade. Essas características formam o acrônimo ACID:

- a) Atomicidade: para uma transação significa que todas as operações são executadas ou ela é abortada sem alteração nos dados.
- b) Consistência: significa que o banco está consistente antes e depois de uma transação ser executada, ou seja, uma transação não pode violar uma restrição de integridade.
- c) Isolamento: uma transação não pode ler informações de outra transação não concluída, problema relacionado à concorrência entre transações.
- d) Durabilidade: uma transação deve ser escrita no sistema de arquivos antes de ser concluída, garantindo que não haja perda de dados por falhas de hardware.

Elmasri e Navathe (ELMASRI e NAVATHE, 2006) também abordam algumas vantagens na utilização dos SGBD, entre elas:

- a) Controle de Redundância: como o mesmo dado é armazenado somente uma vez, o espaço de armazenamento é menor e não são necessárias múltiplas atualizações para garantir a consistência do dado.
- b) Restrição de Acessos Não Autorizados: devido ao fato de que nem todo usuário pode realizar as mesmas operações sobre todos os dados, o sistema possui um controle de autorização garantindo as restrições automaticamente.
- c) Estruturas para o Processamento Eficiente de Consultas: o sistema possui estruturas adicionais, armazenamento temporário (buffer) e índices, para aumentar a velocidade de pesquisa do dado.
- d) Restrições de Integridade: as restrições dos dados (tipo de campo, limites de valores, referência entre estruturas) são automaticamente verificadas e garantidas pelo SGBD.

4.1 Modelo Relacional

Para Carneiro (2004), as entidades (ou tabelas como são chamadas) são os objetos sobre os quais se deseja armazenar informações. Essas entidades possuem propriedades e relacionamentos com outras entidades que também podem conter propriedades. Devido a esses relacionamentos, existe uma correspondência entre as tabelas e o conceito matemático de relação, por isso o nome do modelo (SILBERCHATZ, KORTH e SUDARSHAN, 1999).

A modelagem dos dados “é uma descrição dos tipos de informações que estão armazenadas em um banco de dados” (HEUSER, 1998, p. 16) e para Carneiro (2004, p. 7-8), é:

Uma definição abstrata dos objetos representados por estes dados, dos relacionamentos desses objetos entre si e de um conjunto de operadores e regras que os usuários finais utilizam para interagir com o banco de dados.

A modelagem pode ser feita em diferentes níveis de abstração. Para os Bancos de Dados (BD), podem ser considerados dois níveis de abstração: o modelo conceitual e o modelo lógico. O modelo conceitual é independente do tipo de BD, a representação mais utilizada nos BD relacionais é o diagrama entidade-relacionamento (DER). No modelo lógico, os dados como tabelas e suas colunas são descritos de forma textual.

Em 1974, a IBM criou uma linguagem para acessar o seu BD relacional chamada de Linguagem de Consulta Estruturada (*Structed Query Language* ou *SQL* em inglês). Em 1986, foram estabelecidos os padrões da linguagem pela *American National Standards Institute* (ANSI) e pela *International Standards Organization* (ISO), este primeiro padrão ficou conhecido como SQL1. Em 1992 foi revisado e chamado de SQL/92 ou SQL2. A versão mais recente é de 1999 e é conhecida como SQL:1999 ou SQL3, possuindo suporte aos BD orientados a objetos.

Segundo Carneiro (2004), é uma linguagem simples e fácil, pois permite escrever as consultas quase em linguagem natural (inglês), facilitando a utilização por não conhecedores de lógica de programação. Heuser (HEUSER, 1998) complementa que nas consultas SQL não é especificada nenhuma forma de como acessar os dados, o caminho é definido pelo próprio SGBD durante a execução da instrução.

4.2 NoSQL

Neste capítulo será apresentado um novo paradigma para a persistência de dados, principalmente voltado ao armazenamento de grandes volumes de dados que ultrapassam os terabytes. Este novo modelo vem auxiliando grandes portais como o Facebook, Twitter e Digg a migrar de seus BD MySQL para uma plataforma muito mais robusta (LAI, 2010).

Ainda não existe um padrão de tecnologia estabelecido. Segundo Fowler e Sadalage (2012), este fato gerou o termo *Poliglot Persistence*, que permite que as empresas utilizem a melhor tecnologia para cada um dos cenários existentes, não utilizando somente os RDBMS.

4.2.1 *Big Data*

Um estudo de 2010 aponta que as empresas americanas com mais de mil funcionários armazenavam em média 200 terabytes³ de dados, chegando a um petabyte⁴ em alguns setores (XAVIER, 2012). Outra pesquisa de 2008 apontou que as famílias americanas receberam 34 gigabytes⁵ de informação por pessoa a cada dia (THE ECONOMIST, 2010).

O International Data Corporation (IDC, 2008) calculou o tamanho do universo digital em 2007 como sendo de 281 exabytes⁶ e estimou 1,800 exabytes em 2011. A empresa Cisco estima que em 2013 o tráfego anual da internet será de 667 exabytes (THE ECONOMIST, 2010). Dessa realidade surge o termo Big Data que foi criado para denominar “o crescimento exponencial dos dados que as empresas precisam ou podem tratar para extrair informação útil” (SOARES, 2012).

A Computerworld descreve Big Data como “a mineração de enormes volumes de dados estruturados e não estruturados de informações úteis, usando ferramentas não-tradicionais” (COLLETT, 2011). Para o Gartner, o termo está envolvido com os fatores: Volume, Variedade e Velocidade (PETTEY e GOASDUFF, 2011):

- a) Volume: a quantidade de dados corporativos cresce devido ao aumento na quantidade de transações e no tipo de dados armazenados.
- b) Variedade: para auxiliar as decisões, os dados úteis são originários de dados tabulados, hierárquicos, documentos, e-mails, vídeo, imagens, transações financeiras, etc.

³ Um terabyte equivale a 1×10^{12} bytes

⁴ Um petabyte equivale a 1×10^{15} bytes

⁵ Um gigabyte equivale a 1×10^9 bytes

⁶ Um exabyte equivale a 1×10^{18} bytes

- c) Velocidade: envolve a velocidade que os dados são produzidos e a necessidade de processá-los de acordo com a demanda.

Collett (2011) complementa, afirmando que não são necessários os três fatores para uma solução ser considerada Big Data, dependendo do caso alguns terabytes de dados que necessitam ser processados instantaneamente já a caracterizam. Segundo Taurion (2011), o conceito de Big Data é juntar grandes volumes de informações estruturadas e não estruturadas para que as empresas tomem decisões baseadas em fatos.

Para Jacobs (2009), o que tem tornado os dados realmente grandes é a repetição de informações simples através do tempo e/ou espaço como, por exemplo, operadoras de celular armazenando a posição dos seus celulares a cada 15 segundos. Segundo Sevilla (2011), Big Data é necessário quando o volume de dados cresceu tanto que a área de Tecnologia da Informação (TI) não consegue retornar informações básicas em menos de um minuto.

Além do processamento usual dos dados, novas formas de visualização também estão sendo utilizadas, entre elas: nuvem de *tags*⁷, *clustergramas*⁸ e *history flow*⁹ (TAURION, 2011).

Para Xavier (2012), os desafios para utilizar Big Data estão em otimizar os recursos tanto no hardware quanto no software. Para o hardware, a utilização de computadores baratos para o processamento paralelo ao invés de um único servidor dedicado tem se mostrado melhor na relação custo/benefício (JACOBS, 2009). Para o software, novas maneiras de armazenar os dados com menos restrições e regras do modelo relacional começaram a ser utilizadas (BRITO, 2010). Algumas dessas soluções receberam o nome de NoSQL e assemelham-se aos sistemas de gerenciamento de arquivo.

⁷ Mais informações em http://en.wikipedia.org/wiki/Tag_cloud

⁸ Mais informações em http://www.schonlau.net/publication/02stata_clustergram.pdf

⁹ Mais informações em http://www.research.ibm.com/visual/projects/history_flow/

4.2.2 O Movimento NoSQL

O termo foi utilizado em 1998 para um BD relacional que omitiu o uso da linguagem SQL. Somente em 2009 foi utilizado novamente em uma conferência em São Francisco por defensores de um modelo não relacional. O chamado movimento NoSQL começou neste mesmo ano e vem crescendo rapidamente (STRAUCH, 2011).

A tradução mais aceita para a sigla é Não Apenas SQL (*Not Only SQL* em inglês). Para Brito (BRITO, 2010), os bancos de dados NoSQL não pretendem substituir o modelo relacional, somente apresentar uma possibilidade para casos em que não seja exigida uma estrutura rígida de dados.

Segundo Fowler (2012), o termo NoSQL se refere a um particular conjunto de BD que possuem algumas características em comum:

- a) Não utilizam o modelo relacional nem a linguagem SQL;
- b) São de código aberto;
- c) Desenhados para utilização em grandes clusters;
- d) Baseados nas necessidades web do século 21;
- e) Não existe um esquema, permitindo que campos possam ser adicionados a qualquer registro.

As características desses bancos podem ser definidas como (LÓSCIO, DE OLIVEIRA e PONTES, 2011):

- a) Escalabilidade horizontal: ao invés de utilizar uma escalabilidade vertical que necessita aumentar o processamento e armazenamento do servidor, a escalabilidade horizontal aumenta a quantidade de servidores, dividindo o problema em pequenas partes.
- b) Ausência de esquema ou esquema flexível: a ausência de esquema rígido facilita a escalabilidade e aumenta a disponibilidade, porém não existe a garantia de integridade dos dados.

- c) Suporte nativo a replicação: outra maneira de aumentar a escalabilidade é utilizando a replicação de dados.
- d) API¹⁰ simples para acesso de dados: o principal objetivo não é a forma como os dados são armazenados, e sim como obtê-los de maneira eficiente.
- e) Consistência eventual: essa característica diz que, como os dados estão distribuídos, somente se pode garantir dois dos três pontos do teorema CAP.

Segundo De Diana e Gerosa (2010), o teorema CAP significa: consistência, disponibilidade e tolerância à partição (*Consistency, Availability e Partition tolerance* em inglês). A consistência implica que a leitura de um registro após a sua atualização deve retornar o novo valor, disponibilidade é a garantia de que o serviço em funcionamento responda a todas as requisições que receber, e tolerância à partição implica que o sistema continue em funcionamento mesmo que ocorra um problema de rede particionando o serviço em duas partes.

Outro ponto em discordância ao modelo relacional é a abordagem BASE, que significa disponibilidade básica, estado leve e consistência eventual (*Basically Available, Soft state e Eventual consistency* em inglês), ao invés da necessidade de garantir as propriedades ACID como critério de correção dos dados em SGBDs multiusuário. Ippolito (*apud* STRAUCH, 2011, p. 32) resume a propriedade em: “uma aplicação funciona basicamente todo o tempo (disponibilidade básica), não necessita ser consistente todo o tempo (estado leve), mas terá um estado consistente eventual (consistência eventual)”.

Brito (2010) apresenta o escalonamento como uma das principais vantagens da arquitetura NoSQL sobre a arquitetura dos SGBDs relacionais em virtude de ter sido criado com esse objetivo. O escalonamento pode ser vertical (*scale up* em inglês) realizado através do aumento da capacidade do servidor, ou pode ser horizontal (*scale out* em inglês) realizado através do aumento na quantidade de servidores, um exemplo pode ser visualizado na Figura 2. Os bancos NoSQL adotam, preferencialmente, o escalonamento horizontal particionando os dados (*sharding* em inglês) entre os diversos servidores. O escalonamento não é feito

¹⁰ API significa em inglês Application Programming Interface

automaticamente pelo gerenciador do BD NoSQL, somente é mais facilitado com relação ao modelo relacional (TOTH, 2011).

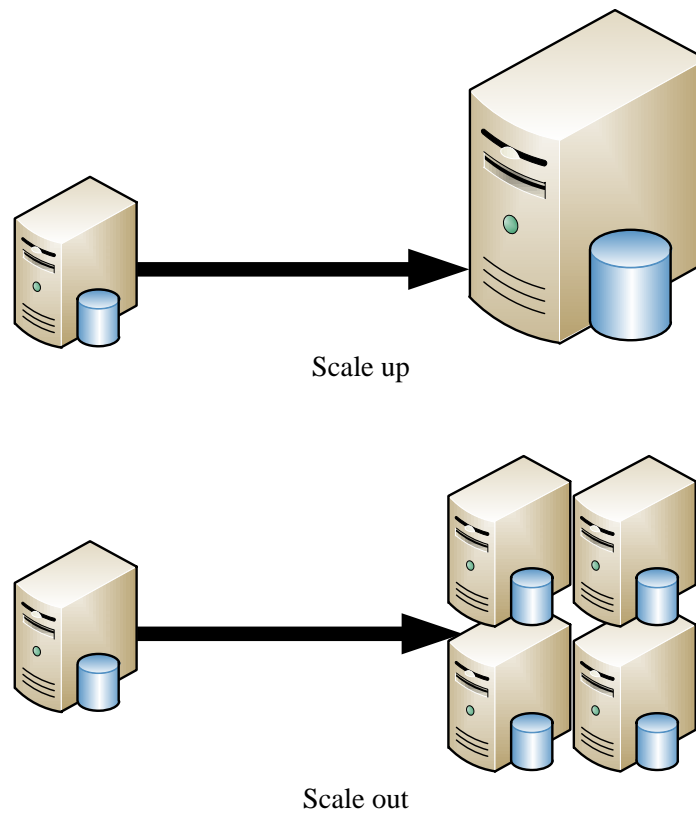


Figura 2: Escalonamento

4.2.3 Técnicas de Implementação

Segundo Lóscio, De Oliveira e Pontes (2011), para a implementação das funcionalidades, os bancos de dados NoSQL utilizam algumas técnicas, entre elas:

- a) *Map/Reduce*: permite o tratamento de grande volume de dados e é dividida em duas partes. Na primeira (Map), as solicitações são divididas em solicitações menores e distribuídas entre os nós de servidores. Já a segunda (Reduce), as solicitações são processadas nos nós filhos e devolvidas para os nós pais.
- b) *Consistent hashing*: forma de armazenamento e recuperação de dados distribuídos, sua utilização evita a migração dos dados entre servidores.

- c) *Multiversion concurrency control* (MVCC): suporte de transações paralelas sem a utilização de locks, permitindo a escrita paralela de valores.
- d) *Vector Clocks*: ordenam os eventos de um sistema e são necessários para identificar a situação atual de um dado, em virtude de que o mesmo pode estar sendo atualizado concorrentemente.

4.3 Categorias de NoSQL

Existem mais de 120 bancos de dados NoSQL (NOSQL, [21--]) que podem ser agrupados em quatro categorias: chave-valor, orientado a documentos, orientado a colunas e orientado a grafos.

4.3.1 Chave-valor (*key-value*)

Essa categoria também é conhecida como tabela de hash distribuída, é um modelo considerado simples e de fácil implementação, pois os dados são armazenados e localizados através de chaves (*hash*), utilizando o conceito de mapa ou dicionário de dados (DE DIANA e GEROSA, 2010).

O armazenamento de chave-valor existe há bastante tempo (STRAUCH, 2011), mas um dos principais bancos de dados desse modelo é o DynamoDB que foi utilizado como base para o Cassandra (LÓSCIO, DE OLIVEIRA e PONTES, 2011). Outras opções são os bancos: Project Voldemort, Memcache, Redis, RIAK, Scalaris e Tokyo Cabinet.

4.3.1.1 *DynamoDB*

O DynamoDB é um dos bancos de dados fornecido pela Amazon em uma estrutura própria de milhares de servidores espalhados pelo mundo.

Mesmo os servidores possuindo capacidades diferentes, cada um dos nós possui a mesma responsabilidade, a estrutura garante a divisão do trabalho de acordo com a

capacidade de processamento do servidor. Quando um novo servidor é adicionado ao anel, vários nós virtuais são criados com capacidades fixas. A quantidade de nós virtuais está ligada a capacidade de processamento do servidor (STRAUCH, 2011).

As informações sobre a adição, remoção ou falha em nós é transmitida através do protocolo Gossip. Esse protocolo, segundo Tiwari (2011), é baseado na forma como as fofocas e rumores se espalham nas redes sociais onde são necessárias interações periódicas entre pares de processos.

Segundo Strauch (2011), o DynamoDB foi criado para ser um sistema de consistência eventual, ou seja, quando uma operação de atualização é executada, o retorno da operação acontece antes de todos os nós serem atualizados, ocasionando que leituras posteriores possam retornar versões diferentes dos dados.

Quando um nó recebe uma requisição de atualização de dados, uma nova versão do dado é gerada com um *vector clock* diferente. Esse nó grava as informações localmente e passa a informação para o próximo nó que também realiza a atualização. A quantidade de nós que recebe a atualização é configurada pelo administrador do sistema (*consistent hashing*). Para o sistema, as informações são vistas como um conjunto de dados. Sendo assim, caso aconteça alguma inconsistência entre os dados, o desenvolvedor do sistema pode interagir para informar qual dado deve ser persistido. Caso nada seja informado o sistema utiliza a abordagem de “*last write wins*”, ou seja, permanece o último dado enviado. Por este motivo, o DynamoDB é caracterizado como um sistema MVCC.

A seção 5 apresenta o detalhamento sobre o banco de dados Dynamo, uma vez que trata-se do BD avaliado neste trabalho.

4.3.1.2 Project Voldemort

Nasceu como solução Java para o LinkedIn (STRAUCH, 2011), mas atualmente possui licença de código aberto (CATTELL, 2010).

Para Strauch (2011), diferentemente do modelo relacional, no qual os programadores podem utilizar *stored procedures* para aumentar o desempenho, misturando a camada lógica com a camada de persistência, na utilização de um modelo chave-valor essa separação é feita obrigatoriamente. A documentação do sistema sugere o particionamento dos dados, bem

como a utilização de *caches* para aumentar a performance devido às operações de E/S no disco.

Segundo Cattell (2010), o particionamento de dados é automático através do conceito de *Consistent hashing* onde os dados são replicados num número definido (pelo administrador) de servidores e as atualizações são assíncronas. Por isso, o sistema não garante a consistência dos dados. Quando novos servidores são adicionados ou removidos, o sistema automaticamente se adapta. Além disso, o gerenciador também detecta e recupera automaticamente falhas nos nodos.

Strauch (2011) explica que o Project Voldemort utiliza, assim como o DynamoDB, o mecanismo de concorrência através do uso de *vector clocks*. Caso ocorram conflitos durante a atualização dos dados, o sistema tenta resolver o problema, se não for possível, o usuário deve resolver o problema na próxima requisição do dado.

4.3.1.3 Memcache, Membrain e Membase

Possui uma distribuição de código-livre, utilizando indexação em memória. As funcionalidades de persistência, replicação, alta disponibilidade, crescimento dinâmico, entre outras foram acrescentadas por Schooner Technologies and MemBase (CATTELL, 2010).

O Memcache é compatível com outros sistemas como o Membrain e o Membase. O Membrain é licenciado por servidor pela Schooner Technologies e seu diferencial é a possibilidade de ajuste fino (*tunning*) utilizando memória *flash*. Membase é distribuído pela empresa Membase e seu diferencial está na capacidade de adicionar ou remover servidores enquanto o sistema está em funcionamento, redirecionando as requisições para outros servidores durante o processo.

Segundo Tiwari (2011), o principal motivo de utilização do Memcache é diminuir o acesso ao BD, por isso é utilizado pelo Facebook, Twitter, Wikipedia e YouTube.

4.3.1.4 Redis

O projeto começou a ser desenvolvido na linguagem C por apenas uma pessoa e atualmente está sobre licença BSD (CATTELL, 2010). Os dados são armazenados na memória RAM do servidor e são copiados para o disco de backup quando há a necessidade de desligamento do sistema. Ao acrescentar novos nós, o desligamento do sistema é obrigatório.

Segundo Tiwari (2011), todos os dados são armazenados no Redis na forma de *strings*, inclusive listas, conjuntos e mapas. Para isso, ele utiliza um sistema próprio de gerenciamento de memória, que ao gravar os dados em disco armazena na memória o endereço físico do dado junto ao hash.

Para Strauch (2011), a maior desvantagem do Redis é que o tamanho dos dados é limitado ao tamanho da memória disponível no servidor.

4.3.1.5 RIAK

Foi escrito na linguagem Erlang e tornou-se um código-aberto em 2009 pela empresa Basho que o descreve como um banco orientado a chave-valor e orientado a documento (CATTELL, 2010). A biblioteca cliente está disponível, entre outras linguagens, em Erlang e Java e segue o padrão web de requisições HTTP RESTful¹¹.

RIAK suporta replicação e *sharding* dos objetos armazenados no formato JSON¹², porém não são suportados índices diferentes da chave-primária.

¹¹ REST significa em inglês REpresentation State Transfer

¹² JSON significa em inglês JavaScript Object Notation

4.3.1.6 *Scalaris*

Assim como o RIAK, Scalaris também foi escrito na linguagem Erlang, seu código aberto foi desenvolvido pelo Zuse Institute de Berlim (CATTELL, 2010).

Os dados são armazenados em memória, mas a replicação síncrona e a recuperação de falhas garantem a durabilidade das atualizações. Para Strauch (2011), o Scalaris possui um modelo de consistência rígida e permite consultas complexas através da sua biblioteca.

4.3.1.7 *Tokyo Cabinet*

A parte servidor é chamada de Tokyo Cabinet e a biblioteca cliente é nomeada Tokyo Tyrant, ambos escritos na linguagem C, fizeram parte de um projeto do Sourcefourge.net, mas atualmente é licenciado e mantido pelo FAL Labs. (CATTELL, 2010).

O servidor possui seis variações: índices *hash* em memória ou em disco, árvores-B em memória ou em disco, tabelas de tamanho fixo ou variável. Strauch (2011), apresenta o gerenciamento dos dados no servidor como similar ao processo de paginação e *swapping*.

Segundo Cattell (2010), não possui documentação muito clara, mas possui suporte a transações ACID, replicação assíncrona entre dois nodos principais ou entre principal/escravo, a recuperação é manual e não possui *sharding* automático.

4.3.2 *Orientado a Colunas*

Segundo Lóscio, De Oliveira e Pontes (2011), nesse modelo os dados são indexados pela tripla formada por linha, coluna e *timestamp*. A linha e a coluna são identificadas por chaves e o *timestamp* é um diferenciador entre as múltiplas versões dos dados. Também existe o conceito de família de colunas (*column family*) que é utilizado para agrupar colunas de mesmo tipo.

Para De Diana e Gerosa (2010), os bancos de dados relacionais são mais adequados aos sistemas de processamento transacionais (*Online Transactional Processing* ou OLTP em

inglês), enquanto os bancos orientados a coluna se enquadram melhor para os sistemas de processamento analítico (*Online Analytical Processing* ou OLAP em inglês).

Segundo Cattell (2010), que denomina esses sistemas como *Extensible Record Stores*, a escalabilidade desses sistemas é resultado da divisão simultânea das linhas e colunas entre diversos servidores.

Os principais bancos nessa categoria são: BigTable, Hypertable, HBase e Cassandra.

4.3.2.1 BigTable

O BigTable da Google, criado em 2004, foi uma das primeiras implementações de um sistema não relacional com o objetivo de promover maior escalabilidade e disponibilidade (BRITO, 2010). Ele foi construído sobre a plataforma Google File System (GFS) e é utilizado em mais de 60 produtos da empresa entre eles: Google Analytics, Orkut e Google Earth (PADHY, PATRA e SATAPATHY, 2011).

Para Orend (2010), o BigTable é comparável ao armazenamento de chave-valor, com a diferença de que ao invés de mapear uma chave com um valor é mapeado uma tripla formada pela linha, coluna e *timestamp*.

Segundo Strauch (2011), as linhas possuem uma chave de até 64KB, são ordenadas alfabeticamente, são particionadas e armazenadas nos chamados *tablets*. As colunas são agrupadas em famílias de colunas e não existe um limite de quantidade de colunas por tabelas. O *timestamp* é representado por um inteiro de 64 bits e é utilizado para diferenciar as versões de uma célula de valor. Além disso, o *timestamp* pode ser gerado pelo servidor ou informado pela API e também é utilizado para ordenar as alterações dos valores, mantendo o valor mais atual como sendo o primeiro a ser lido.

O GFS é utilizado para a persistência dos arquivos e dos logs. Os dados são armazenados utilizando o formato SSTable, que é um mapa persistente e imutável de chaves no qual as aplicações podem iterar sobre todas as chaves/e valores, entre um determinado intervalo de chaves.

Segundo Orend (2010), o *sharding* é realizado atribuindo o tablet a um único servidor. Um servidor principal armazena as informações de mapeamento entre o *tablet* e o servidor na qual foi armazenado. Devido ao *tablet* ser armazenado em somente um servidor, o processo

de replicação não é feito diretamente pelo BigTable e sim pelo próprio GFS. Por não existir replicação, a consistência do dado é garantida.

O único ponto possível de falha da estrutura é o serviço Chubby, que é constituído de cinco servidores (sendo um *master* e quatro clientes). O serviço é responsável, entre outras tarefas, por distribuir os *tablets* entre os demais servidores do cluster. Sendo assim caso o serviço fique inacessível, o BigTable não consegue atualizar seus mapeamentos e também ficará indisponível depois de um tempo.

O BigTable não suporta uniões, sendo assim os relacionamentos devem ser tratados na camada de aplicação. A API é fornecida na linguagem C++.

4.3.2.2 *HyperTable*

Para Cattell (CATTELL, 2010), o HyperTable é muito similar ao BigTable e HBase, foi escrito em C++ e possui código aberto pela Zvents. Além de possuir uma quantidade ilimitada de colunas nas famílias de colunas, os dados são particionados e replicados em memória para depois serem escritos no sistema de arquivos distribuídos da Apache: o Hadoop (*Hadoop Distributed File System* ou HDFS em inglês).

Segundo Orend (2010), possui uma linguagem própria de acesso aos dados chamada de *Hypertable Query Langue* (HQL) e a API pode ser acessada através de Thrift ou C++.

4.3.2.3 *HBase*

É um projeto da Apache escrito em Java e que foi patentado logo após o BigTable. Sua principal diferença está em utilizar a técnica de Map/Reduce, sob o sistema de arquivo Hadoop, ao invés do Google File System (OREND, 2010).

No HBase, os dados são escritos em memória e posteriormente gravados em arquivos periodicamente compactados. O controle de concorrência é otimista, gerando erro caso exista conflito. O particionamento e a distribuição são transparentes, não necessitando de informações adicionais (CATTELL, 2010). Além da API via JAVA, também pode ser acessado através de REST e Thrift.

Segundo Orend (2010), a principal utilização do HBase é no sistema de mensagens em tempo real construído para o Facebook em 2010.

4.3.2.4 Cassandra

O Cassandra é escrito em Java, licenciado pela Apache e tornou-se código livre em 2008. Foi desenvolvido por um engenheiro do Facebook e um engenheiro do projeto Dynamo (CATTELL, 2010), sendo inspirado no BigTable e no próprio Dynamo.

Segundo Cattell (2010), o Cassandra, assim como os Hypertable e HBase, realiza as alterações em memória para posteriormente atualizar o disco que é compactado periodicamente. O processo de detecção e recuperação de falhas é totalmente automatizado, assim como a detecção de novos nós no cluster que utiliza o protocolo Gossip. Um dos pontos fracos do sistema é o sistema de concorrência, que não possui *locks* e as réplicas são atualizadas assincronamente.

Strauch (2011) caracteriza uma instância do Cassandra como uma única tabela que representa “um mapa multidimensional distribuído indexado por uma chave”, onde as dimensões podem ser:

- a) Linha: que são identificadas por uma chave literal de tamanho definido.
- b) Coluna: possui um nome e armazena um número de valores por linha identificados por um *timestamp*. Cada linha pode possuir uma quantidade diferente de colunas.
- c) Supercoluna: também é identificada por um nome e possui uma quantidade variante de colunas de acordo com a linha.
- d) Família de Colunas: É formada por colunas e supercolunas que podem ser adicionadas dinamicamente.

O acesso aos dados de uma linha é atômico, não importando a quantidade de colunas que devem ser lidas ou gravadas. Quando uma requisição é feita, o sistema identifica qual servidor possui a informação. Para a inclusão e alteração de dados, a replicação é feita numa quantidade de servidores definida pelo administrador do sistema. Para otimizar a performance, o sistema armazena índices de colunas e família de colunas para obter os dados diretamente do disco sem necessitar de uma varredura completa de todas as colunas.

A API do Cassandra está disponível em Java, Ruby Python, C#, Thrift entre outras.

4.3.3 Orientado a Documentos

Um documento é um objeto com um identificador único e um conjunto de campos, onde os documentos não possuem um esquema rígido, permitindo que cada documento tenha seu próprio conjunto de campos (LÓSCIO, DE OLIVEIRA e PONTES, 2011).

Os principais bancos no mercado são: CouchDB, SimpleDB e MongoDB.

4.3.3.1 SimpleDB

Segundo Cattell (2010), o SimpleDB é parte da nuvem proprietária pertencente a Amazon e contratada na forma de serviço. O seu nome origina-se de possuir um modelo simples com poucas operações sobre os documentos

O sistema utiliza consistência eventual, realizando replicação assíncrona dos documentos, mas, diferentemente de outros bancos, suporta múltiplos agrupamentos (domínios) dentro da mesma base de dados. Cada domínio pode possuir um conjunto diferente de índices que são atualizados automaticamente quando um documento é atualizado. O SimpleDB não realiza o *sharding* automaticamente, mas cada domínio pode estar armazenado em um nó diferente da Amazon.

4.3.3.2 CouchDB

Foi desenvolvido na linguagem Erlang e pertence a Apache desde 2008, possui interface em diversas linguagens, entre elas: Java, C, PHP e Python. Segundo Strauch (2011), o CouchDB é um descendente do Lotus Notes, já que foi iniciado pelo mesmo desenvolvedor Damien Katz.

A estrutura básica de armazenamento é um documento constituído de campos e valores; são permitidas referências entre documentos através de URL ou URI, porém nenhuma verificação de consistência é realizada.

Segundo Cattell (2010), o CouchDB possui muitas semelhanças com o SimpleDB, entre elas o fato de não realizar *sharding*, ou seja, ao atualizar um documento, precisa ser replicado em todos os servidores para garantir a consistência eventual. O sistema possui suporte a índices secundários que precisam ser criados manualmente e as consultas podem ser paralelizadas entre diversos nós utilizando o mecanismo de Map/Reduce.

O controle de concorrência é realizado através da técnica de MVCC, utilizando identificadores de sequência para cada versão dos documentos. Caso o documento tenha sido alterado por outro usuário desde a sua leitura, a aplicação é notificada e pode combinar ou descartar as suas alterações. Após a confirmação da alteração (*commit*), os dados são gravados em disco para manter a durabilidade dos documentos.

Segundo Strauch (2011), o processo de replicação é incremental e é capaz de determinar conflitos de versões entre os servidores. A arquitetura foi desenhada para que cada servidor tenha a mesma responsabilidade, não existindo um mestre e um escravo. Para dois servidores começarem a se replicar basta que um localize o outro na rede. É possível realizar uma replicação parcial dos dados desde que sejam configurados manualmente filtros em cada servidor.

Como o sistema constantemente está gravando as alterações em disco, periodicamente o arquivo deve ser compactado para restaurar espaço para o sistema de arquivos. Esse processo consiste de copiar todo o conteúdo válido para um novo arquivo. Durante a cópia o arquivo original continua sendo utilizado para pesquisa e alteração. No caso de algum problema acontecer durante a cópia, o arquivo original continua íntegro.

4.3.3.3 MongoDB

Escrito na linguagem C++, é fornecido pela empresa 10gen e possui licença GPL. Cattell (2010) define o MongoDB como sendo similar ao CouchDB, mas com a grande diferença de ter suporte nativo a *sharding*. Além disso, o uso de índices de consulta é automático como em um RBDMS. Outra diferença está no controle de concorrência, enquanto o CouchDB utiliza MVCC, o MongoDB fornece operações atômicas nos campos. As operações atômicas realizam as alterações na memória local. Quando as alterações são

enviadas para o servidor, ele verifica se mesmos documentos não foram alterados por outros usuários

Os dados são armazenados num formato chamado BSON¹³, semelhante ao JSON. Além disso, é utilizado uma especificação chamada GridFS que permite armazenar longos dados binários, como, por exemplo, vídeos. O MongoDB suporta replicação com *failover*¹⁴ e recuperação automática.

Segundo Tiwari (2011), este banco é uma boa escolha para realizar a migração entre o modelo relacional e o NoSQL, uma vez que suporta consultas no estilo SQL, tem um controle rudimentar de referência e a modelagem de dados é semelhante, utilizando o conceito de tabela e coluna.

Segundo Strauch (2011), o banco não possui chaves estrangeiras. Sendo assim o relacionamento é feito manualmente onde um campo de referência é utilizado para informar o caminho completo até o documento de destino. Outra maneira é o chamado *Database Reference* (DBRef), que permite a referência com documentos de coleção externa ao documento atual. Porém, a forma mais eficiente de definir relações entre documentos é através do aninhamento de documentos onde um documento é colocado dentro do outro.

Desde a versão 1.5.2, as operações de atualização e exclusão de documentos não realizam bloqueio, sendo assim alterações em lote podem gerar inconsistência entre os dados. Para amenizar o problema, o programador pode atribuir um *flag*, indicando que a operação deve ser realizada de forma atômica.

Para otimizar o processo de consultas, o usuário pode definir índices que são automaticamente calculados. Por causa do custo de processamento durante a atualização dos dados, os índices são indicados para coleções que possuem mais leitura do que gravação. Caso a coleção tenha mais gravação que leitura, a criação de índices pode prejudicar a performance.

Uma das grandes empresas que utilizam o MongoDB está a Foursquare. Os motivos pela adoção da ferramenta estão, normalmente, ligados ao modelo flexível de dado e a

¹³ BSON é baseado no termo JSON e significa em inglês Binary JSON

¹⁴ Failover é o processo no qual uma máquina assume os serviços de outra, quando esta última apresenta falha.

velocidade de leitura e gravação dos dados (TIWARI, 2011). O banco possui bibliotecas clientes em várias linguagens, entre elas: C, C#, Java, JavaScript, Pearl, PHP, Python e Ruby.

4.3.4 Orientado a Grafos

O modelo de grafos possui três elementos básicos: os vértices dos grafos (nós), as arestas (relacionamentos) e os atributos (propriedades dos nós e dos relacionamentos) (LÓSCIO, DE OLIVEIRA e PONTES, 2011). O BD é um multigrafo rotulado e direcionado, onde cada nó está relacionando com vários outros nós através das arestas.

Para De Diana e Gerosa (2010), este padrão está ligado diretamente ao modelo de grafos, onde os dados são representados como grafos dirigidos ou estruturas que generalizam a noção de grafos. O padrão suporta restrições através de integridade referencial.

Segundo Angles e Gutierrez (*apud* DE DIANA e GEROSA, 2010), o modelo orientado a grafos é mais interessante quando:

Informações sobre a interconectividade ou a topologia dos dados é mais importante, ou tão importante quanto, os dados propriamente ditos.

Um dos principais bancos dessa categoria é o Neo4j, OrientDB e DEX.

4.3.4.1 Neo4j

Segundo Vicknair, *et al.* (2010), o banco foi escrito em Java pela empresa Neo Technology e está em produção desde 2005. Além disso, possui licença *open source* para uso não comercial. O Neo4j foi construído utilizando um mecanismo de busca textual de alta performance desenvolvido pela Apache chamado de Lucene.

Matsushita e Nguessan (2012) explicam que os dados são armazenados em cada vértice utilizando uma tabela de chave-valor, permitindo assim que cada vértice possua informação diferente dos demais. Como todo o banco é visto com uma grande estrutura de grafos, o processo de aprendizagem e visualização de dados específicos pode ser mais complicado do que no modelo relacional.

Uma das vantagens do Neo4J, perante as outras ferramentas NoSQL, é a garantia das propriedades ACID. Outra vantagem está na economia de processamento e baixo tempo de resposta das informações relacionadas, isto devido ao processo de busca em profundidade a partir de um vértice específico.

4.3.4.2 *OrientDB*

É um banco *open source*, escrito em Java e mistura funcionalidades dos modelos orientados a documento e orientados a grafos. Suporta transações ACID e pode trabalhar tanto com *schema* rígido quanto flexível. Um dos motivos pelo qual tem isso bastante utilizado é o suporte a SQL, que permite a migração de programadores acostumados com o modelo relacional (OrientDB, [21--]).

4.3.4.3 *DEX*

É um sistema desenvolvido em Java e C++ pela Sparsity Technologies que o denomina como um *Labeled Directed Attributed Multigraph*. *Labeled* (rotulado em inglês), pois os vértices e arestas do grafo pertencem a um tipo, *Directed* (direcionado em inglês), uma vez que as arestas podem ser direcionadas ou não, *Attributed* (atribuída em inglês), pois permite que os vértices e arestas possuam atributos (SPARSITY TECHNOLOGIES, [21--]).

DEX suporta transações, garantindo somente consistência e isolamento.

4.3.5 *Comparativo entre os Bancos*

Na Tabela 1 é apresentado um resumo com as principais características dos bancos discutidos nas seções anteriores.

Tabela 1: Resumo dos bancos de dados

Orientação	Banco de Dados	Controle de Concorrência	Armazenamento	Replicação	Transação	Linguagem
Chave-Valor	DynamoDB	MVCC	Plug-in	Assíncrona	Não	
Chave-Valor	Membrain	Lock	Flash + disco	Síncrona	Local	
Chave-Valor	Memcache	Lock		Síncrona	Local	
Chave-Valor	Redis	Lock	RAM	Assíncrona	Não	C
Chave-Valor	RIAK	MVCC	Plug-in	Assíncrona	Não	Erlang
Chave-Valor	Scalaris	Lock	RAM	Síncrona	Local	Erlang
Chave-Valor	Tokyo	Lock	RAM ou disco	Assíncrona	Local	C
Chave-Valor	Voldemort	MVCC	RAM ou disco	Assíncrona	Não	Java
Colunas	BigTable	Lock + timestamp	GFS	Assíncrona e Síncrona	Local	
Colunas	Cassandra	MVCC	Disco	Assíncrona	Local	Java
Colunas	HBase	Lock	Hadoop	Assíncrona	Local	Java
Colunas	HyperTable	Lock	Aquivos	Síncrona	Local	C++
Documentos	CouchDB	MVCC	Disco	Assíncrona	Não	Erlang
Documentos	MongoDB	Lock	Disco	Assíncrona	Não	Java
Documentos	SimpleDB	Nenhum	S3	Assíncrona	Não	
Grafo	DEX	Lock			ACID Parcial	Java, C++
Grafo	Neo4j	Lock	Disco		ACID	Java
Grafo	OrientDB	Lock e MVCC	Disco		ACID Parcial	Java, C++

Fonte: Tabela baseada em resumo disponível em Cattell (2010).

Tabela 2: Site oficial dos produtos

Orientação	Banco de Dados	Site
Chave-Valor	DynamoDB	http://aws.amazon.com/pt/dynamodb/
Chave-Valor	Membrain	http://www.schoonerinfotech.com/products/membrain/index.php
Chave-Valor	Memcache	http://memcached.org/
Chave-Valor	Redis	http://try.redis-db.com/
Chave-Valor	RIAK	http://wiki.basho.com/
Chave-Valor	Scalaris	http://code.google.com/p/scalaris/
Chave-Valor	Tokyo	http://fallabs.com/tokyocabinet/
Chave-Valor	Voldemort	http://www.project-voldemort.com
Colunas	BigTable	http://labs.google.com/papers/bigtable.html
Colunas	Cassandra	http://incubator.apache.org/cassandra/
Colunas	HBase	http://hbase.apache.org/
Colunas	HyperTable	http://hypertable.org/
Documentos	CouchDB	http://couchdb.apache.org
Documentos	MongoDB	http://www.mongodb.org/
Documentos	SimpleDB	http://aws.amazon.com/pt/simpledb/
Grafo	DEX	http://www.sparsity-technologies.com/dex.php
Grafo	Neo4j	http://www.neo4j.org/
Grafo	OrientDB	http://www.orientdb.org/index.htm

Na Tabela 2 são apresentados os endereços das páginas oficiais dos produtos.

5 BANCO DE DADOS DYNAMO

O Dynamo foi apresentado em 2007 baseado nos seguintes requisitos e pressupostos (DECANDIA, *et al.*, 2007):

- i) Modelo de consulta: operações de leitura e escrita de dados identificados por uma chave, não permitindo operações em múltiplos registros e nem esquema relacional.
- j) Propriedade ACID: experiências na Amazon mostraram que para prover as características ACID é necessário ter uma baixa disponibilidade, no entanto a disponibilidade é um dos requisitos principais do SGBD. Sendo assim, o objetivo era diminuir a consistência se isso trouxesse maior disponibilidade.
- k) Eficiência: o sistema deve funcionar em uma infraestrutura formada por hardware de baixo custo.
- l) Uso Interno: somente utilizado na infraestrutura interna da Amazon, onde o ambiente é considerado seguro não necessitando de autenticação e autorização.
- m) Escalabilidade Incremental: o sistema deve ser capaz de aumentar o processamento horizontalmente no menor tempo possível na operação e nos sistemas adjacentes.
- n) Simétrico: cada nodo da infraestrutura tem as mesmas responsabilidades, não existindo nodos com funções ou responsabilidades distintas.
- o) Descentralização: além de simétrico, a descentralização *peer-to-peer* deve ser utilizada ao invés de controle centralizado.
- p) Heterogenia: o sistema deve ser capaz de explorar melhor a heterogenia da infraestrutura, ou seja, o processamento do nodo deve ser proporcional às suas capacidades.

5.1 Desenho da Solução

Para que o Dynamo possa ser particionado, replicado, versionado e escalável, foram utilizadas algumas técnicas que são listadas na Tabela 3: Técnicas de implementação.

Tabela 3: Técnicas de implementação

Problema	Técnica	Vantagens
----------	---------	-----------

Particionamento	Consistent Hashing	Escalabilidade incremental
Alta disponibilidade na gravação	Vector clocks com reconciliação na leitura	O tamanho da versão está dissociado da taxa de utilização
Recuperação de falhas permanentes	Antientropia utilizando árvores Merkle	Sincronização de réplicas em <i>background</i>
Sócios e detecção de falhas	Protocolo Gossip-based membership	Mantém a simetria e evita um registro centralizado de informações

Fonte: Tabela baseada em resumo disponível em DeCandia, *et al.* (2007).

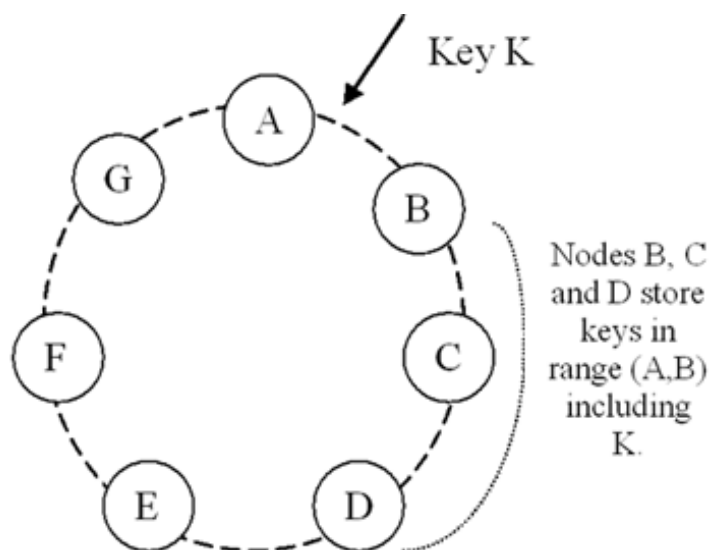
5.2 Particionamento e Replicação

Para realizar o particionamento dos dados é utilizada a técnica de *consistent hashing* onde a função *hash* é tratada como um espaço circular fechado ou anel. A cada nodo é atribuído um valor aleatório que representa a sua posição no anel. Para cada chave de valor é calculado o valor do *hash* que corresponde a uma posição no anel. Com base nesta posição, o anel é percorrido no sentido horário até encontrar a primeira posição maior que a calculada, sendo assim cada nó é responsável pela região no anel entre o seu predecessor e ele mesmo (o processo pode ser visto na Figura 3).

A versão padrão de *consistent hashing* apresenta dois problemas:

- c) A distribuição aleatória de nós gera uma distribuição não uniforme de dados e processamento.
- d) O algoritmo é feito para um ambiente homogêneo de nós.

Para resolver esses problemas foi utilizada uma variação do algoritmo que utiliza o conceito de nós virtuais. Cada nó físico pode gerar múltiplos nós virtuais (de acordo com a capacidade do servidor), que são mapeados no anel.



Fonte: <http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>

Figura 3: Particionamento e Replicação de dados

O processo seguinte é a replicação dos dados em múltiplos servidores, parte fundamental para atingir alta disponibilidade e durabilidade. Após identificar o nó de armazenamento (conforme visto na etapa de particionamento) o dado é armazenado localmente e a operação é confirmada ao programa chamador. Assincronamente este servidor realiza a atualização do dado em nós que estão ao seu alcance, no exemplo da Figura 3 estes nós são o C e D.

Para evitar falhas de hardware, a replicação não pode ocorrer em nós virtuais pertencentes ao mesmo servidor físico.

5.3 Versionamento dos Dados

O Dynamo é um sistema de consistência eventual, ou seja, a operação é dada como terminada após atualizar o dado no nó coordenador. As atualizações das réplicas são feitas de forma assíncrona. Neste cenário pode acontecer de uma nova requisição de leitura retornar o valor anterior a última atualização.

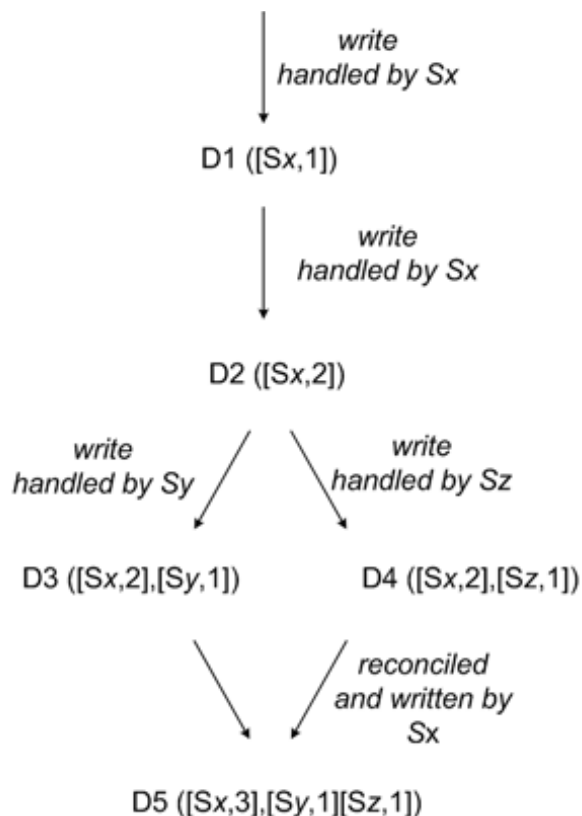
Para prevenir problemas de divergência de cada alteração num registro, gera-se uma nova versão imutável do dado, permitindo que múltiplas versões do dado existam ao mesmo tempo. Na maioria das vezes, as novas versões se sobrepõem automaticamente a versões

anteriores, porém, quando existem múltiplas versões paralelas do registro, o sistema não consegue resolver o conflito, sendo necessário realizar a união entre os dois registros.

Para o controle do versionamento de cada registro é utilizada a técnica de *vector clocks*. Cada versão do registro recebe uma lista de tuplas (*vector clocks*), através da comparação de dois *clocks* é possível determinar a ordem entre os registros ou se houve uma ramificação entre eles.

A Figura 4 apresenta um exemplo onde pode ser analisada a ramificação de um registro. Nela um determinado registro é alterado pelo o usuário S_x gerando a versão D1, em seguida o mesmo usuário altera o registro criando a versão D2. A ramificação ocorre quando os usuários S_y e S_z realizam a alteração no mesmo momento, criando duas versões distintas D3 e D4, respectivamente. No passo final, quando o usuário S_x vai realizar uma nova alteração, precisa realizar a reconciliação entre as versões criando a versão D5.

Os valores apresentados entre parâmetros representam o *vector clock*, apresentando qual foi o caminho que a versão percorreu.



Fonte:
<http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>

Figura 4: Versionamento de registros

5.4 O produto Amazon DynamoDB

O Amazon DynamoDB foi lançado como produto em 18 de janeiro de 2012 e é o resultado de 15 anos de aprendizado nas áreas de banco de dados não relacionais em larga escala e serviços em nuvem. O Amazon DynamoDB foi baseado no Dynamo que também serviu de base para outros bancos de dados. O Dynamo foi originalmente desenhado para ser utilizado nas operações chaves de e-commerce da Amazon, como o carrinho de compras e o serviço de seção (VOGELS, 2012).

Com base nisso, o Amazon DynamoDB é um banco de dados NoSQL que fornece rapidez e performance, além de ser totalmente gerenciado pelo administrador. Ele automaticamente distribui os dados e o tráfego das tabelas a um número de servidores que seja capaz de lidar com as requisições dos clientes. Os dados são armazenados em discos *Solid State Disks* (SSD) e são automaticamente replicados através de múltiplas regiões para prover alta disponibilidade e durabilidade (AMAZON, [21--]).

O DynamoDB possui as seguintes características:

- g) Provisionamento de transferência: Durante a criação ou edição da tabela, a capacidade de requisição desejada pode ser especificada. O SGBD se encarrega de alocar os recursos necessários para garantir a capacidade especificada. O provisionamento é feito através da quantidade de unidades de 1KB que se deseja gravar ou ler.
- h) Escalabilidade de armazenamento: Não existe um limite de quantidade de dados armazenados. O serviço automaticamente aloca mais espaço quando necessário.
- i) Distribuição Total: O escalonamento horizontal e automatizado replica a mesma tabela em vários servidores.
- j) Construído à prova de falhas: O processo automático e assíncrono de replicação de dados através de múltiplos servidores espalhados em diferentes regiões garante, além de alta disponibilidade, também a proteção dos dados contra falhas individuais ou coletivas de hardware.
- k) Consistência forte e contadores atômicos: Diferente de outros BD não relacionais, o DynamoDB facilita o desenvolvimento para garantir a consistência forte durante a leitura, retornando sempre o último valor do registro. Além disso, a API fornece chamadas para incremento e decremento de contadores de forma atômica.

- l) Flexibilidade: O BD não possui um esquema fixo de dados, o que significa que cada registro de uma tabela pode ter um número diferente de atributos e de tipos de dados.

5.4.1 Operações de Banco de Dados

A API fornece as operações de criação, atualização e exclusão de tabelas. A atualização da tabela permite o aumento ou diminuição do provisionamento de transferência. Cada tabela é formada por uma chave única (campo *Hash Key*) e pode ou não ter uma variação (campo *Range Key*). O conteúdo de um atributo pode ser: número, literal, conjunto de números ou conjunto de literais. O tamanho em bytes de cada registro é definido pelo somatório do tamanho dos nomes dos campos mais o tamanho binário dos dados.

Também são fornecidos métodos para adicionar, atualizar e excluir registros das tabelas. Durante a atualização de itens é possível modificar valores e adicionar ou remover colunas. Para otimizar as buscas, pode ser utilizada uma operação de retorno de um ou múltiplos itens através de sua chave primária, inclusive em múltiplas tabelas.

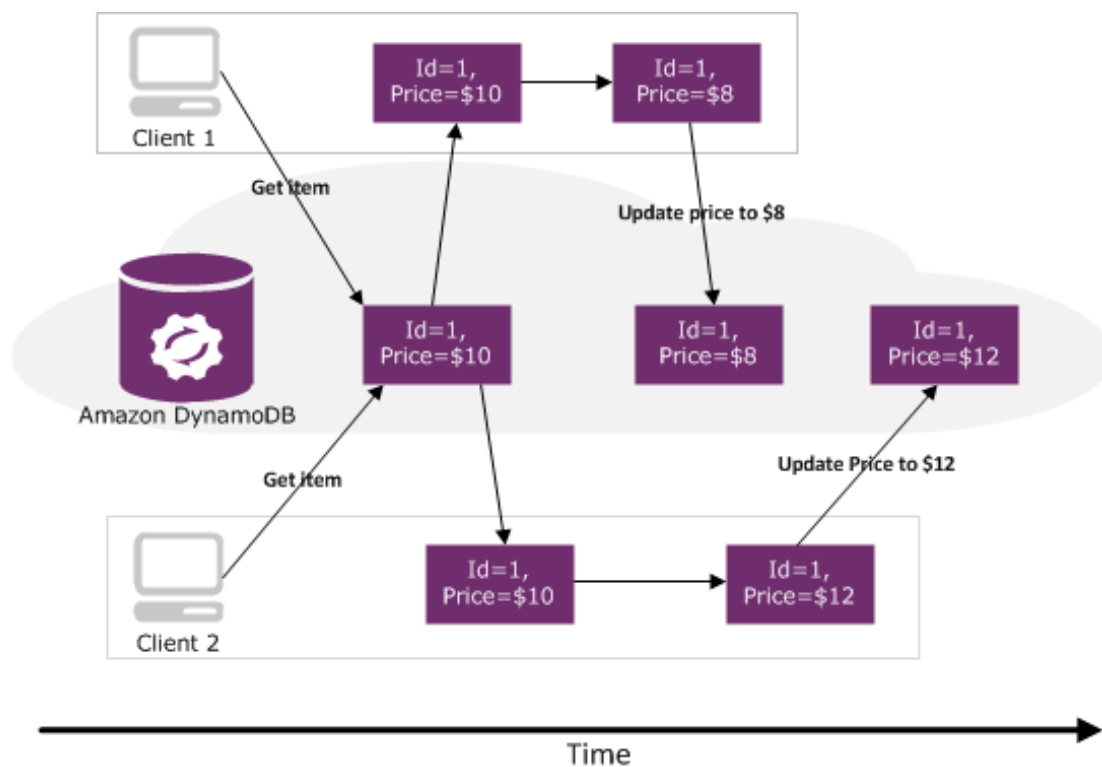
5.4.1.1 Leitura e consistência

O SGBD mantém múltiplas cópias de cada item para garantir durabilidade, para que isso aconteça após uma operação de alteração de dados é necessário que o dado seja gravado em múltiplos servidores, o que demora algum tempo. Essa demora faz com que o dado fique temporariamente inconsistente, ou seja, caso uma leitura seja feita imediatamente o valor antigo pode ser retornado. Essa é a forma padrão de leitura. Mas em alguns casos, é necessário utilizar uma leitura consistente, para isso o DynamoDB retorna o mais recente que reflita todas as operações de escrita. Essa forma de leitura é mais suscetível à lentidão da rede.

5.4.1.2 Controle de Concorrência

Em um ambiente multiusuário, em alguns casos, é necessário garantir que a atualização de um usuário não afete a gravação de outro. Para isso, o DynamoDB suporta a escrita condicional, que nada mais é do que a verificação de valor gravado antes de realizar a gravação de um novo valor. Para contadores de valor, é possível utilizar operações atômicas que incrementam ou decrementam valores sem sofrerem interferências de outras operações de gravação.

A Figura 5 apresenta uma simulação de como os dados são solicitados e gravados, Nela o registro é lido simultaneamente pelos usuários “Client 1” e “Client 2”, enquanto o “Client 1” realiza a alteração do valor para 8, o “Client 2” realiza a alteração para 10 e ambos são atualizados no BD, gerando duas versões do registro.



Fonte: <http://aws.amazon.com/pt/documentation/dynamodb/>

Figura 5: Leitura e gravação no BD

5.4.2 Consultas

Para consultas, existem dois mecanismos: *Query* e *Scan*. A *Query* permite a consulta na tabela utilizando o campo *Hash* e, opcionalmente, um filtro de *Range*. Esse mecanismo é o mais eficiente para buscar itens na tabela.

A operação de *Scan* é a forma menos eficiente, pois realiza uma varredura em todos os dados da tabela. Nessa operação é possível realizar pesquisa por valores que não são chave, porém isso implica em uma busca comparativa registro-a-registro.

Para uma melhor performance, o *Scan* deve somente ser utilizado quando a *Query* não for possível. Para diminuir o tempo de resposta do *Scan* e *Query*, o retorno das requisições são páginas com tamanho máximo de 1MB e a quantidade de registros é delimitada através de parâmetros.

6 DESENVOLVIMENTO

O desenvolvimento de um software conceito, que apresenta uma solução não convencional utilizando um banco NoSQL ao invés de um BD relacional, tem como objetivo oferecer às organizações uma arquitetura que possa ajudá-las a organizar os seus conhecimentos e auxiliar na tomada de decisão.

Neste trabalho, o Twitter é utilizado apenas como uma fonte de informações para o software conceito. Para uma solução corporativa é preciso identificar as fontes de informação a serem utilizadas como: intranet, documentos, e-mails, mensagens instantâneas ou outras fontes que a própria organização dispõe, bem como o cruzamento dessas informações internas com informações externas, entre elas as redes sociais como, por exemplo, o próprio Twitter ou Facebook.

Neste capítulo é apresentada a especificação da arquitetura de dados NoSQL para indexação de Tweets com a apresentação da abordagem utilizada para a definição da taxonomia, modelagem de dados e desenvolvimento de um aplicativo.

6.1 Taxonomia

A taxonomia foi definida utilizando termos relacionados a redes sociais (Figura 6), tecnologia em banco de dados (Figura 8) e gestão do conhecimento (Figura 7).

Cada tag pode possuir nenhum ou múltiplos sinônimos. Os sinônimos servem para prevenir erros de grafia, formas alternativas de escrever o mesmo termo ou detalhamento de siglas. Foram definidos dois tipos distintos de tags:

- a) Não Pesquisável: a tag não é utilizada para pesquisar, mas caso ela exista no texto será marcada. Nas figuras abaixo estão escritas em preto com fundo cinza.
- b) Normal: a tag é utilizada para pesquisa e marcação. Estão escritas em branco com fundo preto nas figuras abaixo.

Foi criada esta separação porque existem termos aplicáveis a qualquer contexto, como por exemplo: os nomes das redes sociais e linguagens de programação. Por isso os termos pesquisáveis estão vinculados à gestão do conhecimento e SGBDs.



Figura 6: Taxonomia de Redes Sociais

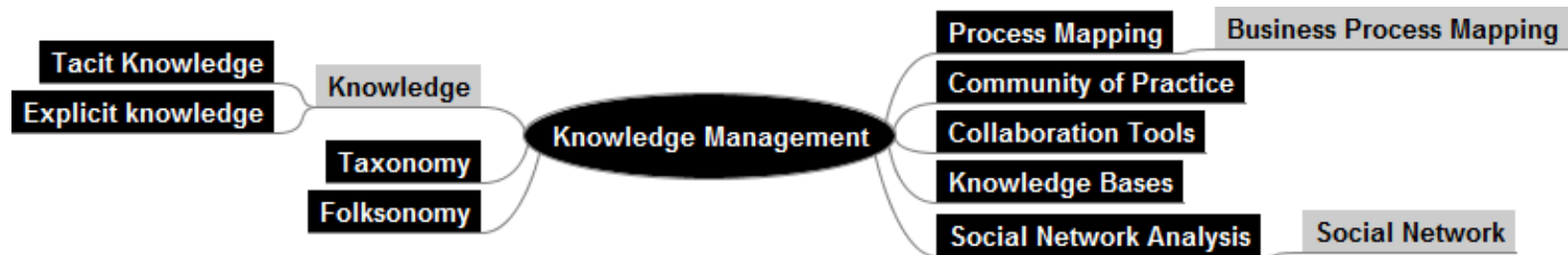


Figura 7: Taxonomia de Gestão do Conhecimento

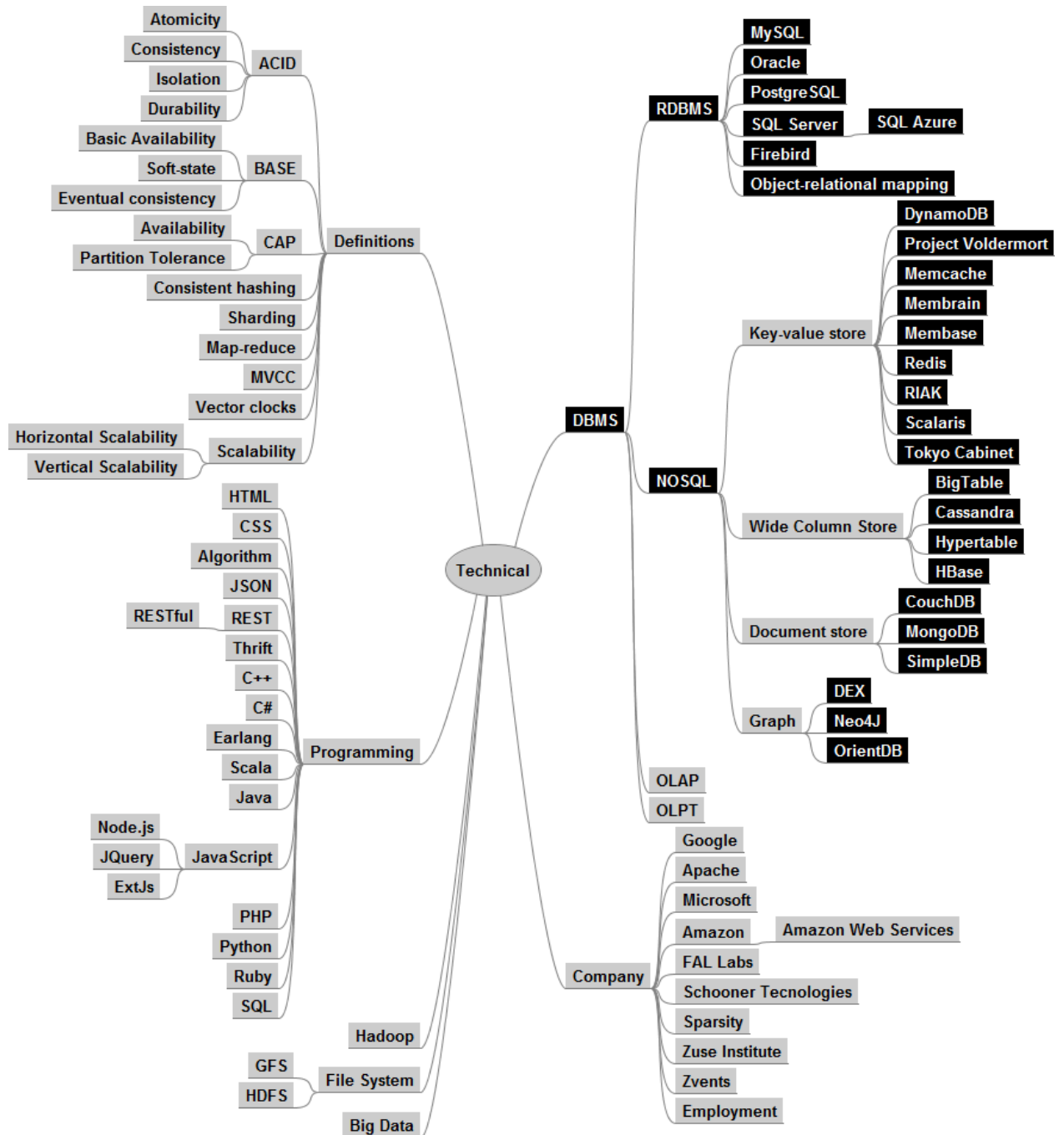


Figura 8: Taxonomia Técnica

Com o objetivo de facilitar à localização de mensagens, quando um texto for marcado com uma tag, todas as tags superiores também serão identificadas para fins de busca. Segue um exemplo de mensagem:

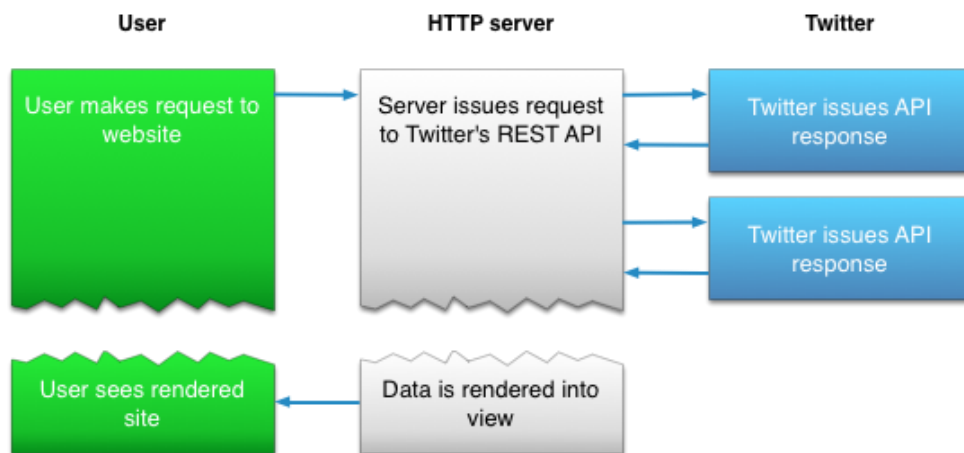
“new version of #Marvin #dynamoDB #Amazon #NoSQL #BigData #Taxonomy #Twitter C# #Jquery <http://t.co/XalPaeaZ>”

Com base na taxonomia definida, a mensagem será marcada com duas listas de tags:

- a) Presentes no texto: essa lista contém as que estão escritas no texto, são elas: Amazon, Big Data, C#, DynamoDB, JQuery, NOSQL e Taxonomy, Twitter.
- b) Hierárquicas: essa lista é gerada através das tags presentes no texto, mas para cada uma delas é acrescentada as respectivas tags de níveis superiores, são elas: Amazon, Big Data, C#, DBMS, DynamoDB, JavaScript, JQuery, Key-value store, Knowledge Management, NOSQL, Programming, SNS, Taxonomy, Twitter.

6.2 Integração com o Twitter

Para a obtenção de Tweets, o Twitter fornece duas APIs distintas: REST API e Streaming API (TWITTER, [21--]). A diferença básica entre as duas APIs é que a conexão HTTP na Streaming é mantida aberta, o que envolve uma utilização diferente pela aplicação, enquanto que na REST, cada requisição de mensagens abre e fecha uma conexão HTTP (Figura 9). Na Streaming API, a conexão HTTP é mantida por um processo paralelo e os Tweets são atualizados por notificações (Figura 10).



Fonte: <https://dev.twitter.com/docs>

Figura 9: REST API do Twitter

Mesmo a Streaming API sendo mais complexa do que a REST, ela possui como benefício a utilização em uma infinidade de aplicativos como o Hootsuite¹⁵ ou a própria interface web e mobile do Twitter¹⁶.

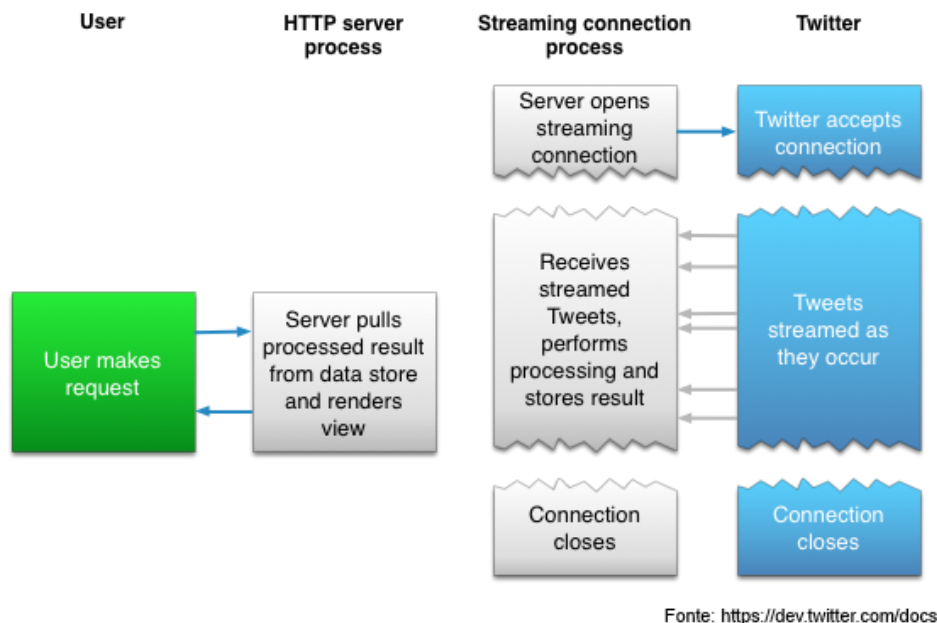


Figura 10: Streaming API do Twitter

A Streaming API fornece um acesso de baixa latência ao fluxo global de Tweets. A streaming funciona através do envio de mensagens indicando novos Tweets. São fornecidos três endereços de acordo com a necessidade:

- Public Streams:** utilizado para seguir usuários específicos ou tópicos, e para mineração de dados;
- User Streams:** retorna os dados de apenas um usuário, utilizado para apresentar uma visão;
- Site Streams:** utilizado por servidores que necessitam conectar com o Twitter em nome de muitos usuários.

¹⁵ Disponível em <http://hootsuite.com>

¹⁶ Disponível em <https://twitter.com/>

Para a importação das mensagens de acordo com a taxonomia, foi utilizado a Public Streams que suporta uma série de parâmetros passados através de HTTP POST¹⁷. Para obter as mensagens desejadas, foi utilizado o parâmetro *track*. Esse parâmetro é uma lista de frases separadas por vírgula. Cada frase pode ter entre 1 a 60 caracteres e um ou mais termos separados por espaço. O Tweet será retornado se ao menos uma das frases possuir todos os seus termos (em ordem e ignorando a caixa alta ou baixa). As frases são comparadas através de UTF-8, levando em consideração até a acentuação.

Para a conexão com a API, é necessário autenticação que pode ser feita de duas formas: Básica e OAuth. Para o desenvolvimento nesse trabalho, optou-se por utilizar a autenticação básica que necessita um usuário e senha válidos do Twitter. Uma conexão pode ser desconectada pelo servidor quando outra conexão é realizada com as mesmas credenciais.

Para evitar super-congestionamento, as conexões podem receber o código HTTP 420 (*Rate Limited*). Essa situação pode acontecer através de sucessivas desconexões e reconexões (por exemplo, para troca de parâmetro). Para evitar que o IP seja bloqueado por um período de tempo, é importante que ao receber uma notificação 420, as tentativas de novas conexões sejam suspensas por alguns minutos.

6.3 Modelo de Dados

Para a construção das tabelas de dados foi utilizado o modelo conceitual definido na Figura 11.

¹⁷ HTTP POST é uma das muitas maneiras de requisitar informações através do protocolo HTTP. Mais informações em <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

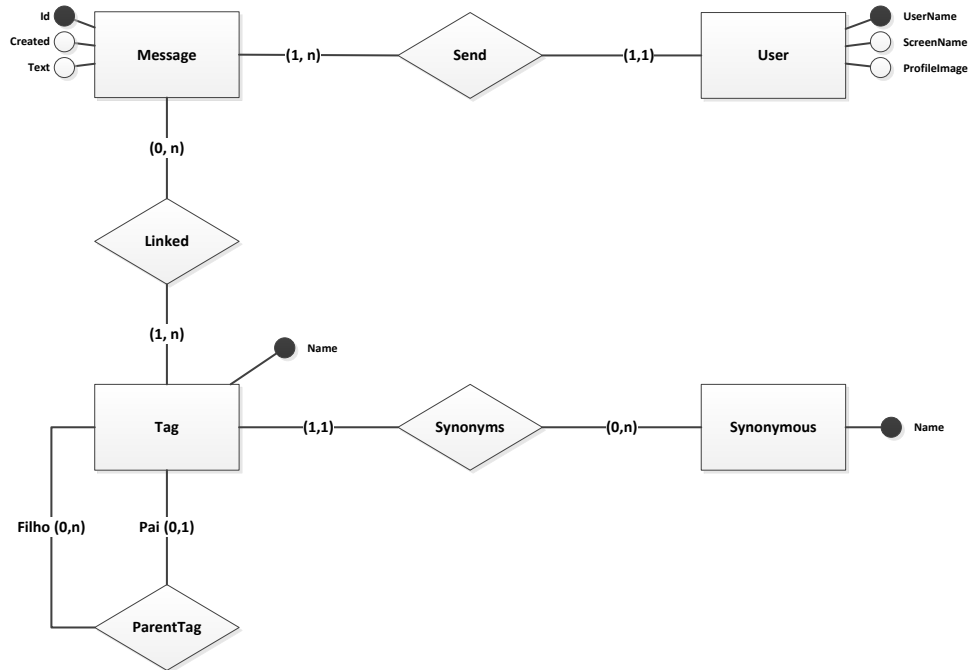


Figura 11: Modelo Conceitual

O BD não trabalha com o conceito de chave primária (*Primary Key* ou PK em inglês), para cada tabela deve ser definida uma Hash Key (HK) composta por um único campo de valor primitivo (literal ou numérico). Para algumas tabelas pode ser necessário criar a *Range Key* (RK) que é utilizada para criar uma variação de valores sobre a *Hash Key*.

Como não existe chave estrangeira (*Foreign Key* ou FK em inglês) as tabelas não estão diretamente ligadas, mas via programação existe o relacionamento entre *Message.UserName* e *User.Name*, assim como cada item de *Message.Tags* se relaciona com *Tag.Name*.

Com base no modelo, foi criada a estrutura de tabelas no DynamoDB conforme a Figura 12. O campo *Message.Tags* foi definido como um conjunto (*array*) de textos.

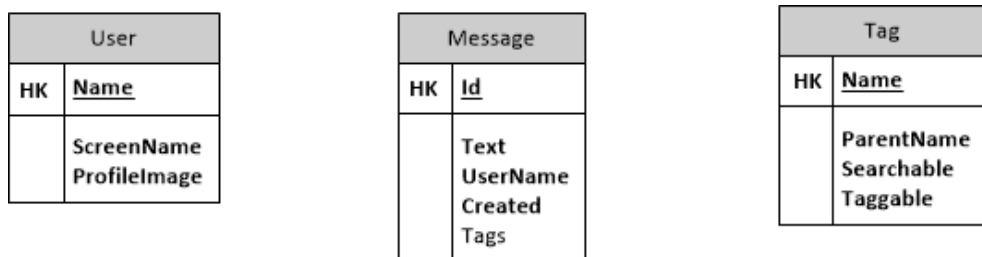


Figura 12: Primeira versão das tabelas

Conforme o projeto foi sendo executado, essa estrutura mostrou-se um pouco lenta, pois para cada mensagem era necessário buscar o usuário. Para otimizar a performance, as

tabelas User e Message foram unidas, desnormalizando assim os usuários. A Figura 13 apresenta como as tabelas ficaram após a alteração.

Message	
HK	<u>Id</u>
	Text UserName Created Tags ScreenName ProfileImage

Tag	
HK	<u>Name</u>
	ParentName Searchable Taggable

Figura 13: Segunda versão das tabelas

Novamente durante teste de execução da aplicação o tempo de respostas estava alto, excedendo 5 minutos nas filtragens por tags e data, pois neste processo ainda era necessário realizar um *Scan*. Para isso, foram criadas três tabelas de índices: *Message_by_Created*, *Message_by_Tag* e *Message_by_TagCreated*, conforme a Figura 14. Outras alterações foram a criação do campo *Message.HierarchicalTags* que representa toda a hierarquia de Tags, não somente as tags escritas no texto e exclusão do campo *Tag.Taggable* já que todas as tags poderão ser identificadas e marcadas.

Message	
HK	<u>Id</u>
	Text Created UserName ScreenName ProfileImage Tags HierarchicalTags

Tag	
HK	<u>Name</u>
	ParentName Searchable

Message_by_Created	
HK RK	<u>CreatedDate</u> <u>Id</u>

Message_by_Tag	
HK RK	<u>KeyTags</u> <u>Id</u>

Message_by_TagCreated	
HK RK	<u>KeyTags_CreatedDate</u> <u>Id</u>

Figura 14: Terceira versão das tabelas

O resultado disso é que ao gravar um registro na tabela *Message* também são gravados outros registros para indexação de dados nas tabelas:

- a) *Message_by_Created*: gravado 1 registro como índice por data de criação

- b) *Message_by_Tag*: gravados no máximo 2^n-1 registros, onde n é o número de tags hierárquicas, como índice por tag hierárquicas
- c) *Message_By_TagCreated*, gravados no máximo 2^n-1 registros, onde n é o número de tags hierárquicas, como índice por tag hierárquicas e data de criação

Um exemplo de registros pode ser visualizado Tabela 4 (no formato JSON), ela apresenta um registro na tabela *Message* e como são criados os registros vinculados nas tabelas demais tabelas.

De acordo com os parâmetros de entrada são adotadas estratégias diferentes para a busca das mensagens. Quando são informados tags e data é utilizada uma busca por *HK* na tabela *Message_by_TagCreated* para obter os Id das mensagens e buscá-las em lote na tabela de *Message*. O processo é semelhante a quando é informado somente tags ou somente data, é buscado nas tabelas *Message_by_Tag* ou *Message_by_Created*, respectivamente, antes de buscar na tabela *Message*. Sem essas três tabelas de índices o processo de filtragem não seria possível, já que o tempo de resposta era altíssimo. A busca sem parâmetros continua realizando um *Scan* na tabela *Message*.

```

Message {
  "Id": "312312312",
  "Text": "Marvin utiliza Taxonomy para buscar mensagens do Twitter",
  "Created": "2012/11/19 14:00 GMT",
  "Username": "calielc",
  "ScreenName": "Caliel Costa",
  "ProfileImage": "http://...",
  "Tags": ["Taxonomy", "Twitter"],
  "HierarchicalTags": ["Taxonomy", "Knowledge Management", "Twitter", "Social
Network Site"]
}

Message_by_Created {"CreatedDate": "2012/11/19 GMT", "Id": "312312312"}

Message_by_Tag {"KeyTags": ["Knowledge Management"], "Id": "312312312"}
Message_by_Tag {"KeyTags": ["Knowledge Management", "Social Network Site"], "Id":
  "312312312"}
Message_by_Tag {"KeyTags": ["Knowledge Management", "Twitter"], "Id": "312312312"}
Message_by_Tag {"KeyTags": ["Taxonomy"], "Id": "312312312"}
Message_by_Tag {"KeyTags": ["Taxonomy", "Social Network Site"], "Id": "312312312"}
Message_by_Tag {"KeyTags": ["Taxonomy", "Twitter"], "Id": "312312312"}
Message_by_Tag {"KeyTags": ["Social Network Site"], "Id": "312312312"}
Message_by_Tag {"KeyTags": ["Twitter"], "Id": "312312312"}

Message_by_TagCreated {"KeyTags_CreatedDate": ["Knowledge Management"] + "2012/11/19
  GMT", "Id": "312312312"}
Message_by_TagCreated {"KeyTags_CreatedDate": ["Knowledge Management", "Social
  Network Site"] + "2012/11/19 GMT", "Id": "312312312"}
Message_by_TagCreated {"KeyTags_CreatedDate": ["Knowledge Management", "Twitter"] +
  "2012/11/19 GMT", "Id": "312312312"}
Message_by_TagCreated {"KeyTags_CreatedDate": ["Taxonomy"] + "2012/11/19 GMT", "Id":
  "312312312"}
Message_by_TagCreated {"KeyTags_CreatedDate": ["Taxonomy", "Social Network Site"] +
  "2012/11/19 GMT", "Id": "312312312"}
Message_by_TagCreated {"KeyTags_CreatedDate": ["Taxonomy", "Twitter"] + "2012/11/19
  GMT", "Id": "312312312"}
Message_by_TagCreated {"KeyTags_CreatedDate": ["Social Network Site"] + "2012/11/19
  GMT", "Id": "312312312"}
Message_by_TagCreated {"KeyTags_CreatedDate": ["Twitter"] + "2012/11/19 GMT", "Id":
  "312312312"}

```

Tabela 4: Gravação de registros

6.4 Metodologia de Desenvolvimento

O desenvolvimento do projeto seguiu uma metodologia baseada no Scrum (SCHWABER e SUTHERLAND, [21--]). A Figura 15 apresenta o ciclo de vida de um projeto segundo a metodologia.

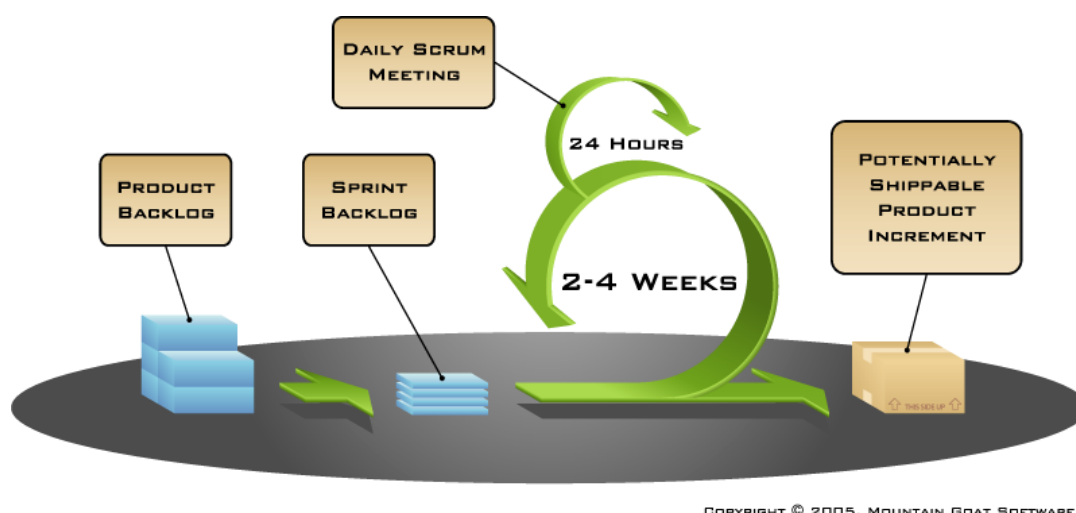


Figura 15: Ciclo de vida SCRUM

- a) Product Backlog: repositório de todas as necessidades do produto é constantemente revisado e priorizado.
- b) Sprint Backlog: Sprint é um período mais curto de desenvolvimento, define-se o período como sendo entre duas e quatro semanas. Com base neste tempo e na disponibilidade de equipe, separa-se um conjunto de necessidades do Product Backlog para compor o Sprint backlog.
- c) Daily Scrum Meeting: Diariamente em horário definido pela equipe, ela deve se reunir para apresentar como foi o andamento desde a última reunião. Cada membro da equipe tenta responder a três perguntas: o que fez desde a última reunião, o que pretendo fazer até a próxima reunião e se existe alguma situação que me impede de fazê-lo.
- d) Pottentially Shippable Product Increment: ao final do Sprint deve-se ter um potencial novo incremento do produto, ou seja, uma versão que pode ser entregue ao cliente com novas funcionalidades.

Para promover mais agilidade e melhor controle, o Sprint desse projeto foi delimitado como sendo de 1 semana (de domingo à sábado) e todo dia de trabalho deveria gerar um novo

potencial entregável. Dessa maneira, os Sprints foram utilizados para controlar o prazo e as expectativas de cada etapa. A primeira atividade do Sprint era definir o que seria feito no próximo ciclo, começando com documentar as atividades feitas no ciclo anterior.

As atividades do Product Backlog foram criadas de forma objetivas e pontual (criar a operação X, escrever o capítulo Y, exportar mensagem, etc...) e estimadas utilizando a notação de “pontos”, onde 1 ponto significa uma atividade muito simples, uma atividade de 2 pontos é duas vezes mais complexa que uma atividade de 1 ponto, e assim sucessivamente. Seguindo as boas práticas de estimativas, segue-se a sequência de Fibonacci para a grandeza de pontos 1, 2, 3, 5, 8, 13,... Neste trabalho, as atividades foram limitadas em 13 pontos.

Para o controle do projeto, foi utilizado o software Acunote¹⁸, que permite a criação de tarefas, preenchimento de Product Backlog e Sprint Backlog. Além disso, permite o acompanhamento do Sprint através de um gráfico *Burndown*. Um exemplo de acompanhamento do projeto pode ser visto na Figura 16.

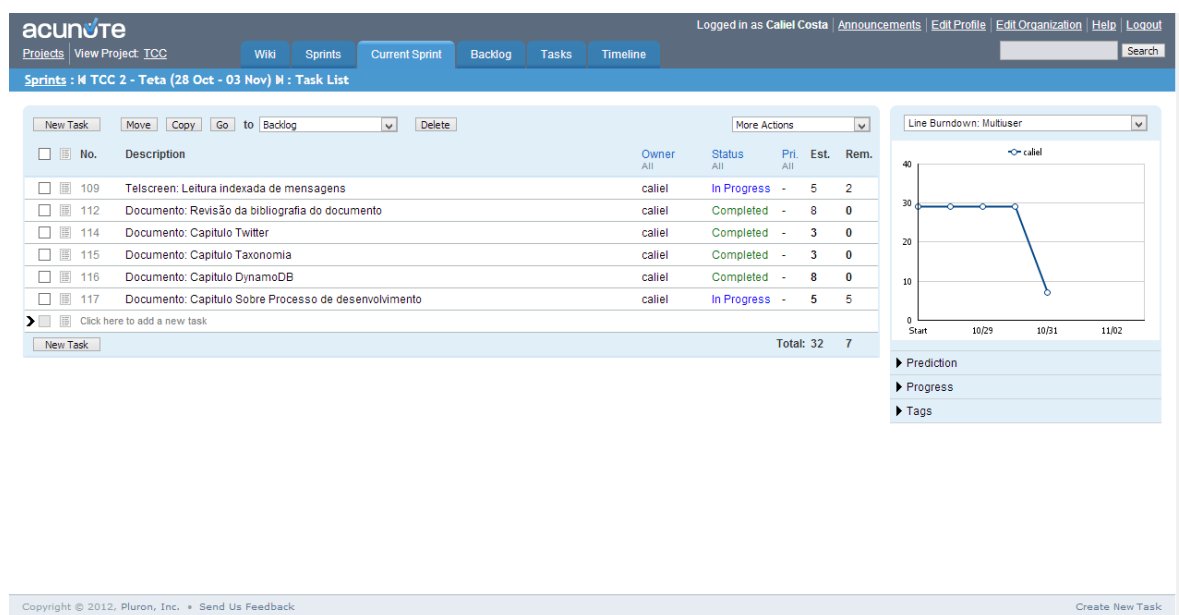


Figura 16: Acunote

Para garantir a correta execução de alguns pontos de código, como, por exemplo, a identificação de tags em um texto e a geração de índices com base em tags, foi utilizada a

¹⁸ Disponível em <http://acunote.com>

técnica de Test Driven Development (TDD). TDD consiste de criar classes de teste antes de criar a classe de implementação. Essa técnica facilita em:

- a) Design de classes mais simples, pois é baseado em agrupamento entre classes para facilitar o teste.
- b) Qualidade futura, pois com o teste pronto, toda a alteração futura na classe poderá gerar erros no teste.
- c) Entendimento do escopo, quando se especifica o resultado final de um método, é mais fácil criar uma execução que retorne o resultado esperado.

TDD não necessita ser feito para todas as classes, mas quanto maior for a cobertura do código mais segura serão as alterações futuras. Segundo a metodologia Extreme Programming (XP), todas as classes devem possuir um teste unitário (EXTREME PROGRAMMING, [21--]).

6.5 Projeto de Desenvolvimento

O projeto de desenvolvimento foi batizado de Kwitter, algo como a fusão de Twitter com Conhecimento (*Knowledge*, em inglês). O projeto foi criado utilizando .NET Framework 4.0 no Visual Studio 2010.

Foram desenvolvidos dois programas: importação dos Tweets e visualização das mensagens.

6.5.1 Marvin

Marvin é o nome do programa de importação de Tweets. Ele foi batizado com esse nome em homenagem ao personagem do livro O Guia do Mochileiro das Galáxias escrito por Douglas Adams em 1979.

O programa foi desenvolvido como sendo um aplicativo de linha de comando, responsável por:

- a) Gerenciamento da taxonomia: o programa é responsável por ler a taxonomia de um arquivo XML e gravá-la na tabela *Tag* do BD. A gravação implica em excluir todos os registros existentes na tabela para só então gravá-los novamente.
- b) Atualização de tags de mensagens: quando a taxonomia sofre grandes alterações (por exemplo: a reorganização de tags entre diferentes níveis hierárquicos ou a inclusão e exclusão de mais níveis), pode ser necessário verificar as tags de todas as mensagens já gravadas. Esse processo pode demorar alguns minutos conforme a quantidade de mensagens existentes.
- c) Observador do Twitter: ao iniciar a importação, a taxonomia gravada no BD é carregada e as tags de busca são identificadas. É criada uma conexão com a Streaming API do Twitter utilizando como filtragem as tags identificadas. Conforme a API retorna Tweets, esses são analisados em busca de tags definidas como marcáveis na taxonomia. Quando a mensagem apresenta uma ou mais tags, é salva na tabela *Message* do BD.

Para a identificação de tags nas mensagens foi utilizada uma expressão regular que busca o termo (e seus sinônimos) como palavras inteiras no texto. Para prevenir ainda mais problemas de gráfica fui desenvolvido um algoritmo que cria variações de termos que contem multiplicas palavras. Por exemplo, para a palavra *Knowledge Management* serão criadas as seguintes variações: KnowledgeManegement, Knowledge-Management e Knowledge_Management.

6.5.2 Telescreen

Telescreen é o site web de visualização das mensagens importadas. Esse nome é uma referência as Teletelas presentes no livro 1984 de George Orwell lançado em 1949.

A única página do site está dividida em três partes:

- a) No cabeçalho é apresentada a filtragem selecionada. As mensagens podem ser filtradas por um conjunto de tags ou por intervalo de datas.
- b) Na lateral direita existe uma árvore com a taxonomia. As tags também podem ser selecionadas. Também são apresentados os 15 termos que mais estão vinculados a seleção atual.

- c) Na região central são apresentadas as mensagens que possuem todas as tags selecionadas e a data está no intervalo informado. As datas de filtragem são consideradas no horário de Greenwich (UTC).

Essas três partes podem ser visualizadas na Figura 17.

Tags: 1

Datas: 1 de [] até []

Aplicar

TELESCREEN

Raimonds Simanovskis *rsim* 08/11/2012 11:47 GMT-2
@metaskills something similar to SET SEARCH_PATH in PostgreSQL or ALTER SESSION SET CURRENT_SCHEMA in Oracle
Oracle • PostgreSQL •

Your Views: Shared *diasforum* 08/11/2012 10:57 GMT-2
DiasForum New Job Posting; Oracle Database Administrator at International Programming & Systems (Redwood City, CA) - <http://t.co/VFg5Fchk>
Employment • Oracle • Programming •

Binus Hacker *binushacker* 08/11/2012 10:46 GMT-2
#Datacenter Oracle Enterprise Manager 12c Cloud Control - Incident Manager - Part Three <http://t.co/iiBzTEBp> #Database #Journal
Oracle •

SHIEYRA RA *ShahirahRosli* 08/11/2012 10:18 GMT-2
kawan aisyah kata aku comel.hurm.malas nak riak bongkak semua.memang comel cantik semulajadi 'ALHAMDULILLAH
RIAK •

Novita Kusumawardani *Novita_kw* 08/11/2012 10:02 GMT-2
Kابه wong do ngmng ngno? Ap salahnya coba? Org iku poto'ne biasa wae,@Dian_Keyboardis: Avatarmu parah dex RT @Novita_kw
DEX •

Writing Jobs *WritingJobs12* 08/11/2012 09:56 GMT-2

Tag cloud: 1
Oracle • Cassandra • DEX • MySQL • RIAK •
PHP • RDBMS • HTML • SQL Server •
MongoDB • Microsoft • You tube • SQL •
Java • CSS •

Taxonomia: 1

- Knowledge Management
- Social Network Site
- Technical
 - Big Data
 - Company
 - DBMS
 - NOSQL
 - Document store
 - Graph
 - Key/value store
 - Wide Column Store
 - OLAP
 - OLPT
 - RDBMS
 - Definitions
 - File System
 - Hadoop

50 mensagens

Figura 17: Interface do site de consulta

6.5.3 Camadas

Além do Marvin e Telescreen, a solução possui outros pacotes, conforme a Figura 18:

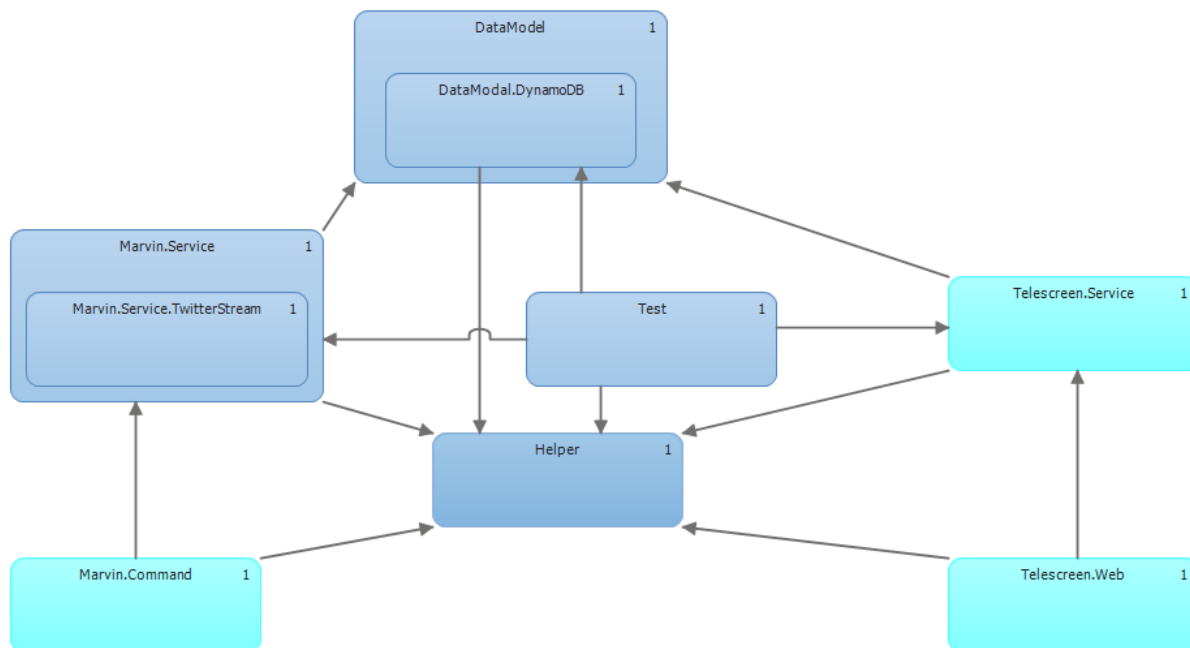


Figura 18: Camadas do projeto

Os pacotes **Marvin.Command** e **Telescreen.Web** já foram apresentados. Os demais são:

- Helper**: pacote contendo algumas classes utilitárias para *strings* e conjuntos (*arrays*).
- DataModel**: pacote contendo as interfaces de modelo de dados e operações de tabelas.
- DataModel.DynamoDB**: implementação das interfaces específicas para o BD DynamoDB.
- Marvin.Service**: Este é o pacote onde estão as classes de gerenciamento da taxonomia, atualização e importação de mensagens.
- Marvin.Command**: Executável de linha de comando que instancia e executa as classes do **Marvin.Service**.
- Telescreen.Service**: É um webservice com os métodos retornar taxonomia e retornar mensagens.

- g) Telescreen.Web: Página web para apresentação do taxonomia e mensagens chamando diretamente o Telescreen.Service.
- h) Test: pacote de testes unitários utilizados em pontos específicos da aplicação como, por exemplo, a identificação de tags de um texto.

7 CONCLUSÃO

A conclusão desse trabalho está dividida em quatro partes:

- a) Analisar a possibilidade de localizar informação no Twitter utilizando uma taxonomia;
- b) Verificar o desempenho do DynamoDB como uma aplicação remota;
- c) Compartilhar a experiência de mudança de paradigma ao desenvolver para o DynamoDB;
- d) Ao final, apresentar algumas sugestões de trabalhos futuros.

7.1 Análise das Informações Encontradas no Twitter

A importação, utilizando a taxonomia definida na seção 6.1, ocorreu por aproximadamente 57 horas (sendo 37 consecutivas) durante os dias 04/11/2012 e 08/11/2012. Foram importadas 43 mil mensagens, ocupando um espaço total de aproximadamente 87 megabytes. No Gráfico 1 pode ser visto a quantidade de registros importados por hora, segundo o horário de Greenwich.

O pico de importação foi no dia 07/11/2012 entre as 16h00min e as 17h00min com 1444 mensagens importadas. Foi importada uma média de 759 mensagens por hora.

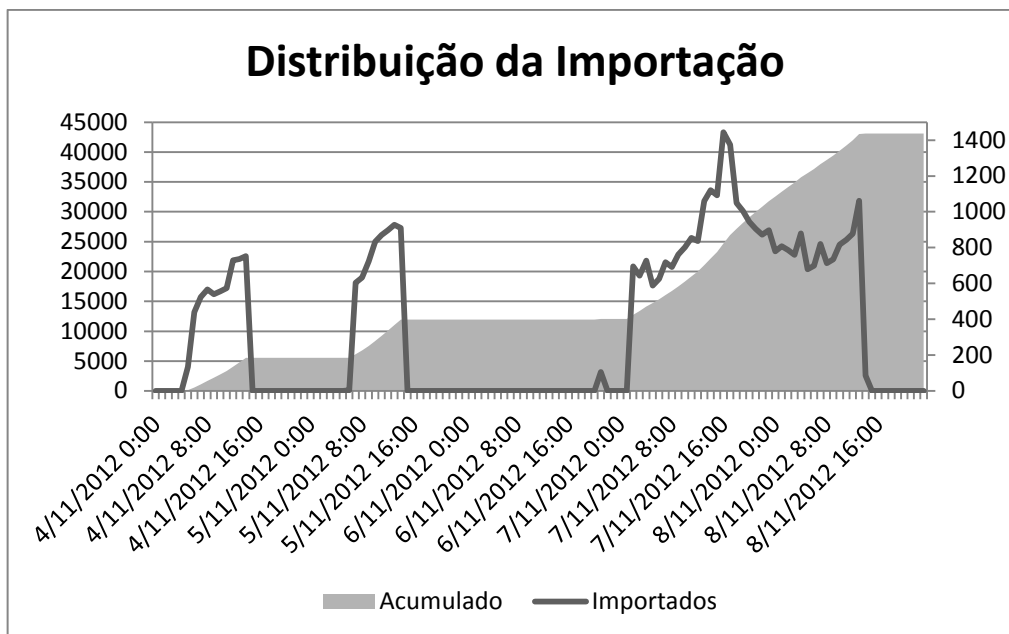


Gráfico 1: Distribuição da importação

Essa análise é baseada em uma série de filtragens, utilizando um cruzamento entre os três tipos de tags: técnicas, gestão do conhecimento e sites de redes sociais. Já, no início da importação, foi possível verificar que algumas mensagens não estariam associadas somente ao contexto das tags citadas anteriormente como, por exemplo:

- a) Cassandra é um nome próprio de origem grega.
- b) O termo “Riak” significa onda, ondulação em indonésio e malaio.
- c) A palavra “Redis” significa repetir em francês.
- d) A palavra “Dex” é utilizada como o diminutivo para o nome “Dexter”, que também é uma série de TV¹⁹.




As traduções de “Riak” e “Redis” foram realizadas utilizando o Google Translator²⁰. Nenhum tratamento adicional foi realizado para essas mensagens, durante a análise de resultados a mensagem foi ignorada se não estivesse vinculada a outra tag.

Na Tabela 5 são apresentadas alguns exemplos de informações que puderam ser extraídas do Twitter.

¹⁹ Mais informações em: <http://www.imdb.com/title/tt0773262/>

²⁰ Disponível em <http://translate.google.com.br>

Tabela 5: Análise de Mensagens

Cruzamento de tags	Análise
Knowledge Management, Social Network Site, Technical	Resultado inexpressivo, não apresentou nenhuma informação relevante.
Knowledge Management, Big Data	<div>  <div> Computer Science&IT <small>rmit_csit</small> Social network analysis, analytics fuelling Big Data growth http://t.co/qr4yxVrs #SocialNetworks #BigData #social #media #analytics #data Big Data • Social Network • Social Network Analysis • </div> </div> <p>08/11/2012 05:40 GMT-2</p> <p>Resultado inexpressivo, com exceção de uma matéria que comenta sobre o investimento de US\$ 28 bilhões em Big Data no ano de 2012, sendo que 45% desse valor serão em análise de redes sociais e análise de conteúdo.</p> <p>Pouco conteúdo referente as tags com os destaques abaixo:</p> <div>  <div> Stéphane Langlois <small>langlois_s</small> knowledge sharing "@henrikkniberg: Spotify is offering free Cassandra courses ... http://t.co/IEK2eg80 " Cassandra • Knowledge • </div> </div> <p>05/11/2012 08:54 GMT-2</p> <p>Compartilhado por Henrik Kniberg²¹ que está ministrando cursos sobre o banco Cassandra.</p> <div>  <div> Julien Duponchelle <small>jduponchelle</small> A social network with Redis http://t.co/zleMVbTI Redis • Social Network • </div> </div> <p>05/11/2012 07:29 GMT-2</p>

²¹ Henrik Kniberg é um importante consultador da comunidade Ágil, escreveu livros como: Scrum and XP from the Trenches e Lean from the Trenches: Managing Large-Scale Projects with Kanban.

Uma apresentação sobre como Redis foi utilizado num site de rede social.

Knowledge Management, Social Network site



Bill Winterberg CFP® [BillWinterberg](#)

Advisors, you can't use collaboration tools like @Yammer unless you have a way to archive its content: <http://t.co/u7zZLTpZ>

07/11/2012 18:41 GMT-2

Collaboration Tools • Yammer •

O link para um podcast que ressalva os usos de ferramentas de colaboração como o Yammer em relação às políticas de *Compliance*.

Knowledge Management, RDBMS



Martijn van der Kamp [MartijnvdKamp](#)

New Oracle Blog How Data and BPM are married to get the right information to the right people at the right time

<http://t.co/cSM3k0NI>

07/11/2012 11:59 GMT-2

Business Process Mapping • Oracle •

No blog Capgemini, um dos blogs da empresa Oracle, um artigo sobre como o Mapeamento de Processos pode ser utilizado para entregar a informação certa para a pessoa certa no tempo certo.

As práticas de gestão do conhecimento mais citadas foram Taxonomia, Ferramentas Colaborativas e Comunidade de Prática.

Knowledge Management



Nicholas Rosellini [n_rosellini](#)

#Asia-Pacific Community of Practice on Women's Political Participation launched #UNDP <http://t.co/MuvbJwvF>

08/11/2012 09:43 GMT-2

Community of Practice •

Uma que se destacou bastante foi a do lançamento de uma comunidade de prática sobre a participação política de mulheres na Ásia.

Explicit knowledge, Tacit Knowledge



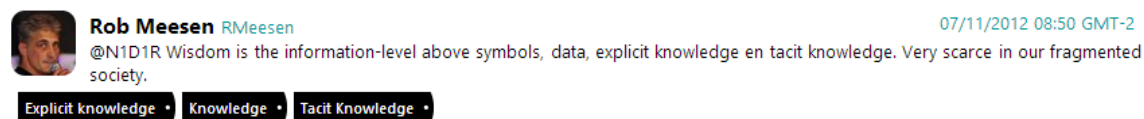
Kim Parker [kimparker054](#)

Explicit Knowledge: 'easy' to manage and exploit. Tacit Knowledge: the hidden gem poorly recognised & ignored to the detriment of business.

08/11/2012 00:54 GMT-2

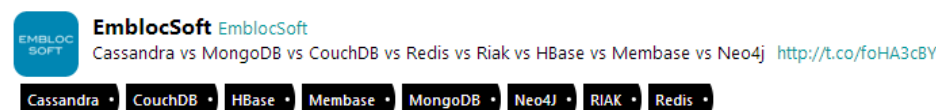
Explicit knowledge • Knowledge • Tacit Knowledge •

O usuário criou a sua definição sobre conhecimento explícito e tácito.

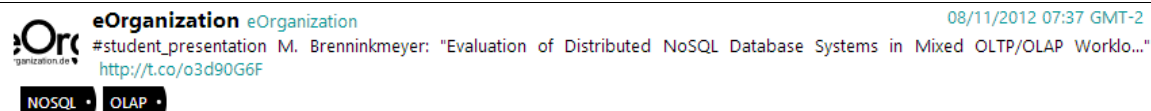


Esta é a quarta mensagem na conversa dos dois usuários na qual é apresentada a definição de Sabedoria.

**Document store, Graph,
Key/value store, Wide
Column Store**

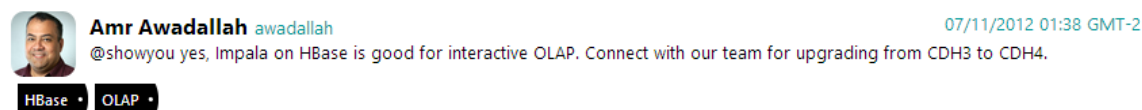
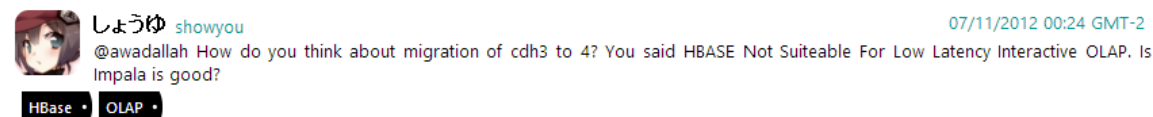


Apresenta um comparativo entre diversos bancos de dados NOSQL.



A mensagem comenta sobre a apresentação de um trabalho acadêmico sobre a utilização de BD NOSQL na implementação de OLTP e OLAP.

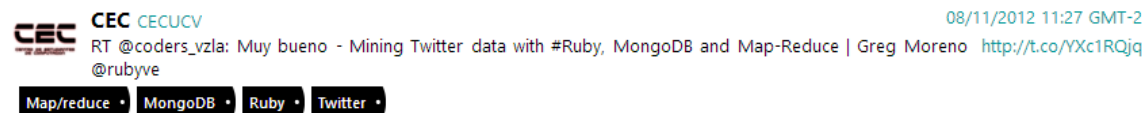
NOSQL, OLAP



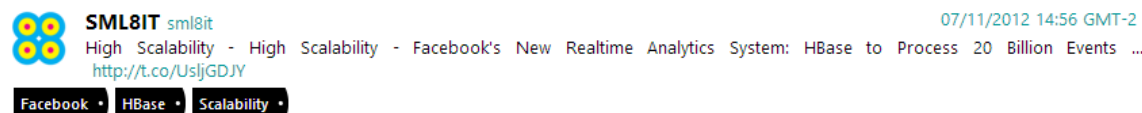
Na primeira mensagem é perguntado sobre a utilização do BD HBase em soluções OLAP, e cerca de uma hora depois a resposta de outro usuário afirmando ser possível.

Definitions, NOSQL

A *definitions* é um agrupamento com todas as definições e algoritmos utilizados, Map/reduce, escalabilidade e sharding foram os que mais apareceram.

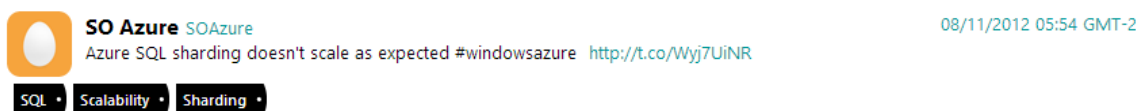


Um exemplo parecido com esse trabalho, porém utilizando Ruby e MongoDB.



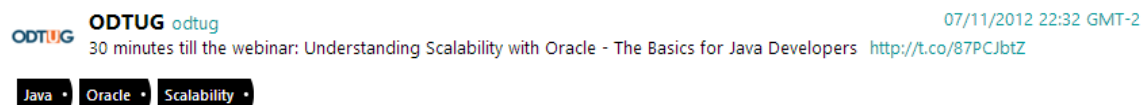
Apresenta um sistema analítico em HBase para o Facebook.

O termo escalabilidade foi uma das palavras mais citadas



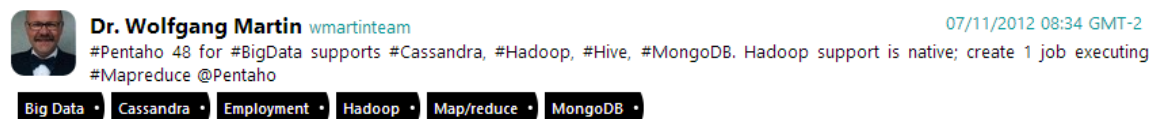
Definitions

Neste caso o termo está vinculado ao banco de dados MS Azure e o usuário está reclamando sobre a sua utilização.



Já aqui uma empresa desenvolveu uma apresentação sobre como funciona a escalabilidade em Oracle.

Map/Reduce



Esta mensagem chamou a atenção por fazer referência ao Pentaho²² um dos mais importantes softwares de *Bussiness Intelligence* (BI) e *Bussiness Analytics*.

Twitter, You Tube e Facebook são as redes sociais que mais se destacam.

DBMS, Social Network Site



stoker stoker

08/11/2012 12:01 GMT-2

RT @MySQL: Election 2012: Twitter Breaks Records with #MySQL - <http://t.co/XepxnMtp>



Uma mensagem sobre como o Twitter conseguiu acompanhar as eleições americanas utilizando uma base MySQL.



José Papo josepapo

07/11/2012 07:02 GMT-2

Join me and nominate DynamoDB for the Best Technology Achievement 2012 Crunchie! <http://t.co/5xqK0tpO> #crunchies



A campanha que José Papo²³ está fazendo para que o DynamoDB seja eleito a melhor tecnologia de 2012. Outras 18 pessoas também postaram a mesma mensagem.

DynamoDB



Raghuraman raghuramanb

05/11/2012 07:46 GMT-2

#AWS #Dynamodb improvements: <http://t.co/rp7ecELf>



Durante o desenvolvimento do trabalho foi lançada uma nova versão do SDK e isso foi citado em algumas mensagens.



Kishorekumar indykish

05/11/2012 09:01 GMT-2

How to interface with DynamoDB using Rails: <http://t.co/w8F8LYT8> via #aws #dynamodb #megam @indykish



²² Mais informações em: <http://www.pentaho.com/>

²³ José Papo trabalha na Amazon como evangelista técnico do Amazon Web Service para a América Latina.

Uma postagem sobre como fazer a integração entre o DynamoDB e a linguagem Ruby on Rails.

Não havia sido prevista uma tag para empregos, porém, durante as primeiras importações, verificou-se que termos relacionados a isso como, por exemplo, *employment*, *job*, *hire*, *carrier*, apareciam com frequência. Sendo assim, foi acrescida uma tag não pesquisável pelo termo. Isso resultou mais de 2500 mensagens contendo como cinco termos mais relevantes:

Employment

- a) Oracle: 1518 mensagens
- b) MySQL: 558 mensagens
- c) SQL: 408 mensagens
- d) SQL Server: 334 mensagens
- e) PHP: 320 mensagens

Employment, NoSQL

Com essa filtragem retornou apenas 80 registros, que estão vinculados em sua maioria (36 vezes) ao banco de dados MongoDB.

Através desses exemplos (e de outros não listados) é possível afirmar que o Twitter possui informações válidas, porém é necessário saber buscá-las e organizá-las. A busca com base em uma taxonomia ajudou bastante, principalmente para fazer a navegabilidade de ir de um conceito mais abstrato até chegar a um conceito mais concreto.

7.2 Performance do Amazon DynamoDB

Como é um banco contratado na forma de serviço, é o próprio administrador do banco de dados (*Database administrator* ou DBA), que estipula a capacidade de entrada e saída de informação, portanto quanto menor a capacidade, mais lenta a solução se torna. Uma tabela configurada para uma leitura e uma gravação por segundo (1KB de dado) custa mensalmente²⁴ US\$ 0,90, enquanto uma configuração mais robusta com 100 leituras e 50 gravações por segundo custa US\$ 44,64 por mês. Além disso, a taxa de tráfego pode ser alterada a qualquer momento, ou seja, a vazão pode ser alterada de acordo com o momento de utilização do serviço. Nesse trabalho, a taxa de tráfego foi alterada em dois momentos:

- a) Quando o Marvin estava realizando a carga de dados, a quantidade de gravações por segundo das tabelas envolvidas foi aumentada.
- b) Na análise dos dados, o número de leituras por segundo da tabela foi aumentada.

O DynamoDB pode gerar gráficos de monitoramento e envio de notificações sobre a utilização do serviço. Por exemplo, pode ser configurado um e-mail que receberá uma notificação quando a utilização das tabelas exceder 80% do delimitado em 60 minutos. Para os gráficos, existe o serviço chamado de CloudWatch que monitora toda a infraestrutura contratada na Amazon.

No Gráfico 2 e Gráfico 3 são apresentadas gráficos gerados pelo CloudWatch durante o processo de importação que foi executado entre 07/11/2012 às 10h00min e 08/11/2012 às 10h00min segundo o horário Tempo Universal Coordenado (*Universal Time Coordinated* ou UTC em inglês).

²⁴ Custo calculado em 10 de novembro de 2012

Nos dois primeiros gráficos, os pontos do gráfico representam o somatório de unidade de gravação acumulados em 5 minutos. Como já foi explicado, nas tabelas *Message* e *Message_by_Created* (Gráfico 2) são gravados apenas 1 registro por *Tweet*, o pico ocorreu em 07/11/2012 às 17h20min, para a tabela *Message*, foram 162 unidade de gravação, realizando a divisão por 300 (quantidade de segundos em 5 minutos) temos o consumo de 0,54 pontos (em média) por segundo, na tabela *Message_by_Created* o valor é de 322 unidade ou seja 1,07 unidade/segundo. Nas tabelas *Message_by_Tag* e *Message_by_TagCreated* (Gráfico 3) a quantidade de registros por *Tweet* pode variar (mas ambas apresentam a mesma quantidade). Nesse caso o pico ocorreu em 07/11/2012 às 17h25min com 3870 unidades consumidas ou 12,9 unidades/segundo. Com base nesses números, as configurações de leitura e gravação das tabelas podem ser corrigidas para gerar a melhor relação custo/benefício.

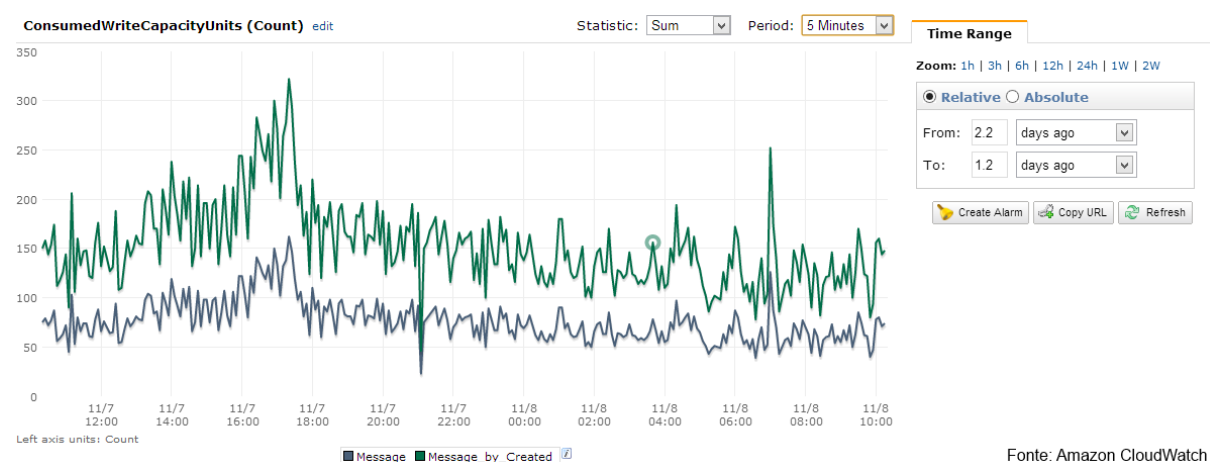


Gráfico 2: Unidades de gravação em Message e Message_by_Created

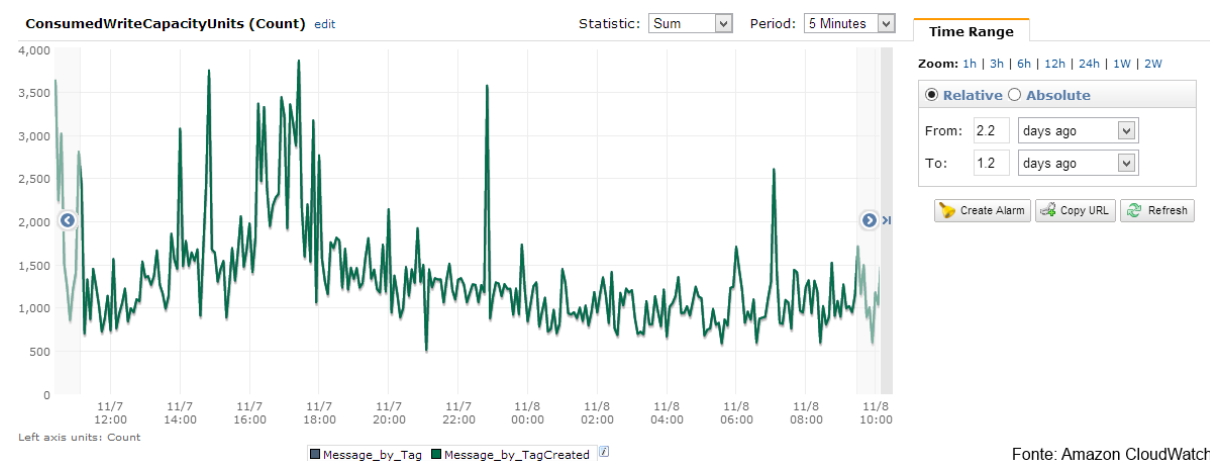


Gráfico 3: Unidades de gravação em Message_By_Tag e Message_by_TagCreated

Nos próximos gráficos são apresentados os tempos máximos e mínimos (em milissegundos) para o retorno da operação de gravação na tabela *Message*. Nota-se no Gráfico 4 que o tempo máximo ficou entre 20 e 40 milissegundos tirando algumas exceções, principalmente em 07/11/2012 às 22h10min quando o valor chegou a 118,1 milissegundos. Já no Gráfico 5 o tempo mínimo oscilou entre três e quatro milissegundos com uma grande exceção em 07/11/2012 às 14h15min onde o tempo foi de 2,56. Com base nisso podemos notar que o sistema possui uma latência muito baixa.

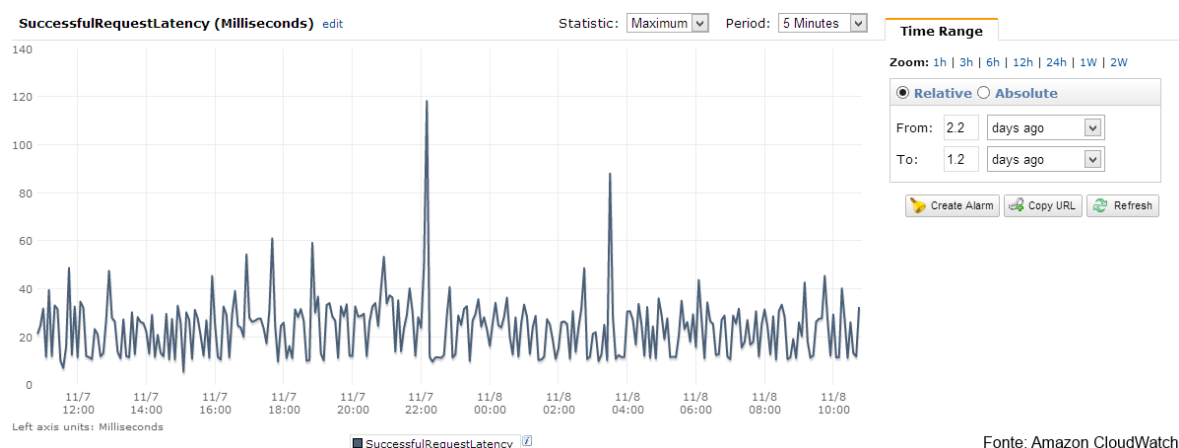


Gráfico 4: Tempo máximo de resposta da gravação na tabela Message

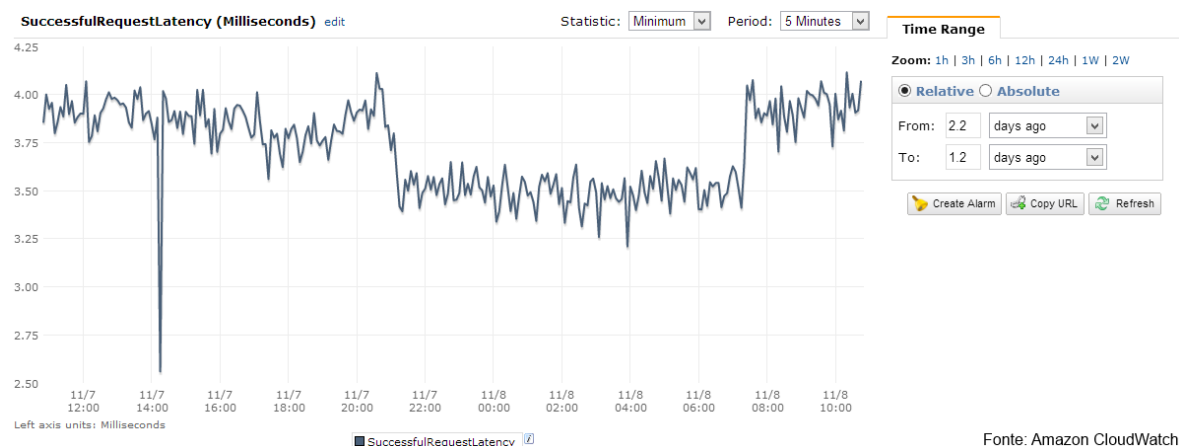


Gráfico 5: Tempo mínimo de resposta da gravação na tabela Message

CloudWatch permite gerar gráficos para outras operações, que não foram utilizadas nesse trabalho. No entanto, gráficos que podem ser gerados pelo CloudWatch são:

- Tempo de resposta para a operação de: *BachGetItem*, *DeleteItem*, *GetItem*, *Scan*, *UpdateItem*, *PutItem* e *Query*.
- Itens retornados/atualizados nas operações de: *BachGetItem*, *DeleteItem*, *GetItem*, *Scan*, *UpdateItem*, *PutItem* e *Query*.

Nesse SGBD, como se pode notar, o principal problema de performance não está no hardware (ou na infraestrutura de serviço) e sim no software e na sua modelagem de tabelas. Durante o desenvolvimento foram testados diversos modelos de dados, desde o primeiro com somente a tabela *Message* até o modelo atual que além da tabela *Message* existem três tabelas de índices: *Message_by_Create*, *Message_by_Tag* e *Message_by_TagCreate*.

Para isso, é necessário não somente entender como o SGBD pode ser utilizado, mas criar algoritmos para gravação e leitura que combinem a distribuição dos dados obtendo o máximo de agilidade e consistência.

Nesta linha foram criados algoritmos para simplificar as tags presentes em uma mensagem, criação de algoritmo de *hash* para gravação de múltiplas tags, gravação em paralelo nas múltiplas tabelas. Dessa maneira, o tamanho e quantidade de registros gravados são reduzidos obtendo maior desempenho.

7.3 Desenvolvimento com Amazon DynamoDB

A Amazon fornece um kit para desenvolvimento (*software development kit* ou SDK em inglês) para as tecnologias: Android, iOS, Java, .NET, PHP e Ruby (AMAZON, [21--]). Esse kit permite o desenvolvimento e integração com toda a plataforma de serviços em nuvem da Amazon, inclusive com o DynamoDB.

Como o Kwitter foi desenvolvido em .NET, existem duas formas de realizar a codificação:

- a) Primeira é utilizando *.NET Low-Level API* que fornece um controle maior sobre a codificação (pode-se realizar a analogia com o ADO.NET).
- b) Segunda é *.NET Object Persistence Model* que possui uma abstração ainda maior agilizando o desenvolvimento de rotinas mais simples (análogo ao *ADO.NET Entity Framework*).

Nesse projeto utilizou-se uma junção de ambos. Na parte de gravação de registros, foi utilizado *.NET Object Persistence Model*, já, na parte de consultas seletivas (*query*), foi utilizada a *.NET Low-Level API*. Isso permitiu um controle melhor e mais específico buscando o melhor desempenho nas consultas.

O principal motivo pelo qual o DynamoDB foi escolhido entre os bancos de dados apresentados foi pelo fornecimento da infraestrutura juntamente com o SGBD. Como já mencionado, esse critério foi plenamente atendido pela Amazon, fornecendo uma arquitetura robusta e escalável a um valor justo.

Uma das principais ressalvas é com relação ao modelo de consistência eventual. Porém, isso não se tornou um problema em vista que a mensagem é um registro estável, uma vez gravada não é mais alterada, o que pode ocorrer é a alteração na taxonomia necessitando uma reanálise de cada mensagem, mas isso não é uma necessidade constante.

Mesmo que as mensagens fossem seguidamente alteradas, seria necessária uma quantidade muito grande de acessos multiusuário ao mesmo registro para causar problemas, já que a latência da aplicação é muito baixa.

Uma das grandes dificuldades percebida é que a equipe de desenvolvimento tem que se adaptar aos novos tempos. O desenvolvimento tem que pensar em como fazer melhor uso da arquitetura, pois agora o custo de uso é medido rapidamente. No modelo de banco de dados relacional quando uma aplicação não está obtendo muito desempenho deixa-se ao cargo do DBA criar índices ou *stored procedure*. Neste paradigma a criação de uma nova tabela vai necessitar de desenvolvimento, além de um monitoramento das métricas do sistema, mas enquanto isso não acontece o custo de aumentar a vazão de uma determinada tabela pode ser alto.

O DynamoDB atendeu todas as expectativas de um banco de dados, o autor desse trabalho com base na experiência de 8 anos utilizando o modelo relacional, o DynamoDB se encaixaria em muitas dessas aplicações, em alguns casos muito melhor que o banco relacional adotado. Um exemplo é o sistema para controle de ponto eletrônico onde é necessário armazenar cada uma das batidas de cada funcionário.

7.4 Trabalhos Futuros

Com o decorrer do trabalho, identificou-se o tipo de relação existente entre a mensagem e as suas tags e percebe-se que se assemelham muito ao modelo de grafos onde cada tag e mensagem são arestas conectadas. Sendo assim, um exemplo de trabalho futuro é

utilizar os bancos de dados orientados a grafos (seção 4.3.4) para organizar mensagens vinculadas a tags.

Outro trabalho é a utilização de outras redes sociais, como Facebook ou LinkedIn para localizar informações relevantes à taxonomia.

A melhoria na análise semântica das mensagens pode ser outro objetivo de trabalho futuro. Como esse trabalho está baseado em expressões regulares, pode-se estudar a utilização de dicionários para identificar sinônimos, conjugações verbais, etc.

E finalmente, a última proposta diz respeito à mineração de dados da web. A mineração pode ser dividida em várias etapas (SILVA, 2012):

- a) Aquisição e gravação de Dados.
- b) Extração da informação e limpeza.
- c) Integração, agregação e representação.
- d) Processamento de consultas, modelagem e análise de dados.

As duas primeiras etapas (aquisição e extração de informação) que são bastante trabalhosas foram realizadas nesse trabalho, podem em trabalhos futuros realizar as demais etapas.

REFERÊNCIAS BIBLIOGRÁFICAS

ABEL, A. Adoção de NoSQL está em crescimento, indicam várias pesquisas. **InfoQ**, 15 fev. 2012. Disponível em: <<http://www.infoq.com/br/news/2012/02/NoSQL-Adoption-Is-on-the-Rise>>. Acesso em: 26 fev. 2012.

ALVES, M. Folksonomia: acostume-se com este termo. **iMaster**, 2007. Disponível em: <<http://imasters.com.br/artigo/7314/tendencias/folksonomia-acostume-se-com-este-termo>>. Acesso em: 13 maio 2012.

AMAZON. Amazon DynamoDB Documentation. **Amazon web services**, [21--]. Disponível em: <<http://aws.amazon.com/pt/documentation/dynamodb/>>. Acesso em: 20 set. 2012.

AMAZON. Sample Code & Libraries. **Amazon Web Service**, [21--]. Disponível em: <<http://aws.amazon.com/code/>>. Acesso em: 10 nov. 2012.

BENEDETTI, F. Gestão do Conhecimento: Um Importante Recurso para a Inteligência Estratégica, 2008. Disponível em: <<http://www.slideshare.net/fabianabenedetti/gesto-do-conhecimento-um-importante-recurso-para-a-inteligencia-estratgica-12852122>>. Acesso em: 08 maio 2012.

BOYD, D. M.; ELLISON, N. B. Social Network Sites: Definition, History, and Scholarship. **Journal of Computer-Mediated Communication**, v. 13, n. 1, p. 210–230, out. 2007. Disponível em: <<http://onlinelibrary.wiley.com/doi/10.1111/j.1083-6101.2007.00393.x/full>>. Acesso em: 29 abr. 2012.

BRITO, R. W. Bancos de Dados NoSQL x SGBDs Relacionais: Análise Comparativa. **InfoBrasil**, 2010. Disponível em: <<http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoSQL.pdf>>. Acesso em: 28 maio 2012.

CARLSON, N. How Many Users Does Twitter REALLY Have? **Business Insider**, 2011. Disponível em: <<http://www.businessinsider.com/chart-of-the-day-how-many-users-does-twitter-really-have-2011-3>>. Acesso em: 06 maio 2012.

CARNEIRO, J. L. Introdução a Banco de dados, 2004. Disponível em: <http://www.jlcarneiro.com/downloads/banco_dados.pdf>. Acesso em: 21 maio 2012.

CATTELL, R. Scalable SQL and NoSQL Data Stores. **ACM SIGMOD Record**, Nova Iorque, v. 39, n. 4, p. 12-27, dez. 2010. Disponível em: <<http://dl.acm.org/citation.cfm?id=1978919>>. Acesso em: 03 mar. 2012.

CHOO, C. W. **A organização do conhecimento**: como as organizações usam a informação para criar significado, construir conhecimento e tomar decisões. 2ª. ed. São Paulo: Senac São Paulo, 2006.

COLLETT, S. Big Data: é um grande negócio? **Computerworld**, 02 set. 2011. Disponível em: <<http://computerworld.uol.com.br/tecnologia/2011/09/02/big-data-e-um-grande-negocio/>>. Acesso em: 22 fev. 2012.

COMM, J. **O poder do Twitter**: estratégias para dominar o seu mercado e atingir ses objetivos com um tweet por vez. Tradução de Leonardo Abramowicz. São Paulo: Editora Gente, 2009. ISBN 978-85-7312-656-3.

DATE, C. J. **Introdução a sistemas de Banco de Dados**. 1. ed. Rio de Janeiro: Campus, 2004.

DE DIANA, M.; GEROSA, M. A. **NOSQL na Web 2.0**: Um Estudo Comparativo de Bancos Não-Relacionais para Armazenamento de Dados na Web 2.0. Workshop de Teses e Dissertações em Banco de Dados. Belo Horizonte: [s.n.]. 2010.

DECANDIA, G. et al. Dynamo: Amazon's Highly Available Key-value Store, Stevenson, 14-17 out. 2007. 205-220. Disponível em: <<http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>>. Acesso em: 2012 out. 31.

DICTIONARY.COM. Social Network. **Dictionary.com**, [21--]. Disponível em: <<http://dictionary.reference.com/browse/social+network>>. Acesso em: 02 maio 2012.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 5ª. ed. [S.l.]: Addison Wesley, 2006. Acesso em: 21 maio 2012.

EXTREME PROGRAMMING. **Extreme Programming**, [21--]. Disponível em: <<http://www.extremeprogramming.org/>>. Acesso em: 30 out. 2012.

FOWLER, M. NosqlDefinition, 2012. Disponível em: <<http://martinfowler.com/bliki/NosqlDefinition.html>>. Acesso em: 22 fev. 2012.

FOWLER, M.; SADALAGE, P. Polyglot Persistence. **Martin Fowler**, 2012. Disponível em: <<http://martinfowler.com/articles/nosql-intro.pdf>>. Acesso em: 19 mar. 2012.

G1. O que é: Rede social. **G1**, 2008. Disponível em: <<http://g1.globo.com/Noticias/0,MUL394839-15524,00.html>>. Acesso em: 29 abr. 2012.

HEUSER, C. A. **Projeto de Banco de Dados**. 6. ed. Porto Alegre: Artmed, v. 4, 1998. Disponível em: <http://www.julianoribeiro.com.br/troca/banco_de_dados/material_der.pdf>. Acesso em: 21 maio 2012.

HUBERMAN, B. A.; ROMERO, D. M.; WU, F. Social Science Research Network. **Social networks that matter**: Twitter under the microscope, 2008. Disponível em: <<http://ssrn.com/abstract=1313405>>. Acesso em: 25 abr. 2012.

IBM. What is big data? **IBM**, [21--]. Disponível em: <<http://www-01.ibm.com/software/data/bigdata/>>. Acesso em: 22 fev. 2012.

IDC. The Diverse and Exploding Digital Universe, 2008. Disponível em: <<http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf>>. Acesso em: 2012 maio 2012.

JACOBS, A. The Pathologies of Big Data. **Communications of the ACM - A Blind Person's Interaction with Technology**, Nova Iorque, v. 52, n. 8, p. 36-44, ago. 2009. Disponível em: <<http://dl.acm.org/citation.cfm?id=1536632>>. Acesso em: 27 maio 2012.

JAIN, S. 40 Most Popular Social Networking Sites of the World. **Socialmedia Today**, 2010. Disponível em: <<http://socialmediatoday.com/node/195917>>. Acesso em: 02 maio 2012.

KATO, D.; GLEDSON, S. Folksonomia: características, funcionamento e aplicações. **Terra Fórum**, 2009. Disponível em: <<http://biblioteca.terraforum.com.br/Paginas/folksonomia-caracteristicas-funcionamento-e-aplicacoes.aspx>>. Acesso em: 13 maio 2012.

KWAK, H. et al. What is Twitter, a Social Network or a News Media? **WWW '10 Proceedings of the 19th international conference on World wide web**, Nova Iorque, p. 591-600, 2010. Disponível em: <<http://dl.acm.org/citation.cfm?id=1772751>>. Acesso em: 25 abr. 2012.

LAI, E. Twitter growth prompts switch from MySQL to 'NoSQL' database, 2010. Disponível em: <http://www.computerworld.com/s/article/9161078/Twitter_growth_prompts_switch_from_MySQL_to_NoSQL_database>. Acesso em: 29 maio 2012.

LÓSCIO, B. F.; DE OLIVEIRA, H. R.; PONTES, J. C. D. S. NoSQL no desenvolvimento de aplicações Web colaborativas. **Simpósio Brasileiro de Sistemas Colaborativos**, Paraty, ago. 2011. Disponível em: <http://www.addlabs.uff.br/sbsc_site/SBSC2011_NoSQL.pdf>. Acesso em: 26 Ferereiro 2012.

MATSUSHITA, R. S. I.; NGUESSAN, D. Framework para integração de serviços móveis baseado em rede social. **FaSci-Tech**, 2012. Disponível em: <<http://www.fatecsaocaetano.edu.br/fascitech/index.php/fascitech/article/view/55>>. Acesso em: 17 jun. 2012.

MONTEIRO, J. M. Gestão do Conhecimento, 2008. Disponível em: <<http://www.administradores.com.br/informe-se/artigos/gestao-do-conhecimento/26902/>>. Acesso em: 07 maio 2012.

NATIONS, D. The Top Social Networking Sites. **About.com**, [21--]. Disponível em: <http://webtrends.about.com/od/socialnetworking/a/social_network.htm>. Acesso em: 24 abr. 2012.

NONAKA, I.; TAKEUCHI, H. **Criação de conhecimento na empresa**: como as empresas japonesas geram a dinâmica da inovação. Rio de Janeiro: Campus, 1997.

NOSQL. NOSQL Databases. **NoSQL - Your Ultimate Guide to the Non - Relational Universe!**, [21--]. Disponível em: <<http://nosql-database.org/>>. Acesso em: 22 fev. 2012.

OREND, K. Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer. **Technical University Munich, Faculty of Informatics**, Munich, 2010. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.184.483&rep=rep1&type=pdf>>. Acesso em: 03 mar. 2012.

ORIENTDB , [21--]. Disponível em: <<http://code.google.com/p/orient/>>. Acesso em: 17 jun. 2012.

OXFORD DICTIONARIES ONLINE. Social Network. **Oxford Dictionaries Online**, [21--]. Disponível em: <<http://oxforddictionaries.com/definition/social%2Bnetwork>>. Acesso em: 02 maio 2012.

PADHY, R. P.; PATRA, M. R.; SATAPATHY, S. C. RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's. **International Journal of Advanced Engineering Sciences and Technologies**, v. 11, n. 1, p. 15-30, set. 2011. Disponível em: <<http://www.ijaest.iserp.org/archieves/19-Sep-15-30-11/Vol-No.11-Issue-No.1/3.IJAEST-Vol-No-11-Issue-No-1-RDBMS-to-NoSQL-Reviewing-Some-Next-Generation-Non-Relational-Database's-015-030.pdf>>. Acesso em: 30 maio 2012.

PETTEY, C.; GOASDUFF, L. Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data. **Gartner**, 2011. Disponível em: <<http://www.gartner.com/it/page.jsp?id=1731916>>. Acesso em: 27 maio 2012.

POWELL, J. **33 Milhões de pessoas na sua rede de contatos**. Tradução de Leonardo Abramowicz. São Paulo: Editora Gente, 2010. ISBN 978-85-7312-676-1.

RODRIGO, H. Redes Sociais, um conceito mais antigo do que você pode imaginar! **Digital Markketing**, 2010. Disponível em: <<http://www.digitalmarkketing.com/2010/08/12/redes-sociais-um-conceito-mais-antigo-do-que-voce-pode-imaginar/#.T54ISKtYsk1>>. Acesso em: 24 abr. 2012.

SANTIAGO JR., J. R. S. **Gestão do Conhecimento A Chave para o Sucesso Empresarial**. São Paulo: Novatec, 2004. Disponível em: <<http://www.olivreiro.com.br/pdf/livros/cultura/764390.pdf>>. Acesso em: 10 maio 2012.

SCHWABER, K.; SUTHERLAND, J. Scrum Guiders. **Scrum.org**, [21--]. Disponível em: <<http://www.scrum.org/Scrum-Guides>>. Acesso em: 31 out. 2012.

SEVILLA, M. What is Big Data? **Capgemini**, 8 nov. 2011. Disponível em: <<http://www.capgemini.com/technology-blog/2011/11/what-is-big-data/>>. Acesso em: 23 fev. 2012.

SILBERCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de Banco de Dados**. 3ª. ed. São Paulo: Pearson Makron Books, 1999. ISBN 85-346-1073-8.

SILVA, A. S. D. **XXVII Brazilian Symposium on Database**. Exploring Structured Data in Textual Content from the Web: Methods, Techniques and Applications. São Paulo: [s.n.]. 2012.

SILVA, S. L. D. Gestão do conhecimento: uma revisão crítica orientada pela abordagem da criação do conhecimento, 2004. Disponível em: <<http://www.scielo.br/pdf/%0D/ci/v33n2/a15v33n2.pdf>>. Acesso em: 07 maio 2012.

SOARES, E. Big Data: proposta arrojada e visão de 360°, 2012. Disponível em: <<http://computerworld.uol.com.br/tecnologia/2012/02/06/big-data-proposta-arrojada-e-visao-de-360b0/>>. Acesso em: 27 maio 2012.

SPARSITY TECHNOLOGIES. DEX. **Sparsity Technologies**, [21--]. Disponível em: <<http://www.sparsity-technologies.com/dex.php>>. Acesso em: 17 jun. 2012.

STRAUCH, C. NoSQL Databases, 2011. Disponível em: <<http://yizero.com/file/nosql dbs.pdf>>. Acesso em: 01 Março 2012.

STRAUCH, C. NoSQL Databases. **iiWAS '11 Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services**, Nova Iorque, dez. 2011. 278-283. Disponível em: <<http://dl.acm.org/citation.cfm?id=2095583>>. Acesso em: 01 mar. 2012.

TAURION, C. Big Data: a nova fronteira da inovação. **iMasters**, 2011. Disponível em: <<http://imasters.com.br/artigo/22095/tendencias/big-data-a-nova-fronteira-da-inovacao>>. Acesso em: 27 maio 2012.

TERRA, J. C. C. Gestão do Conhecimento: aspectos conceituais e estudo exploratório sobre as práticas de empresas brasileiras, 1999. Disponível em: <<http://www.eps.ufsc.br/ergon/exercicios/Caso1.zip>>. Acesso em: 07 maio 2012.

TERRA, J. C. C. **Gestão do Conhecimento: O Grande desafio empresarial!** São Paulo: Elsevier, 2005. Disponível em: <http://www.pucrs.br/uni/poa/feng/civil/professores/giugliani/Gestao_Biblioteca_Terra_Forum.pdf>. Acesso em: 06 maio 2012.

TERRA, J. C. C. et al. **Taxonomia- elemento fundamental para a GC**. Rio de Janeiro: Campus, 1998. Disponível em: <<http://biblioteca.terraforum.com.br/Paginas/Taxonomia-elementofundamentalparaaGC.aspx>>. Acesso em: 13 maio 2012.

THE ECONOMIST. All too much. **The Economist**, 25 fev. 2010. Disponível em: <<http://www.economist.com/node/15557421>>. Acesso em: 22 fev. 2012.

THE ECONOMIST. Data, data everywhere. **The Economist**, 25t fev. 2010. Disponível em: <<http://www.economist.com/node/15557443>>. Acesso em: 2012 fev. 2012.

TIMON, C. R. L. et al. Retenção de Conhecimento em Instituições do Terceiro Setor. **UFRJ/COPPE**, Rio de Janeiro, dez. 2009. Disponível em: <http://portal.crie.coppe.ufrj.br/portal/data/documents/storedDocuments/%7B93787CAE-E94C-45C7-992B-9403F6F40836%7D/%7BA10F0CC6-27B4-47D4-86A5-21EF4EA7DC26%7D/RJ18_projeto%2002.pdf>. Acesso em: 13 maio 2012.

TIWARI, S. **Professional NoSQL**. [S.l.]: John Wiley, 2011. Disponível em: <http://books.google.com.br/books?hl=pt-BR&lr=&id=tv5iO9MnObUC&oi=fnd&pg=PT7&dq=%22Professional+NoSQL&ots=X3NSpExv75&sig=tLopQZs_Vsrl31fi__8HqOeNzLA>. Acesso em: 10 mar. 2012.

TOTH, R. M. Abordagem NoSQL - uma real alternativa, Sorocaba, abr. 2011. Disponível em: <http://www2.sor.ufscar.br/verdi/topicosCloud/nosql_artigo.pdf>. Acesso em: 30 maio 2012.

TSOTSIS, A. Twitter Is At 250 Million Tweets Per Day, iOS 5 Integration Made Signups Increase 3x. **Techcrunch**, 17 out. 2011. Disponível em: <<http://techcrunch.com/2011/10/17/twitter-is-at-250-million-tweets-per-day/>>. Acesso em: 11 mar. 2012.

TWITTER. About Twitter. **Twitter**, [21--]. Disponível em: <<https://twitter.com/about>>. Acesso em: 19 mar. 2012.

TWITTER. Documentation. **Twitter**, [21--]. Disponível em: <<https://dev.twitter.com/docs>>. Acesso em: 10 out. 2012.

VICKNAIR, C. et al. A Comparison of a Graph Database and a Relational Database. **ACM SE '10 Proceedings of the 48th Annual Southeast Regional Conference**, Oxford, abr. 2010. Disponível em: <<http://dl.acm.org/citation.cfm?id=1900008.1900067>>. Acesso em: 17 jun. 2012.

VIJAYAN, J. Big data: análise avançada é vital para os negócios. **Computerworld**, 26 ago. 2011. Disponível em: <<http://computerworld.uol.com.br/tecnologia/2011/08/25/big-data-analise-avancada-e-vital-para-os-negocios/>>. Acesso em: 22 fev. 2012.

VOGEL, M. J. M. Taxonomia: Produto ou Processo? **Terra Fórum**, 2009. Disponível em: <<http://www.terraforum.com.br/biblioteca/Documents/Forms/DispForm.aspx?ID=173>>. Acesso em: 13 maio 2012.

VOGELS, W. Amazon DynamoDB – a Fast and Scalable NoSQL Database Service Designed for Internet Scale Applications. **All Things Distributed**, 2012. Disponível em: <<http://www.allthingsdistributed.com/2012/01/amazon-dynamodb.html>>. Acesso em: 31 out. 2012.

XAVIER, W. O desafio da TI: como armazenar e processar tantas informações no cenário atual. **Olhar Digital**, 2012. Disponível em: <http://olhardigital.uol.com.br/negocios/digital_news/noticias/o-desafio-da-ti-como-armazenar-e-processar-tantas-informacoes-no-cenario-atual>. Acesso em: 26 maio 2012.