Column-Based Databases: Estudo exploratório no âmbito das Bases de Dados NoSQL

José Pedro Cunha [1], José Luís Pereira [2]

1) Universidade do Minho, Portugal
a58322@alunos.uminho.pt

2) Universidade do Minho, Portugal
jlmp@dsi.uminho.pt

Resumo

O aumento da quantidade de dados gerados que se tem verificado nos últimos anos e a que se tem vindo a dar o nome de *Big Data* levou a que a tecnologia relacional começasse a demonstrar algumas fragilidades no seu armazenamento e manuseamento o que levou ao aparecimento das bases de dados NoSQL. Estas estão divididas por quatro tipos distintos nomeadamente chave/valor, documentos, grafos e famílias de colunas. Este artigo é focado nas bases de dados do tipo *column-based* e nele serão analisados os dois sistemas deste tipo considerados mais relevantes: Cassandra e HBase.

Palavras chave: *Big Data*, NoSQL, *Column-based databases*, Cassandra, HBase

1. Introdução

Depois de várias décadas de grande sucesso e bons serviços prestados às organizações, a tecnologia relacional de bases de dados tem vindo a ser desafiada por uma nova classe de tecnologias de bases de dados a que se deu a designação genérica de NoSQL (*Not only* SQL). Para este facto contribuíram decisivamente os recentes desenvolvimentos na área a que se tem vindo a chamar *Big Data*, em que o aumento da quantidade de dados gerados diariamente em diversos domínios de aplicação como a *Web*, dispositivos móveis e principalmente as redes sociais, entre outros, está atualmente na ordem das centenas de *Terabytes* [Kuznetsov & Poskonin 2014]. Como tal, tendo em conta o volume e complexidade dos dados a gerir a tecnologia relacional começa a demonstrar fragilidades substanciais. Em particular, a necessidade de gerir dados cujos formatos são dificilmente acomodáveis em sistemas relacionais, dispersos por múltiplos servidores, levou ao aparecimento das ditas Bases de Dados NoSQL. Estas são principalmente focadas no desempenho, permitindo o processamento de dados de forma rápida e

eficiente. O seu modelo de dados não necessita de seguir os padrões rígidos do modelo relacional, pelo que armazenam tanto dados estruturados como não estruturados. Dentro desta nova classe de tecnologias de Bases de Dados surgiram diferentes propostas, com distintas proveniências e áreas de aplicação, vulgarmente classificadas em quatro grupos, de acordo com o seu modelo de dados: *Column, Document, Key/Value e Graph Based databases*, sendo que cada um destes modelos possui uma grande diversidade de propostas no mercado [Hossain & Moniruzzaman 2013].

Este artigo é focado nas bases de dados NoSQL do tipo *column-based* e tem como objetivo efetuar uma análise comparativa dos produtos tecnológicos mais relevantes deste grupo assim como compará-los com a tecnologia relacional de forma a compreender quais as suas caraterísticas, diferenças e principais vantagens. Assim, tendo em conta o seu prestígio, documentação existente e a quantidade de organizações de renome que utilizam os seus serviços, as bases de dados NoSQL do tipo *column-based* consideradas mais relevantes para análise são o Cassandra e o Apache HBase.

Em termos de estrutura, o artigo começa por abordar as principais caraterísticas das bases de dados NoSQL, assim como fazer uma breve s*í*ntese acerca de cada um dos quatro tipos de base de dados já mencionados. De seguida é feita uma análise individual às ferramentas Cassandra e HBase no que diz respeito ao seu modelo de dados, modelo de consulta e ainda à sua arquitetura. A seção seguinte é acerca do trabalho prático a realizar numa segunda etapa e por fim as conclusões retiradas acerca do estudo efetuado.

2. Caraterísticas das Bases de Dados NoSQL

O termo NoSQL foi utilizado pela primeira vez em 1998 por Carlo Strozzi ao fazer referência a uma base de dados relacional criada por si denominada "Strozzi SQL" que tinha a particularidade de não utilizar a linguagem SQL para efetuar consultas aos dados. O termo voltou a ser utilizado mais tarde em 2009 numa conferência organizada por Johan Oskarsson acerca de bases de dados do tipo *open-source* com o nome "NoSQL *meetup*", sendo que a partir daí passou a ser o termo da moda para referir bases de dados não relacionais [Abramova et al. 2014].

O seu modelo de dados não possui os padrões rígidos do modelo relacional pelo que armazenam dados estruturados como não estruturados. Esta ausência de esquema permite adicionar campos aos registos da base de dados sem que antes tenham que ser feitas alterações na estrutura da mesma. Ao contrário do modelo relacional, as bases de dados NoSQL não suportam consultas aos dados armazenados através da linguagem SQL. Contudo, permitem facilmente armazenar e

recuperar dados de forma rápida e eficiente independentemente da sua estrutura e conteúdo [Leavitt 2010].

De forma a evitar a perda de dados as bases de dados NoSQL possuem uma arquitetura distribuída tolerante a falhas que se baseia na distribuição dos dados por vários servidores pelo que, caso um servidor deixe de funcionar, o sistema manter-se-á em funcionamento garantindo assim elevados índices de disponibilidade. Outra das principais caraterísticas destas bases de dados não relacionais é a escalabilidade horizontal que consiste em aumentar o número de máquinas do sistema, dividindo o processamento dos dados conforme as necessidades do mesmo de forma a obter sempre elevados níveis de desempenho. Uma das técnicas utlizadas para garantir isto é denominada por *sharding* [Sadalage & Fowler 2013, página 38].

Relativamente à consistência, enquanto o modelo relacional segue as propriedades ACID que entre outras coisas providencia uma consistência forte dos dados, no caso das bases de dados NoSQL pode verificar-se apenas uma *eventual consistency* conforme as necessidades do sistema em questão e tendo em consideração o teorema CAP. Este refere que entre as propriedades consistência, disponibilidade e tolerância à partição, apenas duas delas podem ser asseguradas em simultâneo, sendo sempre necessário abdicar de uma delas. Normalmente a grande discussão que se coloca neste tipo de base de dados é entre a disponibilidade e a consistência visto que por se tratarem de sistemas distribuídos partições de rede são uma constante e como tal torna-se necessário que os sistemas sejam tolerantes a este tipo de falhas [Leavitt 2010].

Existem assim, quatro tipos de bases de dados NoSQL:

- ***Key/Value-based*:** destacam-se como sendo as mais simples e onde se pode verificar um melhor desempenho. Cada campo é constituído por uma chave e um valor e funciona como uma tabela *hash*, sendo que o conteúdo do valor pode ser armazenado em qualquer formato de dados. O acesso aos dados é sempre efetuado através da sua chave primária que é única e através das operações *get*, *put* e *delete*, sendo que em geral não são suportadas operações que abranjam conjuntos de valores [Atzeni et al. 2014]. Este tipo de armazenamento suporta uma elevada quantidade de dados e tem como principais vantagens a boa escalabilidade horizontal, simplicidade e o alto desempenho, sendo particularmente adequado para situações em que os dados não se relacionam. Por outro lado não é apropriado para realizar consultas complexas aos dados dado que este modelo não possui uma boa capacidade de indexação efetuando apenas operações *read* e *write* básicas. Num contexto real, este tipo de bases de dados são apropriadas para, por exemplo, gerir *stock* e armazenar informação relativa a perfis de utilizador ou compras em plataformas *online* dada a sua simplicidade e velocidade na recuperação dos dados

[Abramova et al. 2014]. As bases de dados do tipo *key-value* mais relevantes são o Riak e o Redis;

- ***Document-based*:** este tipo de base de dados baseiam-se em modelos de dados semiestruturados onde a informação é armazenada em forma de conjuntos de documentos que podem ter diferentes formatos tais como XML, BSON ou JSON. Cada documento é constituído por um conjunto de campos armazenados de forma não estruturada e que estão associados a uma chave única que os identifica. O seu modelo de consulta é considerado superior ao das restantes categorias de base de dados NoSQL pelo que permitem consultar conjuntos de documentos, efetuar consultas agregadas, incluir restrições sobre o campo de valores pretendido, ordenar resultados, entre outras coisas. Assim, este tipo de base de dados são apropriadas para problemas que envolvem ambientes variados e focam-se no armazenamento de grandes volumes de dados e no bom desempenho na consulta dos mesmos [Kuznetsov & Poskonin 2014]. Num contexto prático, este tipo de bases de dados representam uma boa solução para trabalhar com aplicações *web* com grandes quantidades de documentos que podem ser armazenados em ficheiros estruturados tais como documentos de texto, *emails*, sistemas XML ou sistemas CRM. São também uma boa opção para aplicações de comércio eletrónico [Abramova et al. 2014]. Os sistemas deste tipo mais adotados são o MongoDB e o Apache CouchDB;
- ***Graph-based*:** neste tipo de sistema o modelo é baseado em nós que se relacionam entre si por meio de uma aresta. Os nós contêm propriedades na forma de pares chave/valor e os relacionamentos têm sempre um nome associado e uma direção contendo um nó remetente e outro destinatário. As relações também podem conter propriedades [Robinson et al. 2013]. A organização do grafo permite que os dados depois de armazenados sejam interpretados de forma diferente com base nas diversas relações, sendo que um nó pode ter um número ilimitado de relações [Sadalage & Fowler 2013, página 113]. Estes sistemas são considerados ideais para lidar com dados que estejam interligados sendo importantes para, por exemplo analisar relacionamentos entre utilizadores de redes sociais, analisar sistemas de transportes e plataformas que sugerem produtos a utilizadores com base nos seus gostos e preferências [Abramova et al. 2014]. As principais referências deste tipo de bases de dados são Neo4J e o InfoGrid;
- ***Column-based:*** este tipo de sistema NoSQL destaca-se pela sua capacidade de armazenar grandes quantidades de dados. Ao contrário dos tradicionais modelos relacionais, uma linha corresponde a um conjunto de colunas associadas à mesma chave

primária, não existindo um esquema previamente definido em que cada coluna de uma linha tem o seu espaço atribuído mesmo que seja do tipo *NULL*. Cada linha pode conter um conjunto diferentes de colunas, o que se traduz num menor esforço computacional e contribui para aumentar o desempenho do sistema. Cada coluna é constituída pelo par nome e valor, sendo que o nome funciona como a chave da coluna. Além disto cada um destes é armazenado em conjunto com um valor *timestamp* que fornece informações acerca do valor temporal em que aqueles dados foram inseridos ou atualizados permitindo entre outras coisas resolver conflitos relacionados com *writes* e lidar com dados que estejam desatualizados. Para além disto as colunas são armazenadas por famílias de colunas de forma a facilitar a organização e a distribuição dos dados. Este modelo é o ideal para lidar com estruturas de dados complexas e volumosas [Sadalage & Fowler 2013, página 115]. Num contexto prático este tipo de base de dados é a ideal para, por exemplo, lidar com sistemas em que se verifica um elevado registo de eventos constante [Abramova et al. 2014]. As bases de dados mais populares deste tipo são o Cassandra e o Apache HBase.

3. Cassandra Vs HBase

De seguida apresentam-se sucintamente os dois produtos considerados mais representativos da classe de bases de dados NoSQL do tipo column-based.

3.1. Cassandra

O Cassandra foi desenvolvido pelo Facebook em 2008 como um projeto *open-source*, com o intuito inicial de melhorar o mecanismo do motor de busca da plataforma. Em 2009 foi adotado pela Apache Software Foundation e é hoje em dia, utilizado por grandes empresas como o Ebay, o Twitter e a Cisco Systems para além de órgãos governamentais como a NASA [Weber & Strauch 2010].

Os seus criadores, Avinash Lakshman e Prashant Malik, definem-na como uma base de dados distribuída e altamente escalável criada para armazenar grandes quantidades de dados distribuídos por vários servidores oferecendo alta disponibilidade e uma eventual consistência dos dados [Laksham & Prashant 2010].

3.1.1. Modelo de Dados

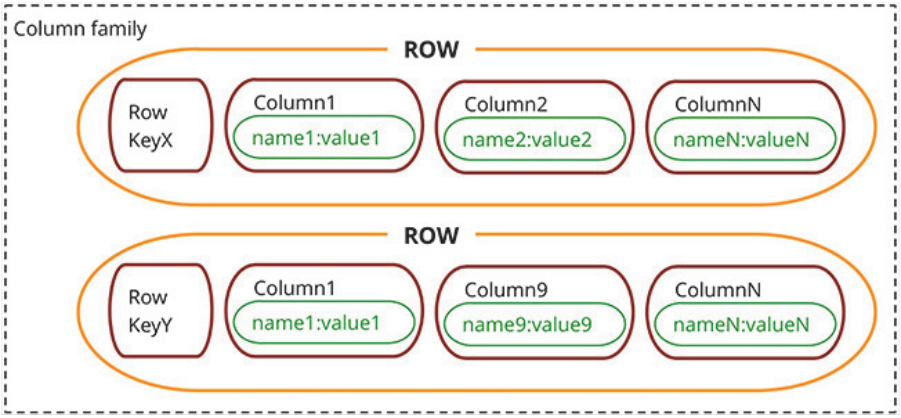

Figura 1 - Estrutura de uma família de colunas do Cassandra (retirado de [Sadalage & Fowler 2013])

O modelo de dados Cassandra é muito semelhante ao do BigTable e é constituído por [Datastax 2015]:

- **Coluna:** no Cassandra a unidade básica de armazenamento é a coluna. Uma coluna é constituída pelo par nome e valor, sendo que o nome funciona como chave da coluna. Além disto cada coluna possui um valor *timestamp* que fornece informações acerca do valor temporal em que os dados foram inseridos ou atualizados permitindo entre outras coisas resolver conflitos em operações *write* ou lidar com dados que estejam desatualizados. Uma coluna pode ser interpretada como um registo numa base de dados relacional;
- **Super Coluna:** pode ser vista como uma coluna com sub-colunas. Da mesma forma que uma coluna, uma super coluna contem um par chave/valor embora neste caso o valor esteja associado a um mapa que contem várias colunas;
- **Linha:** Ao contrário do que acontece num modelo relacional, uma linha é um conjunto de colunas associadas à mesma chave primária. Relativamente ao armazenamento dos dados enquanto num modelo relacional existe um esquema previamente definido em que cada coluna de uma dada linha tem o seu espaço atribuído mesmo que o seu valor seja *NULL*, neste caso só é atribuído espaço às colunas presentes nessa linha o que se traduz num menor esforço computacional e contribui para o aumento do desempenho;
- **Família de Colunas:** são constituídas por um número infinito de linhas, sendo que cada linha contem uma *row key* e um conjunto de colunas ordenadas pela sua chave. Uma família de colunas pode ser comparada a um conjunto de linhas de uma tabela do modelo relacional em que cada linha possui uma chave e está associada a um conjunto

de colunas. A grande diferença entre os dois modelos é o fato de no modelo não relacional não existir nenhum esquema pré-definido pelo que as linhas não têm que ter exatamente as mesmas colunas e estas podem ser adicionadas a uma só linha a qualquer instante sem que seja necessário que isso aconteça nas restantes [Sadalage & Fowler 2013, página 100];

- **Super Família de Colunas:** Semelhante a uma família de colunas, uma super família de colunas é constituída por um conjunto de super colunas, sendo que são úteis para manter dados que estão relacionados juntos [Sadalage & Fowler 2013, página 101].
- ***Keyspace*:** é semelhante a uma base de dados no sistema relacional onde se encontram armazenadas todas as famílias de colunas relacionadas com uma mesma aplicação. O *keyspace* aquando da sua criação define a forma como os dados serão replicados ao longo do *cluster*;

Relativamente às famílias de coluna, embora sejam muito flexíveis, na prática não são inteiramente *"schema-less"*, pelo que cada uma deve ser criada para contar apenas um tipo de dados [Datastax 2015].

3.1.2. Modelo de Consulta

Relativamente à API do Cassandra esta consiste apenas em três simples operações, nomeadamente:

- *get(table, key, columnName)*
- *insert(table, key, rowMutation)*
- *delete(table, key, columnName)*

Aqui o argumento *columnName* pode ser relativo tanto a uma coluna que pertença a uma família de colunas, como a uma família de colunas inteira, uma super família de colunas ou até mesmo a uma coluna pertencente a uma super coluna [Laksham & Prashant 2010].

Todos os acessos aos dados são feitos através da chave, sendo retornado o valor da coluna associada a essa chave. Existem também métodos adicionais que permitem obter vários valores assim como várias colunas tendo todos como base as chaves.

O Cassandra possui uma linguagem de consulta muito similar ao SQL denominada de CQL – *Cassandra Query Language*. Uma vez que a sua *syntax* é muito similar ao SQL torna-se mais fácil aos utilizadores interagir com os dados armazenados, verificando-se uma curva de aprendizagem reduzida [Abramova et al. 2014]. No entanto, CQL não permite efetuar *joins* ou subconsultas dos dados [Sadalage & Fowler 2013, página 114].

De forma a efetuar consultas mais complexas aos dados é possível executar trabalhos *MapReduce* na base de dados através da implementação Apache Hadoop destinada ao processamento de dados distribuídos [Datastax 2015].

3.1.3. Arquitetura do Sistema

Particionamento

Uma das principais caraterísticas do Cassandra é ser altamente escalável, ou seja, distribui os dados por um conjunto de nós pertencentes ao *cluster*, sendo que o Cassandra o faz através de um *consistent hashing*. A distribuição dos nós dentro do *cluster* pode ser compreendida mais facilmente fazendo uma analogia a um espaço circular ou um anel onde a cada nó do sistema é atribuído um valor aleatório chamado *token* que define a sua posição no anel. A cada conjunto de dados identificado por uma chave é atribuído um nó através de um *hashing* à sua chave, sendo que o sistema percorre o anel no sentido dos ponteiros do relógio e o primeiro nó encontrado que seja maior que o valor da chave define a sua posição no anel. Esse nó é considerado o nó coordenador para essa chave. Cada nó torna-se assim responsável pela região do anel entre si e o seu antecessor [Karger et al. 1997]. A principal vantagem da função *consistent hashing* é que a adição ou remoção de um nó apenas afeta os seus nós vizinhos. Esta função também apresenta alguns desafios nomeadamente em relação à distribuição não uniforme dos dados e o facto de ser alheio à heterogeneidade em relação ao desempenho dos nós. O Cassandra de forma a lidar com estes problemas tem como auxilio o denominado *"Gossip Protocol"* que garante que todos os nós do *cluster* estejam em constante comunicação, estando sempre atualizados relativamente ao estado dos outros nós [Hewitt 2010, página 88]. Assim o sistema faz uma análise dos dados presentes no anel e move os nós que tenham poucos dados de forma a aliviar aqueles que estejam sobrecarregados equilibrando a quantidade de dados presentes em todos os nós [Laksham & Prashant 2010].

Esta é uma solução utilizada antes da versão 1.2 do Cassandra, sendo que após a mesma o paradigma foi alterado e em vez de um nó estar associado a um *token* e um intervalo de valores, passou a estar associado a vários *tokens*, sendo este paradigma denominado de nós virtuais (*VNodes*). Este permite que cada nó possua pequenas partições que estão distribuídas por todo o *cluster* e utiliza igualmente um *consistent hashing* para distribuir os dados, sendo que neste caso não necessita de gerar e atribuir um *token* [Datastax 2015].

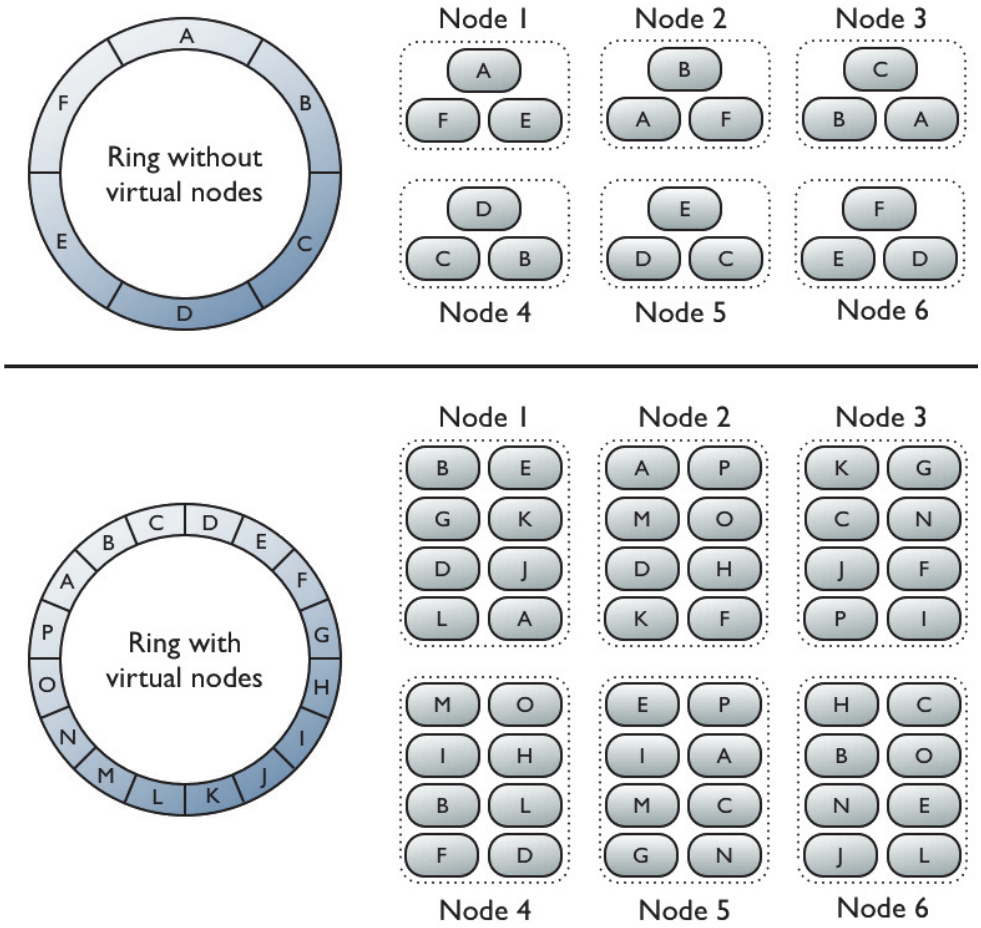

Figura 2 - Estrutura de Cluster Cassandra antes e pós versão 1.2 (retirado de [Datastax 2015])

Na figura 2 pode-se verificar ambas as estruturas, sendo que a parte superior representa o *cluster* ainda sem *VNodes*. Aqui pode-se verificar que por exemplo o intervalo "A" é replicado para os nós um, dois e três e também que cada nó possui exatamente partições consecutivas no anel, seguindo a direção do ponteiro dos relógios. A segunda parte da figura representa o paradigma *VNodes*, sendo estes escolhidos aleatoriamente e sem seguir qualquer sentido lógico [Datastax 2015].

Consistência

O Cassandra utiliza uma abordagem *quorum-based* configurável [Gifford 1979]. O cliente é capaz de especificar o nível de consistência por consulta o que lhe permite gerir a taxa de disponibilidade em relação à consistência. Um elevado nível de consistência implica que mais nós tenham que responder à consulta, oferecendo assim garantias de que os dados presentes em cada réplica são os mesmos. Caso dois nós respondam a um mesmo pedido com valores diferentes é feita uma análise ao valor *timestamp* de cada um e aquele que for mais recente é sempre retornado ao cliente, ocorrendo posteriormente uma operação *read repair* em que o nó que respondeu ao pedido com

um valor obsoleto é atualizado, de forma que todas as réplicas estejam consistentes [Hewitt 2010, página 130].

Tanto para operações *write* como *read* verifica-se a existência de vários níveis de consistência, sendo que estes têm interpretações diferentes conforme o tipo de operação. Para ambas as operações o nível que apresenta maior consistência é o *ALL*, enquanto que aquele onde se verifica uma menor consistência em prol de uma maior disponibilidade é o nível *ANY*. Para ambos os casos no nível *QUORUM* verifica-se um equilíbrio entre consistência e disponibilidade, sendo que existe a garantia de que 51% dos nós irão responder ao pedido efetuado [Datastax 2015].

O Cassandra permite assim que o cliente regule o nível de consistência que pretende tanto para operações *read* como *write* conforme as suas necessidades [Datastax 2015].

Replicação

O Cassandra recorre à replicação para alcançar elevados níveis de disponibilidade e de durabilidade do seu *cluster*, sendo cada conjunto de dados replicados por N *hosts* em que N diz respeito ao fator de replicação que pode ser configurável por instância [Sadalage & Fowler 2013, página 109].

O Cassandra oferece várias estratégias de replicação que devem ser definidas aquando da criação do *keyspace* entre as quais se destacam [Datastax 2015]:

- ***Simple Strategy*** – esta estratégia é recomendada quando se utiliza um único *data center*, sendo este definido como um grupo de nós relacionados no *cluster* com o objetivo de replicar. Nesta estratégia, a primeira réplica é definida pelo administrador do sistema e as restantes são colocadas nos seguintes nós do anel, seguindo a direção do ponteiro dos relógios e sem considerar qualquer topologia, como por exemplo, a localização do *data center;*
- ***Network Topology Strategy*** - é a estratégia recomendada para a maioria das implementações pois é mais fácil de se expandir por vários *data centers* quando necessário e especifica o número de réplicas que o utilizador pretender para *data center*. Esta estratégia tenta colocar réplicas em diferentes *racks*, de forma a prevenir problemas resultantes de quebras de energia ou falhas de rede que ao atingir um *rack* afetam todos os nós do mesmo. A decisão do número de réplicas a configurar por *data center* deve ser feita com base em algumas considerações como ser capaz de satisfazer operações *read* localmente sem problemas de latência e tendo em conta possíveis cenários de falha. As maneiras mais comuns que se verificam são duas ou três réplicas

em cada *data center* de forma a manter o sistema tolerante a falhas e também consistente.

No Cassandra verifica-se uma replicação do tipo *peer-to-peer* pois todos os nós se regem pelas mesmas regras, não havendo nenhum nó *master,* o que garante alta disponibilidade e escalabilidade [Sadalage & Fowler 2013, página 105].

Armazenamento Físico

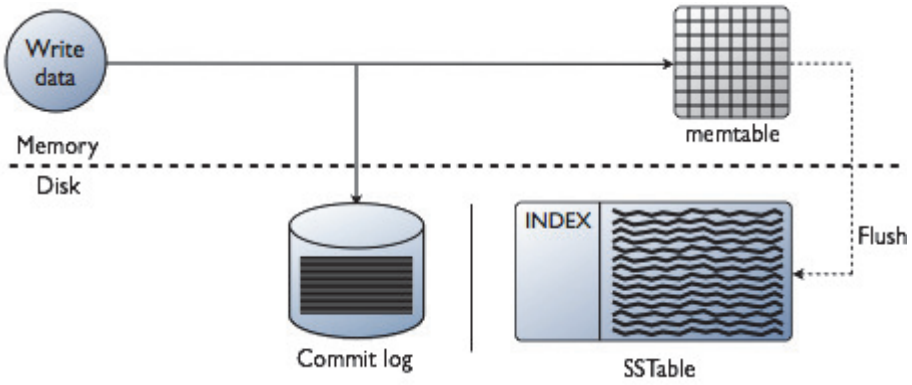

Figura 3 - Estrutura de armazenamento de *writes (*retirado de [Datastax 2015]*)*

Quando o Cassandra recebe um pedido *write*, os dados são inicialmente guardados numa estrutura do disco designada de *commitlog* e de seguida armazenados na memória RAM numa estrutura denominada *memtable*. Uma operação *write* apenas é bem-sucedida quando armazenada nas duas estruturas. Quando as operações *write* são executadas na base de dados o Cassandra disponibiliza automaticamente mais memória para a *memtable,* sendo as *writes* agrupadas e periodicamente descarregadas no disco numa estrutura designada por *SSTable*. Uma família de colunas pode ter associada vários *memtables,* sendo que apenas um se encontra em utilização e os restantes em espera para serem libertados. Caso ocorram alterações nos dados, é escrita uma nova *SSTable*, sendo que para que estas sejam apagadas é necessário que em primeiro lugar o seu estado seja definido como "apagada", para mais tarde durante o processo de compactação serem removidas do disco. O processo de compactação depois de eliminar as tabelas, agrupa os fragmentos de linhas existentes de forma a otimizar a procura de dados no disco [Hewitt 2010, página 91].

Disponibilidade

Como já foi referido anteriormente, no Cassandra todos os nós têm o mesmo peso, pelo que não existe nenhum nó *master*. Isto traduz-se em alta disponibilidade dos dados, sendo que o Cassandra

permite diversas configurações que permitem aumentar a disponibilidade em detrimento de alguma consistência nos pedidos, dependendo das necessidades da aplicação. Considere-se [Hewitt 2010, página 275]:

- R – número mínimo de nós que devem responder de forma síncrona a um pedido *read;*
- W – número mínimo de nós que devem responder de forma síncrona a um pedido *write;*
- N – corresponde ao fator de replicação, ou seja, é o número de nós que participam na replicação dos dados.

A disponibilidade é regulada pela formula (R + W) > N, sendo que sempre que esta é cumprida os dados a serem acedidos encontram-se na sua versão mais recente. Caso este cenário não se verifique a resposta aos pedidos *read* podem não se apresentar na versão mais recente dos dados pois a operação *write* associada pode ainda estar a ser replicada pelos respetivos nós, dando-se um caso de "*eventual consistency*" [Hewitt 2010, página 275].

Por outro lado, sempre que W assume o valor W=1, é dada a garantia que operações *write* estarão sempre disponíveis sem falhas a menos que todos os nós do *cluster* estejam indisponíveis, sendo que o mesmo se sucede com R. Quanto maiores os valores de R e W maior é o nível de consistência, o que nem sempre é o indicado para o bom desempenho das operações *read* e *write*. No que diz respeito ao N quanto maior o seu valor maior é o nível de durabilidade dos dados [Sadalage & Fowler 2013, página 106].

3.2. HBase

HBase é uma base de dados NoSQL que tem como base o BigTable da Google e foi desenvolvido em *Java* pela Powerset tendo-se tornado mais tarde parte do projeto Hadoop da Apache Software Foundations. O HBase é uma implementação *open-source* que executa sobre um sistema de ficheiros distribuídos denominado *Hadoop Distributed File System* (HDFS) fornecendo capacidades do BigTable ao Hadoop e que proporciona alta tolerância a falhas ao armazenar grandes quantidades de dados esparsos [Carstoiu et al. 2010].

3.2.1. Modelo de Dados

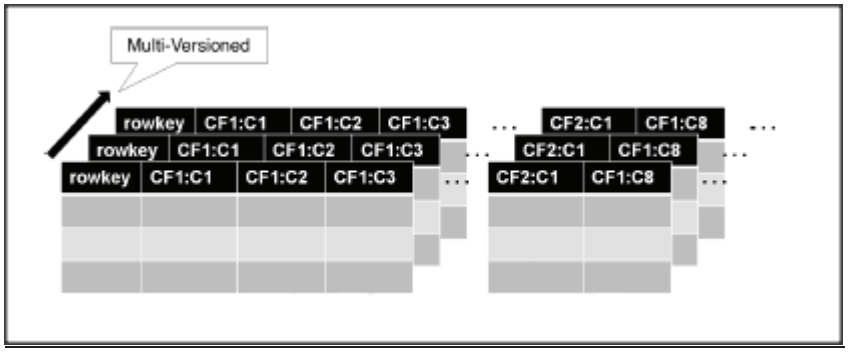

Figura 4- Modelo de Dados HBase (retirado de [Silva 2011])

O modelo de dados do HBase, também ele similar ao do BigTable é constituído pelos seguintes elementos [HBase 2015]:

- ***Namespace* –** É constituído por tabelas e corresponde a uma base de dados no modelo relacional;
- **Tabela** – Os dados no HBase encontram-se organizados por tabelas, sendo cada tabela constituída por várias linhas;
- **Linha** – Dentro de cada tabela os dados estão armazenados de acordo com a sua linha, sendo que cada linha é composta por uma *row key* e um número arbitrário de colunas. As linhas encontram-se ordenadas lexicograficamente pela sua *row key* pelo que a conceção das mesmas se torna um fator importante de forma a armazenar linhas que estão relacionadas próximas umas das outras de forma a otimizar o desempenho;
- **Família de coluna** – Os dados dentro de cada linha estão agrupados por famílias de colunas, sendo que estas devem ser definidas aquando da criação das tabelas. Cada linha de uma tabela possui as mesmas famílias de coluna, embora algumas não armazenem qualquer informação. As famílias de coluna possuem um *column qualifier* de forma a classificar o formato dos dados como por exemplo *pdf* ou *html*, sendo que estes podem variar bastante;
- **Coluna** – As colunas são agrupadas em famílias de colunas, sendo que todas as colunas da mesma família de colunas possuem o mesmo prefixo. As colunas possuem o seguinte formato: "*column family : column qualifier";*
- **Célula** – Uma célula representa a combinação de uma linha, uma família de coluna e um *column qualifier* e contem um valor aliado a um *timestamp* que representa a versão deste. Desta forma quando os valores são consultados é sempre retornada a versão mais

recente dos dados, sendo que o utilizador pode especificar qual versão ou quais versões pretende.

As atualizações das linhas são consideradas atómicas, abrangendo deste modo operações *read* e *write* ao mesmo tempo. As tabelas HBase são particionadas horizontalmente de forma automática em regiões pelo que cada região contém um conjunto de linhas delimitadas pela primeira linha e a última, sendo que esta já não pertence à região. Cada região possui ainda um identificador gerado aleatoriamente [HBase 2015].

3.2.2. Modelo de Consulta

O HBase fornece uma *API Java* que pode ser utilizada para realizar consultas, fornecendo operações básicas como *get*, *put*, *delete* e *update*. A função *scan* permite ainda lidar com intervalos de linhas, sendo capaz de selecionar quais as colunas a serem retornadas ou o número de versões de cada célula. Através da utilização de filtros é possível combinar colunas e selecionar as versões pretendidas, utilizando intervalos de tempo em que o utilizador deve especificar o tempo de início e de fim para a consulta pretendida. O HBase não possui nenhuma linguagem específica tal como o SQL para efetuar consultas aos dados pelo que recorre a outro tipo de técnicas [Lars 2011, página 75].

Para consultas mais complexas recorre a trabalhos *MapReduce* que executam subjacentemente a uma infraestrutura Hadoop denominada *Pig* que permite efetuar consultas aos dados através de uma linguagem de programação de alto nível muito semelhante ao SQL, conhecida como *Pig Latin*. A combinação desta linguagem com a *framework MapReduce* permite o processamento de grandes quantidades de dados em períodos de tempo razoáveis [Lars 2011, página 263].

Também é possível aceder ao HBase através das API *REST*, *Thrift* e ainda através de uma *Shell* HBase que permite aos utilizadores uma interação direta com o mesmo [HBase 2015].

3.2.3. Arquitetura do Sistema

Armazenamento Físico

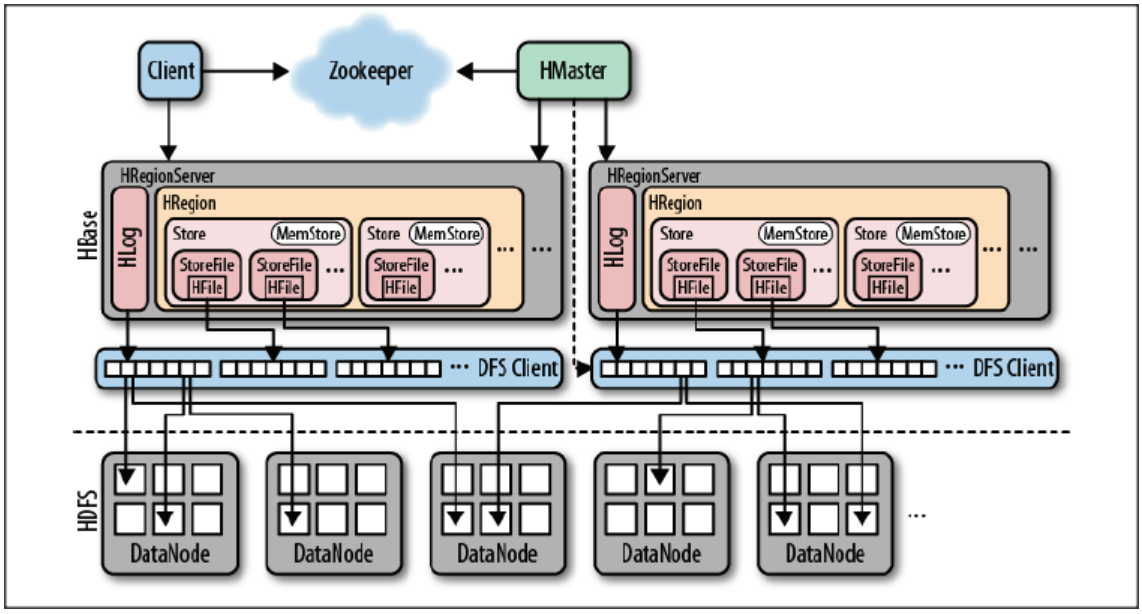

Figura 5- Visão geral de como o HBase armazena os seus ficheiros no HDFS (retirado de [Lars 2011])

Fisicamente no HBase as tabelas encontram-se distribuídas por intervalos de linhas denominados regiões. Uma região é a menor unidade de armazenamento distribuído, sendo que regiões diferentes são armazenadas em diferentes *RegionServers* pelo *master*. Uma tabela quando é criada está associada a apenas uma região, sendo que à medida que os dados vão sendo inseridos na tabela a região vai crescendo e quando o seu tamanho atinge o limite, o intervalo de linhas que a região abrange é divido para metade. Desta forma, à medida que mais dados vão sendo inseridos, mais regiões vão sendo criadas. Cada região possui ainda um *Hlog* e um *HStore*. Um *HStore* é o ponto central do armazenamento no HBase e consiste num ficheiro *MemStore* e um ou mais *StoreFiles*. Um *HStore* gere as famílias de coluna, daí ser importantes as colunas localizadas nas famílias de coluna terem caraterísticas comuns. Os dados inicialmente são gravados na *MemStore* e quando esta está cheia são libertados para o disco, criando um novo *StoreFile*. Aqui um indicador é enviado para o *HLog* que é onde são registadas todas as alterações que ocorrem aos dados no HBase. Assim que o número de *StoreFiles* atinge um limite previamente configurado é realizada uma compactação que reúne todos os *StoreFiles* num só [Yang et al. 2011].

O Hbase possui ainda uma tabela "hbase:meta" armazenada no Zookeeper que contem uma lista atualizada do estado, histórico e localização de todas as regiões do sistema [HBase 2015].

Sistema de Ficheiros Distribuídos Hadoop

Dado que o HBase executa sobre um HDFS torna-se necessário compreender a arquitetura deste, principalmente em relação ao armazenamento de ficheiros, à forma de lidar com falhas e à replicação de blocos.

O HDFS adota uma arquitetura *master-slave* em que um nó é o *master*, também conhecido como *NameNode* enquanto os restantes nós são denominados *DataNodes*. O *NameNode* coordena o *namespace* e operações no sistema de ficheiros como abrir, fechar ou alterar o nome dos mesmos. Além disto tem também a função de manter os metadados associados ao sistema tais como as permissões de acesso aos dados e localização dos ficheiros. No que diz respeito aos *DataNodes* encontram-se espalhados pelo *cluster* e têm como responsabilidade o armazenamento de dados que é feito em blocos [Borthakur 2013].

Um *DataNode* é responsável por atender aos pedidos *read* e *write* dos clientes aos ficheiros, sendo também encarregue de criar os blocos, apagá-los e replicá-los conforme as instruções dadas pelo *NameNode*. Dado que o HDFS foi desenvolvido em *Java* qualquer máquina que suporte *Java*, pode executar *software* para *NameNode* e *DataNode*. As decisões relacionadas com a replicação de blocos são sempre tomadas pelo *NameNode*, que recebe regularmente de cada *DataNode* do *cluster* informações acerca do movimento e proporção dos blocos utilizados. O mecanismo de replicação consiste em manter cópias redundantes dos dados no sistema, de forma que em caso de ocorrência de alguma falha exista sempre uma cópia disponível. O HDFS, por defeito, armazena três cópias em cada bloco para assegurar a confiabilidade, disponibilidade e a tolerância a falhas do sistema. A política de replicação habitual para um *data center* é ter duas máquinas réplica para o mesmo *rack* e uma réplica para um nó localizado noutro *rack*, sendo que isto limita o tráfego de dados entre *racks* e a hipótese de falha de um *rack* inteiro é muito menor que a hipótese de falha de um nó. Para minimizar a latência no acesso aos dados, o HDFS tenta aceder aos dados a partir da réplica mais próxima, pelo que se esta se encontrar no mesmo *rack* terá prioridade de escolha [Vora 2011].

De acordo com um estudo realizado por Zhou Yinan, o HDFS oferece o melhor desempenho no processamento de grandes volumes de dados. A estratégia de replicação é o seu ponto central e exerce grande influência no desempenho das operações *read* e *write*, sendo o facto de a posição dos dados redundantes poder ser ajustada, a grande diferença entre o HDFS e outros sistemas de ficheiros distribuídos [Yang et al. 2011].

Replicação

O HBase fornece um mecanismo de replicação do *cluster* que permite manter os estados entre *clusters* sincronizados utilizando um *write-ahead log* (*WAL*) do *cluster master* para propagar as alterações aos dados. Um *WAL* garante que as alterações aos dados sejam inicialmente gravadas num *Hlog* antes de serem aplicadas, fornecendo assim garantias de atomicidade e durabilidade dos dados [HBase 2015].

O modelo de replicação utilizado pelo HBase é do tipo *master-slave* em que um *cluster* HBase é o *master*, ou seja, o responsável por criar e replicar os dados e os restantes são denominados *slaves*, ou seja, os que recebem os dados por via da replicação. A replicação é feita de forma assíncrona, o que significa que os *clusters* podem estar geograficamente distantes, que a ligação entre eles pode falhar durante algum tempo e que as linhas inseridas no servidor *master* podem não estar disponíveis ao mesmo tempo nos servidores *slave*, verificando-se uma *eventual consistency* dos dados [HBase 2015].

A figura 6 representa o processo de replicação no HBase.

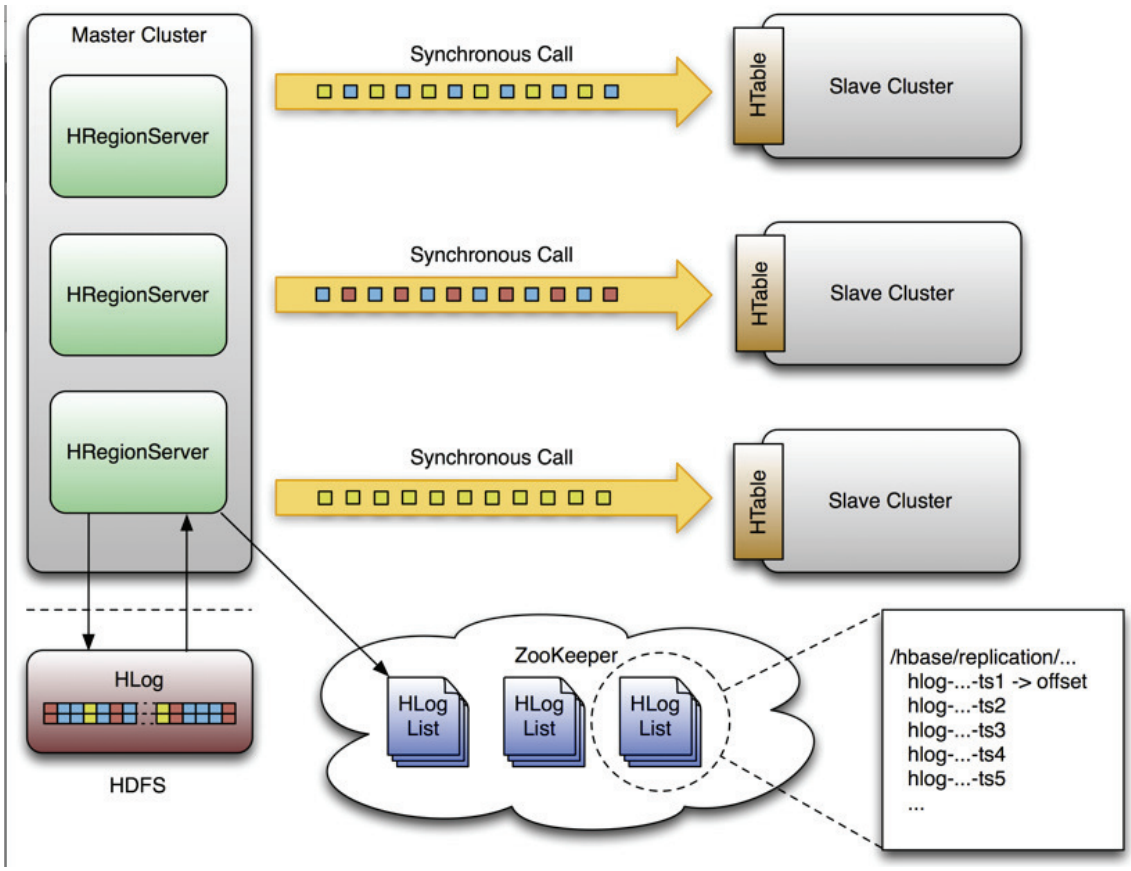

Figura 6 - Processo de Replicação HBase (retirado de [HBase 2015])

Os *WALs* de cada *RegionServer* são a base da replicação do HBase e devem manter-se no *HDFS* enquanto são necessários para replicar dados para qualquer *cluster slave*. Cada *RegionServer* lê a partir do *log* mais antigo que necessite de replicar e mantem o registo do seu progresso dentro do Zookeeper de forma a simplificar a recuperação de falhas. O Zookeeper é um serviço que faz o papel de coordenador e gere as principais atividades de replicação de forma a simplificar o processo de recuperação de falhas. O marcador da posição que indica o progresso, assim como a quantidade de *WAL's* por processar para todos os *clusters slave* pode ser diferente para todos [HBase 2015].

Consistência

Relativamente à consistência o HBase apresenta-se por defeito como estritamente consistente, dado que todos os valores surgem numa única região definida por um intervalo de chaves e cada região está atribuída a um único *RegionServer* de cada vez, o que garante que todas as operações *write* se realizam numa ordem e que todas as operações *read* acedem aos dados confirmados mais recentes.

No entanto, uma vez que a replicação no HBase é feita de forma assíncrona, encontrando-se a replicação ativa, quando são efetuados *reads* através de um servidor *slave*, apenas é garantida uma *eventual consistency* pois as atualizações aos dados por parte do *master* não chegam ao mesmo tempo a todos os *slaves* [HBase 2015].

4. Conclusões e trabalho futuro

Apesar de serem várias as semelhanças entre o Cassandra e o HBase existem também algumas diferenças importantes a verificar. Ao nível da infraestrutura o Cassandra possui apenas um único tipo de nó em que todos eles possuem as mesmas responsabilidades, enquanto no HBase isso não se verifica, existindo um nó considerado *master* e os restantes *slaves* com diferentes responsabilidades. Como tal, o HBase carateriza-se por ter uma forte consistência e ser otimizado para operações *read*. Por sua vez o Cassandra carateriza-se pela alta disponibilidade, sendo capaz de regular o nível de consistência pretendido, garantindo uma *eventual consistency* dos dados. O sistema Cassandra é otimizado para operações *write*. Assim, relativamente ao teorema CAP o HBase é considerado um sistema do tipo CP (consistente e tolerante a falhas) e o Cassandra um sistema AP (disponível e tolerante a falhas).

Relativamente à comunicação entre os nós, o Cassandra utiliza uma estratégia denominada *Gossip Protocol* enquanto o HBase recorre a um sistema externo denominado *Zookeeeper*. Quanto ao particionamento dos dados o HBase adota uma estratégia ordenada enquanto o Cassandra o faz

aleatoriamente. A replicação dos dados no HBase é feita de forma assíncrona enquanto no Cassandra se verifica de forma síncrona. Em relação ao modelo de consulta o Cassandra possui uma linguagem CQL muito semelhante ao SQL que permite ao utilizador uma rápida aprendizagem e suporta índices secundários por coluna de forma a melhorar a consulta aos dados. O HBase apesar de não suportar índices secundários suporta outros mecanismos que fornecem as mesmas funcionalidades, sendo também capaz de filtrar o acesso aos dados o que resulta em pesquisas mais rápidas. O HBase, ao contrário do Cassandra é também capaz de suportar nas suas consultas agregados como SUM, AVG, MAX. Independentemente disto, tanto num sistema como noutro é importante que colunas com padrões de acesso semelhantes sejam mantidas na mesma família de colunas de forma otimizar o desempenho das consultas. Daí ser importante planear previamente a organização das famílias de colunas.

Posto isto o trabalho futuro passa por identificar um caso de estudo, com dados a rondar os milhões de registos e comparar o desempenho dos dois sistemas ao nível do acesso e consulta aos dados em diferentes cenários e retirar conclusões acerca do desempenho de cada um dos sistemas.

5. Referências

Abramova, V., Bernardino, J. e Furtado, P.," Experimental Evaluation of NoSQL Databases" *International Journal of Database Management Systems*, 6, 3 (2014), 1–16.

Atzeni, P., Bugiotti, F. e Rossi, L., "Uniform access to NoSQL systems". *Information Systems*, 43, 2014, 117–133.

Bordhakur, D., *HDFS Architecture Guide*, http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, (20 de Julho de 2015), 2013.

Carstoiu, D., Cernian, A. e Olteanu, A., "Hadoop Hbase-0.20.2 performance evaluation", *New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on*, 2010, 84-87.

Datastax, *DataStax Cassandra 2.2 for Linux*, http://docs.datastax.com/en/cassandra/2.2/cassandra/cassandraAbout.html, (5 de Maio de 2015), 2015

Gifford, D.K.,"Weighted voting for replicated data", *Proceedings of the seventh symposium on Operating systems principles - SOSP '79*, 1979, 150–162.

HBase, A., *Apache HBase ™ Reference Guide*, http://hbase.apache.org/book.html, (5 de Maio de 2015), 2015.

Hewitt, E., *Cassandra - The Definitive Guide*, O'Reilly Media, Inc., 2010

Hossain, S.A. e Moniruzzaman, A., "NoSQL Database : New Era of Databases for Big data Analytics - Classification , Characteristics and Comparison", *International Journal of Database Theory and Application*, 6, 4 (2013), 1–14.

Karger, D. et al., "Consistent Hashing and Random Trees", *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*, 1997, 654–663.

Kuznetsov, S.D. e Poskonin, A. V., "NoSQL data management systems", *Programming and Computer Software*, 40, 6 (2014), 323–332.

Laksham, A. & Prashant, M., "Cassandra: a decentralized structured storage system", *ACM SIGOPS Operating Systems Review*, 2010, 1–6.

Lars, G., *HBase: The Definitive Guide*, O'Reilly Media, Inc., 2011.

Leavitt, N., "Will NoSQL Databases Live Up to Their Promise?", *Computer*, 43, 2010, 12–14.

Robinson, I., Webber, J. e Eifrem, E., *Graph Databases*, O'Reilly Media, Inc., 2013.

Sadalage, P. e Fowler, M., *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*, Addison-Wesley, 2013.

Silva, C., *Data Modeling with NoSQL: How, When and Why*, Tese de mestrado integrado, Engenharia Informática e Computação, Faculdade de Engenharia da Universidade do Porto, 2010.

Vora, M.N., "Hadoop-HBase for large-scale data", *Proceedings of 2011 International Conference on Computer Science and Network Technology, ICCSNT 2011*, 2011, 601–605.

Weber, S. e Strauch, C., "NoSQL Databases", *Lecture Notes Stuttgart Media*, 2010, 1–8.

Yang, J., Tang, D. e Zhou, Y., "A distributed storage model for EHR based on HBase", *Proceedings - 2011 4th International Conference on Information Management, Innovation Management and Industrial Engineering, ICIII 2011*, 2011, 369–372.