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

Gonçalo Sousa [1], José Luís Pereira [2]

1) Universidade do Minho, Portugal
a58333@alunos.uminho.pt

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

Resumo

Com este artigo pretende-se dar um contributo para a clarificação de uma nova área da tecnologia de bases de dados que emergiu, recentemente, como resposta às dificuldades apresentadas pelo tradicional modelo relacional, no que diz respeito ao suporte do chamado *Big Data* – as bases de dados *NoSQL*. Em particular, pretende-se abordar uma das quatro classes de bases de dados em que o mundo *NoSQL* se divide – as *Document-Based databases*, de que produtos como o MongoDB e o *CouchDB* são alguns dos principais representantes. Como propósito do trabalho em que este artigo se enquadra, pretende-se perceber quais as diferenças, a nível de desempenho, entre soluções suportadas por estes dois produtos, quando utilizados no suporte de bases de dados de dimensão crescente, procurando evidenciar também as diferenças entre estas e soluções construídas recorrendo à bem conhecida tecnologia relacional.

Palavras chave: *Big Data*, bases de dados *NoSQL*, *Document-Based Databases*.

1. Introdução

O crescimento exponencial da chamada Web 2.0 resultou na geração de cada vez maiores volumes de dados, caraterizados por uma grande diversidade de formatos, de que é necessário tirar partido. O armazenamento, a gestão e o processamento destes novos dados veio evidenciar as limitações das bases de dados relacionais, tanto em termos de capacidade de lidar com os grandes volumes de dados, como dos tipos de dados a suportar.

Atualmente são gerados enormes volumes de dados a todo o instante e por todo o tipo de dispositivos, desde eletrodomésticos inteligentes, até ao comércio *online*. Existem cada vez mais sistemas ligados à internet que comunicam entre si. Nos últimos anos, os avanços nas tecnologias Web, o incremento na utilização das redes sociais e a proliferação de sensores e dispositivos móveis conectados à internet, resultou na geração de imensos dados que necessitam de ser armazenados e posteriormente processados. Esta situação forçou uma evolução ao nível das tecnologias de bases de dados para responder aos novos desafios:

- Grandes níveis de concorrência na leitura e escrita de dados;

- Para não defraudar as expectativas dos utilizadores, as bases de dados têm de fornecer tempos de resposta suficientemente baixos;
- Armazenamento de grandes volumes de dados (*Big Data*) eficiente e com determinados requisitos de acesso;
- Aplicações muito exigentes em termos de processamento, como motores de pesquisa e plataformas para redes sociais, têm de responder em tempo útil às solicitações de milhões de utilizadores;
- Para além da escalabilidade é fundamental uma alta disponibilidade das soluções;
- Cada vez mais o volume de dados e os pedidos concorrentes aumentam, pelo que as bases de dados necessitam de ser facilmente expandidas sem interrupções;
- A juntar a tudo isto, exigem-se baixos custos de gestão e operação das soluções.

Neste contexto, uma nova tecnologia de bases de dados emergiu e está a tornar-se cada vez mais importante no mercado – as bases de dados *NoSQL* [Han et al. 2011]. Interessa, pois, conhecer esta nova tecnologia e perceber o contributo que pode dar para suportar as necessidades do *Big Data*. Neste contexto será aqui abordada a classe das bases de dados *NoSQL* designadas por *Document-Based*.

2. Big Data

Atualmente o armazenamento e processamento de grandes volumes de dados na área da computação é um grande desafio. O mundo digital está a crescer a olhos vistos e a tornar-se cada vez mais complexo, quer relativamente ao volume de dados a tratar (do *Terabyte* ao *Petabyte*), à sua variedade (dados estruturados, semiestruturados e não-estruturados) e à velocidade a que têm de ser processados (ritmos cada vez mais rápidos).

A este fenómeno chama-se *Big Data*, e pode ser definido como uma grande coleção de dados que cresceu tanto e de forma tão diversificada que não pode ser gerido ou explorado eficientemente com as tradicionais ferramentas da tecnologia relacional [Moniruzzaman & Hossain 2013].

O termo *Big Data* surgiu como referência a um conjunto de problemas derivados das enormes quantidades de dados produzidos por empresas, organizações e pessoas particulares. Posteriormente, passou a descrever "o conjunto de soluções que visam tratar esses dados, desde o seu armazenamento até à transformação destes em informações relevantes para auxiliarem os gestores nas tomadas de decisões" [Basso & Padua 2014].

Segundo Alexandre e Cavique [2013], o fenómeno *Big Data* assenta nos chamados 3V's:

- **Volume**: A necessidade de lidar, de forma sustentada, com o crescimento exponencial de dados gerados;
- **Velocidade**: A necessidade de processar os dados em tempo-real, devido à importância que isso tem para as organizações atuais;
- **Variedade**: O facto de os dados se apresentarem nos mais variados formatos, alguns deles dificilmente acomodáveis em estruturas rígidas.

A quantidade de dados gerada diariamente está-se a tornar abismal, desde redes sociais, web, entre outros. Isto leva a problemas ao nível do processamento, armazenamento e manipulação de dados, que faz com que os sistemas relacionais não sejam os mais indicados para lidar com o problema do *Big Data*. Por essa razão foram então criados os Sistemas *NoSQL*, que dispõem de melhorias ao nível do armazenamento e processamento de grandes quantidade de dados, principalmente híbridos e não estruturados [Vieira et al. 2012].

O *NoSQL* veio resolver o problema da grande quantidade de dados que são gerados e têm de ser tratados. Para além disso deve levar em consideração a necessidade das organizações manterem esses dados durante longos períodos de tempo [Alexandre & Cavique 2013].

3. Surgimento do conceito *NoSQL*

Para entender melhor este novo conceito é importante perceber em que consiste uma base de dados. Esta pode ser definida, muito simplesmente, como um repositório onde se armazenam dados de vários tipos, interligados através de algum mecanismo que permita manter as referências entre si.

E.F.Codd propôs o modelo relacional de base de dados em 1970. Na altura, este veio simplificar o armazenamento de dados e o seu processamento, ao migrar os dados para tabelas relacionadas entre si por campos comuns, manipulados por uma linguagem de alto-nível – a SQL. O que autor não previu foi que a quantidade de dados que se teria de armazenar no futuro se tornaria tão gigantesca como é hoje [Cardoso 2012].

Em relação ao conceito de *NoSQL*, este foi inicialmente usado em 1998 por Carlo Strozzi para referir uma base de dados *open source* que não utilizava a referida interface SQL [Abramova et al. 2014; Strauch 2010].

Pozzani [2013] refere que esta tecnologia foi desenvolvida inicialmente na Google, sendo que o primeiro *research paper* foi publicado em 2003. Uma dos primeiros movimentos de mudança em direção ao *NoSQL* ocorreu em 2007, quando a *Amazon* apresentou o *NoSQL*

Dynamo. Esta foi das primeiras empresas a guardar os seus dados numa base dados de características não-relacionais [Leavitt 2010]. O exemplo do sistema *Dynamo* veio depois inspirar vários outros projetos com um armazenamento de dados alternativo ao modelo relacional.

Strauch [2010] acrescenta que, com o aparecimento da *web2.0*, empresas famosas como *Google*, *Amazon*, entre outras, começaram a largar as tradicionais bases de dados relacionais como o *Oracle* e o *DB2* para começarem a construir os seus próprios sistemas, isto para armazenar e processar grandes quantidades de dados.

Tomando como exemplo uma das maiores redes sociais do momento, o *Facebook*, só nesta rede 2.4 biliões de conteúdos são partilhados entre amigos todos os dias [Grolinger et al. 2013; Parikh 2013]. Neste tipo de contextos a tecnologia relacional começou a apresentar limitações especialmente relacionadas com a escalabilidade e o desempenho.

As novas exigências fizeram com que a indústria das TIC surgisse com novas propostas para estes dois problemas, resultantes das limitações dos sistemas de armazenamento relacional. Empresas como o *Facebook* e a *Google* concluíram que seria impossível gerir grandes quantidades de dados com as bases de dados relacionais e começaram então a procurar soluções alternativas aos sistemas relacionais. A alternativa que surgiu denominou-se *NoSQL* [Cardoso 2012; Guimarães et al. 2013; Hecht & Jablonski 2011]. A tecnologia *NoSQL* não foi criada com o intuito de substituir o modelo relacional mas sim preencher uma necessidade do mercado que este não servia adequadamente.

Diferentemente das bases de dados relacionais que, de uma forma ou outra, são todas relativamente semelhantes, a área das bases de dados *NoSQL* caracteriza-se por possuir diferentes tipos que variam muito entre si, tendo cada uma as suas próprias caraterísticas e finalidades [Vieira et al. 2012].

4. NoSQL

NoSQL é um termo utilizado para descrever uma classe de tecnologias que fornecem uma alternativa para o armazenamento de dados em comparação com os sistemas de gestão de bases de dados relacionais. Caracterizam-se por fornecer soluções efetivas e de baixo custo para os problemas de disponibilidade e escalabilidade [Schram & Anderson 2012].

Inicialmente, o termo NoSQL era suposto significar "No SQL", mas devido a poucos sistemas eliminarem completamente a tecnologia relacional, foi então atualizado para "Not only SQL" [Schram & Anderson 2012]. Esta tecnologia é uma alternativa viável relativamente às bases de

dados relacionais, pois garante melhor desempenho nas consultas enquanto mantêm um certo nível de escalabilidade e flexibilidade [Lee et al. 2013].

Mesmo que não exista um consenso no que exatamente constitui uma solução NoSQL, o seguinte conjunto de caraterísticas é normalmente atribuído a estas [Grolinger et al. 2013]:

- São completamente livres de esquemas, de modo a poderem lidar com uma grande variedade de tipos de dados;
- Possibilitam escalar horizontalmente as soluções através da adição incremental de novos nodos (commodity servers);
- Focadas na disponibilidade e na tolerância ao particionamento. Sendo assim, estão apontadas para cenários de alta distribuição. Portanto, escolhem comprometer a consistência em favor da disponibilidade;
- Normalmente, não suportam transações ACID, tal como são fornecidas pelas bases de dados relacionais. Neste âmbito as bases de dados NoSQL são normalmente referidas como sistemas BASE [Pritchett, 2008]. No entanto, algumas bases de dados NoSQL, como o CouchDB, suportam transações ACID.

Este tipo de tecnologia começa a encontrar-se em operação em grandes centros de dados, onde as falhas são uma ocorrência constante, tanto devido a problemas de disco, como comunicação de rede. Esta tecnologia torna-se apropriada nesses contextos já que possui recursos para permitir a replicação de dados em várias máquinas, o que se traduz num aumento da disponibilidade dos sistemas [Queiroz et al. 2013].

5. ACID VS BASE

Enquanto o ACID fornece um conjunto de propriedades que garantem que as transações das bases de dados são processadas com confiança, o oposto modelo BASE, que é derivado diretamente do teorema CAP, aponta para fornecer um diferente conjunto de propriedades das que o ACID fornece [Silva 2011].

O acrónimo ACID significa, *Atomicity*, *Consistency*, *Isolation* e *Durability*, e é usado pela generalidade dos sistemas de gestão de base de dados relacionais para assegurar a integridade das bases de dados. As suas caraterísticas constituintes significam o seguinte:

- ***Atomicity*** – Caso exista uma falha na transação, o efeito das suas operações não é aplicado na base de dados;
- ***Consistency*** – A integridade da base de dados é garantida no final da execução de

uma transação;

- ***Isolation*** – O resultado de uma execução de uma transação concorrente é o mesmo do que uma transação isolada;
- ***Durability*** – Após a confirmação de uma transação, mesmo que ocorram falhas os resultados são refletidos na base de dados.

ACID fornece um conjunto de propriedades que garantem que as transações das bases de dados são processadas com confiança e que a base de dados é consistente em acessos concorrentes e falhas do sistema [Silva 2011].

Em relação ao modelo BASE, Toth [2011] refere que para que seja possível trabalhar com um sistema distribuído foi proposto enfraquecer ou relaxar o requisito de consistência permanente de dados, focando antes na disponibilidade e na tolerância ao particionamento. Deste modo surge o conceito BASE *(Basically Available, Soft state, Eventual consistency)* com propriedades e filosofia oposta.

Toth [2011] refere ainda que é considerado um cenário que lida bem com transações distribuídas, tolerância a falhas de consistência, e replicação otimista num sistema distribuído.

Vieira et al. [2012] assim como Toth [2011], referem que o paradigma BASE foi criado a partir do teorema CAP. O primeiro autor ainda refere que o facto de não se controlar a consistência, faz com que exista uma sensível diminuição no custo computacional para a garantia de consistência dos dados em relação às bases de dados relacionais. Já o segundo autor refere que este modelo de consistência está normalmente associado a sistemas distribuídos. Refere também, que todas as atualizações pendentes serão propagadas por todos os nós do sistema, caso durante um determinado tempo não existam atualizações, ficando assim no fim, todo o sistema consistente. A partir deste conceito vários tipos de bases de dados *NoSQL* foram criadas que, como já foi referido, não garantem temporariamente a consistência dos dados de uma aplicação. Toth [2011] refere grandes empresas, como *Google* e a *Amazon*, que aproveitaram a vantagem deste paradigma, isto porque utilizaram um modelo *NoSQL*, aplicado ao conceito de distribuição horizontal. Como não existe garantia de consistência, as configurações de fragmentação no modelo são mais fáceis de efetuar e, por conseguinte, o modelo torna-se mais escalável e custo reduzido.

Resumindo, *Basically Available* significa que as bases de dados estão disponíveis sempre que são acedidas, mesmo que uma parte esteja indisponível. *Soft state* diz que não precisam sempre de ser consistentes e podem tolerar inconsistência por um certo período de tempo; e *Eventual*

consistency enfatiza que, depois de um certo período de tempo, a base de dados vem para um estado de consistência.

6. Teorema CAP

O CAP (*Consistency, Availability, Partition tolerance*) foi introduzido em 2000 pelo cientista de computação Eric Brewer [2000] como uma conjetura e foi formalmente provada em 2002 por Seth Gilbert e Nancy Lynch [2002], sendo assim, estabelecida como teorema [Cardoso 2012; Silva 2011].

Toth [2011] refere que o teorema CAP, enquadrando-se na área da computação distribuída, assenta em três requisitos que descreve da seguinte forma:

- **Consistência**: Todos os nós de um sistema distribuído necessitam que a versão dos dados que armazenam seja a mesma;
- **Disponibilidade**: Ainda que um nó da rede esteja desligado, todos os clientes podem encontrar, sempre que pretenderem, pelo menos uma cópia dos dados que necessitam;
- **Tolerância ao particionamento**: Caso o sistema seja implementado em servidores diferentes, continua com os seus dados, propriedades e caraterísticas, sempre transparentes para o utilizador.

De acordo com este teorema, num sistema distribuído torna-se impossível garantir, simultaneamente, mais que dois daqueles três requisitos. Na figura seguinte estão representadas as combinações possíveis.

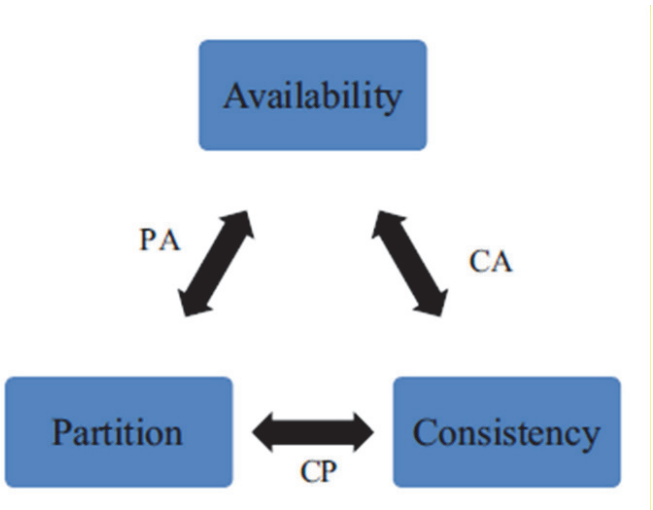

Figura 1 - Teorema CAP [Indrawan-Santiago 2012, pag.47]

Segundo Dimitrov [2010], as combinações possíveis dos três requisitos processa-se da seguinte forma:

- CA - Consistência e Disponibilidade
  - *Clusters* únicos, fácil de assegurar devido a todos os nodos estarem sempre em contacto;
  - Exemplo: 2 PC;
  - Quando um particionamento ocorre o sistema bloqueia.
- CP - Consistência e Tolerância ao particionamento
  - Alguns dados podem ficar inacessíveis (disponibilidade sacrificada), mas o resto é consistente;
  - Exemplo: *sharded database*.
- PA - Disponibilidade e Tolerância ao particionamento
  - O sistema está disponível mesmo durante o particionamento, mas alguns dados podem-se revelar inconsistentes;
  - Exemplo: *DNS*, *caches*, *Master*/*Slave replication*;
  - Necessita de alguma estratégia de resolução de conflitos.

Tipicamente as bases de dados *NoSQL* são PA (*Partition tolerance*/*Availability*) enquanto a maioria das relacionais são CA (Consistency/Availability).

7. Tipos de bases de dados *NoSQL*

Segundo diversos autores (Toth [2011], Bonnet et al. [2011] ou Guimarães et al. [2013], entre outros), as bases de dados *NoSQL* dividem-se em quatro tipos principais:

- *Document-based databases*, que têm como finalidade armazenar documentos em coleções, coleções estas que contêm objetos idênticos. Não possuem esquema desde que o número de campos não seja limitado e pode ser dinamicamente adicionada a um documento. (por exemplo, o *CouchDB* e o MongoDB);
- *Column-based Databases*, que têm como finalidade armazenar os dados como

triplas contendo linha, coluna e rótulo de tempo (por exemplo, o Cassandra e o HBase);

- *Key/Value-based Databases*, que têm como finalidade armazenar os dados como chaves e valores (por exemplo, o Redis e o DynamoDB);
- *Graph-based Databases*, que têm como finalidade armazenar relacionamentos entre os dados (por exemplo, o Neo4J e o *BigData*).

Neste trabalho interessam-nos particularmente as *Document-based Databases*, pelo que serão sucintamente apresentadas de seguida.

8. Document-based databases

Estas bases de dados são organizadas em coleções de documentos, ao contrário das bases de dados relacionais em que se usam tabelas estruturadas com campos para cada registo. Também é permitido ao utilizador adicionar vários campos de diferentes tamanhos a um documento [Leavitt 2010].

Nestas bases de dados o objetivo é gerir e armazenar documentos. Estes documentos podem assumir os formatos *XML, JSON (Javascript Object Notation) ou BSON (Binary JSON)*, que são formatos padrão na troca de dados [Moniruzzaman & Hossain 2013].

Uma vez que estas bases de dados não possuem um esquema definido para lidar com os dados faz com que sejam uma boa solução para as típicas aplicações da *web 2.0*, assim como para gerir conteúdos associados às redes sociais [Queiroz et al. 2013].

As *Document-based databases* agrupam pares de chave-valor em documentos. A cada documento é atribuída uma chave especial "ID", sendo que esta é atribuída a cada um destes separadamente. Desta forma, é possível tratar dados estruturados complexos como *Nested objects* mais convenientemente [Hecht & Jablonski 2011].

Dentro de uma coleção, os documentos podem ser vistos de forma sequencial, podem ser recuperados individualmente utilizando o seu identificador, ou efetuar uma consulta respeitando a sua estrutura de campos. É igualmente possível introduzir um novo documento a qualquer altura numa coleção [Atzeni et al. 2013].

O facto de as relações entre os dados estarem integradas dentro do próprio documento pode originar dados duplicados, no entanto possibilita uma consulta direta. Desta forma, tendo como base uma estrutura que tenha uma relação Cliente com Encomendas (um cliente pode ter várias encomendas), se uma consulta fosse criada para retornar as encomendas do Cliente "X", neste

tipo de base de dados bastaria obter o documento correspondente ao Cliente "X", que este por sua vez já traria os documentos correspondentes às suas encomendas. Já nas bases de dados relacionais seria necessário fazer uma junção das tabelas Clientes e Encomendas, o que levaria a uma diminuição de desempenho na sua execução, mais sentida caso as tabelas tenham alguma dimensão [Almeida & Brito 2012]. Uma outra vantagem de possuir dados duplicados é que "facilita a distribuição do sistema, já que a quantidade de nós a serem consultados numa busca envolvendo várias entidades relacionadas é menor caso elas estejam próximas" [Diana & Gerosa 2010].

Relativamente aos documentos abaixo apresentados pode-se visualizar que eles são similares mas podem ter diferenças em termos de estrutura e conteúdo. Isto é permitido nas *Document-based databases*. Ainda que a estrutura de dados possa diferir entre documentos, estes podem pertencer à mesma coleção, ao contrário das bases de dados relacionais em que cada linha de uma tabela tem de seguir a mesma estrutura [Sadalage & Fowler 2012].

```
{
    nome: "Gonçalo",
    idade: 24,
    morada: "Azurém, Guimarães",
    Gostos: ["Ginásio", "Futebol"]
    Data_Nascimento: "04-06-1991",
    Emprego: "Estudante",
    Estado Civil: "Solteiro",

}
```

```
{
    nome: "Luís",
    idade: 45,
    morada: {Freguesia: "Azurém",
             Cidade: "Guimarães",
             CódigoPostal: 4800},
    Gostos: ["Bicleta", "Futebol"]
    Data_Nascimento: "03-06-1970",
    Emprego: "Engenheiro Informático",
    Estado Civil: "Casado",

}
```

Figura 2 - Exemplo de dois documentos de uma mesma coleção

As bases de dados que serão alvo de atenção neste trabalho são o MongoDB e o CouchDB, já que são as mais representativas no mercado e consideradas como sendo as mais completas da família de bases de dados *NoSQL* do tipo *Document-based databases*.

8.1 MongoDB

O MongoDB é uma *Document-based database*, *open-source*, escrita na linguagem C++. A companhia 10gen também fornece suporte a esta base de dados. Não possui qualquer tipo de esquema e gere documentos JSON como o CouchDB. Foca-se no alto desempenho e desenvolvimento ágil, fornecendo ao programador um conjunto de caraterísticas para facilmente consultar dados, ao mesmo tempo que facilita a escalabilidade do sistema.

Em relação ao modelo de dados, o MongoDB utiliza uma base de dados composta por coleções

de documentos no formato BSON, que é uma versão muito semelhante ao JSON mas numa representação binária, por razões de eficiência e por disponibilizar tipos de dados adicionais em comparação com o JSON" [Strauch 2010]. Segundo Queiroz et al. [2013], estas coleções de documentos relembram os conceitos de tabelas e linhas da tecnologia relacional, mas com a particularidade dos documentos não necessitarem de ter o mesmo esquema. Sendo assim, os documentos podem conter campos que são diferentes de outros documentos (ver Figura 3).

```
{
     "_id" : "001",
     "tipo": "hospital",
     "nome": "Santa Casa",
"endereco" : {
       "logradouro": "R. das Rosas",
           "numero": 95,
           "cidade": "Ouro Preto" }
},
{
        "_id": "999",
       "tipo": "foco_queimada",
   "long_lat": [-46.76, -20.54],
   "satelite": "GOES-12",
 "observacao": "2012-07-05 05:15:00"
}
```

Figura 3 - Exemplo de documentos diferentes de uma mesma coleção [Queiroz et al. 2013, pág.485]

Cada documento possui um identificador único (_id), que pode ser fornecido pelo utilizador quando um documento é criado, ou automaticamente gerado pela base de dados. No campo _id é automaticamente criado um índice, embora outros índices possam ser criados manualmente com o objetivo de melhorar o desempenho das consultas.

Em relação ao modelo de consultas o MongoDB suporta consultas dinâmicas (ad hoc) sobre todos os documentos existentes numa coleção, em analogia com as consultas SQL disponibilizadas pelos sistemas relacionais. Segundo Silva [2011], "*as consultas são expressadas como objetos JSON e são enviadas para o MongoDB por database drivers (normalmente usando o método "find").*" Por exemplo, na figura seguinte apresenta-se um exemplo simples de uma consulta a uma base de dados de todos os livros contendo o título "Flatland".

```
1 db.books.find({'title': 'Flatland'})
```

Figura 4 - Consulta de todos os livros com o título Flatland [Silva 2011, pág.17]

Como se pode constatar na Figura 5, também é possível fazer a ordenação e a contagem de resultados. Neste caso, na primeira consulta todos os livros que estão escritos em inglês são recuperados, ordenando-os pelo seu título. Já a segunda consulta fornece o número de livros escritos em inglês. Do mesmo modo, estão também disponíveis operadores de agregação ($max, $min, $avg, $sum, etc.), que é possível fazer atuar sobre grupos de documentos.

```
1 db.books.find({'language': 'English'}).sort({'title':1})
2 db.books.find({'language': 'English'}).count()
```

Figura 5 - Exemplo de duas consultas [Silva 2011, pág.17]

Em relação ao modelo de replicação o MongoDB fornece replicação *master-slave* e *replica sets*, sendo que os dados são replicados assincronamente entre servidores. Em ambos os casos, para operações de escrita, apenas um servidor é usado, enquanto as operações de leitura podem ser redirecionadas para os servidores *slave*. Os dados são replicados através dos servidores do conjunto.

Em relação ao modelo de consistência o facto de o MongoDB possuir apenas um *master* ativo, a consistência total é possível se todas as operações de leitura forem realizadas no master. Se a replicação para os servidores *slave* for feita assincronamente e o MongoDB não necessitar de fornecer controlo de concorrência das versões, a leitura nos servidores *slave* será de *eventual consistency*. O facto de poder existir leitura nos servidores *slave* possibilita o balanceamento de carga e, sendo assim, o cliente também pode fazer com que a escrita seja replicada para servidores *slave*. Isto ajuda a lidar com escritas importantes, em que a *eventual consistency* possa não ser adequada, porque assim consegue-se flexibilidade de leitura para ler através dos servidores *slave*.

8.2 *CouchDB*

A base de dados Apache CouchDB é também uma *Document-based database open-source*. Foi criada pela Fundação Apache e desenvolvida em Erlang - uma linguagem de programação utilizada para aplicações concorrentes e distribuídas. As propriedades ACID são cumpridas nesta base de dados, fornecendo atualizações serializadas. É já, pela sua natureza, distribuída e suporta vários tipos de esquemas de replicação e resolução de conflitos.

Em relação ao seu modelo de dados uma base de dados CouchDB é formada por uma coleção independente de documentos em formato JSON [Queiroz et al. 2013]. Estes documentos são

compostos por pares de chave:valor. Os valores podem ser de vários tipos: números, strings, boleanos, listas e dicionários. Os documentos não possuem qualquer tipo de esquema e, sendo assim, não têm de seguir nenhuma estrutura, bastando respeitar a sintaxe do JSON.

A cada documento são atribuídos pelo menos dois atributos: um número de revisão (_rev), necessário para a resolução de conflitos durante atualizações, e um identificador único (_id). Na Figura 6 está representado um exemplo de um livro nesta linguagem.

```
{
 "_id":"282e1890a8095bcc0cb1318bed85bcb377e2700f",
 "_rev":"946B7D1C",
 "Type":"Book"
 "Title":"Flatland",
 "Author":"Edwin A. Abbot",
 "Date":"2009-10-09",
 "Language":"English",
 "ISBN":"1449548660",
 "Tags":["flatland", "mathematics", "geometry"],
}
```

Figura 6 - Documento com a descrição de um livro [Silva 2011, pág.14]

Nas palavras de Silva [2011], esta base de dados "*é um simples contentor de uma coleção de documentos e não estabelece nenhuma relação obrigatória entre eles*." (Anderson et al. 2010)
Embora o CouchDB não imponha nenhum tipo de relação entre os documentos, é importante possuir algum tipo de estrutura que retrate como os documentos são organizados. Sendo assim, o CouchDB fornece um *view model*.
Segundo Silva [2011], *views* são métodos de agregar documentos existentes na base de dados. Já Queiroz et al. [2013] refere que estas permitem a filtragem de partes específicas ou criar índices para acelerar a procura de um documento. As *views* não afetam os dados de uma base de dados, simplesmente mudam a maneira como os dados são representados, definindo assim o design da aplicação. Silva [2011] refere também que "*para definir uma view o utilizador deve fornecer uma função javascript que atua como uma função map da operação Mapreduce. Esta função leva um documento como um argumento e é permitido manipular os dados do documento da maneira que se desejar. A view resultante é um documento que pode ter múltiplas linhas*."
Na figura anterior (Figura 6) foi apresentado um extrato de uma coleção de documentos de livros. Partindo do princípio que é pretendido obter o título de todos os livros que estão armazenados na base de dados, criamos uma função *map* correspondente para ser usada na *view* como é mostrado na Figura 7.

```
1 function(doc) {
2   if (doc.Type == "Book") {
3     emit(null, {Title: doc.Title};
4   }
5 }
```

Figura 7 - Exemplo de uma função utilizada para obter o título de todos os livros armazenados na base de dados [Silva 2011, pág.14]

Em relação ao modelo de consultas o CouchDB utiliza uma RESTful HTTP API para efetuar operações CRUD básicas em todos os dados armazenados e utiliza métodos, como os HTTP POST, GET, PUT e DELETE para as realizar. Consultas mais complexas podem ser traduzidas através de *views*, como se viu anteriormente, e o resultado pode ser lido utilizando esta *REST API* [Anderson et al. 2010; Silva 2011].

Relativamente ao modelo de replicação o CouchDB é um sistema de base de dados distribuído do tipo *peer-based*, permitindo *peers* para atualização e alteração de dados, sincronizando bidireccionalmente as mudanças. Portanto, utilizando o *master-slave setup*, em que as sincronizações são feitas unilateralmente, ou o *master-master setup*, em que as mudanças podem acontecer em qualquer um dos nodos, estes devem ser sincronizados bidireccionalmente.

Para isto acontecer, o CouchDB utiliza uma replicação otimista e tem a capacidade de lidar com conflitos, que podem acontecer devido a replicações de alterações. Como já foi referido anteriormente, um número de revisão (_rev) é atribuído a cada documento, isto porque sempre que um documento é atualizado, a versão antiga é mantida e atribuído à versão atualizada um número de revisão diferente. Portanto, quando um conflito é detetado, a versão que prevalecer é gravada como a mais recente e a versão que não prevalecer é também gravada no histórico do documento. Isto é realizado de forma consistente em todos os nós do sistema. Existe também a possibilidade da aplicação escolher lidar com o conflito por ela própria, ignorando uma versão [Anderson et al. 2010].

Em relação à consistência, o CouchDB fornece *eventual consistency*. Como vários *masters* são permitidos, as alterações têm a necessidade de serem propagadas aos nodos restantes. Deste modo não existem bloqueios de escrita. Portanto, até estas alterações se propagarem de nodo para nodo, a base de dados permanece num estado de inconsistência. Como são suportados *master setups* únicos com múltiplos *slaves*, a consistência total (*strong consistency*) pode ser alcançada (Anderson et al. 2010).

9. Conclusões e trabalho futuro

Durante este trabalho, a revisão de literatura realizada permitiu clarificar alguns aspetos relativos às bases de dados NoSQL. Nomeadamente, as razões da sua entrada fulgurante no mercado e o papel que o fenómeno *Big Data* teve no impulsionamento destas tecnologias.

Exploradas as características principais das duas *Document-based databases* selecionadas é já claro que, embora pertencentes à mesma família NoSQL, o MongoDB e o CouchDB, distinguem-se uma da outra em alguns aspetos importantes. Nomeadamente, em algumas particularidades do modelo de dados, no modo como efetuam a replicação, como fazem as consultas e como implementam a consistência.

Como trabalho futuro pretende-se comparar os dois produtos, em termos do desempenho que é possível obter, relativamente ao suporte de bases de dados com dimensões sucessivamente maiores. Este estudo, para cada um dos volumes de dados em análise, envolverá a definição de algumas consultas de complexidade diversa, verificando-se como as duas bases de dados se comportam. Adicionalmente, pretende-se também comparar este tipo de soluções NoSQL com as soluções que lhes corresponderiam usando a tradicional tecnologia relacional, de modo a tirar outras conclusões.

10. Referências

Abramova, V., Bernardino, J., & Pedro, F., "*Experimental Evaluation of NoSQL Databases*", International Journal of Database Management Systems (IJDMS), (2014), 1-16.

Alexandre, J., & Cavique, L., "*NoSQL no suporte à análise de grande volume de dados*", Revista de Ciencias de Computação, 8, (2013), 37 – 48.

Almeida, R. C. De, & Brito, P. F. De., *Utilização da Classe de Banco de Dados NOSQL como Solução para Manipulação de Diversas Estruturas de Dados*, Palmas, 2012.

Anderson, J. C., Lehnardt, J., & Slater, N., *CouchDB: The Definitive Guide*, O. M. Inc., Ed, 2010.

Atzeni, P., Bugiotti, F., & Rossi, L., "Uniform access to *NoSQL* systems", Information Systems, 43, (2013), 117–133.

Basso, C., & Padua, R. de., *O USO DE BIG DATA NAS UNIVERSIDADES*, Seminário

de Iniciação Científica, 94, 2014.

Bonnet, L., Laurent, A., Sala, M., Laurent, B., & Sicard, N., "*Reduce, you say: What NoSQL can do for data aggregation and BI in large repositories*", Proceedings -

International Workshop on Database and Expert Systems Applications, DEXA, (2011), 483–488.

Brewer, E., "*Towards Robust Distributed Systems"*, In Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, Vol. 19, (2000), 1–12.

Cardoso, R., *Bases de Dados NoSQL*, Instituto Superior de Engenharia do Porto, 2012.

Diana, M. De. Gerosa, M. A., *NoSQL na Web 2.0: Um estudo comparativo de bancos Não-Relacionais para Armazenamento de Dados na Web 2.0*, WTDBD 2010-IX Workshop de Teses E Dissertações Em Banco de Dados, 2010.

Dimitrov, M., *NoSQL Databases*, Ontotext, 2010.

Guimarães, T., Nakai, A., & Vieira, L., *Banco de dados NoSQL para integração de bases de dados de gases de efeito estufa*, MOSTRA DE ESTAGIÁRIOS E BOLSISTAS DA EMBRAPA INFORMÁTICA AGROPECUÁRIA, 9, Brasília, 2013.

Gilbert, S., & Lynch, N., *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*, ACM SIGACT News, 33, 51, 2002.

Grolinger, K., Higashino, W. a, Tiwari, A., & Capretz, M. A., *Data management in cloud environments: NoSQL and NewSQL data stores*, Journal of Cloud Computing: Advances, Systems and Applications, 2, 22, (2013).

Han, J., Haihong, E., Le, G., & Du, J., "*Survey on NoSQL database"*, Proceedings International Conference on Pervasive Computing and Applications, ICPCA, (2011), 363– 366.

Hecht, R., & Jablonski, S., "*NoSQL evaluation: A use case oriented survey"*, International Conference on Cloud and Service Computing, (2011), 336–341.

Leavitt, N., "*Will NoSQL Databases Live Up to Their Promise?"* Computer, 43, 2 (2010), 12–14.

Lee, K. K. Y., Tang, W. C., & Choi, K. S., "*Alternatives to relational database: Comparison of NoSQL and XML approaches for clinical data storage"*, Computer Methods and Programs in Biomedicine, 110, 1 (2013), 99–109.

Moniruzzaman, A., & Hossain, S., "*NoSQL database: New era of databases for big data analytics-classification, characteristics and comparison"*, arXiv Preprint arXiv: 1307.0191, 6, 4 (2013), 1–14.

Parikh, J. (2013). *Facebook Newsroom : A new Datacenter for Iowa*, http://newsroom.fb.com/ (11 de Fevereiro de 2015), 2013.

Pozzani, G., *Introduction to NOSQL*, *NoSQL* Seminar, 2013.

Queiroz, G. R. De, Miguel, A., Monteiro, V., & Câmara, G., "Geographic Databases and *NoSQL* : Accomplishments and Future Directions", Revista Brasileira de Cartografia, (2013), 479 – 492.

Sadalage, P. J., & Fowler, M., *NoSQL Distilled*, 2012.

Schram, A., & Anderson, K. M., "*MySQL to NoSQL"*, Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity - SPLASH '12, 191 (2012).

Silva, C. A. R. F. O. Da., *Data modeling with NoSQL: How, when and why,* Faculdade de Engenharia da Universidade do Porto, 2011.

Strauch, C., *NoSQL Databases*, Stuttgart, 2010.

Toth, R. M., *Abordagem NoSQL- Uma real Alternativa*, Sorocaba, 2011.

Vieira, M., FIGUEIREDO, J., Liberatti, G., & Viebrantz, A., *"Bancos de Dados NoSQL: conceitos, ferramentas, linguagens e estudos de casos no contexto de Big Data*.", Simpósio Brasileiro DE Banco de Dados, 1 (2012), 1 – 30.