Bancos de Dados NoSQL x SGBDs Relacionais: Análise Comparativa*

Ricardo W. Brito, Faculdade Farias Brito e Universidade de Fortaleza, ricardow@ffb.edu.br

***Resumo*—O Modelo Relacional tem sido amplamente utilizado em praticamente todos os tipos de sistemas de bancos de dados nas últimas décadas. Porém, com o crescimento cada vez mais intenso do volume de dados de certas organizações, certos fatores limitantes têm propiciado que modelos alternativos de banco de dados sejam utilizados em tais cenários. Motivados principalmente pela questão da escalabilidade do sistema, uma nova geração de bancos de dados – conhecidos como NoSQL – vem ganhando força e espaço tanto na academia quanto no mercado. Neste artigo, são apresentadas as principais características desses bancos de dados e se discute de que forma essas novas soluções podem abordar certas questões atualmente enfrentadas.**

***Index Terms* — Banco de Dados, Modelo Relacional, NoSQl, Escalabilidade, Disponibilidade, Performance.**

I. Introdução

Desde sua criação no início dos anos 1970, o Modelo Relacional de dados tem sido utilizado em larga escala pela grande maioria dos sistemas de gerenciamento de banco de dados.

Tendo surgido como sucessor dos modelos hierárquico e de rede, o modelo relacional tornou-se padrão para a grande maioria dos SGBDs (Sistemas Gerenciadores de Banco de Dados), tais como o SQL Server, Oracle, PostgreSQL, MySQL, etc. Seus elementos básicos são as relações (ou tabelas), as quais são compostas de linhas (ou tuplas) e colunas (ou atributos). Os dados estão estruturados conforme esse modelo [5].

Outra característica fundamental desse modelo é a utilização de restrições de integridade. Esses elementos são utilizados para garantir que a integridade dos dados seja mantida. As restrições de integridade mais comuns são as chaves, mais especificamente, as chaves primárias e as chaves estrangeiras.

A chave primária tem o objetivo de assegurar a identificação única das tuplas das tabelas. A chave estrangeira torna os valores de determinado atributo dependentes dos valores existentes em outro atributo, normalmente de outra tabela.

Outra característica importante do Modelo Relacional é o processo de Normalização. Seu objetivo é a aplicação de certas regras sobre as tabelas do banco de dados, de forma a garantir o projeto adequado dessas tabelas. Uma característica básica da normalização consiste na separação dos dados referentes a elementos distintos em tabelas distintas, associadas através da utilização das chaves.

Adicionalmente, o modelo relacional passou a adotar como linguagem de definição, manipulação e consulta de dados a SQL (*Structured Query Language*). Desenvolvida originalmente pela IBM, o SQL é uma linguagem declarativa de para banco de dados relacional inspirada na álgebra relacional. Sua simplicidade e alto poder de expressão fizeram do SQL a linguagem de consulta de dados mais utilizada no mundo e ajudou a consolidar a posição dominando do modelo relacional.

II. SGBDs Relacionais

Os SGBDs relacionais oferecem aos usuários processos de validação, verificação e garantias de integridade dos dados, controle de concorrência, recuperação de falhas, segurança, controle de transações, otimização de consultas, dentre outros.

A utilização de tais recursos facilitou a vida dos desenvolvedores de aplicações, possibilitando que estes pudessem se preocupar exclusivamente com o foco da aplicação.

Como um dos conceitos mais básicos do modelo relacional, as chaves representam uma forma simples e eficaz de associação entre as tabelas do banco de dados. A *chave primária* foi criada com o objetivo de identificar de forma única as tuplas da tabela e ainda de determinar a ordem física dessas tuplas. A *chave estrangeira* permite uma relação de dependência entre atributos de tabelas distintas, de forma que os valores permitidos em um atributo dependam dos valores existentes em outro atributo.

Tais recursos são amplamente utilizados em bancos de dados relacionais e servem como base para a utilização de outros componentes, como é o caso dos *índices*. Estes elementos tornaram-se padrão para todo tipo de tabela por propiciarem um significativo ganho de performance no processamento de consultas.

Além desses componentes, os SGDBs Relacionais possibilitam que múltiplos usuários possam acessar e manipular um mesmo banco de dados simultaneamente de forma eficiente, recurso indispensável para sistemas de grande porte.

* Este trabalho teve o apoio do CNPq-RHAE e da UNUM.

Outra característica importante dos SGDBs relacionais consiste na possibilidade do sistema se recuperar de forma adequada de possíveis falhas. O sistema tem a capacidade de retornar ao ponto anterior â falha ocorrida, garantindo que o banco permanecerá em um estado consistente.

Todos esses recursos ajudaram a manter os SGDBs Relacionais em posição de destaque entre os mais diversos tipos de ambientes computacionais, mas não impediram o surgimento de certos problemas, principalmente devido ao crescimento vertiginoso do volume de dados presentes nos bancos de certas organizações.

III. LIMITAÇÕES

Devido ao intenso crescimento do número de aplicações, soluções, recursos e tudo o que se refere a sistemas computacionais nas últimas décadas, o volume de dados associados a tais tipos de sistemas também teve um ritmo de crescimento acelerado.

Atualmente, o volume de dados de determinadas organizações, como é o caso do Google, atinge o nível de petabytes, o que corresponde a 10^{15} bytes. Em cenários nos quais o gerenciamento de grande volume de dados tem se mostrado problemático, a utilização de SGBDs relacionais, ou em outras palavras do próprio Modelo Relacional, já não se mostra tão eficiente.

De um modo geral, os principais problemas encontrados com a utilização do Modelo Relacional estavam concentrados na dificuldade em se conciliar tal modelo com a demanda por escalabilidade cada vez mais frequente.

Como exemplo, tomemos uma determinada aplicação web sendo executada sobre um SGBD relacional. Com o crescimento do número de usuários, o sistema começa a ter uma queda de performance. Para superar esse problema, restam duas alternativas promover um *upgrade* no servidor ou aumentar o número de servidores.

Tais opções não seriam suficientes se o número de usuários continuar a crescer em ritmo intenso porque o problema passa a se concentrar no próprio acesso à base de dados. A solução, portanto, passa a ser escalar o banco de dados, distribuindo-o em várias máquinas. Tal abordagem é possível em um SGBD relacional, porém, não é realizada de maneira tão simples.

Devido a sua natureza estruturada, os desenvolvedores começaram a perceber a dificuldade em se organizar os dados do Modelo Relacional em um sistema distribuído trabalhando com particionamento de dados [8]. É justamente nesse ponto em que o foco das soluções não-relacionais estão concentradas.

IV. BANCOS DE DADOS NOSQL

As mudanças ocorridas na tentativa de se propor alternativas ao uso do Modelo Relacional levaram os desenvolvedores a pensar em um modo alternativo de se modelar as bases de dados. A estrutura pouco flexível utilizada até então passou a ser um problema a ser contornado e as soluções propostas tinham como base a eliminação ou minimização dessa estruturação.

O objetivo dos projetistas de banco de dados de organizações de grande porte passou a ser desenvolver uma nova estratégia de armazenamento na qual pudessem estar livres de certas estruturas e regras presentes no Modelo Relacional. Assim, foram surgindo soluções que pareciam voltar no tempo, retornando ao simples sistemas de gerenciamento de arquivos.

Se, por um lado, tais soluções perdiam todo o arcabouço de regras de consistência presentes no Modelo Relacional, por outro lado, poderiam ganhar em performance, flexibilizando os sistemas de banco de dados para as características particulares de cada organização.

O termo NoSQL surgiu em 1998, a partir de uma solução de banco de dados que não oferecia uma interface SQL, mas esse sistema ainda era baseado na arquitetura relacional [14]. Posteriormente, o termo passou a representar soluções que promoviam uma alternativa ao Modelo Relacional, tornando-se uma abreviação de *Not Only SQL* (não apenas SQL), sendo utilizado principalmente em casos em que o Modelo Relacional não apresentava performance adequada.

O propósito, portanto, das soluções NoSQL não é substituir o Modelo Relacional como um todo, mas apenas em casos nos quais seja necessária uma maior flexibilidade da estruturação do banco. Em português já existe o acrônimo MRNN para Modelo Relacional Não-Normalizado.

Uma das primeiras implementações de um sistema realmente não-relacional surgiu em 2004 quando o Google lançou o BigTable, um banco de dados proprietário de alta performance que tinha como objetivo promover maior escalabilidade e disponibilidade. A idéia central era justamente flexibilizar a forte estruturação utilizada pelo Modelo Relacional [4].

Outra implementação foi realizada em 2007, pela Amazon, com a apresentação do sistema Dynamo, o qual tinha como característica básica ser um banco de dados não-relacional, de alta disponibilidade, utilizado pelos *web services* da Amazon [6].

Em 2008, iniciou-se outro importante projeto. O Cassandra foi projetado para ser um sistema de banco de dados distribuído e de alta disponibilidade, desenvolvido pelo site Facebook para lidar com grandes volumes de dados [7]. No início de 2010, o Cassandra desbancou o MySQL como banco de dados do Twitter, demonstrando sua importância cada vez mais crescente [17].

Na mesma época, começou o desenvolvimento do Apache CouchDB, um banco de dados livre e orientado a documentos, os quais armazenam os dados sem a necessidade de qualquer esquema pré-definido, e que utiliza o conceito de arquitetura *MapReduce*, especialmente projetada para suportar computação distribuída em larga escala [1], [3], [9].

Outro solução emergente, lançada em 2009, foi o MongoDB, um banco de dados orientado a documentos,

escalável, de alta performance e livre de esquemas, com várias características semelhantes ao CouchDB [15].

Todas essas soluções apresentam certas características em comum e outras particulares que os diferenciam. O capítulo a seguir descreve e categoriza as principais características dos principais sistemas de banco de dados NoSQL.

V. Classificação de Bancos de Dados NoSQL

Apesar de possuírem certas características em comum, tais como serem livres de esquema, promoverem alta disponibilidade e maior escalabilidade, os sistemas de bancos de dados NoSQL existentes possuem diversas singularidades.

Quando à distribuição de dados, certos sistemas promovem o particionamento e a replicação dos dados, enquanto outros deixam essa tarefa para o cliente. A maioria das soluções é distribuída, como é caso do Amazon Dynamo, do CouchDB, do MongoDb, do BigTable e do Cassandra.

Quando ao modelo de dados, existem quatro categorias básicas: os sistemas baseados em armazenamento chave-valor, como é o caso do Amazon Dynamo; os sistemas orientados a documentos, entre os quais temos o CouchDB e o MongoDB; os sistemas orientados a coluna, que tem como exemplos o Cassandra e o BigTable; e os sistemas baseados em grafos, como são os casos do Neo4j e do InfoGrid.

Na primeira categoria, existe uma coleção de chaves únicas e de valores, os quais são associados com as chaves.

No segundo grupo, os documentos são as unidades básicas de armazenamento e estes não utilizam necessariamente qualquer tipo de estruturação pré-definida, como é o caso das tabelas do Modelo Relacional. Um documento do CouchDB [1], [16], por exemplo, é um objeto que possui um identificador único e que consiste de certos campos. Esse documento segue o formato JSON (JavaScript Object Notation), padrão que visa uma fácil legibilidade e independência de linguagem [12]. No exemplo ilustrado na Figura 1, o documento possui cinco campos com seus respectivos valores.

```
"Assunto": "Banco de Dados"
"Autor": "Wagner"
"Data": "15/03/2010"
"Tags": ["NoSQL", "banco de dados", "couchdb"]
"Body": "O CouchDB é um interessante banco de dados NoSQL"
```

Figura 1. Exemplo de documento do CouchDB

Na terceira categoria, muda-se o paradigma de orientação a registros (ou tuplas) para orientação a atributos (ou colunas). O Cassandra possui colunas (tuplas que contém nome, valor e *timestamp*), famílias de colunas (um repositório para colunas, análogo a uma tabela do Modelo Relacional) e super-colunas (compostas por *arrays* de colunas) [7].

Na quarta categoria, os dados são armazenados em nós de um grafo cujas arestas representam o tipo de associação entre esses nós.

Quanto ao modo de armazenamento dos dados, temos os bancos que mantêm suas informações em memória realizando persistências ocasionais; aqueles que mantêm suas informações em disco, como são os casos do CouchDb e do MongoDb; e aqueles configuráveis, tais como BigTable e o Cassandra.

A seguir, são confrontadas certas características entre os Sistemas Gerenciados de Bancos de Dados relacionais e as soluções NoSQL.

VI. SGBDs x NoSQL

Quando se analisa a possibilidade de se optar por uma estratégia NoSQL em detrimento de um SGBD tradicional, é preciso levar em consideração algumas questões básicas, como, por exemplo, os critérios de escalonamento, consistência de dados e disponibilidade. A seguir, são apresentadas discussões comparativas a respeito desses três critérios.

A questão do escalonamento é essencial porque é justamente nesse ponto em que os bancos NoSQL apresentam as principais vantagens em relação aos SGBDs relacionais, basicamente pelo fato de terem sido criados para esse fim, enquanto os sistemas relacionais possuem uma estruturação menos flexível e menos adaptada para cenários em que o escalonamento faz-se necessário.

Seja uma aplicação determinada aplicação web, com arquitetura simples conforme ilustrado na Figura 2, para se promover opções de atendimento ao número crescente de usuários, os responsáveis pelo sistema poderiam optar pelo *upgrade* no servidor, solução também conhecida como Escalonamento Vertical (*scale up*); ou pelo aumento na quantidade de servidores, solução também conhecida como Escalonamento Horizontal (*scale out*), o qual pode ser visualizado na Figura 3.

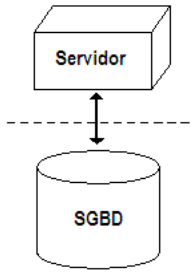

Figura 2. Arquitetura Tradicional de uma Aplicação Web

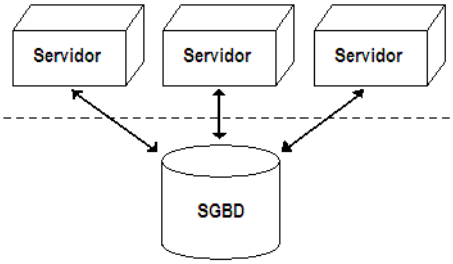

Figura 3. Utilização do Escalonamento Horizontal

O Escalonamento Vertical, que se caracteriza por sua simplicidade, tem sido tradicionalmente mais indicado para as camadas de banco de dados, enquanto o Escalonamento Horizontal tem sido mais utilizado nas camadas das aplicações, principalmente para a Web.

O próximo passo consiste em tentar escalonar o próprio banco de dados. A idéia básica é distribuir o banco em várias máquinas, utilizando-se do particionamento dos dados. Esse processo é também conhecido como *sharding*. No entanto, esta abordagem esbarra nas características do SGBD, dada a dificuldade em se adaptar a toda a estrutura lógica do Modelo Relacional à solução proposta. A Figura 4 ilustra essa abordagem.

A dificuldade em se aplicar o processo de *sharding* em bancos de dados relacionais está relacionada com alguns fatores importantes. Primeiro, enquanto os dados de um SGBD relacional obedecem o critério de normalização, o processo de *sharding* se caracteriza justamente pelo inverso: a desnormalização dos dados. Dados que são utilizados em conjunto são armazenados em conjunto.

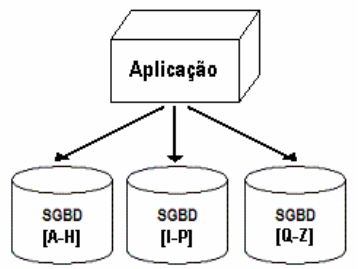

Figura 4. Esquema de *sharding*

Segundo, existe uma mudança de paradigma em relação ao processo de escalonamento. Enquanto os SGBDs relacionais aplicam a estratégia de escalonamento vertical, ou seja, reforçar o servidor; o processo de *sharding* objetiva trabalhar com o escalonamento horizontal, paralelizando seus dados em vários servidores.

Terceiro, o próprio volume dos dados por máquina é sensivelmente minimizado devido ao processo de distribuição. Conjuntos de dados menores são mais fáceis de serem acessados, atualizados e gerenciados.

Finalmente, o fator mais importante, o grau de disponibilidade do sistema é otimizado em relação à abordagem relacional, justamente pelo critério já discutido anteriormente referente ao fato de que a queda de uma máquina não causa a interrupção de todo o sistema.

Como exemplo disso, pode ser citado o caso do Twitter. Em 2008, enquanto ainda utilizava o PostgreSQL, o site que funciona como rede social ficou 84 horas for do ar. Em 2009, após a utilização do Cassandra, esse tempo foi sensivelmente reduzido para 23 horas e 45 minutos [17].

Pelos fatores anteriormente expostos, os principais benefícios dessa abordagem podem ser identificados como: maior disponibilidade, menor tempo de resposta para consultas, paralelismo de atualização de dados e maior grau de concorrência.

Os bancos de dados NoSQL foram especialmente projetados para atender essas características de forma mais natural. O MongoDB inclui um módulo de *sharding* automático que permite a construção de um *cluster* de banco de dados escalável horizontalmente projetado para incorporar novas máquinas de forma dinâmica [15].

Nessa arquitetura, conforme ilustrado na Figura 5, um *cluster* consiste de dois ou mais blocos de servidores chamados de *shards*, um ou mais servidores de configuração – os quais armazenam os metadados do banco – e qualquer número de processos roteadores chamados de *mongos* através dos quais os servidores da aplicação se conectam [15].

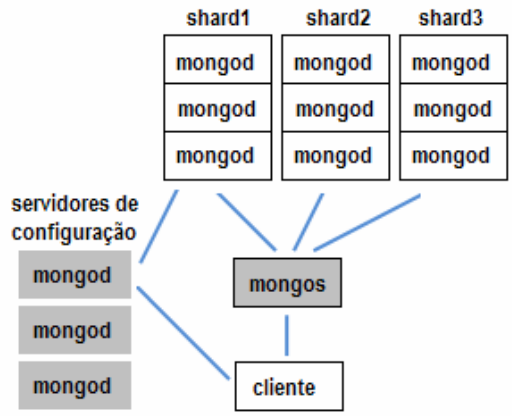

Figura 5. Arquitetura de *sharding* do MongoDB

Seja um cenário no qual existem inicialmente, por exemplo, três servidores, cada qual armazenando partes distintas do banco de dados, a aplicação se conecta ao *cluster* de nós através de um *mongos* o qual é responsável pelo roteamento das operações ao destino apropriado. Esse local de destino, conhecido como *shard*, consiste de dois ou mais servidores, cada um contendo uma réplica da porção do banco armazenado pelo *shard* [13].

O gerenciamento de falhas se dá através da substituição automática do servidor falho por um novo servidor. Assim, cada *shard* sempre estará on-line.

Apesar de, para a aplicação, o cluster continuar sendo visto como um banco de dados não-distribuído, a capacidade do sistema é otimizada. Atualizações e leituras de dados são distribuídas através dos três servidores *shard*.

Outra comparação interessante entre as abordagens relacionais e NoSQL se refere ao controle de concorrência. Enquanto nos primeiros esse processo se utiliza dos bloqueios (*locks*) para garantir que dois usuários não estejam atualizando o mesmo item de dados no mesmo instante, nas soluções NoSQL são utilizadas outras estratégias que permitem um maior grau de concorrência.

O CouchDB, por exemplo, utilizada uma estratégia chamada de Controle de Concorrência de Multi-versões (MVCC). Neste caso, as requisições aos dados podem ser realizadas em paralelo. A idéia é criar múltiplas versões dos

documentos e permitir a atualização sobre uma dessas versões, mantendo ainda a versão desatualizada. Assim, não há necessidade de se bloquear os itens de dados. A Figura 6 ilustra o contraste entre o sistema de *locking* dos SGBDs tradicionais e o MVCC do CouchDB [1].

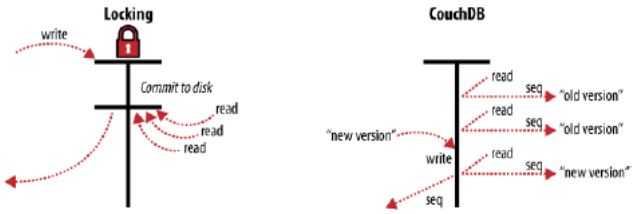

Figura 6. Controle de Concorrência: Bloqueios x MVCC. Fonte: [1]

Ao se substituir um SGBD relacional por uma solução NoSQL, a arquitetura perde em consistência, mas pode ganhar em flexibilidade, disponibilidade e performance. Tal idéia se baseia no Teorema de Eric Brewer conhecido como Teorema CAP (*Consistency*, *Availability* e *Partition Tolerance*), o qual afirma que é impossível para um sistema computacional distribuído garantir, de forma simultânea, consistência, disponibilidade e tolerância ao particionamento. Segundo esse teorema, um sistema distribuído pode garantir apenas duas dessas três características simultaneamente [2].

O tipo de consistência utilizada nos bancos de dados NoSQL é chamada de Consistência Eventual: o sistema de armazenamento garante que se nenhuma nova atualização for realizada sobre o objeto, eventualmente todos os acessos à esse objeto retornarão o último valor atualizado. Quando uma escrita for realizada no banco, não se pode garantir que, a partir daquele momento, todos os outros processos terão acesso apenas ao dado atualizado. Estes processos estarão apenas eventualmente capacitados para perceber tais mudanças [11].

Esse conceito é estendido para o paradigma chamado BASE (***B**asically **A**vailable, **S**oft state, **E**ventual consistency*) que se caracteriza por ser basicamente disponível, ou seja, o sistema parece estar funcionando o tempo todo; em estado leve, o sistema não precisa ser consistente o tempo todo; e eventualmente consistente, o sistema torna-se consistente no momento devido [10].

Esse modelo entra em contraste com o paradigma ACID (***A**tomicity, **C**onsistency, **I**solation, **D**urability*) comumente associado aos SGBDs relacionais. Enquanto o modelo ACID força a consistência ao final de cada operação, o modelo BASE permite que o banco de dados esteja eventualmente em um estado consistente. A disponibilidade do modelo BASE está relacionada ao fato de que a queda de uma máquina do sistema não leva o sistema como um todo a ser interrompido, representando apenas uma máquina a menos disponível [10].

VII. Conclusões

A decisão de se realizar uma mudança dessa natureza – optar por um abordagem NoSQL em contraste com uma abordagem de um SGSBD tradicional – deve levar em consideração as necessidades do problema.

Os fatores que devem ser utilizados como critérios de comparação são de tipos diversos, indo desde a escalabilidade do sistema, passando por questões de consistência de dados, até problemas de facilidade de uso ou existência ou não de linguagens de consulta.

A comparação realizada neste trabalho foi baseada principalmente em três fatores: escalabilidade, consistência dos dados e disponibilidade do sistema. A Tabela 1 apresenta um quadro comparativo entre o paradigma relacional e o paradigma NoSQL conforme esses três critérios escolhidos e analisados.

Como pontos positivos dos SGBDs relacionais deve-se observar que tais sistemas são soluções muito mais maduras e experimentadas, enquanto que as implementações NoSQL ainda estão definindo um padrão próprio. Além disso, a consistência de dados continua sendo um fator a ser considerado com atenção e as transações dos SGBDs relacionais ainda são a melhor forma de se trabalhar com esse problema.

Além disso, a utilização de operações de junções entre os dados, ainda que pontuais, pode tornar-se bastante custosa. Apesar do fato dos bancos NoSQL serem projetados para não precisar das junções, ainda podem ocorrer consultas que necessitem de operações dessa natureza, ainda que não no mesmo nível que faz-se necessário em sistemas relacionais comuns.

Tabela 1. Análise Comparativa Modelo Relacional x NoSQL

	Relacional	NoSQL
Escalonamento	Possível, mas complexo. Devido à natureza estruturada do modelo, a adição de forma dinâmica e transparente de novos nós no *grid* não é realizada de modo natural.	Uma das principais vantagens desse modelo. Por não possuir nenhum tipo de esquema pré-definido, o modelo possui maior flexibilidade o que favorece a inclusão transparente de outros elementos.
Consistência	Ponto mais forte do modelo relacional. As regras de consistência presentes propiciam uma maior grau de rigor quanto à consistência das informações.	Realizada de modo eventual no modelo: só garante que, se nenhuma atualização for realizada sobre o item de dados, todos os acessos a esse item devolverão o último valor atualizado.
Disponibilidade	Dada a dificuldade de se conseguir trabalhar de forma eficiente com a distribuição dos dados, esse modelo pode não suportar a demanda muito grande de informações do banco.	Outro fator fundamental do sucesso desse modelo. O alto grau de distribuição dos dados propicia que um maior número de solicitações aos dados seja atendida por parte do sistema e que o sistema fique menos tempo não-disponível.

Outro fator importante a ser considerado é a ausência de

uma linguagem de consulta, como é o caso do SQL para os SGBDs relacionais. Não existe, em qualquer abordagem noSQL, nada que se aproxime da simplicidade e expressividade oferecida pelo SQL. Adicionalmente, perde-se toda a funcionalidade oferecida pela linguagem, tais como funções, rotinas, etc. Além disso, deixa-se de utilizar a mais simples restrição de integridade sobre o banco, o que pode tornar a aplicação mais pesada.

Adicionalmente, deve-se ter em mente que a escalabilidade em alto grau faz-se necessária apenas em bancos de grande porte, nos quais a alta disponibilidade é imprescindível. Utilizar uma solução NoSQL para bancos de dados nos quais a disponibilidade não seja um fator imprescindível ainda é uma abordagem discutível.

REFERENCES

[1] J. C. Anderson, N. Slater, J. Lehnardt,, “CouchDB: The Definitive Guide”, 1ª edição, O'Reilly Media, 2009.

[2] E. Brewer. “Towards Robust Distributed Systems”. (Invited Talk). Principles of Distributed Computing (PODC), Portland, Oregon, Julho 200.

[3] C. Chandler, “CouchDB in Action”, 1ª edição, Manning Publications, 2009.

[4] F. Chang , J. Dean , S. Ghemawat , W. C. Hsieh , D. A. Wallach , M. Burrows , T. Chandra , A. Fikes , R. E. Gruber, “Bigtable: A distributed storage system for structured data”, In Proceedings of the 7th Conference on Usenix Symposium on Operating Systems Design And Implementation, Volume 7, 2006.

[5] E. F. Codd, “A Relational Model of Data for Large Shared Data Banks”, Communications of the ACM, Volume 13, nº 6, Junho de 1970, p. 377-387.

[6] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall e W. Vogels. "Dynamo: Amazon's Highly Available Key-Value Store", ACM SIGOPS Operating Systems Review, Volume 41, Edição 6, Dezembro 2007, SOSP '07, p. 205–220.

[7] A. Lakshman, P. Malik, “Cassandra - A Decentralized Structured Storage System”, LADIS 2009

[8] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," Computer, vol. 43, no. 2, pp. 12-14, Feb. 2010.

[9] J. Lennon, “Beginning CouchDB”, 1ª edição, Apress, 2009.

[10] D. Pritchett, “BASE: An Acid Alternative”, ACM Queue vol. 6, no. 3, Julho 2008.

[11] W. Vogels, “Eventually Consistent”, Scalable Web Services, Volume 6 No. 6, Outubro de 2008.

[12] “Introducing JSON”. http://json.org/. Acessado em 14/05/2010.

[13] “MongoDB: Sharding Introduction”. http://api.mongodb.org/wiki/current/Sharding Introduction.html. Acessado em 14/05/2010.

[14] “NoSQL Relational Database Management System: Home Page”. Strozzi.it. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home Page. Acessado em 13/04/2010.

[15] “Official MongoDB Project Website”. http://www.mongodb.org/display/DOCS/Home” . Acessado em 13/04/2010.

[16] “The CouchDBProject”. http://couchdb.apache.org/index.html. Acessado em 14/05/2010

[17] “Twitter growth prompts switch from MySQL to 'NoSQL' database”. http://www.computerworld.com/s/article/9161078/Twitter_growth_prompts_switch_from_MySQL_to_NoSQL_database. Acessado em 14/05/2010.